Bioinformatics Toolbox™

Reference

MATLAB®

MathWorks®

How to Contact MathWorks

Bioinformatics Toolbox™ Reference

Revision History

Contents

Alphabetical List

1

Alphabetical List

aa2int

Convert amino acid sequence from letter to integer representation

Syntax

SeqInt = aa2int(*SeqChar*)

Input Arguments

SeqChar	One of the following: • String of single-letter codes specifying an amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers. Unknown characters are mapped to 0. Integers are arbitrarily assigned to IUB/IUPAC letters. • MATLAB® structure containing a Sequence field that contains an amino acid sequence, such as returned by fastaread, getgenpept, genpeptread, getpdb, or pdbread.

Output Arguments

SeqInt	Amino acid sequence specified by a row vector of integers.

Description

SeqInt = aa2int(*SeqChar*) converts *SeqChar*, a character string of single-letter codes specifying an amino acid sequence, to *SeqInt*, a row vector of integers specifying the same amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers.

Mapping Amino Acid Letter Codes to Integers

Amino Acid	Code	Integer
Alanine	A	1
Arginine	R	2
Asparagine	N	3
Aspartic acid (Aspartate)	D	4
Cysteine	C	5
Glutamine	Q	6
Glutamic acid (Glutamate)	E	7
Glycine	G	8
Histidine	H	9
Isoleucine	I	10
Leucine	L	11
Lysine	K	12
Methionine	M	13
Phenylalanine	F	14
Proline	P	15
Serine	S	16
Threonine	T	17
Tryptophan	W	18
Tyrosine	Y	19
Valine	V	20
Asparagine or Aspartic acid (Aspartate)	B	21
Glutamine or Glutamic acid (Glutamate)	Z	22
Unknown amino acid (any amino acid)	X	23
Translation stop	*	24
Gap of indeterminate length	-	25
Unknown character (any character or symbol not in table)	?	0

Examples

Converting a Simple Sequence

Convert the sequence of letters MATLAB to integers.

```
SeqInt = aa2int('MATLAB')

SeqInt =

   13    1   17   11    1   21
```

Converting a Random Sequence

1   Create a random string to represent an amino acid sequence.

```
SeqChar = randseq(20, 'alphabet', 'amino')

SeqChar =

    dwcztecakfuecvifchds
```

2   Convert the amino acid sequence from letter to integer representation.

```
SeqInt = aa2int(SeqChar)

SeqInt =

  Columns 1 through 13
     4   18    5   22   17    7    5    1   12   14    0    7    5

  Columns 14 through 20
    20   10   14    5    9    4   16
```

See Also

aminolookup | int2aa | int2nt | nt2int

getStop

Class: BioMap

Compute stop positions of aligned read sequences from `BioMap` object

Syntax

Stop = getStop(*BioObj*)
Stop = getStop(*BioObj*, *Subset*)

Description

Stop = getStop(*BioObj*) returns *Stop*, a vector of integers specifying the stop position of aligned read sequences with respect to the position numbers in the reference sequence from a `BioMap` object.

Stop = getStop(*BioObj*, *Subset*) returns a stop position for only read sequences specified by *Subset*.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector

- Cell array of strings containing valid sequence headers

Note: If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

Stop

Vector of integers specifying the stop position of aligned read sequences with respect to the position numbers in the reference sequence. *Stop* includes the stop positions for only read sequences specified by *Subset*.

Examples

Construct a `BioMap` object, and then compute the stop position for different sequences in the object:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Compute the stop position of the second sequence in the object
Stop_2 = getStop(BMObj1, 2)

Stop_2 =

        37

% Compute the stop positions of the first and third sequences in
% the object
Stop_1_3 = getStop(BMObj1, [1 3])

Stop_1_3 =

        36
        39

% Compute the stop positions of all sequences in the object
```

```
Stop_All = getStop(BMObj1);
```

See Also
getStart | BioMap

How To
- "Manage Short-Read Sequence Data in Objects"

Related Links
- Sequence Read Archive
- SAM format specification

filterByFlag

Class: BioMap

Filter sequence reads by SAM flag

Syntax

```
Indices = filterByFlag(BioObj, FlagName, FlagValue)
Indices = filterByFlag(BioObj, Subset, FlagName, FlagValue)
Indices = filterByFlag(..., FlagName1, FlagValue1, FlagName2,
FlagValue2, ...)
```

Description

Indices = filterByFlag(*BioObj*, *FlagName*, *FlagValue*) returns *Indices*, a vector of logical indices, indicating the read sequences in *BioObj*, a `BioMap` object, with *FlagName* set to *FlagValue*.

Indices = filterByFlag(*BioObj*, *Subset*, *FlagName*, *FlagValue*) returns *Indices*, a vector of logical indices, indicating the read sequences that meet the specified criteria from a subset of entries in a `BioMap` object.

Indices = filterByFlag(..., *FlagName1*, *FlagValue1*, *FlagName2*, *FlagValue2*, ...) applies multiple flag filters in a single statement.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

Subset

Either of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector

Default:

FlagName

String specifying one of the following flags to filter by:

- `'pairedInSeq'` — The read is paired in sequencing, regardless if it is mapped as a pair.
- `'pairedInMap'` — The read is mapped in a proper pair.
- `'unmappedQuery'` — The read is unmapped.
- `'unmappedMate'` — The mate is unmapped.
- `'strandQuery'` — Strand direction of the read (`0` = forward, `1` = reverse).
- `'strandMate'` — Strand direction of the mate (`0` = forward, `1` = reverse).
- `'readIsFirst'` — The read is first in a pair.
- `'readIsSecond'` — The read is second in a pair.
- `'alnNotPrimary'` — The read's alignment is not primary.
- `'failedQualCheck'` — The read fails platform or vendor quality checks.
- `'duplicate'` — The read is a PCR or optical duplicate.

Default:

FlagValue

Logical value indicating the status of a flag. A `0` indicates `false` or forward, and a `1` indicates `true` or reverse.

Default:

Output Arguments

Indices

Vector of logical indices, indicating the read sequences in *BioObj* with *FlagName* set to *FlagValue*.

Examples

Construct a `BioMap` object, and then determine the read sequences that are both mapped in a proper pair and first in a pair:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Filter the elements using 'pairedInMap' and 'readIsFirst' flags
Indices = filterByFlag(BMObj1, 'pairedInMap', true,...
                       'readIsFirst', true);
% Return the headers of the filtered elements
filtered_Headers = BMObj1.Header(Indices);
```

See Also
`BioMap`

How To
• "Manage Short-Read Sequence Data in Objects"

Related Links
• Sequence Read Archive
• SAM format specification

getAlignment

Class: BioMap

Construct alignment represented in `BioMap` object

Syntax

```
Alignment = getAlignment(BioObj, StartPos, EndPos)
Alignment = getAlignment(BioObj, StartPos, EndPos, R)
Alignment = getAlignment(..., 'ParameterName', ParameterValue)
[Alignment, Indices] = getAlignment(...)
```

Description

Alignment = getAlignment(*BioObj*, *StartPos*, *EndPos*) returns *Alignment*, a character array containing the aligned read sequences from *BioObj*, a `BioMap` object. The read sequences must align within a specific region of the reference sequence, which is defined by *StartPos* and *EndPos*, two positive integers such that *StartPos* is less than *EndPos*, and both are smaller than the length of the reference sequence.

Alignment = getAlignment(*BioObj*, *StartPos*, *EndPos*, *R*) selects the reference where `getAlignment` reconstructs the alignment.

Alignment = getAlignment(..., '*ParameterName*', *ParameterValue*) accepts one or more comma-separated parameter name/value pairs. Specify *ParameterName* inside single quotes.

[*Alignment*, *Indices*] = getAlignment(...) returns *Indices*, a vector of indices specifying the read sequences that align within a specific region of the reference sequence.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

StartPos

Positive integer that defines the start of a region of the reference sequence. *StartPos* must be less than *EndPos*, and smaller than the total length of the reference sequence.

Default:

EndPos

Positive integer that defines the end of a region of the reference sequence. *EndPos* must be greater than *StartPos*, and smaller than the total length of the reference sequence.

Default:

R

Positive integer indexing the SequenceDictionary property of *BioObj*, or a string specifying the actual name of the reference.

Parameter Name/Value Pairs

'OffsetPad'

Specifies if padding blanks are added at the beginning of each aligned sequence to represent the offset of the start position of each aligned sequence with respect to the reference. Choices are true or false (default).

Default:

Output Arguments

Alignment

Character array containing the aligned read sequences from *BioObj* that align within a specific region of the reference sequence. Each row of the character array contains one aligned sequence, that is, the sequence positions that fall within the specified region of the reference sequence. Each aligned sequence can include gaps.

Indices

Vector of indices specifying the read sequences from *BioObj* that align within a specific region of the reference sequence.

Examples

Construct a `BioMap` object, and then reconstruct the alignment between positions 10 and 25 of the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Construct the alignment between positions 10 and 25 of the
% reference sequence.
Alignment = getAlignment(BMObj1, 10, 25)

Alignment =

CTCATTGTAAATGTGT
CTCATTGTAAATGTGT
CTCATTGTAAATGTGT
CTCATTGTAATTTTTT
CTCATTGTAAATGTGT
   ATTGTAAATGTGT
   ATTGTAAATGTGT
     TGTAAATGTGT
         AAATGTGT
             GTGT
             GTGT
               GT
```

Algorithms

`getAlignment` assumes the reference sequence has no gaps. Therefore, positions in reads corresponding to insertions (I) and padding (P) do not appear in the alignment.

Because soft clipped positions (S) are not associated with positions that align to the reference sequence, they do not appear in the alignment.

A skipped position (N) appears as a `.` (period) in the alignment.

Hard clipped positions (H) do not appear in the sequences or the alignment.

See Also
`BioMap` | `getBaseCoverage` | `getCompactAlignment` | `align2cigar` | `cigar2align`

How To
- "Manage Short-Read Sequence Data in Objects"

Related Links
- Sequence Read Archive
- SAM format specification

getBaseCoverage

Class: BioMap

Return base-by-base alignment coverage of reference sequence in `BioMap` object

Syntax

```
Cov = getBaseCoverage(BioObj, StartPos, EndPos)
Cov = getBaseCoverage(BioObj, StartPos, EndPos, R)
Cov = getBaseCoverage(..., Name,Value)
[Cov, BinStart] = getBaseCoverage(...)
```

Description

Cov = getBaseCoverage(*BioObj*, *StartPos*, *EndPos*) returns *Cov*, a row vector of nonnegative integers. This vector indicates the base-by-base alignment coverage of a range or set of ranges in the reference sequence in *BioObj*, a `BioMap` object. The range or set of ranges are defined by *StartPos* and *EndPos*. *StartPos* and *EndPos* can be two nonnegative integers such that *StartPos* is less than *EndPos*, and both integers are smaller than the length of the reference sequence. *StartPos* and *EndPos* can also be two column vectors representing a set of ranges (overlapping or segmented). When *StartPos* and *EndPos* specify a segmented range, *Cov* contains NaN values for base positions between segments.

Cov = getBaseCoverage(*BioObj*, *StartPos*, *EndPos*, *R*) selects the reference where getBaseCoverage calculates the coverage.

Cov = getBaseCoverage(..., Name,Value) returns alignment coverage information with additional options specified by one or more Name,Value pair arguments.

[*Cov*, *BinStart*] = getBaseCoverage(...) returns *BinStart*, a row vector of positive integers specifying the start position of each bin (when binning occurs).

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

StartPos

Either of the following:

- Nonnegative integer that defines the start of a range in the reference sequence. *StartPos* must be less than *EndPos* and smaller than the total length of the reference sequence.
- Column vector of nonnegative integers, each defining the start of a range in the reference sequence.

Default:

EndPos

Either of the following:

- Nonnegative integer that defines the end of a range in the reference sequence. *EndPos* must be greater than *StartPos* and smaller than the total length of the reference sequence.
- Column vector of nonnegative integers, each defining the end of a range in the reference sequence.

Default:

R

Positive integer indexing the `SequenceDictionary` property of *BioObj*, or a string specifying the actual name of the reference.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'binWidth'

Positive integer specifying the bin width, in number of base pairs (bp). Bins are centered within min(*StartPos*) and max(*EndPos*). Thus, the first and last bins span approximately equally outside the range from min(*StartPos*) to max (*EndPos*).

Note: You cannot specify both binWidth and numberOfBins.

Default:

'numberOfBins'

Positive integer specifying the number of equal-width bins to use to span the requested region. Bins are centered within min(*StartPos*) and max(*EndPos*). Thus, the first and last bins span approximately equally outside the range from min(*StartPos*) to max (*EndPos*).

Note: You cannot specify both binWidth and numberOfBins.

Default:

'binType'

String specifying the binning algorithm. Choices are:

- 'max' — From the bin, getBaseCoverage selects the base position with the most reads aligned to it, then uses its alignment coverage value for the bin.
- 'min' — From the bin, getBaseCoverage selects the base position with the least reads aligned to it, then uses its alignment coverage value for the bin.
- 'mean' — Uses the average alignment coverage, computed from all base positions within the bin.

Default: 'max'

'complementRanges'

Specifies whether to return the alignment coverage for the base positions between segments, instead of within segments. If true, the length of *Cov* is

numel(min(*StartPos*):max(*EndPos*)), and *Cov* contains NaN values for base positions within segments.

Default: false

'Spliced'

Logical specifying whether short reads are spliced during mapping (as in mRNA-to-genome mapping). N symbols in the Signature property of the object are not counted.

Default: false

Output Arguments

Cov

Row vector of nonnegative integers. This vector specifies the number of read sequences that align with each base position or bin in the requested regions. A set of ranges can be overlapping or segmented. For a range, the length of *Cov* is numel(*StartPos:EndPos*). For a segmented range, the length of *Cov* is numel(min(*StartPos*):max(*EndPos*)). *Cov* contains NaN values for base positions between segments. When binning occurs, the number of elements in *Cov* equals the number of bins.

BinStart

Row vector of positive integers specifying the start position of each bin. *BinStart* is the same length as *Cov*. If no binning occurs, then *BinStart* equals min(*StartPos*):max(*EndPos*).

Examples

Construct a BioMap object, and then return the alignment coverage of each of the first 12 base positions of the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the number of reads that align to each of
% the first 12 base positions of the reference sequence
cov = getBaseCoverage(BMObj1, 1, 12)
```

```
cov =

     1     1     2     2     3     4     4     4     5     5     5     5
```

Construct a `BioMap` object, and then return the alignment coverage of the range between 1 and 1000, on a bin-by-bin basis, using bins with a width of 100 bp:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the number of reads that align to each 100-bp bin
% in the 1:1000 range of the reference sequence. Also return the
% start position of each bin
[cov, bin_starts] = getBaseCoverage(BMObj1, 1, 1000, 'binWidth', 100)

cov =

    17    20    41    44    45    48    48    45    46    42


bin_starts =

     1   101   201   301   401   501   601   701   801   901
```

See Also

getCounts | getAlignment | BioMap | getIndex | getCompactAlignment | align2cigar | cigar2align

How To

• "Manage Short-Read Sequence Data in Objects"

Related Links

• Sequence Read Archive
• SAM format specification

getCompactAlignment

Class: BioMap

Construct compact alignment represented in `BioMap` object

Syntax

```
CompAlignment = getCompactAlignment(BioObj, StartPos, EndPos)
CompAlignment = getCompactAlignment(BioObj, StartPos, EndPos, R)
CompAlignment = getCompactAlignment(..., 'ParameterName',
ParameterValue)
[CompAlignment, Indices] = getCompactAlignment(...)
[CompAlignment, Indices, Rows] = getCompactAlignment(...)
```

Description

CompAlignment = getCompactAlignment(*BioObj*, *StartPos*, *EndPos*) returns *CompAlignment*, a character array containing the aligned read sequences from *BioObj*, a `BioMap` object, in a compact format. The read sequences must align within a specific region of the reference sequence, which is defined by *StartPos* and *EndPos*, two positive integers such that *StartPos* is less than *EndPos*, and both are smaller than the length of the reference sequence.

CompAlignment = getCompactAlignment(*BioObj*, *StartPos*, *EndPos*, *R*) selects the reference where `getCompactAlignment` reconstructs the alignment.

CompAlignment = getCompactAlignment(..., '*ParameterName*', *ParameterValue*) accepts one or more comma-separated parameter name/value pairs. Specify *ParameterName* inside single quotes.

[*CompAlignment*, *Indices*] = getCompactAlignment(...) returns *Indices*, a vector of indices specifying the read sequences that align within a specific region of the reference sequence.

[*CompAlignment*, *Indices*, *Rows*] = getCompactAlignment(...) returns *Rows*, a vector of positive numbers specifying the row in *CompAlignment* where each read sequence is best displayed.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

StartPos

Positive integer that defines the start of a region of the reference sequence. *StartPos* must be less than *EndPos*, and smaller than the total length of the reference sequence.

Default:

EndPos

Positive integer that defines the end of a region of the reference sequence. *EndPos* must be greater than *StartPos*, and smaller than the total length of the reference sequence.

Default:

R

Positive integer indexing the `SequenceDictionary` property of *BioObj*, or a string specifying the actual name of the reference.

Parameter Name/Value Pairs

'Full'

Specifies whether or not to include only the read sequences that fully align with the defined region of the reference sequence, that is, they are completely contained within the region, and do not extend beyond the region. Choices are `true` or `false` (default).

Default: `false`

'TrimAlignment'

Specifies whether or not to trim empty leading and trailing columns from the alignment. Choices are `true` or `false`. Default is `false`, which does not trim the alignment, but includes any empty leading or trailing columns, and returns an alignment always of length *EndPos* − *StartPos* + 1.

Default: `false`

Output Arguments

`CompAlignment`

Character array containing the aligned read sequences from *`BioObj`* that align within the requested region. The character array represents a compact alignment, that is each row of the character array contains one or more aligned sequences, such that the number of rows in the character array is minimized. Each aligned sequence includes only the sequence positions that fall within the requested region, and each aligned sequence can include gaps.

`Indices`

Vector of indices specifying the read sequences from *`BioObj`* that align within the requested region.

`Rows`

Vector of positive numbers specifying the row in *`CompAlignment`* where each read sequence is best displayed.

Examples

Construct a `BioMap` object, and then construct the compact alignment between positions 30 and 59 of the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Construct the compact alignment between positions 30 and 59 of
% the reference sequence, and return the indices of the reads in the
% compact alignment, as well as the row each read is in.
[CompAlignment, Ind, Row] = getCompactAlignment(BMObj1, 30, 59)

CompAlignment =

TAACTCG       GCCCAGCATTAGGGAGC
TAACTCGT             CATTAGGGAGC
TAACTCGTCC            ATTAGGGAGC
TAACTCTTCTCT          TTAGGGAGC
TAACTCGTCCATGG        TAGGGAGC
```

```
TAACTCGTCCCTGGCCCA                C
TAACTCGTCCATGGCCCAG
TAACTCGTCCATTGCCCAGC
TAACTCGTCCATGGCCCAGCATT
TAACTCGTCCATGGCCCAGCATTTGGG
TAACTCGTCCATGGCCCAGCATTAGGG
TAACTCGTCCATGGCCCAGCATTAGGGAGC
TAACTCGTCCATGGCCCAGCATTAGGGATC
TAACTCGTCCATGGCCCAGCATTAGGGAGC
 AACTCGTCCATGGCCCAGCATTAGGGAGC
        GTACATGGCCCAGCATTAGGGAGC
          TCCATGGCCCAGCATTAGGGCGC
```

```
Ind =

     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
    11
    12
    13
    14
    15
    16
    17
    18
    19
    20
    21
    22
    23


Row =

     1
```

```
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
1
2
3
4
5
6
```

Algorithms

getCompactAlignment assumes the reference sequence has no gaps. Therefore, positions in reads corresponding to insertions (I) and padding (P) do not appear in the alignment.

Because soft clipped positions (S) are not associated with positions that align to the reference sequence, they do not appear in the alignment.

A skipped position (N) appears as a - (hyphen) in the alignment.

Hard clipped positions (H) do not appear in the sequences or the alignment.

See Also
getAlignment | BioMap | getBaseCoverage | align2cigar | cigar2align

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links

- Sequence Read Archive
- SAM format specification

getCounts

Class: BioMap

Return count of read sequences aligned to reference sequence in `BioMap` object

Syntax

```
Count = getCounts(BioObj, StartPos, EndPos)
GroupCount = getCounts(BioObj, StartPos, EndPos, Groups)
GroupCount = getCounts(BioObj, StartPos, EndPos, Groups, R)
... = getCounts(..., Name,Value)
```

Description

Count = getCounts(*BioObj*, *StartPos*, *EndPos*) returns *Count*, a nonnegative integer specifying the number of read sequences in *BioObj*, a `BioMap` object, that align to a specific range or set of ranges in the reference sequence. The range or set of ranges are defined by *StartPos* and *EndPos*. *StartPos* and *EndPos* can be two nonnegative integers such that *StartPos* is less than *EndPos*, and both integers are smaller than the length of the reference sequence. *StartPos* and *EndPos* can also be two column vectors representing a set of ranges (overlapping or segmented).

By default, `getCounts` counts each read only once. Therefore, if a read spans multiple ranges, that read instance is counted only once. When *StartPos* and *EndPos* specify overlapping ranges, the overlapping ranges are considered as one range.

GroupCount = getCounts(*BioObj*, *StartPos*, *EndPos*, *Groups*) specifies *Groups*, a row vector of integers or strings, the same size as *StartPos* and *EndPos*. This vector indicates the group to which each range belongs. *GroupCount* is a column vector containing a number of elements equal to the number of unique elements in *Groups*. *GroupCount* specifies the number of reads that align to each group, in the ascending order of unique groups in *Groups*.

Each group is treated independently. Therefore, a read can be counted in more than one group.

GroupCount = getCounts(*BioObj*, *StartPos*, *EndPos*, *Groups*, *R*) specifies a reference for each of the segmented ranges defined by *StartPos*, *EndPos*, and *Groups*.

`... = getCounts(..., Name,Value)` returns counts with additional options specified by one or more Name,Value pair arguments.

Input Arguments

`BioObj`

Object of the `BioMap` class.

Default:

`StartPos`

Either of the following:

- Nonnegative integer that defines the start of a range in the reference sequence. *StartPos* must be less than *EndPos*, and smaller than the total length of the reference sequence.
- Column vector of nonnegative integers, each defining the start of a range in the reference sequence.

Default:

`EndPos`

Either of the following:

- Nonnegative integer that defines the end of a range in the reference sequence. *EndPos* must be greater than *StartPos*, and smaller than the total length of the reference sequence.
- Column vector of nonnegative integers, each defining the end of a range in the reference sequence.

Default:

`Groups`

Row vector of integers or strings, the same size as *StartPos* and *EndPos*. This vector indicates the group to which each range belongs.

Default:

R

Vector of positive integers indexing the `SequenceDictionary` property of *BioObj*, or a cell array of strings specifying the actual names of references. *R* must be ordered and have the same number of elements as the unique elements in *Groups*. If *R* has the same number of elements as *Groups*, then all of the entries in *R* for each unique value in *Groups* must be the same.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`'Independent'`

Logical that specifies whether to treat the ranges defined by *StartPos* and *EndPos* independently. If `true`, *Count* is a column vector containing the same number of elements as *StartPos* and *EndPos*. In this case, a read that spans multiple ranges, is counted once in each range.

Note: This name-value pair argument is ignored when using the *Groups* input argument, because `getCounts` assumes that each group of ranges is independent.

Default: `false`

`'Overlap'`

Specifies the minimum number of base positions that a read must overlap in a range or set of ranges, to be counted. This value can be any of the following:

- Positive integer
- `'full'` — A read must be fully contained in a range or set of ranges to be counted.
- `'start'` — A read's start position must lie within a range or set of ranges to be counted.

Default: 1

'Spliced'

Logical specifying whether short reads are spliced during mapping (as in mRNA-to-genome mapping). N symbols in the `Signature` property of the object are not counted.

Default: `false`

'Method'

String specifying the method to measure the abundance of reads. Choices are:

- `'raw'` — Raw counts
- `'rpkm'` — Counts of reads per kilobase pairs per million aligned reads
- `'mean'` — Average coverage depth computed base-by-base
- `'max'` — Maximum coverage depth computed base-by-base
- `'min'` — Minimum coverage depth computed base-by-base
- `'sum'` — Sum of all aligned bases in all the reads

Default: `'raw'`

Output Arguments

Count

Either of the following:

- When `Independent` is `false`, this value is a nonnegative integer. The integer specifies the number of reads that align to a range or set of ranges (overlapping or segmented) of the reference sequence in *BioObj*, a `BioMap` object. Each read is counted only once, even if the read spans multiple ranges.
- When `Independent` is `true`, this value is a column vector of nonnegative integers. This vector indicates the number of reads that align to the independent ranges specified by *StartPos* and *EndPos*. This vector contains the same number of elements as *StartPos* and *EndPos*.

GroupCount

Column vector containing a number of elements equal to the number of unique elements in *Groups*. The vector specifies the number of reads that align to each group, in the order

of unique groups in *Groups*. The groups of ranges are treated independently. Therefore, a single read can be counted in more than one group.

Examples

Construct a `BioMap` object, and then return the number of reads that align to at least one base position in two ranges of the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the number of reads that align to the segmented range 1:50 and 71:100
counts_1 = getCounts(BMObj1,[1;71],[50;100])

counts_1 =

    37
```

Construct a `BioMap` object, and then return the number of reads that align to at least one base position in two independent ranges of the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the number of reads that align to each of the ranges,
% 1:50 and 71:100, independent of each other
counts_2 = getCounts(BMObj1,[1;71],[50;100],'independent',true)

counts_2 =

    20
    21
```

Notice that the total number of reads reported in `counts_2` is greater than the number of reads reported in `counts_1`. This difference occurs because there are four reads that span the two ranges, and are counted twice in the second example.

Construct a `BioMap` object, and then return the number of reads that align to two separate groups of ranges of the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the number of reads that align to a group containing range 30:60,
% and also the number of reads that align to a group containing range 1:10
% and range 50:60
```

```
counts_3 = getCounts(BMObj1,[1;30;50],[10;60;60],[2 1 2])

counts_3 =

    25
    22
```

Construct a `BioMap` object, and then return the total number of reads aligned to the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the number of sequences that align to the entire reference sequence
getCounts(BMObj1,min(getStart(BMObj1)),max(getStop(BMObj1)))

ans =

      1482
```

See Also

getIndex | getAlignment | BioMap | getBaseCoverage | getCompactAlignment | align2cigar | cigar2align

How To

· "Manage Short-Read Sequence Data in Objects"

Related Links

· Sequence Read Archive
· SAM format specification

getCoverage

Class: BioMap

Compute read coverage in `BioMap` object

Note: `getCoverage` has been removed. Use `getBaseCoverage`, `getCounts`, or `getIndex` instead.

Syntax

```
Cov = getCoverage(BioObj, StartPos, EndPos)
[Cov, Indices] = getCoverage(BioObj, StartPos, EndPos)
[Cov, Indices, Seqs] = getCoverage(BioObj, StartPos, EndPos)
... = getCoverage(BioObj, StartPos, EndPos, 'ParameterName',
ParameterValue)
```

Description

Cov = getCoverage(*BioObj*, *StartPos*, *EndPos*) returns *Cov*, a nonnegative integer indicating the number of read sequences that cover (align within) a specific region of the reference sequence in *BioObj*, a `BioMap` object. The specific region of the reference sequence is defined by *StartPos* and *EndPos*. *StartPos* and *EndPos* can be two nonnegative integers such that *StartPos* is less than *EndPos*, and both are smaller than the length of the reference sequence. *StartPos* and *EndPos* can also be two column vectors representing a collection of regions of the reference sequence. In this case, *Cov* is a column vector of nonnegative integers indicating the number of read sequences that cover each region.

[*Cov*, *Indices*] = getCoverage(*BioObj*, *StartPos*, *EndPos*) also returns *Indices*, a vector of indices specifying the read sequences that align within a specific region of the reference sequence.

[*Cov*, *Indices*, *Seqs*] = getCoverage(*BioObj*, *StartPos*, *EndPos*) also returns *Seqs*, a cell array of strings containing the read sequences that align within a specific region of the reference sequence.

```
... = getCoverage(BioObj, StartPos, EndPos, 'ParameterName',
ParameterValue) accepts one or more comma-separated parameter name/value pairs.
```
Specify *ParameterName* inside single quotes.

Tips

Use the *Indices* output from the `getCoverage` method as input to other `BioMap` methods. Doing so lets you determine other information about the read sequences in the coverage region, such as header, start position, mapping quality, etc.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

StartPos

Either of the following:

- Nonnegative integer that defines the start of a region of the reference sequence. *StartPos* must be less than *EndPos*, and smaller than the total length of the reference sequence.
- Column vector of nonnegative integers, each defining the start of a region of the reference sequence.

Default:

EndPos

Either of the following:

- Nonnegative integer that defines the end of a region of the reference sequence. *EndPos* must be greater than *StartPos*, and smaller than the total length of the reference sequence.

- Column vector of nonnegative integers, each defining the end of a region of the reference sequence.

Default:

Parameter Name/Value Pairs

`'Base'`

Specifies if the output *Cov* is computed base-by-base, that is determining the number of nongap symbols that align with each position in the specified region of the reference sequence. If `true`, *Cov* is a vector of positive integers corresponding to the base positions in the specified region of the reference sequence.

Default: `false`

`'Full'`

Specifies to include only the read sequences that fully align with the defined region of the reference sequence, that is, they are completely contained within the region, and do not extend beyond the region.

Default: `false`

Output Arguments

`Cov`

Either of the following:

- Nonnegative integer indicating the number of read sequences that cover (align within) a specific region of the reference sequence in *BioObj*.
- Column vector of nonnegative integers indicating the number of read sequences that cover each region specified by *StartPos* and *EndPos*, when they are both column vectors. In this case, *Cov* is the same length as *StartPos* and *EndPos*.

`Indices`

Vector of indices specifying the read sequences from *BioObj* that align within a specific region of the reference sequence.

Seqs

Cell array of strings containing the read sequences from *BioObj* that align within a specific region of the reference sequence. Each string is a sequence read without alignment information.

Examples

Construct a `BioMap` object, and then retrieve the coverage of the first 50 positions of the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the number of sequences that cover the first 50
% positions of the reference sequence
cov = getCoverage(BMObj1, 1, 50)

cov =

    20
```

Construct a `BioMap` object, and then retrieve the starting positions for the read sequences that cover the first 50 positions of the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the number of sequences that cover the first 50
% positions of the reference sequence
% Also retrieve the indices of these sequences
[cov, idx] = getCoverage(BMObj1, 1, 50);
% Use the indices for these sequences to determine their start
% positions
startPositions = getStart(BMObj1, idx);
```

Construct a `BioMap` object, and then retrieve the coverage of the first 50 positions of the reference sequence, considering only read sequences that align fully within the region:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the number of sequences that cover the first 50
% positions of the reference sequence
% Consider only read sequences that align fully within the region
fullCov = getCoverage(BMObj1, 1, 50, 'full', true)
```

```
fullCov =

     8
```

Construct a `BioMap` object, and then retrieve the coverage for the first 10 positions of the reference sequence, on a base-by-base basis:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the number of sequences that cover each base position of
% the first 10 positions of the reference sequence
baseCov = getCoverage(BMObj1, 1, 10, 'base', true)

baseCov =

     1
     1
     2
     2
     3
     4
     4
     4
     5
     5
```

See Also
getAlignment | BioMap | getCompactAlignment | align2cigar | cigar2align

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive
· SAM format specification

getFlag

Class: BioMap

Retrieve read sequence flags from `BioMap` object

Syntax

```
Flag = getFlag(BioObj)
Flag = getFlag(BioObj, Subset)
```

Description

Flag = getFlag(*BioObj*) returns *Flag*, a vector of nonnegative integers indicating the bit-wise information that specifies the status of the 11 flags described by the SAM format specification. Each integer corresponds to one read sequence from a `BioMap` object.

Flag = getFlag(*BioObj*, *Subset*) returns flag integers for only object elements specified by *Subset*.

Tips

After using the `getFlag` method to return the integer specifying the bit-wise information for the SAM flags, use the `bitget` function to determine the status of a specific SAM flag. For more information, see "Examples" on page 1-38.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

Flag

Vector of nonnegative integers. Each integer corresponds to one read sequence and indicates the bit-wise information that specifies the status of the 11 flags described by the SAM format specification. These flags describe different sequencing and alignment aspects of a read sequence. *Flag* includes flag integers for only read sequences specified by *Subset*.

Examples

Construct a `BioMap` object, and then retrieve the SAM flag values for different elements in the object:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve integer specifying bit-wise information for 11
% SAM flags of the second element
flagValue = getFlag(BMObj1, 2)

flagValue =

     73
```

```
% Retrieve integers specifying bit-wise information for 11
% SAM flags of the first and third elements
flagValues = getFlag(BMObj1, [1 3])

flagValues =

     73
    137

% Retrieve integers specifying bit-wise information for 11
% SAM flags of all elements
allFlagValues = getFlag(BMObj1);

% Determine the status of the fourth flag (mate is unmapped)
% for the second element, which has a flag value of 73
bitget(73, 4)

ans =

     1
```

Alternatives

An alternative to using the getFlag method is to use dot indexing with the Flag property:

BioObj.Flag(*Indices*)

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers.

See Also
setFlag | bitget | BioMap

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive

- SAM format specification

getIndex

Class: BioMap

Return indices of read sequences aligned to reference sequence in `BioMap` object

Syntax

```
Indices = getIndex(BioObj, StartPos, EndPos)
Indices = getIndex(BioObj, StartPos, EndPos, R)
Indices = getIndex(..., Name,Value)
```

Description

Indices = getIndex(*BioObj*, *StartPos*, *EndPos*) returns *Indices*, a column vector of indices specifying the read sequences that align to a range or set of ranges in the reference sequence in *BioObj*, a `BioMap` object. The range or set of ranges are defined by *StartPos* and *EndPos*. *StartPos* and *EndPos* can be two nonnegative integers such that *StartPos* is less than *EndPos*, and both integers are smaller than the length of the reference sequence. *StartPos* and *EndPos* can also be two column vectors representing a set of ranges (overlapping or segmented).

`getIndex` includes each read only once. Therefore, if a read spans multiple ranges, the index for that read appears only once.

Indices = getIndex(*BioObj*, *StartPos*, *EndPos*, *R*) selects the reference associated with the range specified by *StartPos* and *EndPos*.

Indices = getIndex(..., Name,Value) returns indices with additional options specified by one or more Name,Value pair arguments.

Tips

Use the *Indices* output from the `getIndex` method as input to other `BioMap` methods. Doing so lets you retrieve other information about the reads in the range, such as header, start position, mapping quality, sequences, etc.

Input Arguments

`BioObj`

Object of the `BioMap` class.

Default:

`StartPos`

Either of the following:

- Nonnegative integer that defines the start of a range in the reference sequence. *StartPos* must be less than *EndPos*, and smaller than the total length of the reference sequence.
- Column vector of nonnegative integers, each defining the start of a range in the reference sequence.

Default:

`EndPos`

Either of the following:

- Nonnegative integer that defines the end of a range in the reference sequence. *EndPos* must be greater than *StartPos*, and smaller than the total length of the reference sequence.
- Column vector of nonnegative integers, each defining the end of a range in the reference sequence.

Default:

`R`

Positive integer indexing the `SequenceDictionary` property of *BioObj*, or a string specifying the actual name of the reference.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single

quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'Overlap'

Specifies the minimum number of base positions that a read must overlap in a range or set of ranges, to be included. This value can be any of the following:

- Positive integer
- `'full'` — A read must be fully contained in a range or set of ranges to be counted.
- `'start'` — A read's start position must lie within a range or set of ranges to be counted.

Default: 1

'Depth'

Specifies to decimate the output indices. The coverage depth at any base position is less than or equal to `Depth`, a positive integer.

Default: Inf

'Spliced'

Logical specifying whether short reads are spliced during mapping (as in mRNA-to-genome mapping). N symbols in the `Signature` property of the object are not counted.

Default: false

Output Arguments

Indices

Column vector of indices specifying the reads that align to a range or set of ranges in the specified reference sequence in *BioObj*, a `BioMap` object.

Examples

Construct a `BioMap` object, and then use the indices of the reads to retrieve the start and stop positions for the reads that are fully contained in the first 50 positions of the reference sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the indices of reads that are fully contained in the
% first 50 positions of the reference sequence
indices = getIndex(BMObj1, 1, 50, 'overlap', 'full');
% Use these indices to return the start and stop positions of
% the reads
starts = getStart(BMObj1, indices)
stops = getStop(BMObj1, indices)

starts =

          1
          3
          5
          6
          9
         13
         13
         15


stops =

         36
         37
         39
         41
         43
         47
         48
         49
```

Construct a `BioMap` object, and then use the indices of the reads to retrieve the sequences for the reads whose alignments overlap a segmented range by at least one base pair:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Return the indices of the reads that overlap the
% segmented range 98:100 and 198:200, by at least 1 base pair
indices = getIndex(BMObj1, [98;198], [100;200], 'overlap', 1);
% Use these indices to return the sequences of the reads
sequences = getSequence(BMObj1, indices);
```

See Also

getStart | getSequence | getAlignment | BioMap | getStop | getCounts |
getBaseCoverage | getCompactAlignment | align2cigar | cigar2align

How To

- "Manage Short-Read Sequence Data in Objects"

Related Links

- Sequence Read Archive
- SAM format specification

getInfo

Class: BioMap

Retrieve information for single element of `BioMap` object

Syntax

Info = getInfo(*BioObj*, *Element*)

Description

Info = getInfo(*BioObj*, *Element*) returns *Info*, a tab-delimited string containing information about a single element in *BioObj*, a `BioMap` object.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

Element

One of the following to specify one element in *BioObj*:

- Scalar specifying an element index
- Logical vector
- String containing a valid sequence header

Default:

Output Arguments

Info

Tab-delimited string containing information about a single element in *BioObj*, a BioMap object. The string contains the information from the following properties in order:

- Header
- Flag
- Start
- MappingQuality
- Signature
- Sequence
- Quality

Examples

Construct a BioMap object, and then retrieve information for the second element in the object:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve information for the second element in the object
element2Info = getInfo(BMObj1, 2)

element2Info =

EAS54_65:7:152:368:113 73 3 99 35M
CTAGTGGCTCATTGTAAATGTGTGGTTTAACTCGT
<<<<<<<<<0<<<<655<<7<<<:9<<3/:<6):
```

See Also
BioMap

How To
- "Manage Short-Read Sequence Data in Objects"

Related Links

- Sequence Read Archive
- SAM format specification

getMappingQuality

Class: BioMap

Retrieve sequence mapping quality scores from `BioMap` object

Syntax

```
MappingQuality = getMappingQuality(BioObj)
MappingQuality = getMappingQuality(BioObj, Subset)
```

Description

MappingQuality = getMappingQuality(*BioObj*) returns *MappingQuality*, a vector of integers specifying mapping quality scores for each read sequence in *BioObj*, a `BioMap` object.

MappingQuality = getMappingQuality(*BioObj*, *Subset*) returns mapping quality scores for only object elements specified by *Subset*.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

MappingQuality

MappingQuality property of a subset of elements in *BioObj.MappingQuality* is a vector of integers specifying the mapping quality scores for read sequences specified by *Subset*.

Examples

Construct a BioMap object, and then retrieve the mapping quality scores for different elements in the object:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the mapping quality property of the second element in
% the object
MQ_2 = getMappingQuality(BMObj1, 2)

MQ_2 =

    99

% Retrieve the mapping quality properties of the first and third
% elements in the object
MQ_1_3 = getMappingQuality(BMObj1, [1 3])

MQ_1_3 =

    99
    99

% Retrieve the mapping quality properties of all elements in the
% object
MQ_All = getMappingQuality(BMObj1);
```

Alternatives

An alternative to using the `getMappingQuality` method is to use dot indexing with the `MappingQuality` property:

BioObj.MappingQuality(*Indices*)

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers.

See Also
BioMap | setMappingQuality

How To
- "Manage Short-Read Sequence Data in Objects"

Related Links
- Sequence Read Archive
- SAM format specification

getMatePosition

Class: BioMap

Retrieve mate positions of read sequences from `BioMap` object

Syntax

```
MatePos = getMatePosition(BioObj)
MatePos = getMatePosition(BioObj,Subset)
```

Description

`MatePos = getMatePosition(BioObj)` returns MatePos, a vector of nonnegative integers specifying the mate positions of read sequences with respect to the position numbers in the reference sequence from a `BioMap` object.

`MatePos = getMatePosition(BioObj,Subset)` returns mate positions for only read sequences specified by Subset.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

Subset

One of the following to specify a subset of the elements in BioObj:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: If you use a cell array of header strings to specify Subset, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

MatePos

`MatePosition` property of all or a subset of elements in BioObj. MatePos is a vector of nonnegative integers specifying the mate positions of read sequences with respect to the position numbers in the reference sequence. MatePos includes the mate positions for only read sequences specified by Subset.

Not all values in the `MatePosition` vector represent valid mate positions, for example, mates that map to a different reference sequence or mates that do not map. To determine if a mate position is valid, use the `filterByFlag` method with the `'pairedInMap'` flag.

Examples

Construct a `BioMap` object, and then retrieve the mate position for different sequences in the object:

```
% Construct a BioMap object from a SAM file and determine the header for the 17th eleme
BMObj1 = BioMap('ex1.sam');
BMObj1.Header(17)

ans =

    'EAS114_32:5:78:583:499'

% Retrieve the MatePosition property of the 17th element in the object using the header
MatePos_17 = getMatePosition(BMObj1,{'EAS114_32:5:78:583:499'})

MatePos_17 =

        229
         37
```

Notice the previous example returned two mate positions. This is because the header EAS114_32:5:78:583:499 is a repeated header in the BMObj1 object. The getMatePosition method returns mate positions for all elements in the object with that header.

```
% Retrieve the MatePosition properties of the 37th and 47th elements in
% the object
MatePos_37_47 = getMatePosition(BMObj1, [37 47])

MatePos_37_47 =

          95
         283

% Retrieve the MatePosition properties of all elements in the object
MatePos_All = getMatePosition(BMObj1);
```

Alternatives

An alternative to using the getMatePosition method is to use dot indexing with the MatePosition property:

BioObj.MatePosition(*Indices*)

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers.

See Also
filterByFlag | BioMap | setMatePosition

How To
• "Manage Short-Read Sequence Data in Objects"

Related Links
• Sequence Read Archive
• SAM format specification

getReference

Class: BioMap

Retrieve reference sequence from `BioMap` object

Syntax

Ref = getReference(*BioObj*)

Description

Ref = getReference(*BioObj*) returns the name of the reference sequence from a `BioMap` object. This is the `Reference` property of the object.

Input Arguments

BioObj

Object of the `BioRead` or `BioMap` class.

Default:

Output Arguments

Ref

`Reference` property of *BioObj*, the `BioMap` object. It is a string specifying the name of the reference sequence.

Examples

Construct a `BioMap` object, and then retrieve the reference sequence from the object:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the reference sequence from the object
refSeq = getReference(BMObj1)

refSeq =

seq1
```

Alternatives

An alternative to using the `getReference` method is to use dot indexing with the `Reference` property:

BioObj.Reference

See Also
setReference | BioMap

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive
· SAM format specification

getSignature

Class: BioMap

Retrieve signature (alignment information) from `BioMap` object

Syntax

```
Signature = getSignature(BioObj)
Signature = getSignature(BioObj, Subset)
```

Description

Signature = getSignature(*BioObj*) returns *Signature*, a cell array of CIGAR-formatted strings, each representing how a read sequence in a `BioMap` object aligns to the reference sequence.

Signature = getSignature(*BioObj*, *Subset*) returns signature strings for only object elements specified by *Subset*.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

`Signature`

`Signature` property of a subset of elements in *BioObj*. *Signature* is a cell array of CIGAR-formatted strings, each representing how read sequences, specified by *Subset*, align to the reference sequence.

Examples

Construct a `BioMap` object, and then retrieve the signatures for different elements in the object:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the signature property of the second element in
% the object
Sig_2 = getSignature(BMObj1, 2)

Sig_2 =

    '35M'

% Retrieve the signature properties of the first and third
% elements in the object
Sig_1_3 = getSignature(BMObj1, [1 3])

Sig_1_3 =

    '36M'
    '35M'

% Retrieve the signature properties of all elements in the object
Sig_All = getSignature(BMObj1);
```

Alternatives

An alternative to using the getSignature method is to use dot indexing with the Signature property:

BioObj.Sgnature(*Indices*)

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers.

See Also
setSignature | BioMap | getAlignment

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive
· SAM format specification

getStart

Class: BioMap

Retrieve start positions of aligned read sequences from `BioMap` object

Syntax

```
Start = getStart(BioObj)
Start = getStart(BioObj, Subset)
```

Description

Start = getStart(*BioObj*) returns *Start*, a vector of integers specifying the start position of aligned read sequences with respect to the position numbers in the reference sequence from a `BioMap` object.

Start = getStart(*BioObj*, *Subset*) returns a start position for only read sequences specified by *Subset*.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

<hr>

Note: If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

<hr>

Default:

Output Arguments

Start

Start property of a subset of elements in *BioObj*. It is a vector of integers specifying the start position of aligned read sequences with respect to the position numbers in the reference sequence. It includes the start positions for only read sequences specified by *Subset*.

Examples

Construct a BioMap object, and then retrieve the start position for different sequences in the object:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Retrieve the start property of the second element in the object
Start_2 = getStart(BMObj1, 2)

Start_2 =

         3

% Retrieve the start properties of the first and third elements in
% the object
Start_1_3 = getStart(BMObj1, [1 3])

Start_1_3 =

         1
         5

% Retrieve the start properties of all elements in the object
Start_All = getStart(BMObj1);
```

Alternatives

An alternative to using the getStart method is to use dot indexing with the Start property:

BioObj.Start(*Indices*)

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers.

See Also
setStart | BioMap | getStop

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive
· SAM format specification

getSummary

Class: BioMap

Print summary of `BioMap` object

Syntax

```
getSummary(BioObj)
ds = getSummary(BioObj)
```

Description

`getSummary(BioObj)` prints a summary of a `BioMap` object. The summary includes the names of references, the number of sequences mapped to each reference, and the genomic range that the sequences cover in each reference.

`ds = getSummary(BioObj)` returns the summary information in a `dataset` array.

Input Arguments

BioObj

Object of the `BioMap` class.

Output Arguments

ds

`dataset` array containing the summary of the `BioMap` object, BioObj. The dataset array has an observation (row) for each reference in `BioObj`, and two variables (columns): the number of sequences mapped to each reference and the genomic range that the sequences cover in each reference.

`getSummary` stores additional metadata for the `BioMap` object in the `UserData` property of `ds`, which you can access using `ds.Properties.UserData`.

Examples

Construct a `BioMap` object, and then display a summary of the object:

```
% Construct a BioMap object from a SAM file
BMObj2 = BioMap('ex2.sam');
getSummary(BMObj2)

BioMap summary:
                                  Name: ''
                        Container_Type: 'Data is file indexed.'
             Total_Number_of_Sequences: 3307
    Number_of_References_in_Dictionary: 2

            Number_of_Sequences    Genomic_Range
    seq1    1501                    1   1569
    seq2    1806                    1   1567
```

See Also
`BioMap`

How To
·   "Manage Short-Read Sequence Data in Objects"

Related Links
·   Sequence Read Archive
·   SAM format specification

setFlag

Class: BioMap

Set read sequence flags for `BioMap` object

Syntax

```
NewObj = setFlag(BioObj, Flag)
NewObj = setFlag(BioObj, MappingQuality, Subset)
```

Description

NewObj = setFlag(*BioObj*, *Flag*) returns *NewObj*, a new `BioMap` object, constructed from *BioObj*, an existing `BioMap` object, with the `Flag` property set to *Flag*, a vector of nonnegative integers indicating the bit-wise information that specifies the status of each of the 11 flags described by the SAM format specification.

NewObj = setFlag(*BioObj*, *MappingQuality*, *Subset*) returns *NewObj*, a new `BioMap` object, constructed from *BioObj*, an existing `BioMap` object, with the `Flag` property of a subset of the elements set to *Flag*, a vector of nonnegative integers indicating the bit-wise information that specifies the status of each of the 11 flags described by the SAM format specification. It sets the `Flag` property for only the object elements specified by *Subset*.

Tips

To update the `Flag` property in an existing `BioMap` object, use the same object as the input *BioObj* and the output *NewObj*.

Input Arguments

BioObj

Object of the `BioMap` class.

Note: If *BioObj* was constructed from a `BioIndexedFile` object, you cannot set its `Flag` property.

Default:

`Flag`

Vector of nonnegative integers. Each integer corresponds to one read sequence and indicates the bit-wise information that specifies the status of each of the 11 flags described by the SAM format specification. These flags describe different sequencing and alignment aspects of a read sequence.

Default:

`Subset`

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: A one-to-one relationship must exist between the number and order of elements in *Flag* and *Subset*. If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

`NewObj`

Object of the `BioMap` class.

Examples

Construct a `BioMap` object, and then set a subset of the flags:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Set the Flag property of the second element to a new value
BMObj1 = setFlag(BMObj1, 75, 2);
```

Alternatives

An alternative to using the `setFlag` method to update an existing object is to use dot indexing with the `Flag` property:

BioObj.Flag(*Indices*) = *NewFlag*

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers. *NewFlag* is a vector of nonnegative integers indicating the bit-wise information that specifies the status of each of the 11 flags described by the SAM format specification. Each integer corresponds to one read sequence in a `BioMap` object. *Indices* and *NewFlag* must have the same number and order of elements.

See Also
getFlag | BioMap

How To
- "Manage Short-Read Sequence Data in Objects"

Related Links
- Sequence Read Archive
- SAM format specification

setMappingQuality

Class: BioMap

Set sequence mapping quality scores for `BioMap` object

Syntax

```
NewObj = setMappingQuality(BioObj, MappingQuality)
NewObj = setMappingQuality(BioObj, MappingQuality, Subset)
```

Description

NewObj = setMappingQuality(*BioObj*, *MappingQuality*) returns *NewObj*, a new `BioMap` object, constructed from *BioObj*, an existing `BioMap` object, with the `MappingQuality` property set to *MappingQuality*, a vector of integers specifying the mapping quality scores for read sequences.

NewObj = setMappingQuality(*BioObj*, *MappingQuality*, *Subset*) returns *NewObj*, a new `BioMap` object, constructed from *BioObj*, an existing `BioMap` object, with the `MappingQuality` property of a subset of the elements set to *MappingQuality*, a vector of integers specifying the mapping quality scores for read sequences. It sets the mapping quality scores for only the object elements specified by *Subset*.

Tips

To update mapping quality scores in an existing `BioMap` object, use the same object as the input *BioObj* and the output *NewObj*.

Input Arguments

BioObj

Object of the `BioMap` class.

Note: If *BioObj* was constructed from a `BioIndexedFile` object, you cannot set its `MappingQuality` property.

Default:

MappingQuality

Vector of integers specifying the mapping quality scores for read sequences.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: A one-to-one relationship must exist between the number and order of elements in *MappingQuality* and *Subset*. If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

NewObj

Object of the `BioMap` class.

Examples

Construct a `BioMap` object, and then set a subset of the mapping quality scores:

```
% Construct a BioMap object from a SAM file
```

```
BMObj1 = BioMap('ex1.sam');
% Set the Mapping Quality property of the second element to a new
% value
BMObj1 = setMappingQuality(BMObj1, 74, 2);
```

Alternatives

An alternative to using the setMappingQuality method to update an existing object is to use dot indexing with the MappingQuality property:

BioObj.MappingQuality(*Indices*) = *NewMappingQuality*

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers. *NewMappingQuality* is a vector of integers specifying the mapping quality scores for read sequences. *Indices* and *NewQuality* must have the same number and order of elements.

See Also
BioMap | getMappingQuality

How To
•   "Manage Short-Read Sequence Data in Objects"

Related Links
•   Sequence Read Archive
•   SAM format specification

setMatePosition

Class: BioMap

Set mate positions of read sequences in `BioMap` object

Syntax

```
NewObj = setMatePosition(BioObj,MatePos)
NewObj = setMatePosition(BioObj,MatePos,Subset)
```

Description

`NewObj = setMatePosition(BioObj,MatePos)` returns NewObj, a new `BioMap` object, constructed from BioObj, an existing `BioMap` object, with the `MatePosition` property set to MatePos, a vector of nonnegative integers specifying the mate positions of the read sequences with respect to the position numbers in the reference sequence.

`NewObj = setMatePosition(BioObj,MatePos,Subset)` returns NewObj, a new `BioMap` object, constructed from BioObj, an existing `BioMap` object, with the `MatePosition` property of a subset of the elements set to MatePos, a vector of nonnegative integers specifying the mate positions of the read sequences with respect to the position numbers in the reference sequence. The `setMatePosition` method sets the mate positions for only the object elements specified by Subset.

Tips

- To update mate positions in an existing `BioMap` object, use the same object as the input BioObj and the output NewObj.

Input Arguments

BioObj

Object of the `BioMap` class.

Note: If BioObj was constructed from a `BioIndexedFile` object, you cannot set its `MatePosition` property.

Default:

MatePos

Vector of nonnegative integers specifying the mate positions of the read sequences with respect to the position numbers in the reference sequence.

Default:

Subset

One of the following to specify a subset of the elements in BioObj:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: A one-to-one relationship must exist between the number and order of elements in MatePos and Subset. If you use a cell array of header strings to specify Subset, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

NewObj

Object of the `BioMap` class.

Examples

Construct a `BioMap` object, and then set a subset of the sequence mate position values:

```
% Construct a BioMap object from a SAM file and determine the header for the second ele
BMObj1 = BioMap('ex1.sam');
BMObj1.Header(2)

ans =

    'EAS54_65:7:152:368:113'

% Set the MatePosition property of the second element to a new value of 5
BMObj1 = setMatePosition(BMObj1, 5, {'EAS54_65:7:152:368:113'});

% Set the MatePosition properties of the first and third elements in
% the object to 6 and 7 respectively
BMObj1 = setMatePosition(BMObj1, [6 7], [1 3]);

% Set the MatePosition property of all elements in the object to zero
y = zeros(1,BMObj1.NSeqs);
BMObj1 = setMatePosition(BMObj1,y);
```

Alternatives

An alternative to using the `setMatePosition` method to update an existing object is to use dot indexing with the `MatePosition` property:

BioObj.MatePosition(*Indices*) = *NewMatePos*

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers. *NewMatePos* is a vector of integers specifying the mate positions of the read sequences with respect to the position numbers in the reference sequence. *Indices* and *NewMatePos* must have the same number and order of elements.

See Also
BioMap | getMatePosition

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive

- SAM format specification

setReference

Class: BioMap

Set name of reference sequence for `BioMap` object

Syntax

NewObj = setReference(*BioObj, Reference*)

Description

NewObj = setReference(*BioObj, Reference*) returns *NewObj*, a new `BioMap` object, constructed from *BioObj*, an existing `BioMap` object, with the `Reference` property set to *Reference*, a string specifying the name of the reference sequence.

Tips

Rename the reference sequence of an existing `BioMap` object, by using the same object as the input *BioObj* and the output *NewObj*.

Input Arguments

BioObj

Object of the `BioMap` class.

Default:

Reference

String specifying the name of the reference sequence.

Default:

Output Arguments

NewObj

Object of the `BioMap` class.

Examples

Construct a `BioMap` object, and then set the reference sequence to a new sequence:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Create a random reference sequence
newRefSeq = randseq(50);
% Set the Reference property of the object
BMObj1 = setReference(BMObj1, newRefSeq);
```

Alternatives

An alternative to using the `setReference` method to update an existing object is to use dot indexing with the `Reference` property:

BioObj.Reference = *NewReference*

In the previous syntax, *NewReference* is a string of single-letter codes specifying a reference sequence.

See Also
getReference | BioMap

How To
·   "Manage Short-Read Sequence Data in Objects"

Related Links
·   Sequence Read Archive
·   SAM format specification

setSignature

Class: BioMap

Set signature (alignment information) for `BioMap` object

Syntax

```
NewObj = setSignature(BioObj, Signature)
NewObj = setSignature(BioObj, Signature, Subset)
```

Description

NewObj = setSignature(*BioObj*, *Signature*) returns *NewObj*, a new `BioMap` object, constructed from *BioObj*, an existing `BioMap` object, with the `Signature` property set to *Signature*, a cell array of CIGAR-formatted strings, each representing how a read sequence aligns to the reference sequence.

NewObj = setSignature(*BioObj*, *Signature*, *Subset*) returns *NewObj*, a new `BioMap` object, constructed from *BioObj*, an existing `BioMap` object, with the `Signature` property of a subset of the elements set to *Signature*, a cell array of CIGAR-formatted strings, each representing how read sequences, specified by *Subset*, align to the reference sequence. It sets the signature for only the object elements specified by *Subset*.

Tips

- To update signatures in an existing `BioMap` object, use the same object as the input *BioObj* and the output *NewObj*.
- If you modify sequences or start positions in an object, you may need to use the `setSignature` method to modify the `Signature` property of modified sequences accordingly.

Input Arguments

BioObj

Object of the `BioMap` class.

Note: If *BioObj* was constructed from a `BioIndexedFile` object, you cannot set its `Signature` property.

Default:

Signature

Cell array of CIGAR-formatted strings, each representing how a read sequence aligns to the reference sequence. Signature strings can be empty.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: A one-to-one relationship must exist between the number and order of elements in *Signature* and *Subset*. If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

NewObj

Object of the `BioMap` class.

Examples

Construct a `BioMap` object, and then set a subset of the signatures:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Set the Signature property of the second element to a new value
BMObj1 = setSignature(BMObj1, {'36M'}, 2);
```

Alternatives

An alternative to using the `setSignature` method to update an existing object is to use dot indexing with the `Signature` property:

BioObj.Signature(*Indices*) = *NewSignature*

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers. *NewSignature* is a string or a cell array of CIGAR-formatted strings, each representing how a read sequence aligns to the reference sequence. Signature strings can be empty. *Indices* and *NewSignature* must have the same number and order of elements.

See Also
getSignature | setStart | BioMap | setSequence | getAlignment

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive
· SAM format specification

setStart

Class: BioMap

Set start positions of aligned read sequences in `BioMap` object

Syntax

```
NewObj = setStart(BioObj, Start)
NewObj = setStart(BioObj, Start, Subset)
```

Description

NewObj = setStart(*BioObj, Start*) returns *NewObj*, a new `BioMap` object, constructed from *BioObj*, an existing `BioMap` object, with the `Start` property set to *Start*, a vector of positive integers specifying the start positions of the aligned read sequences with respect to the position numbers in the reference sequence. Modifying the `Start` property shifts the aligned sequences.

NewObj = setStart(*BioObj, Start, Subset*) returns *NewObj*, a new `BioMap` object, constructed from *BioObj*, an existing `BioMap` object, with the `Start` property of a subset of the elements set to *Start*, a vector of positive integers specifying the start positions of the aligned read sequences with respect to the position numbers in the reference sequence. It sets the start positions for only the object elements specified by *Subset*.

Tips

- To update start positions in an existing `BioMap` object, use the same object as the input *BioObj* and the output *NewObj*.
- If you modify sequences or signatures in an object, you may need to use the `setStart` method to modify the `Start` property to shift the alignment of modified sequences accordingly.

Input Arguments

BioObj

Object of the BioMap class.

Note: If *BioObj* was constructed from a BioIndexedFile object, you cannot set its Start property.

Default:

Start

Vector of positive integers specifying the start positions of the aligned read sequences with respect to the position numbers in the reference sequence.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: A one-to-one relationship must exist between the number and order of elements in *Start* and *Subset*. If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

NewObj

Object of the BioMap class.

Examples

Construct a `BioMap` object, and then set a subset of the sequence start values:

```
% Construct a BioMap object from a SAM file
BMObj1 = BioMap('ex1.sam');
% Set the Start property of the second element to a new value
BMObj1 = setStart(BMObj1, 5, 2);
```

Alternatives

An alternative to using the `setStart` method to update an existing object is to use dot indexing with the `Start` property:

BioObj.Start(*Indices*) = *NewStart*

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers. *NewStart* is a vector of integers specifying the start positions of the aligned read sequences with respect to the position numbers in the reference sequence. *Indices* and *NewStart* must have the same number and order of elements.

See Also
getStart | setSignature | BioMap | setSequence

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive
· SAM format specification

BioMap class

Superclasses: BioRead

Contain sequence, quality, alignment, and mapping data

Description

The `BioMap` class contains data from short-read sequences, including sequence headers, read sequences, quality scores for the sequences, and data about how each sequence aligns to a given reference. This data is typically obtained from a high-throughput sequencing instrument.

Construct a `BioMap` object from short-read sequence data. Each element in the object has a sequence, header, quality score, and alignment/mapping information associated with it. Use the object properties and methods to explore, access, filter, and manipulate all or a subset of the data, before analyzing or viewing the data.

Construction

`BioMapobj = BioMap` constructs BioMapobj, which is an empty `BioMap` object.

`BioMapobj = BioMap(File)` constructs BioMapobj, a `BioMap` object, from File, a SAM- or BAM-formatted file whose reads are ordered by start position in the reference sequence. The data remains in the source file, and the `BioMap` object accesses it using one or two auxiliary index files. For a SAM-formatted file, MATLAB uses or creates one index file that must have the same name as the source file, but with an `.idx` extension. For a BAM-formatted file, MATLAB uses or creates two index files that must have the same name as the source file, but with `*.bai` and `*.linearindex` extensions. If the index files are not found in the same folder as the source file, the `BioMap` constructor function creates the index files in that folder.

When you pass in an unordered BAM-formatted file, the constructor automatically orders the file and writes the data to an ordered file using the same base name and extension with an added string ".ordered" before the extension. The new file is indexed and used to instantiate the new `BioMap` object.

Note: Because the data remains in the source file and is accessed using the index files:

- Do not delete the source file (SAM or BAM).

- Do not delete the index files (`*.idx,*.bai,` or `*.linearindex`).
- You cannot modify BioMapobj properties.

Tip To determine the number of reference sequences included in your source file, use the `saminfo` or `baminfo` function. Use SAMtools to check if the reads in your source file are ordered by position in the reference sequence, and also to reorder them, if needed.

`BioMapobj = BioMap(Struct)` constructs BioMapobj, a `BioMap` object, from Struct, a MATLAB structure containing sequence and alignment information, such as returned by the `samread` or `bamread` function. The data from Struct remains in memory, which lets you modify the BioMapobj properties.

`BioMapobj = BioMap( ___ ,'Name',Value)` constructs the `BioMap` object using any of previous input arguments and additional options, specified as name-value pair arguments as follows.

`BioMapobj = BioMap( ___ ,'SelectReference',SelectRefValue)` selects one or more references when the source data contains sequences mapped to more than one reference. By default, the constructor includes all of the references in the header dictionary of the source file. When the header dictionary is not available, the constructor defaults to including all reference names found in the source data. SelectRefValue is either a string of a cell array of strings. By using this option, you can prevent the `BioMap` constructor from creating auxiliary index files for references that you will not use in your analysis.

`BioMapobj = BioMap(File,'InMemory',InMemoryValue)` specifies whether to place the data in memory or leave the data in the source file. Leaving the data in the source file and accessing via an index file is more memory efficient, but does not let you modify properties of BioMapobj. Choices are `true` or `false` (default). If the first input argument is not a file name, then this name-value pair argument is ignored, and the data is automatically placed in memory.

Tip Set the 'InMemory' name-value pair argument to `true` if you want to modify the properties of BioMapobj.

`BioMapobj = BioMap( ___ ,'IndexDir',IndexDirValue)` specifies the path to the folder where the index files (`*.idx,*.bai,` or `*.linearindex`) either exist or will be created.

Tip Use the 'IndexDir' name-value pair argument if you do not have write access to the folder where the source file is located.

`BioMapobj = BioMap( ___ ,'Sequence',SequenceValue)` constructs BioMapobj, a `BioMap` object, from SequenceValue, a cell array of strings containing the letter representations of nucleotide sequences. This name-value pair works only if the data is read into memory.

`BioMapobj = BioMap( ___ ,'Header',HeaderValue)` constructs BioMapobj, a `BioMap` object, from HeaderValue, a cell array of strings containing header text for nucleotide sequences. This name-value pair works only if the data is read into memory.

`BioMapobj = BioMap( ___ ,'Quality',QualityValue)` constructs BioMapobj, a `BioMap` object, from QualityValue, a cell array of strings containing the ASCII representation of per-base quality scores for nucleotide sequences. This name-value pair works only if the data is read into memory.

`BioMapobj = BioMap( ___ ,'Reference',ReferenceValue)` constructs BioMapobj, a `BioMap` object, and sets the `Reference` property to ReferenceValue, a cell array of strings containing the name of the reference sequences. This name-value pair works only if the data is read into memory.

`BioMapobj = BioMap( ___ ,'Signature',SignatureValue)` constructs BioMapobj, a `BioMap` object, from SignatureValue, a cell array of strings containing information describing the alignment of each read sequence with the reference sequence. This name-value pair works only if the data is read into memory.

`BioMapobj = BioMap( ___ ,'Start',StartValue)` constructs BioMapobj, a `BioMap` object, from StartValue, a vector of positive integers specifying the position in the reference sequence where the alignment of each read sequence starts. This name-value pair works only if the data is read into memory.

`BioMapobj = BioMap( ___ ,'Flag',FlagValue)` constructs BioMapobj, a `BioMap` object, from FlagValue, a vector of positive integers indicating the bit-wise information for the status of the 11 flags specified by the SAM format specification. These flags

describe different sequencing and alignment aspects of the read sequences. This name-value pair works only if the data is read into memory.

`BioMapobj = BioMap( ___ ,'MappingQuality',MappingQualityValue)` constructs *BioMapobj*, a `BioMap` object, from MappingQualityValue, a vector of positive integers specifying the mapping quality for each read sequence. This name-value pair works only if the data is read into memory.

`BioMapobj = BioMap( ___ ,'MatePosition',MatePositionValue)` constructs BioMapobj, a `BioMap` object, from MatePositionValue, a vector of nonnegative integers specifying the mate position for each read sequence. This name-value pair works only if the data is read into memory.

Input Arguments

File

String specifying a SAM- or BAM-formatted file that contains only one reference sequence and whose reads are ordered by start position in the reference sequence.

Default:

Struct

MATLAB structure containing sequence and alignment information, such as returned by the `samread` or `bamread` function. The structure must have a one-based start position.

Default:

SelectRefValue

String or cell array of strings specifying the name of the reference sequences in File or Struct. Use `saminfo` or `baminfo` to see a complete list of reference sequences in File.

InMemoryValue

Logical specifying whether to place the data in memory or leave the data in the source file. Leaving the data in the source file and accessing it via an index file is more memory efficient, but does not let you modify properties of the `BioMap` object. If the first input argument is not a file name, then this name-value pair argument is ignored, and the data is automatically placed in memory.

Default: `false`

`IndexDirValue`

String specifying the path to the folder where the index file either exists or will be created.

Default: Folder where File is located

`SequenceValue`

Cell array of strings containing the letter representations of nucleotide sequences. This information populates the `BioMap` object's `Sequence` property. The `samread` and `bamread` functions return this information in the `Sequence` field of the output structure.

Default:

`QualityValue`

Cell array of strings containing the ASCII representation of per-base quality scores for nucleotide sequences. This information populates the `BioMap` object's `Quality` property. The `samread` and `bamread` functions return this information in the `Quality` field of the output structure.

Default:

`HeaderValue`

Cell array of strings containing header text for nucleotide sequences. This information populates the `BioMap` object's `Header` property. The `samread` and `bamread` functions return this information in the `QueryName` field of the return structure.

Default:

`NameValue`

String describing the `BioMap` object. This information populates the object's `Name` property.

Default: ' ', an empty string

`ReferenceValue`

Cell array of strings containing the names of the reference sequences. This information populates the object's `Reference` property. The `samread` function returns this

information in the `ReferenceName` field of the `SAMStruct` output argument. The `bamread` function returns this information in the `Reference` field of the `HeaderStruct` output structure.

Default:

`SignatureValue`

Cell array of strings containing information describing the alignment of each read sequence with the reference sequence. The `samread` and `bamread` functions return this information in the `CigarString` field of the return structure. This information populates the object's `Signature` property.

Default:

`StartValue`

Vector of positive integers specifying the position in the reference sequence where the alignment of each read sequence starts. This information populates the object's `Start` property. The `samread` and `bamread` functions return this information in the `Position` field of the output structure.

Default:

`FlagValue`

Vector of positive integers indicating the bit-wise information for the status of the 11 flags specified by the SAM format specification. These flags describe different sequencing and alignment aspects of the read sequences. This information populates the object's `Flag` property. The `samread` and `bamread` functions return this information in the `Flag` field of the output structure.

Default:

`MappingQualityValue`

Vector of positive integers specifying the mapping quality for each read sequence. This information populates the object's `MappingQuality` property. The `samread` and `bamread` functions return this information in the `MappingQuality` field of the output structure.

Default:

MatePositionValue

Vector of nonnegative integers specifying the mate position for each read sequence. This information populates the object's `MatePosition` property. The `samread` and `bamread` functions return this information in the `MatePosition` field of the output structure.

Default:

Properties

Flag

Flags associated with all read sequences represented in the `BioMap` object.

Vector of positive integers such that there is an integer for each read sequence in the object. Each integer indicates the bit-wise information that specifies the status of the 11 flags described by the SAM format specification. These flags describe different sequencing and alignment aspects of a read sequence. A one-to-one relationship exists between the number and order of elements in `Flag` and `Sequence`, unless `Flag` is an empty vector.

Header

Headers associated with all read sequences represented in the `BioMap` object.

Cell array of strings, such that there is a header for each read sequence in the object. Header strings can be empty. A one-to-one relationship exists between the number and order of elements in `Header` and `Sequence`, unless `Header` is an empty cell array.

MatePosition

Positions of the mates for all read sequences represented in the `BioMap` object.

Vector of nonnegative integers such that there is an integer for each read sequence in the object. Each integer indicates the position of the corresponding mate sequence, relative to the reference sequence. A one-to-one relationship exists between the number and order of elements in `MatePosition` and `Sequence`, unless `MatePosition` is an empty vector.

Not all values in the `MatePosition` vector represent valid mate positions, for example, mates that map to a different reference sequence or mates that do not map. To determine if a mate position is valid, use the `filterByFlag` method with the `'pairedInMap'` flag.

`MappingQuality`

Mapping quality scores associated with all read sequences represented in the `BioMap` object.

Vector of integers, such that there is a mapping quality score for each read sequence in the object. A one-to-one relationship exists between the number and order of elements in `MappingQuality` and `Sequence`, unless `MappingQuality` is an empty vector.

`Name`

Description of the `BioMap` object.

Single string describing the `BioMap` object.

Default: ' ', an empty string

`NSeqs`

Number of sequences in the `BioMap` object.

This information is read only.

`Quality`

Per-base quality scores associated with all read sequences represented in the `BioMap` object.

Cell array of strings, such that there is a quality string for each read sequence in the object. Each quality string is an ASCII representation of per-base quality scores for a read sequence. Quality strings can be empty. A one-to-one relationship exists between the number and order of elements in `Quality` and `Sequence`, unless `Quality` is an empty cell array.

`Reference`

Reference sequences in the `BioMap` object.

`BioMapobj.NSeqs`-by-1 cell array of strings specifying the names of the reference sequences.

The reference sequences are the sequences against which the read sequences are aligned.

`Sequence`

Read sequences in the `BioMap` object.

Cell array of strings containing the letter representations of the read sequences.

SequenceDictionary

Cell array of strings that catalogs the names of the references available in the `BioMap` object.

This information is read only.

Signature

Alignment information associated with all read sequences represented in the `BioMap` object.

Cell array of CIGAR strings, such that there is alignment information for each read sequence in the object. Each string represents how a read sequence aligns to the reference sequence. Signature strings can be empty. A one-to-one relationship exists between the number and order of elements in `Signature` and `Sequence`, unless `Signature` is an empty cell array.

Start

Start positions of all aligned read sequences represented in the `BioMap` object.

Vector of integers, such that there is a start position for each read sequence in the object. Each integer specifies the start position of the aligned read sequence with respect to the position numbers in the reference sequence. A one-to-one relationship exists between the number and order of elements in `Start` and `Sequence`, unless `Start` is an empty vector.

Methods

getStop

Compute stop positions of aligned read sequences from `BioMap` object

filterByFlag

Filter sequence reads by SAM flag

getAlignment

Construct alignment represented in `BioMap` object

setFlag	Set read sequence flags for `BioMap` object
setMappingQuality	Set sequence mapping quality scores for `BioMap` object
setMatePosition	Set mate positions of read sequences in `BioMap` object
setReference	Set name of reference sequence for `BioMap` object
setSignature	Set signature (alignment information) for `BioMap` object
setStart	Set start positions of aligned read sequences in `BioMap` object

Inherited Methods

combine	Combine two objects
get	Retrieve property of object
getHeader	Retrieve sequence headers from object
getQuality	Retrieve sequence quality scores from object
getSequence	Retrieve sequences from object
getSubsequence	Retrieve partial sequences from object

getSubset	Create object containing subset of elements from object
plotSummary	Plot summary statistics of BioRead object
set	Set property of object
setHeader	Set sequence headers for object
setQuality	Set sequence quality scores for object
setSequence	Set sequences for object
setSubsequence	Set partial sequences for object
setSubset	Set elements for object
write	Write contents of BioRead or BioMap object to file

Copy Semantics

Value. To learn how value classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Indexing

`BioMap` objects support dot . indexing to extract, assign, and delete data.

Examples

Construct a BioMap object

This example shows how to construct a BioMap object from a SAM file and from a structure.

Construct a BioMap object from a SAM-formatted file that is provided with Bioinformatics Toolbox™ and set the Name property.

```
BMObj1 = BioMap('ex1.sam', 'Name', 'MyObject')
```

```
BMObj1 =

  BioMap with properties:

    SequenceDictionary: 'seq1'
             Reference: [1501x1 File indexed property]
             Signature: [1501x1 File indexed property]
                 Start: [1501x1 File indexed property]
        MappingQuality: [1501x1 File indexed property]
                  Flag: [1501x1 File indexed property]
          MatePosition: [1501x1 File indexed property]
               Quality: [1501x1 File indexed property]
              Sequence: [1501x1 File indexed property]
                Header: [1501x1 File indexed property]
                 NSeqs: 1501
                  Name: 'MyObject'
```

Construct a structure containing information from a SAM file.

```
SAMStruct = samread('ex1.sam');
```

Construct a BioMap object from this structure.

```
BMObj2 = BioMap(SAMStruct)
```

```
BMObj2 =

  BioMap with properties:
```

```
    SequenceDictionary: {'seq1'}
              Reference: {1501x1 cell}
              Signature: {1501x1 cell}
                  Start: [1501x1 uint32]
         MappingQuality: [1501x1 uint8]
                   Flag: [1501x1 uint16]
           MatePosition: [1501x1 uint32]
                Quality: {1501x1 cell}
               Sequence: {1501x1 cell}
                 Header: {1501x1 cell}
                  NSeqs: 1501
                   Name: ''
```

See Also

BioIndexedFile | BioRead | saminfo | samread | baminfo | bamread |
bamindexread | align2cigar | cigar2align

How To

·   "Manage Short-Read Sequence Data in Objects"

Related Links

·   Sequence Read Archive

·   SAM format specification

·   SAMtools

aa2nt

Convert amino acid sequence to nucleotide sequence

Syntax

SeqNT = aa2nt(*SeqAA*)
SeqNT = aa2nt(*SeqAA*, ...'GeneticCode', *GeneticCodeValue*, ...)
SeqNT = aa2nt(*SeqAA*, ...'Alphabet' *AlphabetValue*, ...)

Input Arguments

SeqAA	One of the following:
	• String of single-letter codes specifying an amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers. Unknown characters are mapped to 0.
	• Row vector of integers specifying an amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes.
	• MATLAB structure containing a Sequence field that contains an amino acid sequence, such as returned by fastaread, getgenpept, genpeptread, getpdb, or pdbread.
	Examples: 'ARN' or [1 2 3]
GeneticCodeValue	Integer or string specifying a genetic code number or code name from the table Genetic Code. Default is 1 or 'Standard'.
	Tip If you use a code name, you can truncate the name to the first two letters of the name.
AlphabetValue	String specifying a nucleotide alphabet. Choices are:

| | • 'DNA' (default) — Uses the symbols A, C, G, and T. |
| | • 'RNA' — Uses the symbols A, C, G, and U. |

Output Arguments

SeqNT	Nucleotide sequence specified by a character string of letter codes.

Description

SeqNT = aa2nt(*SeqAA*) converts an amino acid sequence, specified by *SeqAA*, to a nucleotide sequence, returned in *SeqNT*, using the standard genetic code.

In general, the mapping from an amino acid to a nucleotide codon is not a one-to-one mapping. For amino acids with multiple possible nucleotide codons, this function randomly selects a codon corresponding to that particular amino acid. For the ambiguous characters B and Z, one of the amino acids corresponding to the letter is selected randomly, and then a codon sequence is selected randomly. For the ambiguous character X, a codon sequence is selected randomly from all possibilities.

SeqNT = aa2nt(*SeqAA*, ...'*PropertyName*', *PropertyValue*, ...) calls aa2nt with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

SeqNT = aa2nt(*SeqAA*, ...'GeneticCode', *GeneticCodeValue*, ...) specifies a genetic code to use when converting an amino acid sequence to a nucleotide sequence. *GeneticCodeValue* can be an integer or string specifying a code number or code name from the table Genetic Code. Default is 1 or 'Standard'. The amino acid to nucleotide codon mapping for the Standard genetic code is shown in the table Standard Genetic Code.

Tip If you use a code name, you can truncate the name to the first two letters of the name.

SeqNT = aa2nt(*SeqAA*, ...'Alphabet' *AlphabetValue*, ...) specifies a nucleotide alphabet. *AlphabetValue* can be 'DNA', which uses the symbols A, C, G, and T, or 'RNA', which uses the symbols A, C, G, and U. Default is 'DNA'.

Genetic Code

Code Number	Code Name
1	Standard
2	Vertebrate Mitochondrial
3	Yeast Mitochondrial
4	Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma
5	Invertebrate Mitochondrial
6	Ciliate, Dasycladacean, and Hexamita Nuclear
9	Echinoderm Mitochondrial
10	Euplotid Nuclear
11	Bacterial and Plant Plastid
12	Alternative Yeast Nuclear
13	Ascidian Mitochondrial
14	Flatworm Mitochondrial
15	Blepharisma Nuclear
16	Chlorophycean Mitochondrial
21	Trematode Mitochondrial
22	Scenedesmus Obliquus Mitochondrial
23	Thraustochytrium Mitochondrial

Standard Genetic Code

Amino Acid Name	Amino Acid Code	Nucleotide Codon
Alanine	A	GCT GCC GCA GCG
Arginine	R	CGT CGC CGA CGG AGA AGG
Asparagine	N	ATT AAC

Amino Acid Name	Amino Acid Code	Nucleotide Codon
Aspartic acid (Aspartate)	D	GAT GAC
Cysteine	C	TGT TGC
Glutamine	Q	CAA CAG
Glutamic acid (Glutamate)	E	GAA GAG
Glycine	G	GGT GGC GGA GGG
Histidine	H	CAT CAC
Isoleucine	I	ATT ATC ATA
Leucine	L	TTA TTG CTT CTC CTA CTG
Lysine	K	AAA AAG
Methionine	M	ATG
Phenylalanine	F	TTT TTC
Proline	P	CCT CCC CCA CCG
Serine	S	TCT TCC TCA TCG AGT AGC
Threonine	T	ACT ACC ACA ACG
Tryptophan	W	TGG
Tyrosine	Y	TAT, TAC
Valine	V	GTT GTC GTA GTG
Asparagine or Aspartic acid (Aspartate)	B	Random codon from D and N
Glutamine or Glutamic acid (Glutamate)	Z	Random codon from E and Q
Unknown amino acid (any amino acid)	X	Random codon
Translation stop	*	TAA TAG TGA
Gap of indeterminate length	-	- - -

Amino Acid Name	Amino Acid Code	Nucleotide Codon
Unknown character (any character or symbol not in table)	?	???

Examples

- Convert an amino acid sequence to a nucleotide sequence using the standard genetic code.

  ```
  aa2nt('MATLAP')

  ans =

  ATGGCGACGTTAGCGCCG
  ```

- Convert an amino acid sequence to a nucleotide sequence using the Vertebrate Mitochondrial genetic code.

  ```
  aa2nt('MATLAP', 'GeneticCode', 2)

  ans =

  ATGGCAACTCTAGCGCCT
  ```

- Convert an amino acid sequence to a nucleotide sequence using the Echinoderm Mitochondrial genetic code and the RNA alphabet.

  ```
  aa2nt('MATLAP','GeneticCode','ec','Alphabet','RNA')

  ans =

  AUGGCCACAUUGGCACCU
  ```

- Convert an amino acid sequence with the ambiguous character B.

  ```
  aa2nt('abcd')

  Warning: The sequence contains ambiguous characters.

  ans =

  GCCACATGCGAC
  ```

See Also

aminolookup | baselookup | geneticcode | nt2aa | revgeneticcode |
seqviewer | rand

aacount

Count amino acids in sequence

Syntax

```
AAStruct = aacount(SeqAA)
AAStruct = aacount(SeqAA, ...'Ambiguous', AmbiguousValue, ...)
AAStruct = aacount(SeqAA, ...'Gaps', GapsValue, ...)
AAStruct = aacount(SeqAA, ...'Chart', ChartValue, ...)
```

Input Arguments

SeqAA	One of the following: • String of single-letter codes specifying an amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers. Unknown characters are mapped to 0. • Row vector of integers specifying an amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes. • MATLAB structure containing a Sequence field that contains an amino acid sequence, such as returned by fastaread, getgenpept, genpeptread, getpdb, or pdbread. Examples: 'ARN' or [1 2 3]
AmbiguousValue	String specifying how to treat ambiguous amino acid characters (B, Z, or X). Choices are: • 'ignore' (default) — Skips ambiguous characters • 'bundle' — Counts ambiguous characters and reports the total count in the Ambiguous field. • 'prorate' — Counts ambiguous characters and distributes them proportionately in the appropriate fields. For example,

	the counts for the character B are distributed evenly between the D and N fields.
	• 'individual' — Counts ambiguous characters and reports them in individual fields.
	• 'warn' — Skips ambiguous characters symbols and displays a warning.
GapsValue	Specifies whether gaps, indicated by a hyphen (-), are counted or ignored. Choices are true or false (default).
ChartValue	String specifying a chart type. Choices are 'pie' or 'bar'.

Output Arguments

| *AAStruct* | 1-by-1 MATLAB structure containing fields for the standard 20 amino acids (A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, and V). |

Description

AAStruct = aacount(*SeqAA*) counts the number of each type of amino acid in *SeqAA*, an amino acid sequence, and returns the counts in *AAStruct*, a 1-by-1 MATLAB structure containing fields for the standard 20 amino acids (A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, and V).

- Ambiguous amino acid characters (B, Z, or X), gaps, indicated by a hyphen (-), and end terminators (*) are ignored by default.

- Unrecognized characters are ignored and cause the following warning message.

  Warning: Unknown symbols appear in the sequence. These will be ignored.

AAStruct = aacount(*SeqAA*, ...'*PropertyName*', *PropertyValue*, ...) calls aacount with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

AAStruct = aacount(*SeqAA*, ...'Ambiguous', *AmbiguousValue*, ...) specifies how to treat ambiguous amino acid characters (B, Z, or X). Choices are:

- 'ignore' (default)
- 'bundle'
- 'prorate'
- 'individual'
- 'warn'

AAStruct = aacount(*SeqAA*, ...'Gaps', *GapsValue*, ...) specifies whether gaps, indicated by a hyphen (-), are counted or ignored. Choices are true or false (default).

AAStruct = aacount(*SeqAA*, ...'Chart', *ChartValue*, ...) creates a chart showing the relative proportions of the amino acids. *ChartValue* can be 'pie' or 'bar'.

Examples

1 Create an amino acid sequence.

```
Seq = 'MATLAB';
```

2 Count the amino acids in the sequence and return the results in a structure.

```
AA =  aacount(Seq)

AA =

    A: 2
    R: 0
    N: 0
    D: 0
    C: 0
    Q: 0
    E: 0
    G: 0
    H: 0
    I: 0
    L: 1
    K: 0
    M: 1
    F: 0
    P: 0
```

```
        S: 0
        T: 1
        W: 0
        Y: 0
        V: 0
```

3 Get the count for alanine (A) residues.

```
AA.A

ans =

        2
```

4 Use the `fastaread` function to read the sequence for the human p53 tumor protein into a MATLAB structure.

```
p53 = fastaread('p53aa.txt')

p53 =

        Header: 'gi|8400738|ref|NP_000537.2| tumor protein p53 [Homo sapiens]'
      Sequence: [1x393 char]
```

5 Count the amino acids in the sequence, return the results in a structure, and display the results in a pie chart.

```
AA =  aacount(p53, 'chart', 'pie');
```
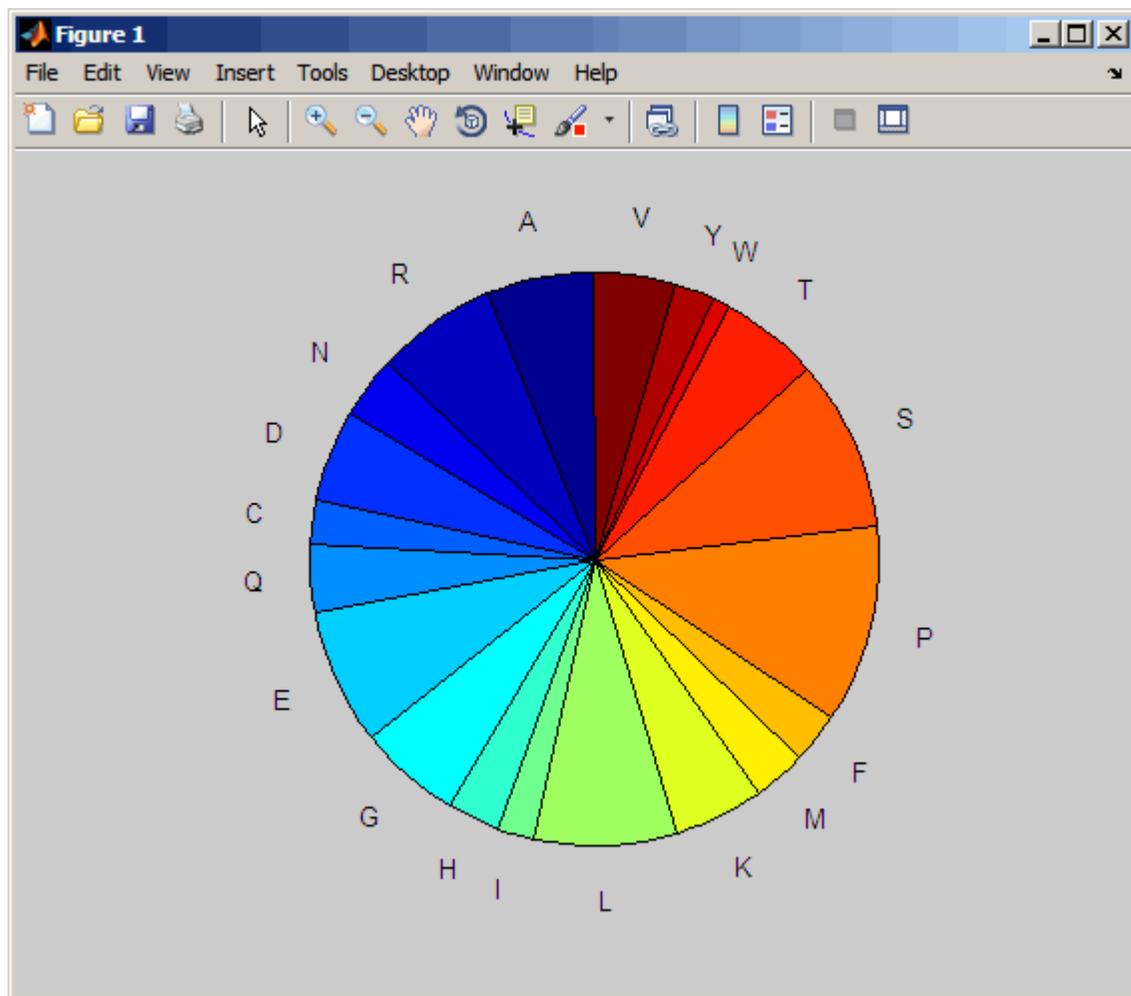
See Also

aminolookup | atomiccomp | basecount | codoncount | dimercount | isoelectric | molweight | proteinplot | proteinpropplot | seqviewer

abstract

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set abstract describing experiment in ExpressionSet object

Syntax

```
Abstract = abstract(ESObj)
NewESObj = abstract(ESObj, NewAbstract)
```

Description

Abstract = abstract(*ESObj*) returns a string containing the abstract information describing the experiment from a MIAME object in an ExpressionSet object.

NewESObj = abstract(*ESObj*, *NewAbstract*) replaces the abstract information in the MIAME object in *ESObj*, an ExpressionSet object, with *NewAbstract*, a string containing new abstract information, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the bioma.ExpressionSet class.

Default:

NewAbstract

String containing new abstract information.

Default:

Output Arguments

Abstract

String containing the abstract information describing the experiment from a MIAME object in an ExpressionSet object.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing the abstract information.

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1-   section of the `bioma.ExpressionSet` class reference page. Retrieve the abstract information stored in the MIAME object stored in the ExpressionSet object:

```
% Retrieve abstract text from the MIAME object
Abstract = abstract(ESObj)
```

See Also
bioma.ExpressionSet | bioma.data.MIAME

How To

· "Managing Gene Expression Data in Objects"

addTitle (clustergram)

Add title to clustergram

Syntax

```
addTitle(CGObject, Title)
addTitle(CGObject, Title, 'Property1Name', Property1Value,
'Property2Name', Property2Value, ...)
H = addTitle(CGObject)
```

Input Arguments

CGObject	Clustergram object created with the function `clustergram`.
Title	String used as the title in the Clustergram window.

Output Arguments

H	Handle to a MATLAB text object used as the title for the clustergram.

Description

addTitle(*CGObject*, *Title*) adds a title above the clustergram displayed in the Clustergram window.

addTitle(*CGObject*, *Title*, 'Property1Name', *Property1Value*, 'Property2Name', *Property2Value*, ...) specifies text object properties for the title. For more information on the property name/property value pairs you can use to modify the text, see Text Properties.

H = addTitle(*CGObject*) returns the handle to the text object used as the title for the clustergram.

Examples

Supply a title for the clustergram object created in the first two steps of the "Examples" on page 1-447 section of the clustergram function reference page. Use 14-point, italic text for the title.

```
addTitle(cgo, 'Expression Levels During Diauxic Shift', ...
          'FontSize', 14, 'FontAngle', 'Italic')
```

Return a handle to the title text object, then use the set function to change the font size to 16 points.

```
h = addTitle(cgo)
set(h, 'FontSize', 16)
```

More About

•   clustergram object

See Also

clustergram | addYLabel | plot | set | addXLabel | get | view

addTitle (HeatMap)

Add title to heat map

Syntax

```
addTitle(HMObject, Title)
addTitle(HMObject, Title, 'Property1Name', Property1Value,
'Property2Name', Property2Value, ...)
H = addTitle(HMObject)
```

Input Arguments

HMObject	HeatMap object created with the function `HeatMap`.
Title	String used as the title in the HeatMap window.

Output Arguments

H	Handle to a MATLAB text object used as the title for the heat map.

Description

addTitle(*HMObject*, *Title*) adds a title above the heat map displayed in the HeatMap window.

addTitle(*HMObject*, *Title*, 'Property1Name', *Property1Value*, 'Property2Name', *Property2Value*, ...) specifies text object properties for the title. For more information on the property name/property value pairs you can use to modify the text, see Text Properties.

H = addTitle(*HMObject*) returns the handle to the text object used as the title for the heat map.

Examples

Supply a title for the HeatMap object created in the "Examples" on page 1-   section of the `HeatMap` function reference page. Use 14-point, italic text for the title.

```
addTitle(hmo, 'Example Heat Map', 'FontSize', 14, ...
              'FontAngle','Italic')
```

Return a handle to the title text object, then use the `set` function to change the font size to 16 points.

```
h = addTitle(hmo)
set(h, 'FontSize', 16)
```

More About

· HeatMap object

See Also

`HeatMap | addYLabel | view | addXLabel | plot`

addXLabel (clustergram)

Label *x*-axis of clustergram

Syntax

```
addXLabel(CGObject, Label)
addXLabel(CGObject, Label, 'Property1Name', Property1Value,
'Property2Name', Property2Value, ...)
H = addXLabel(CGObject)
```

Input Arguments

CGObject	Clustergram object created with the function `clustergram`.
Label	String used as the *x*-axis label in the Clustergram window.

Output Arguments

H	Handle to a MATLAB text object used as the *x*-axis label for the clustergram.

Description

addXLabel(*CGObject*, *Label*) adds a label below the *x*-axis of a clustergram displayed in the Clustergram window.

addXLabel(*CGObject*, *Label*, 'Property1Name', *Property1Value*, 'Property2Name', *Property2Value*, ...) specifies text object properties for the *x*-axis label. For more information on the property name/property value pairs you can use to modify the text, see Text Properties.

H = addXLabel(*CGObject*) returns the handle to the text object used as the *x*-axis label for the clustergram.

Examples

Supply an *x*-axis label for the clustergram object created in the first two steps of the "Examples" on page 1-447 section of the `clustergram` function reference page. Use 12-point, italic text for the label.

```
addXLabel(cgo, 'Diauxic Shift Times', 'FontSize', 12, ...
         'FontAngle', 'Italic')
```

Return a handle to the *x*-axis label text object, then use the `set` function to change the font size to 14 points.

```
h = addXLabel(cgo)
set(h, 'FontSize', 14)
```

More About

· clustergram object

See Also

`clustergram` | `addYLabel` | `plot` | `set` | `addTitle` | `get` | `view`

addXLabel (HeatMap)

Label *x*-axis of heat map

Syntax

```
addXLabel(HMObject, Label)
addXLabel(HMObject, Label, 'Property1Name', Property1Value,
'Property2Name', Property2Value, ...)
H = addXLabel(HMObject)
```

Input Arguments

HMObject	HeatMap object created with the function `HeatMap`.
Label	String used as the *x*-axis label in the HeatMap window.

Output Arguments

H	Handle to a MATLAB text object used as the *x*-axis label for the heat map.

Description

addXLabel(*HMObject*, *Label*) adds a label below the *x*-axis of a heat map displayed in the HeatMap window.

addXLabel(*HMObject*, *Label*, 'Property1Name', *Property1Value*, 'Property2Name', *Property2Value*, ...) specifies text object properties for the *x*-axis label. For more information on the property name/property value pairs you can use to modify the text, see Text Properties.

H = addXLabel(*HMObject*) returns the handle to the text object used as the *x*-axis label for the heat map.

Examples

Supply an *x*-axis label for the HeatMap object created in the "Examples" on page 1- section of the `HeatMap` function reference page. Use 12-point, italic text for the label.

```
addXLabel(hmo, 'Times', 'FontSize', 12, 'FontAngle', 'Italic')
```

Return a handle to the *x*-axis label text object, then use the `set` function to change the font size to 14 points.

```
h = addXLabel(hmo)
set(h, 'FontSize', 14)
```

More About

·   HeatMap object

See Also

`HeatMap` | `addYLabel` | `view` | `addTitle` | `plot`

addYLabel (clustergram)

Label *y*-axis of clustergram

Syntax

```
addYLabel(CGObject, Label)
addYLabel(CGObject, Label, 'Property1Name', Property1Value,
'Property2Name', Property2Value, ...)
H = addYLabel(CGObject)
```

Input Arguments

CGObject	Clustergram object created with the function `clustergram`.
Label	String used as the *y*-axis label in the Clustergram window.

Output Arguments

H	Handle to a MATLAB text object used as the *y*-axis label for the clustergram.

Description

addYLabel(*CGObject*, *Label*) adds a label to the left of the *y*-axis of a clustergram displayed in the Clustergram window.

addYLabel(*CGObject*, *Label*, 'Property1Name', *Property1Value*, 'Property2Name', *Property2Value*, ...) specifies text object properties for the *y*-axis label. For more information on the property name/property value pairs you can use to modify the text, see Text Properties.

H = addYLabel(*CGObject*) returns the handle to the text object used as the *y*-axis label for the clustergram.

Examples

Supply a *y*-axis label for the clustergram object created in the first two steps of the "Examples" on page 1-447 section of the `clustergram` function reference page. Use 12-point, italic text for the label.

```
addYLabel(cgo, 'Genes', 'FontSize', 12, 'FontAngle', 'Italic')
```

Return a handle to the *y*-axis label text object, then use the `set` function to change the font size to 14 points.

```
h = addYLabel(cgo)
set(h, 'FontSize', 14)
```

More About

- clustergram object

See Also

`clustergram` | `addXLabel` | `plot` | `set` | `addTitle` | `get` | `view`

addYLabel (HeatMap)

Label *y*-axis of heat map

Syntax

```
addYLabel(HMObject, Label)
addYLabel(HMObject, Label, 'Property1Name', Property1Value,
'Property2Name', Property2Value, ...)
H = addYLabel(HMObject)
```

Input Arguments

HMObject	HeatMap object created with the function `HeatMap`.
Label	String used as the *y*-axis label in the HeatMap window.

Output Arguments

H	Handle to a MATLAB text object used as the *y*-axis label for the heat map.

Description

addYLabel(*HMObject*, *Label*) adds a label to the left of the *y*-axis of a heat map displayed in the HeatMap window.

addYLabel(*HMObject*, *Label*, 'Property1Name', *Property1Value*, 'Property2Name', *Property2Value*, ...) specifies text object properties for the *y*-axis label. For more information on the property name/property value pairs you can use to modify the text, see Text Properties.

H = addYLabel(*HMObject*) returns the handle to the text object used as the *y*-axis label for the heat map.

Examples

Supply a *y*-axis label for the HeatMap object created in the "Examples" on page 1-section of the `HeatMap` function reference page. Use 12-point, italic text for the label.

```
addYLabel(hmo, 'Samples', 'FontSize', 12, 'FontAngle', 'Italic')
```

Return a handle to the *y*-axis label text object, then use the `set` function to change the font size to 14 points.

```
h = addYLabel(hmo)
set(h, 'FontSize', 14)
```

More About

• HeatMap object

See Also

`HeatMap` | `addXLabel` | `view` | `addTitle` | `plot`

affygcrma

Perform GC Robust Multi-array Average (GCRMA) procedure on Affymetrix microarray probe-level data

Syntax

```
Expression = affygcrma(CELFiles, CDFFile, SeqFile)
Expression = affygcrma(ProbeStructure, Seq)
Expression = affygcrma(CELFiles, CDFFile, SeqFile, ...'CELPath',
CELPathValue, ...)
Expression = affygcrma(CELFiles, CDFFile, SeqFile, ...'CDFPath',
CDFPathValue, ...)
Expression = affygcrma(CELFiles, CDFFile, SeqFile, ...'SeqPath',
SeqPathValue, ...)
Expression = affygcrma(..., 'ChipIndex', ChipIndexValue, ...)
Expression = affygcrma(..., 'OpticalCorr', OpticalCorrValue, ...)
Expression = affygcrma(..., 'CorrConst', CorrConstValue, ...)
Expression = affygcrma(..., 'Method', MethodValue, ...)
Expression = affygcrma(..., 'TuningParam', TuningParamValue, ...)
Expression = affygcrma(..., 'GSBCorr', GSBCorrValue, ...)
Expression = affygcrma(..., 'Median', MedianValue, ...)
Expression = affygcrma(..., 'Output', OutputValue, ...)
Expression = affygcrma(..., 'Showplot', ShowplotValue, ...)
Expression = affygcrma(..., 'Verbose', VerboseValue, ...)
```

Input Arguments

CELFiles	Any of the following: • String specifying a single CEL file name. • `'*'`, which reads all CEL files in the current folder. • `' '`, which opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold **Ctrl** or **Shift** while clicking to select multiple CEL files.

	• Cell array of CEL file names.
CDFFile	Either of the following:
	• String specifying a CDF file name.
	• ' ', which opens the Select CDF File dialog box from which you select the CDF file.
SeqFile	Either of the following:
	• String specifying a file name of a sequence file (tab-separated or FASTA) that contains the following information for a specific type of Affymetrix® GeneChip® array:
	• Probe set IDs
	• Probe *x*-coordinates
	• Probe *y*-coordinates
	• Probe sequences in each probe set
	• Affymetrix GeneChip array type (FASTA file only)
	The sequence file (tab-separated or FASTA) must be on the MATLAB search path or in the Current Folder (unless you use the `SeqPath` property). In a tab-separated file, each row represents a probe; in a FASTA file, each header represents a probe.
	• An *N*-by-25 matrix of sequence information, such as returned by `affyprobeseqread`.
Seq	An *N*-by-25 matrix of sequence information, such as returned by `affyprobeseqread`.
ProbeStructure	MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs, returned by the `celintensityread` function.
CELPathValue	String specifying the path and folder where the files specified in *CELFiles* are stored.
CDFPathValue	String specifying the path and folder where the file specified in *CDFFile* is stored.

SeqPathValue	String specifying a folder or path and folder where *SeqFile* is stored.
ChipIndexValue	Positive integer specifying a chip. This chip's sequence information and mismatch probe intensity data is used to compute probe affinities. Default is 1.
OpticalCorrValue	Controls the use of optical background correction on the input probe intensity values. Choices are true (default) or false.
CorrConstValue	Value that specifies the correlation constant, rho, for log background intensity for each PM/MM probe pair. Choices are any value ≥ 0 and ≤ 1. Default is 0.7.
MethodValue	String that specifies the method to estimate the signal. Choices are 'MLE', a faster, ad hoc Maximum Likelihood Estimate method, or 'EB', a slower, more formal, empirical Bayes method. Default is 'MLE'.
TuningParamValue	Value that specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB). **Tip** For information on determining a setting for this parameter, see Wu et al., 2004.
GSBCorrValue	Specifies whether to perform gene-specific binding (GSB) correction using probe affinity data. Choices are true (default) or false. If there is no probe affinity information, this property is ignored.
MedianValue	Specifies the use of the median of the ranked values instead of the mean for normalization. Choices are true or false (default).

OutputValue	Specifies the scale of the returned gene expression values. Choices are: • `'log'` • `'log2'` • `'log10'` • `'linear'` • `@functionname` In the last instance, the data is transformed as defined by the function *functionname*. Default is `'log2'`.
ShowplotValue	Controls the display of a plot showing the $\log_2$ of mismatch (MM) probe intensity values from a specified chip (CEL file), versus that chip's MM probe affinities. The plot also shows the LOWESS fit for computing NSB data of the specified chip. Choices are `true`, `false`, or *I*, an integer specifying a chip. If set to `true`, the first chip is plotted. Default is: • `false` — When return values are specified. • `true` — When return values are not specified.
VerboseValue	Controls the display of the status of the reading of files and GCRMA processing. Choices are `true` (default) or `false`.

Output Arguments

Expression	DataMatrix object containing the $\log_2$ gene expression values that have been background adjusted, normalized, and summarized using the GC Robust Multi-array Average (GCRMA) procedure. Each row in *Expression* corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file.

Description

Expression = affygcrma(*CELFiles*, *CDFFile*, *SeqFile*) reads the specified Affymetrix CEL files, the associated CDF library file (created from Affymetrix GeneChip arrays for expression or genotyping assays), and the associated sequence file or matrix. It then processes the probe intensity values using GCRMA background adjustment, quantile normalization, and median-polish summarization procedures, then returns *Expression*, a DataMatrix object containing the $\log_2$ based gene expression values in a matrix, the probe set IDs as row names, and the CEL file names as column names. Note that each row in *Expression* corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)

CELFiles is a string or cell array of CEL file names. *CDFFile* is a string specifying a CDF file name. If you set *CELFiles* to '*', then it reads all CEL files in the current folder. If you set *CELFiles* or *CDFFile* to ' ', then it opens the Select Files dialog box from which you select the CEL files or CDF file. From this dialog box, you can press and hold **Ctrl** or **Shift** while clicking to select multiple CEL files. *SeqFile* is a file or matrix containing sequence information for probes on a specific type of Affymetrix GeneChip array.

Note: For details on the reading of files and GCRMA processing, see celintensityread, affyprobeseqread, affyprobeaffinities, gcrma, gcrmabackadj, quantilenorm, and rmasummary.

Expression = affygcrma(*ProbeStructure*, *Seq*) uses GCRMA background adjustment, quantile normalization, and median-polish summarization procedures to process the probe intensity values in *ProbeStructure*. *ProbeStructure* is a MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs, returned by the celintensityread function. *Seq* is a matrix containing sequence information for probes on a specific type of Affymetrix GeneChip array.

Expression = affygcrma(..., '*PropertyName*', *PropertyValue*, ...) calls affygcrma with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Expression = affygcrma(*CELFiles*, *CDFFile*, *SeqFile*, ...'CELPath', *CELPathValue*, ...) specifies a path and folder where the files specified by *CELFiles* are stored.

Expression = affygcrma(*CELFiles*, *CDFFile*, *SeqFile*, ...'CDFPath', *CDFPathValue*, ...) specifies a path and folder where the file specified by *CDFFile* is stored.

Expression = affygcrma(*CELFiles*, *CDFFile*, *SeqFile*, ...'SeqPath', *SeqPathValue*, ...) specifies a path and folder where the file specified by *SeqFile* is stored.

Expression = affygcrma(..., 'ChipIndex', *ChipIndexValue*, ...) computes probe affinities from MM probe intensity data using sequence information and mismatch probe intensity values from the chip specified by *ChipIndexValue*. Default *ChipIndexValue* is 1.

Expression = affygcrma(..., 'OpticalCorr', *OpticalCorrValue*, ...) controls the use of optical background correction on the input probe intensity values. Choices are true (default) or false.

Expression = affygcrma(..., 'CorrConst', *CorrConstValue*, ...) specifies the correlation constant, rho, for background intensity for each PM/MM probe pair. Choices are any value ≥ 0 and ≤ 1. Default is 0.7.

Expression = affygcrma(..., 'Method', *MethodValue*, ...) specifies the method to estimate the signal. Choices are 'MLE', a faster, ad hoc Maximum Likelihood Estimate method, or 'EB', a slower, more formal, empirical Bayes method. Default is 'MLE'.

Expression = affygcrma(..., 'TuningParam', *TuningParamValue*, ...) specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

Tip For information on determining a setting for this parameter, see Wu et al., 2004.

Expression = affygcrma(..., 'GSBCorr', *GSBCorrValue*, ...) specifies whether to perform gene-specific binding (GSB) correction using probe affinity data.

Choices are `true` (default) or `false`. If there is no probe affinity information, this property is ignored.

Expression = affygcrma(..., 'Median', *MedianValue*, ...) specifies the use of the median of the ranked values instead of the mean for normalization. Choices are `true` or `false` (default).

Expression = affygcrma(..., 'Output', *OutputValue*, ...) specifies the scale of the returned gene expression values. *OutputValue* can be:

- `'log'`
- `'log2'`
- `'log10'`
- `'linear'`
- @*functionname*

In the last instance, the data is transformed as defined by the function *functionname*. Default is `'log2'`.

Expression = affygcrma(..., 'Showplot', *ShowplotValue*, ...) controls the display of a plot showing the $\log_2$ of mismatch (MM) probe intensity values from a specified chip (CEL file), versus that chip's MM probe affinities. The plot also shows the LOWESS fit for computing NSB data of the specified chip. Choices are `true`, `false`, or *I*, an integer specifying a chip. If set to `true`, the first chip is plotted. Default is:

- `false` — When return values are specified.
- `true` — When return values are not specified.

Expression = affygcrma(..., 'Verbose', *VerboseValue*, ...) controls the display of the status of the reading of files and GCRMA processing. Choices are `true` (default) or `false`.

Examples

The following example assumes that you have the `HG_U95Av2.CDF` library file stored at `D:\Affymetrix\LibFiles\HGGenome`, and that your current folder points to a location containing CEL files and a sequence file associated with this CDF library file. In this example, the `affygcrma` function reads all the CEL files and the sequence

file in the current folder and a CDF file in a specified folder. It also performs GCRMA background adjustment, quantile normalization, and summarization procedures on the PM probe intensity values, and returns a DataMatrix object, containing the metadata and processed data.

```
Expression = affygcrma('*', 'HG_U95Av2.CDF','HG-U95Av2_probe_tab',...
                       'CDFPath', 'D:\Affymetrix\LibFiles\HGGenome');
```

References

[1] Naef, F., and Magnasco, M.O. (2003). Solving the Riddle of the Bright Mismatches: Labeling and Effective Binding in Oligonucleotide Arrays. Physical Review E *68*, 011906.

[2] Wu, Z., Irizarry, R.A., Gentleman, R., Murillo, F.M., and Spencer, F. (2004). A Model Based Background Adjustment for Oligonucleotide Expression Arrays. Journal of the American Statistical Association *99(468)*, 909–917.

[3] Wu, Z., and Irizarry, R.A. (2005). Stochastic Models Inspired by Hybridization Theory for Short Oligonucleotide Arrays. Proceedings of RECOMB 2004. J Comput Biol. *12(6)*, 882–93.

[4] Wu, Z., and Irizarry, R.A. (2005). A Statistical Framework for the Analysis of Microarray Probe-Level Data. Johns Hopkins University, Biostatistics Working Papers 73.

[5] Wu, Z., and Irizarry, R.A. (2003). A Model Based Background Adjustment for Oligonucleotide Expression Arrays. RSS Workshop on Gene Expression, Wye, England, http://biosun01.biostat.jhsph.edu/%7Eririzarr/Talks/gctalk.pdf.

[6] Speed, T. (2006). Background models and GCRMA. Lecture 10, Statistics 246, University of California Berkeley. http://www.stat.berkeley.edu/users/terry/Classes/s246.2006/Week10/Week10L1.pdf.

[7] Abd Rabbo, N.A., and Barakat, H.M. (1979). Estimation Problems in Bivariate Lognormal Distribution. Indian J. Pure Appl. Math *10(7)*, 815–825.

[8] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary

prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

[9] Irizarry, R.A., Hobbs, B., Collin, F., Beazer-Barclay, Y.D., Antonellis, K.J., Scherf, U., Speed, T.P. (2003). Exploration, Normalization, and Summaries of High Density Oligonucleotide Array Probe Level Data. Biostatistics. *4*, 249–264.

[10] Mosteller, F., and Tukey, J. (1977). Data Analysis and Regression (Reading, Massachusetts: Addison-Wesley Publishing Company), pp. 165–202.

See Also

affyprobeaffinities | affyprobeseqread | affyrma | celintensityread | gcrma | gcrmabackadj | mafdr | mattest | quantilenorm | rmasummary

affyinvarsetnorm

Perform rank invariant set normalization on probe intensities from multiple Affymetrix CEL or DAT files

Syntax

```
NormData = affyinvarsetnorm(Data)
[NormData, MedStructure] = affyinvarsetnorm(Data)
... affyinvarsetnorm(..., 'Baseline', BaselineValue, ...)
... affyinvarsetnorm(..., 'Thresholds', ThresholdsValue, ...)
... affyinvarsetnorm(..., 'StopPercentile',
StopPercentileValue, ...)
... affyinvarsetnorm(..., 'RayPercentile', RayPercentileValue, ...)
... affyinvarsetnorm(..., 'Method', MethodValue, ...)
... affyinvarsetnorm(..., 'Showplot', ShowplotValue, ...)
```

Arguments

Data	Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL or DAT file. (Each CEL or DAT file is generated from a separate chip. All chips should be of the same type.)
MedStructure	Structure of each column's intensity median before and after normalization, and the index of the column chosen as the baseline.
BaselineValue	Property to control the selection of the column index *N* from *Data* to be used as the baseline column. Default is the column index whose median intensity is the median of all the columns.
ThresholdsValue	Property to set the thresholds for the lowest average rank and the highest average rank, which are used to determine the invariant set. The rank invariant set is a set of data points whose proportional rank difference is smaller than a given threshold. The threshold for each data point is

	determined by interpolating between the threshold for the lowest average rank and the threshold for the highest average rank. Select these two thresholds empirically to limit the spread of the invariant set, but allow enough data points to determine the normalization relationship.
	ThresholdsValue is a 1-by-2 vector [*LT, HT*] where *LT* is the threshold for the lowest average rank and *HT* is threshold for the highest average rank. Values must be between 0 and 1. Default is [0.05, 0.005].
StopPercentileValue	Property to stop the iteration process when the number of data points in the invariant set reaches *N* percent of the total number of data points. Default is 1. **Note:** If you do not use this property, the iteration process continues until no more data points are eliminated.
RayPercentileValue	Property to select the *N* percentage of the highest ranked invariant set of data points to fit a straight line through, while the remaining data points are fitted to a running median curve. The final running median curve is a piecewise linear curve. Default is 1.5.
MethodValue	Property to select the smoothing method used to normalize the data. Enter 'lowess' or 'runmedian'. Default is 'lowess'.

ShowplotValue	Property to control the plotting of two pairs of scatter plots (before and after normalization). The first pair plots baseline data versus data from a specified column (chip) from the matrix *Data*. The second is a pair of M-A scatter plots, which plots M (ratio between baseline and sample) versus A (the average of the baseline and sample). Enter either `'all'` (plot a pair of scatter plots for each column or chip) or specify a subset of columns (chips) by entering the column number(s) or a range of numbers.
	For example:
	• ..., `'Showplot'`, `3`, `...)` plots data from column 3.
	• ..., `'Showplot'`, `[3,5,7]`, `...)` plots data from columns 3, 5, and 7.
	• ..., `'Showplot'`, `3:9`, `...)` plots data from columns 3 to 9.

Description

NormData = affyinvarsetnorm(*Data*) normalizes the values in each column (chip) of probe intensities in *Data* to a baseline reference, using the invariant set method. *NormData* is a matrix of normalized probe intensities from *Data*.

Specifically, affyinvarsetnorm:

- Selects a baseline index, typically the column whose median intensity is the median of all the columns.
- For each column, determines the proportional rank difference (*prd*) for each pair of ranks, *RankX* and *RankY*, from the sample column and the baseline reference. $prd = \text{abs}(RankX - RankY)$
- For each column, determines the invariant set of data points by selecting data points whose proportional rank differences (*prd*) are below *threshold*, which is a predetermined threshold for a given data point (defined by the *ThresholdsValue* property). It repeats the process until either no more data points are eliminated, or a predetermined percentage of data points is reached.

The invariant set is data points with a *prd* < *threshold*.

- For each column, uses the invariant set of data points to calculate the lowess or running median smoothing curve, which is used to normalize the data in that column.

[*NormData*, *MedStructure*] = affyinvarsetnorm(*Data*) also returns a structure of the index of the column chosen as the baseline and each column's intensity median before and after normalization.

Note: If *Data* contains NaN values, then *NormData* will also contain NaN values at the corresponding positions.

... affyinvarsetnorm(..., '*PropertyName*', *PropertyValue*, ...) calls affyinvarsetnorm with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... affyinvarsetnorm(..., 'Baseline', *BaselineValue*, ...) lets you select the column index *N* from *Data* to be the baseline column. Default is the index of the column whose median intensity is the median of all the columns.

... affyinvarsetnorm(..., 'Thresholds', *ThresholdsValue*, ...) sets the thresholds for the lowest average rank and the highest average rank, which are used to determine the invariant set. The rank invariant set is a set of data points whose proportional rank difference is smaller than a given threshold. The threshold for each data point is determined by interpolating between the threshold for the lowest average rank and the threshold for the highest average rank. Select these two thresholds empirically to limit the spread of the invariant set, but allow enough data points to determine the normalization relationship.

ThresholdsValue is a 1-by-2 vector [*LT, HT*], where *LT* is the threshold for the lowest average rank and *HT* is threshold for the highest average rank. Values must be between 0 and 1. Default is [0.05, 0.005].

... affyinvarsetnorm(..., 'StopPercentile', *StopPercentileValue*, ...) stops the iteration process when the number of data points in the invariant set reaches *N* percent of the total number of data points. Default is 1.

Note: If you do not use this property, the iteration process continues until no more data points are eliminated.


`... affyinvarsetnorm(..., 'RayPercentile', *RayPercentileValue*, ...)` selects the *N* percentage of the highest ranked invariant set of data points to fit a straight line through, while the remaining data points are fitted to a running median curve. The final running median curve is a piecewise linear curve. Default is `1.5`.

`... affyinvarsetnorm(..., 'Method', *MethodValue*, ...)` selects the smoothing method for normalizing the data. When *MethodValue* is `'lowess'`, `affyinvarsetnorm` uses the lowess method. When *MethodValue* is `'runmedian'`, `affyinvarsetnorm` uses the running median method. Default is `'lowess'`.

`... affyinvarsetnorm(..., 'Showplot', *ShowplotValue*, ...)` plots two pairs of scatter plots (before and after normalization). The first pair plots baseline data versus data from a specified column (chip) from the matrix *Data*. The second is a pair of M-A scatter plots, which plots M (ratio between baseline and sample) versus A (the average of the baseline and sample). When *ShowplotValue* is `'all'`, `affyinvarsetnorm` plots a pair of scatter plots for each column or chip. When *ShowplotValue* is a number(s) or range of numbers, `affyinvarsetnorm` plots a pair of scatter plots for the indicated column numbers (chips).

For example:

- `..., 'Showplot', 3)` plots the data from column 3 of *Data*.
- `..., 'Showplot', [3,5,7])` plots the data from columns 3, 5, and 7 of *Data*.
- `..., 'Showplot', 3:9)` plots the data from columns 3 to 9 of *Data*.
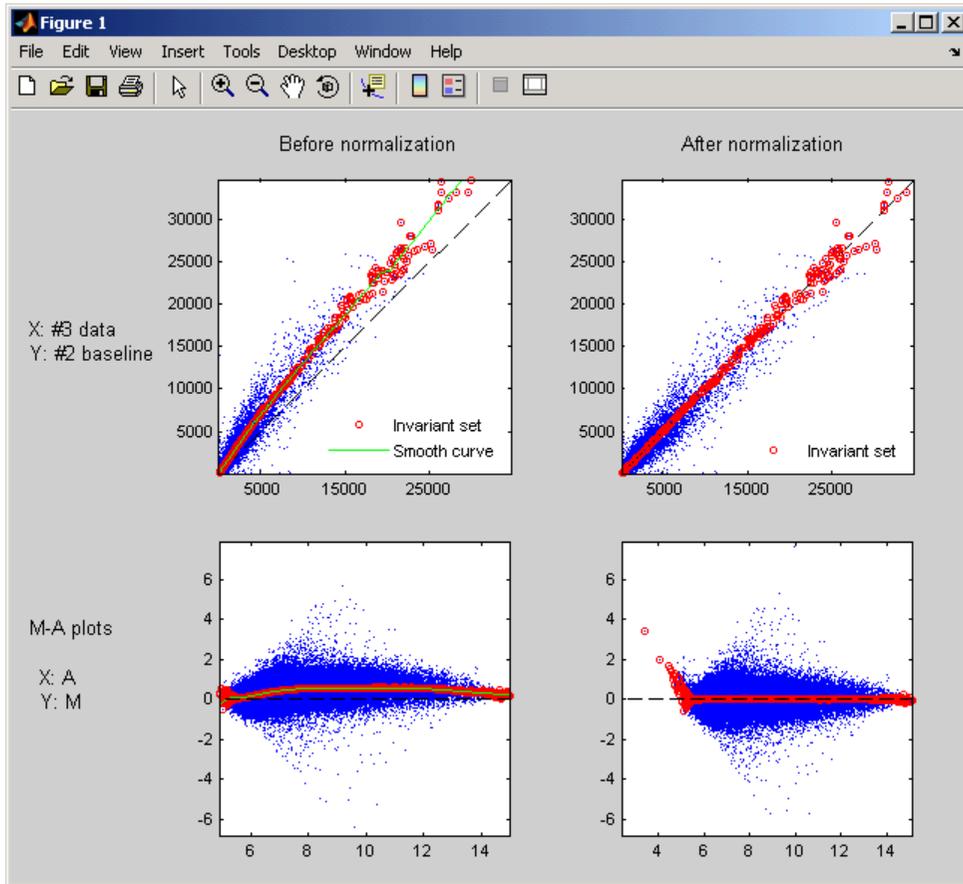
Examples

Normalize Affymetrix data

This example shows how to normalize affymetrix data. The prostatecancerrawdata.mat file used in the example contains data from Best et al., 2005.

Load a MAT-file, included with the Bioinformatics Toolbox™ software, which contains Affymetrix data variables, including `pmMatrix`, a matrix of PM probe intensity values from multiple CEL files.

```
load prostatecancerrawdata
```

Normalize the data in pmMatrix and plot data from columns (chips) 2 and 3. Column 1 is the baseline.

```
NormMatrix = affyinvarsetnorm(pmMatrix, 'Showplot',[2 3]);
```
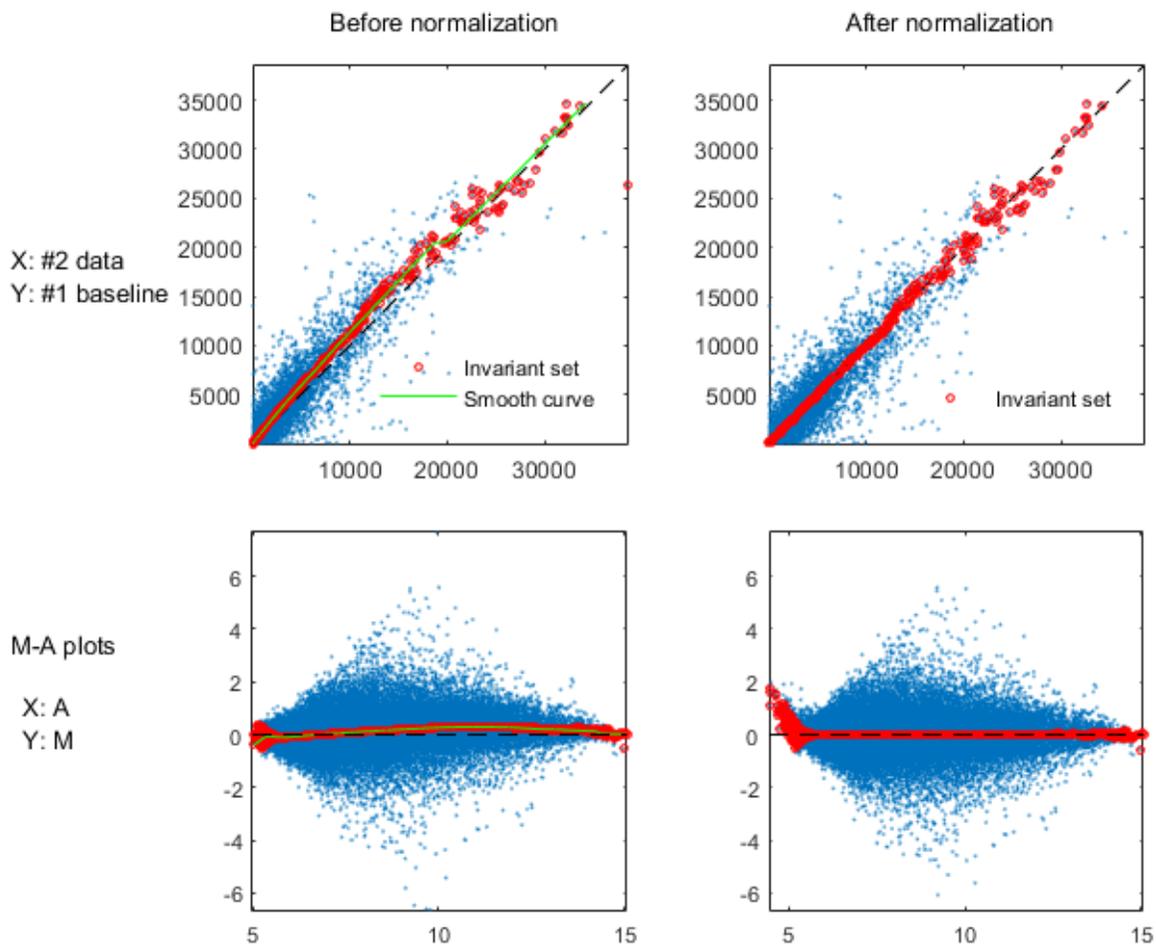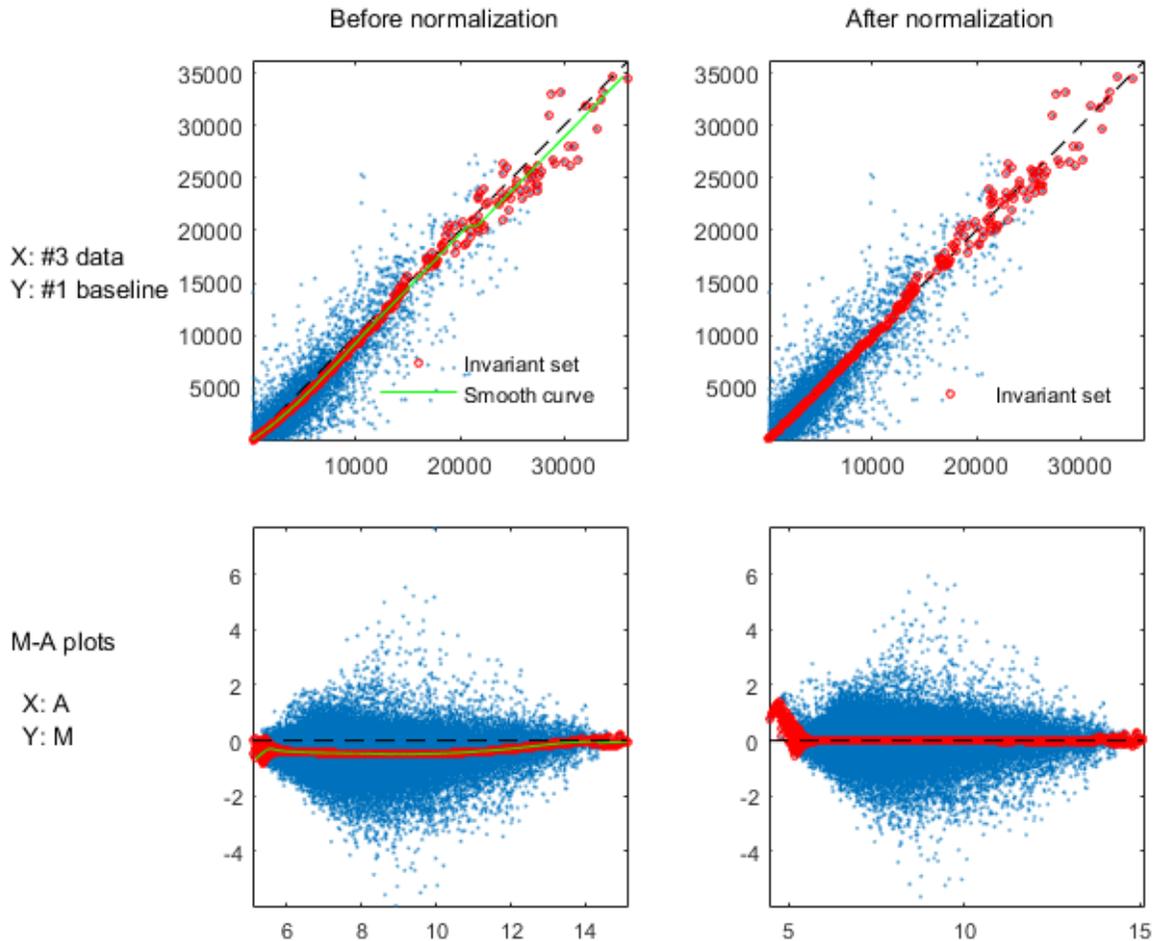
References

[1] Li, C., and Wong, W.H. (2001). Model-based analysis of oligonucleotide arrays: model validation, design issues and standard error application. Genome Biology *2(8)*: research0032.1-0032.11.

[2] http://www.hsph.harvard.edu/cli/complab/dchip/manual.htm

[3] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

See Also

affyread | celintensityread | mainvarsetnorm | malowess | manorm | quantilenorm | rmabackadj | rmasummary

affyprobeaffinities

Compute Affymetrix probe affinities from their sequences and MM probe intensities

Syntax

[*AffinPM*, *AffinMM*] = affyprobeaffinities(*SequenceMatrix*, *MMIntensity*)
[*AffinPM*, *AffinMM*, *BaseProf*] = affyprobeaffinities(*SequenceMatrix*, *MMIntensity*)
[*AffinPM*, *AffinMM*, *BaseProf*, *Stats*] = affyprobeaffinities(*SequenceMatrix*, *MMIntensity*)
... = affyprobeaffinities(*SequenceMatrix*, *MMIntensity*, ...'ProbeIndices', *ProbeIndicesValue*, ...)
... = affyprobeaffinities(*SequenceMatrix*, *MMIntensity*, ...'Showplot', *ShowplotValue*, ...)

Input Arguments

SequenceMatrix	An *N*-by-25 matrix of sequence information for the perfect match (PM) probes on an Affymetrix GeneChip array, where *N* is the number of probes on the array. Each row corresponds to a probe, and each column corresponds to one of the 25 sequence positions. Nucleotides in the sequences are represented by one of the following integers:
	• 0 — None
	• 1 — A
	• 2 — C
	• 3 — G
	• 4 — T
	Tip You can use the affyprobeseqread function to generate this matrix. If you have this sequence information

	in letter representation, you can convert it to integer representation using the `nt2int` function.
MMIntensity	Column vector containing mismatch (MM) probe intensities from a CEL file, generated from a single Affymetrix GeneChip array. Each row corresponds to a probe.
	Tip You can extract this column vector from the `MMIntensities` matrix returned by the `celintensityread` function.
ProbeIndicesValue	Column vector containing probe indexing information. Probes within a probe set are numbered 0 through *N* - 1, where *N* is the number of probes in the probe set.
	Tip You can use the `affyprobeseqread` function to generate this column vector.
ShowplotValue	Controls the display of a plot showing the affinity values of each of the four bases (A, C, G, and T) for each of the 25 sequence positions, for all probes on the Affymetrix GeneChip array. Choices are `true` or `false` (default).

Output Arguments

AffinPM	Column vector of PM probe affinities, computed from their probe sequences and MM probe intensities.
AffinMM	Column vector of MM probe affinities, computed from their probe sequences and MM probe intensities.
BaseProf	4-by-4 matrix containing the four parameters for a polynomial of degree 3, for each base, A, C, G, and T. Each row corresponds to a base, and each column corresponds to a parameter. These values are estimated from the probe sequences and intensities, and represent all probes on an Affymetrix GeneChip array.
Stats	Row vector containing four statistics in the following order: • R-square statistic

	• F statistic
	• p-value
	• Error variance

Description

[*AffinPM*, *AffinMM*] = affyprobeaffinities(*SequenceMatrix*, *MMIntensity*) returns a column vector of PM probe affinities and a column vector of MM probe affinities, computed from their probe sequences and MM probe intensities. Each row in *AffinPM* and *AffinMM* corresponds to a probe. NaN is returned for probes with no sequence information. Each probe affinity is the sum of position-dependent base affinities. For a given base type, the positional effect is modeled as a polynomial of degree 3.

[*AffinPM*, *AffinMM*, *BaseProf*] = affyprobeaffinities(*SequenceMatrix*, *MMIntensity*) also estimates affinity coefficients using multiple linear regression. It returns *BaseProf*, a 4-by-4 matrix containing the four parameters for a polynomial of degree 3, for each base, A, C, G, and T. Each row corresponds to a base, and each column corresponds to a parameter. These values are estimated from the probe sequences and intensities, and represent all probes on an Affymetrix GeneChip array.

[*AffinPM*, *AffinMM*, *BaseProf*, *Stats*] = affyprobeaffinities(*SequenceMatrix*, *MMIntensity*) also returns *Stats*, a row vector containing four statistics in the following order:

- R-square statistic
- F statistic
- p-value
- Error variance

... = affyprobeaffinities(*SequenceMatrix*, *MMIntensity*, ...'*PropertyName*', *PropertyValue*, ...) calls affyprobeaffinities with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

```
... = affyprobeaffinities(SequenceMatrix,
MMIntensity, ...'ProbeIndices', ProbeIndicesValue, ...) uses probe
```
indices to normalize the probe intensities with the median of their probe set intensities.

Tip Use of the `ProbeIndices` property is recommended only if your *MMIntensity* data are not from a nonspecific binding experiment.

```
... = affyprobeaffinities(SequenceMatrix,
MMIntensity, ...'Showplot', ShowplotValue, ...) controls the display of a
```
plot of the probe affinity base profile. Choices are `true` or `false` (default).

Examples

Calculate Affymetrix probe affinities

This example shows how to calculate Affymetrix PM and MM probe affinities from their sequences and MM probe intensities.

Load the MAT-file, included with the Bioinformatics Toolbox™ software, that contains Affymetrix data from a prostate cancer study. The variables in the MAT-file include `seqMatrix`, a matrix containing sequence information for PM probes, `mmMatrix`, a matrix containing MM probe intensity values, and `probeIndices`, a column vector containing probe indexing information.

```
load prostatecancerrawdata
```

Compute the Affymetrix PM and MM probe affinities from their sequences and MM probe intensities, and also plot the affinity values of each of the four bases (A, C, G, and T) for each of the 25 sequence positions, for all probes on the Affymetrix GeneChip array.

```
[apm, amm] = affyprobeaffinities(seqMatrix, mmMatrix(:,1),...
              'ProbeIndices', probeIndices, 'showplot', true);
```
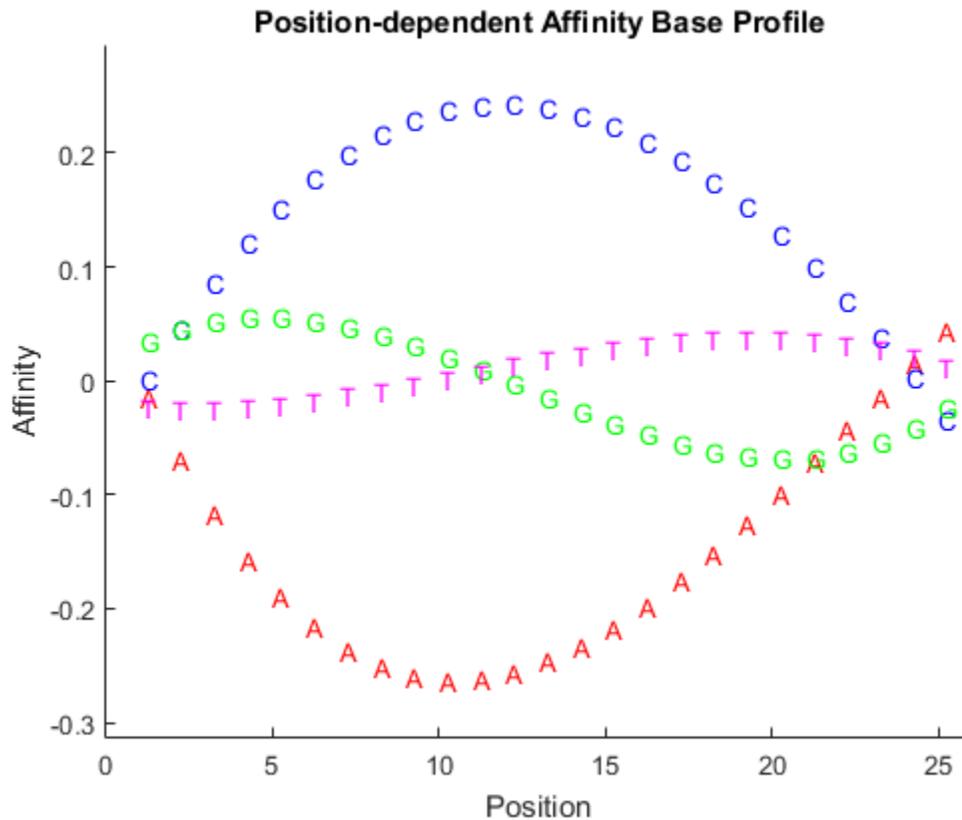
Position-dependent Affinity Base Profile

The prostatecancerrawdata.mat file used in this example contains data from Best et al., 2005.

References

[1] Naef, F., and Magnasco, M.O. (2003). Solving the Riddle of the Bright Mismatches: Labeling and Effective Binding in Oligonucleotide Arrays. Physical Review E *68*, 011906.

[2] Wu, Z., Irizarry, R.A., Gentleman, R., Murillo, F.M. and Spencer, F. (2004). A Model Based Background Adjustment for Oligonucleotide Expression Arrays. Journal of the American Statistical Association *99(468)*, 909–917.

[3] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

See Also

affygcrma | affyprobeseqread | affyread | celintensityread | probelibraryinfo

affyprobeseqread

Read data file containing probe sequence information for Affymetrix GeneChip array

Syntax

```
Struct = affyprobeseqread(SeqFile, CDFFile)
Struct = affyprobeseqread(SeqFile, CDFFile, ...'SeqPath',
SeqPathValue, ...)
Struct = affyprobeseqread(SeqFile, CDFFile, ...'CDFPath',
CDFPathValue, ...)
Struct = affyprobeseqread(SeqFile, CDFFile, ...'SeqOnly',
SeqOnlyValue, ...)
```

Input Arguments

SeqFile	String specifying a file name of a sequence file (tab-separated or FASTA) that contains the following information for a specific type of Affymetrix GeneChip array: • Probe set IDs • Probe *x*-coordinates • Probe *y*-coordinates • Probe sequences in each probe set • Affymetrix GeneChip array type (FASTA file only) The sequence file (tab-separated or FASTA) must be on the MATLAB search path or in the Current Folder (unless you use the SeqPath property). In a tab-separated file, each row represents a probe; in a FASTA file, each header represents a probe.
CDFFile	Either of the following: • String specifying a file name of an Affymetrix CDF library file, which contains information that specifies which probe set each probe belongs to on a specific type of Affymetrix GeneChip array. The CDF library file must be on the MATLAB search

	path or in the MATLAB Current Folder (unless you use the `CDFPath` property).
	• CDF structure, such as returned by the `affyread` function, which contains information that specifies which probe set each probe belongs to on a specific type of Affymetrix GeneChip array.
	Caution Make sure that *SeqFile* and *CDFFile* contain information for the same type of Affymetrix GeneChip array.
SeqPathValue	String specifying a folder or path and folder where *SeqFile* is stored.
CDFPathValue	String specifying a folder or path and folder where *CDFFile* is stored.
SeqOnlyValue	Controls the return of a structure, *Struct*, with only one field, `SequenceMatrix`. Choices are `true` or `false` (default).

Output Arguments

Struct	MATLAB structure containing the following fields:
	• `ProbeSetIDs`
	• `ProbeIndices`
	• `SequenceMatrix`

Description

Struct = affyprobeseqread(*SeqFile, CDFFile*) reads the data from files *SeqFile* and *CDFFile*, and stores the data in the MATLAB structure *Struct*, which contains the following fields.

Field	Description
`ProbeSetIDs`	Cell array containing the probe set IDs from the Affymetrix CDF library file.

Field	Description
ProbeIndices	Column vector containing probe indexing information. Probes within a probe set are numbered 0 through N - 1, where N is the number of probes in the probe set.
SequenceMatrix	An N-by-25 matrix of sequence information for the perfect match (PM) probes on the Affymetrix GeneChip array, where N is the number of probes on the array. Each row corresponds to a probe, and each column corresponds to one of the 25 sequence positions. Nucleotides in the sequences are represented by one of the following integers: • 0 — None • 1 — A • 2 — C • 3 — G • 4 — T **Note:** Probes without sequence information are represented in SequenceMatrix as a row containing all 0s. **Tip** You can use the int2nt function to convert the nucleotide sequences in SequenceMatrix to letter representation.

Struct = affyprobeseqread(*SeqFile*, *CDFFile*, ...'*PropertyName*', *PropertyValue*, ...) calls affyprobeseqread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Struct = affyprobeseqread(*SeqFile*, *CDFFile*, ...'SeqPath', *SeqPathValue*, ...) lets you specify a path and folder where *SeqFile* is stored.

Struct = affyprobeseqread(*SeqFile*, *CDFFile*, ...'CDFPath', *CDFPathValue*, ...) lets you specify a path and folder where *CDFFile* is stored.

Struct = affyprobeseqread(*SeqFile*, *CDFFile*, ...'SeqOnly', *SeqOnlyValue*, ...) controls the return of a structure, *Struct*, with only one field, SequenceMatrix. Choices are true or false (default).

Examples

1 Read the data from a FASTA file and associated CDF library file, assuming both are located on the MATLAB search path or in the Current Folder.

```
S1 = affyprobeseqread('HG-U95A_probe_fasta', 'HG_U95A.CDF');
```

2 Read the data from a tab-separated file and associated CDF structure, assuming the tab-separated file is located in the specified folder and the CDF structure is in your MATLAB Workspace.

```
S2 = affyprobeseqread('HG-U95A_probe_tab',hgu95aCDFStruct,...
     'seqpath','C:\Affymetrix\SequenceFiles\HGGenome');
```

3 Access the nucleotide sequences of the first probe set (rows 1 through 20) in the SequenceMatrix field of the S2 structure.

```
seq = int2nt(S2.SequenceMatrix(1:20,:))
```

See Also
affygcrma | affyinvarsetnorm | affyread | celintensityread | int2nt | probelibraryinfo | probesetlink | probesetlookup | probesetplot | probesetvalues

affyread

Read microarray data from Affymetrix GeneChip file

Syntax

AffyStruct = affyread(*File*)
AffyStruct = affyread(*File*, *LibraryPath*)

Description

AffyStruct = affyread(*File*) reads an Affymetrix file and creates a MATLAB structure. The affyread function can read Affymetrix EXP, DAT, CEL, CLF, BGP, CDF, and GIN files associated with Affymetrix GeneChip arrays for expression, genotyping (SNP), or resequencing assays. It can read Affymetrix CHP files associated with Affymetrix GeneChip arrays for expression assays only.

AffyStruct = affyread(*File*, *LibraryPath*) specifies the path and folder of a CDF or GIN library file.

Input Arguments

File

String specifying a file name or a path and file name of one of the following Affymetrix file types associated with Affymetrix GeneChip arrays for expression, genotyping (SNP), or resequencing assays. However, if the file name is for a CHP file, it must be associated with an Affymetrix GeneChip array for an expression assay.

- **EXP** — Data file containing information about experimental conditions and protocols.
- **DAT** — Data file containing raw image data (pixel intensity values).
- **CEL** — Data file containing information about the intensity values of the individual probes.
- **CHP** — Data file containing summary information of the probe sets, including intensity values.

- **CLF** — Cell layout file that maps probe IDs to a location (*x*- and *y*-coordinates) in the CEL file.
- **BGP** — Background probe file that lists the probes to use for background correction.
- **CDF** — Library file containing information about which probes belong to which probe set.
- **GIN** — Library file containing information about the probe sets, such as the gene name associated with the probe set.

If you specify only a file name, put that file on the MATLAB search path or in the current folder. If you specify only a file name of a CDF or GIN library file, you can specify the path and folder in the *LibraryPath* input argument.

Tip You can learn more about the Affymetrix GeneChip files and download sample files from:

```
http://www.affymetrix.com/support/technical/sample_data/demo_data.affx
```

Note: Some Affymetrix sample data files (DAT, EXP, CEL, and CHP) are combined in a DTT or CAB file. Download and use the Affymetrix Data Transfer Tool to extract these files from the DTT or CAB file. You can download the Data Transfer Tool from:

```
http://www.affymetrix.com/browse/products.jsp?productId=131431&navMode=34000&navAction=
```

You will have to register and log in at the Affymetrix Web site to download the Data Transfer Tool.

Default:

LibraryPath

String specifying the path and folder of a:

- CDF library file associated with *File* when *File* is a CHP file
- CDF library file when *File* is a CDF file
- GIN library file when *File* is a GIN file

Note: If you do not specify *LibraryPath* when reading a CHP file, `affyread` looks in the current folder for the CDF file. If it does not find the CDF file, it still reads the

CHP file. However, it omits the probe set names and types from the return value, *AffyStruct*.

Output Arguments

AffyStruct

MATLAB structure containing information from an Affymetrix data or library file, for expression, genotyping (SNP), or resequencing assay types.

The following tables describe the fields in *AffyStruct* for the different Affymetrix file types.

EXP, DAT, CEL, CHP, CLF, BGP, CDF, and GIN Files

Field	Description
Name	File name.
DataPath	Path and folder of the file.
LibPath	Path and folder of the CDF and GIN library files associated with the file you are reading.
FullPathName	Path and folder of the file.
ChipType	Name of the Affymetrix GeneChip array (for example, DrosGenome1 or HG-Focus).
Date or CreateDate	File creation date.

EXP File

Field	Description
ChipLot Operator SampleType SampleDesc Project Comments Reagents ReagentLot Protocol	Information about experimental conditions and protocols captured by the Affymetrix software.

Field	Description
Station Module HybridizeDate ScanPixelSize ScanFilter ScanDate ScannerID NumberOfScans ScannerType NumProtocolSteps ProtocolSteps	

DAT File

Field	Description
NumPixelsPerRow	Number of pixels per row in the image created from the GeneChip array (number of columns).
NumRows	Number of rows in the image created from the GeneChip array.
MinData	Minimum intensity value in the image created from the GeneChip array.
MaxData	Maximum intensity value in the image created from the GeneChip array.
PixelSize	Size of one pixel in the image created from the GeneChip array.
CellMargin	Size of gaps between cells in the image created from the GeneChip array.
ScanSpeed	Speed of the scanner used to create the image.
ScanDate	Date the scan was performed.
ScannerID	Name of the scanning device used.
UpperLeftX UpperLeftY UpperRightX UpperRightY LowerLeftX LowerLeftY	Pixel coordinates of the scanned image.

Field	Description
LowerRightX LowerRightY	
ServerName	Not used.
Image	A `NumRows`-by-`NumPixelsPerRow` image of the scanned GeneChip array.

CEL File

Field	Description
FileVersion	Version of the CEL file format.
Algorithm	Algorithm used in the image-processing step that converts from DAT format to CEL format.
AlgParams	String containing parameters used by the algorithm in the image-processing step.
NumAlgParams	Number of parameters in `AlgParams`.
CellMargin	Size of gaps between cells in the image created from the GeneChip array, used for computing the intensity values of the cells.
Rows	Number of rows of probes.
Cols	Number of columns of probes.
NumMasked	Number of masked probes, which are not used in subsequent processing.
NumOutliers	Number of cells identified as outliers (extremely high or extremely low intensity) by the image-processing step.
NumProbes	Number of probes (`Rows * Cols`) on the GeneChip array.
UpperLeftX UpperLeftY UpperRightX UpperRightY LowerLeftX LowerLeftY LowerRightX LowerRightY	Pixel coordinates of the scanned image.

Field	Description
ProbeColumnNames	Cell array containing the eight column names in the Probes field: • PosX — *x*-coordinate of the cell • PosY — *y*-coordinate of the cell • Intensity — Intensity value of the cell • StdDev — Standard deviation of intensity value • Pixels — Number of pixels in the cell • Outlier — True/false flag indicating if the cell was marked as an outlier • Masked — True/false flag indicating if the cell was masked • ProbeType — Integer indicating the probe type (for example, 1 = expression)
Probes	NumProbes-by-8 array of information about the individual probes, including intensity values. The ProbeColumnNames field contains the column names of this array.

CHP File

Field	Description
AssayType	Type of assay associated with the GeneChip array (for example, Expression, Genotyping, or Resequencing).
CellFile	File name of the CEL file from which the CHP file was created.
Algorithm	Algorithm used to convert from CEL format to CHP format.
AlgVersion	Version of the algorithm used to create the CHP file.
NumAlgParams	Number of parameters in AlgParams.
AlgParams	String containing parameters used in steps required to create the CHP file (for example, background correction).
NumChipSummary	Number of entries in ChipSummary.
ChipSummary	Summary information for the GeneChip array, including background average, standard deviation, max, and min.

Field	Description
BackgroundZones	Structure containing information about the zones used in the background adjustment step.
Rows	Number of rows of probes.
Cols	Number of columns of probes.
NumProbeSets	Number of probe sets on the GeneChip array.
NumQCProbeSets	Number of QC probe sets on the GeneChip array.
ProbeSets (Expression GeneChip array)	NumProbeSets-by-1 structure array containing information for each expression probe set, including the following fields: • Name — Name of the probe set. • ProbeSetType — Type of the probe set. • CompDataExists — True/false flag indicating if the probe set has additional computed information. • NumPairs — Number of probe pairs in the probe set. • NumPairsUsed — Number of probe pairs in the probe set used for calculating the probe set signal (not masked). • Signal — Summary intensity value for the probe set. • Detection — Indicator of statistically significant difference between the intensity value of the PM probes and the intensity value of the MM probes in a single probe set (Present, Absent, or Marginal). • DetectionPValue — P-value for the Detection indicator. • CommonPairs — When CompDataExists is true, contains the number of common pairs between the experiment and the baseline after the removal of outliers and masked probes. • SignalLogRatio — When CompDataExists is true, contains the change in signal between the experiment and baseline. • SignalLogRatioLow — When CompDataExists is true, contains the lowest ratios of probes between the experiment and the baseline. • SignalLogRatioHigh — When CompDataExists is true, contains the highest ratios of probes between the experiment and the baseline.

Field	Description
	• Change — When CompDataExists is true, describes how the probe changes versus a baseline experiment. Choices are Increase, Marginal Increase, No Change, Decrease, or Marginal Decrease. • ChangePValue — When CompDataExists is true, contains the p-value associated with Change.
ProbeSets (Genotyping GeneChip array)	NumProbeSets-by-1 structure array containing information for each genotyping probe set, including the following fields: • Name — Name of the probe set. • AlleleCall — Allele that is present for the probe set. Possibilities are AA (homozygous for the major allele), AB (heterozygous for the major and minor allele), BB (homozygous for the minor allele), or NoCall (unable to determine allele). • Confidence — Measure of the accuracy of the allele call. • RAS1 — Relative Allele Signal 1 for the SNP site, which is calculated using sense probes. • RAS2 — Relative Allele Signal 2 for the SNP site, which is calculated using antisense probes. • PValueAA — p-value for an AA call. • PValueAB — p-value for an AB call. • PValueBB — p-value for a BB call. • PValueNoCall — p-value for a NoCall call.
ProbeSets (Resequencing GeneChip array)	NumProbeSets-by-1 structure array containing information for each resequencing probe set, including the following fields: • CalledBases — 1-by-NumProbeSets character array containing the bases called by the resequencing algorithm. Possible values are a, c, g, t, and n. • Scores — 1-by-NumProbeSets array containing the score associated with each base call.

CLF File

Field	Description
LibSetName	Name of a collection of related library files for a given chip. There is only one LibSetName for a CLF file. For example, PGF and CLF files intended for use together must have the same LibSetName.
LibSetVersion	Version of a collection of related library files for a given chip. There is only one LibSetVersion for a CLF file. For example, PGF and CLF files intended for use together must have the same LibSetVersion.
GUID	Unique identifier for the CLF file.
CLFFormatVersion	Version of the CLF file format.
Rows	Number of rows in the CEL file. **Note:** The CLF file is 1 base, which means the first row and column are designated 1,1, not 0,0.
Cols	Number of columns in the CEL file. **Note:** The CLF file is 1 base, which means the first row and column are designated 1,1, not 0,0.
StartID	Starting number for the numbering of elements in the CLF file. **Tip** This information is useful when numbering does not start with 1.
EndID	Ending number for the numbering of elements in the CLF file. **Tip** This information is useful when numbering does not start with 1 and/or there are gaps in the numbering.
Order	Order in which the probe IDs are numbered in the CEL file, either 'row_major' or 'col_major'.
DataColNames	Names of the columns in the CEL file that contain data.

Field	Description
Data	If the numbering of elements in the CLF file is sequential, this field contains a function handle that calculates the *x*- and *y*-coordinates of each element in the file from the probe ID. If the numbering of elements in the CLF file is not sequential, this field contains a matrix indicating the number value of each element in the file.

BGP File

Field	Description
LibSetName	Name of a collection of related library files for a given chip. There is only one `LibSetName` for a BGP file.
LibSetVersion	Version of a collection of related library files for a given chip. There is only one `LibSetVersion` for a BGP file.
GUID	Unique identifier for a BGP file.
ExecGUID	Information about the algorithm used to generate the BGP file.
ExecVersion	
Cmd	
Data	Structure containing the following fields: • `probe_id` — ID of the probe to use for background correction. • `probeset_id` — ID of the probe set in the PGF file to which the probe belongs. • `type` — Classification information for the probe. • `gc_count` — Combined number of G and C bases in the probe. • `probe_length`— Length of the probe in base pairs. • `interrogation_position` — Interrogation position of the probe. It is typically 13 for 25-mer PM/MM probes. • `probe_sequence` — Sequence of the probe on the array, going in the direction from array surface to solution. For most standard Affymetrix arrays, this direction is from 3' to 5'. For example, for a sense target (st) probe (see the `probe_type` field), complement the sequence in this field before looking for

Field	Description
	matches to transcript sequences. For an antisense target (at), reverse this sequence.
	• `atom_id` — ID of the atom to which the probe belongs.
	• `x` — Column coordinate of the probe in the CEL file.
	• `y` — Row coordinate of the probe in the CEL file.
	• `probeset_type` — Classification information for the probe set, such as control, affx, or spike. This type information can include multiple classifications and can also be nested.
	• `probe_type` — Classification information for the probe, such as pm (perfect match), mm (mismatch), st (sense target), or at (antisense target). This type information can include multiple classifications and can also be nested.

CDF File

Field	Description
Rows	Number of rows of probes.
Cols	Number of columns of probes.
NumProbeSets	Number of probe sets on the GeneChip array.
NumQCProbeSets	Number of QC probe sets on the GeneChip array.
ProbeSetColumnNames	Cell array containing the six column names in the `ProbePairs` field in the `ProbeSets` array:
	• `GroupNumber` — Number identifying the group to which the probe pair belongs. For expression arrays, this value is always 1. For genotyping arrays, this value is typically 1 (allele A, sense), 2 (allele B, sense), 3 (allele A, antisense), or 4 (allele B, antisense).
	• `Direction` — Number identifying the direction of the probe pair. 1 = sense and 2 = antisense.
	• `PMPosX` — *x*-coordinate of the perfect match probe.
	• `PMPosY` — *y*-coordinate of the perfect match probe.
	• `MMPosX` — *x*-coordinate of the mismatch probe.
	• `MMPosY` — *y*-coordinate of the mismatch probe.

Field	Description
ProbeSets	`NumProbeSets`-by-1 structure array containing information for each probe set, including the following fields: • `Name` — Name of the probe set. • `ProbeSetType` — Type of the probe set. • `CompDataExists` — True/false flag indicating if the probe set has additional computed information. • `NumPairs` — Number of probe pairs in the probe set. • `NumQCProbes` — Number of QC probes in the probe set. • `QCType` — Type of QC probes. • `GroupNames` — Name of the group to which the probe set belongs. For expression arrays, this field contains the name of the probe set. For genotyping arrays, this field contains the name of the alleles, for example `{'A' 'C' 'A' 'C'}'`. • `ProbePairs` — `NumPairs`-by-6 array of information about the probe pairs. The column names of this array are contained in the `ProbeSetColumnNames` field.

GIN File

Field	Description
`Version`	GIN file format version.
`ProbeSetName`	Probe set ID/name.
`ID`	Identifier for the probe set (gene ID).
`Description`	Description of the probe set.
`SourceNames`	Source or sources of the probe sets.
`SourceURL`	Source URL or URLs for the probe sets.
`SourceID`	Vector of numbers specifying which `SourceNames` or `SourceURL` each probe set is associated with.

Examples

The following example uses the sample data and CDF library file from the *E. coli* Antisense Genome array, which you can download from:

```
http://www.affymetrix.com/support/technical/sample_data/demo_data.affx
```

After downloading the sample data, you need the Affymetrix Data Transfer Tool to extract the CEL, DAT, and CHP files from a DTT file. You can download the Data Transfer Tool from:

```
http://www.affymetrix.com/browse/products.jsp?productId=131431&navMode=34000&navAction=
```

The following example assumes that you have stored the files `Ecoli-antisense-121502.CEL`, `Ecoli-antisense-121502.dat`, and `Ecoli-antisense-121502.chp` on the MATLAB search path or in the current folder. It also assumes that you have stored the associated CDF library file, `Ecoli_ASv2.CDF`, at `D:\Affymetrix\LibFiles\Ecoli`.

1   Read the contents of a CEL file into a MATLAB structure.

```
celStruct = affyread('Ecoli-antisense-121502.CEL');
```

2   Display a spatial plot of the probe intensities.

```
maimage(celStruct, 'Intensity')
```
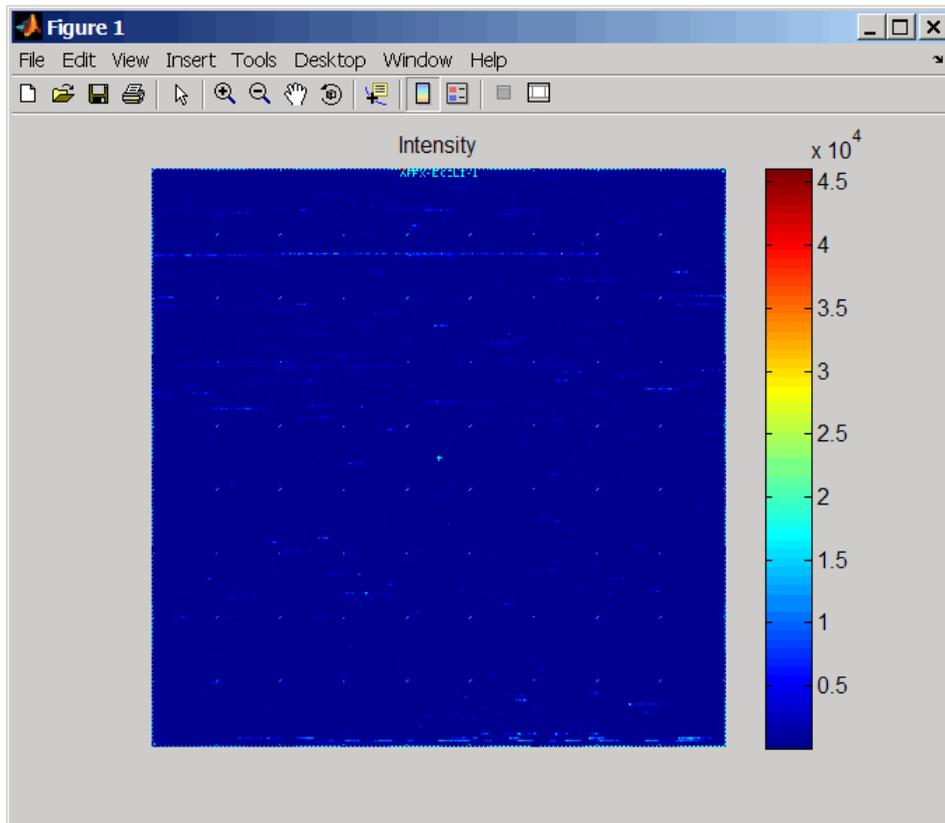
**3** Zoom in on a specific region of the plot.

```
axis([200 340 0 70])
```
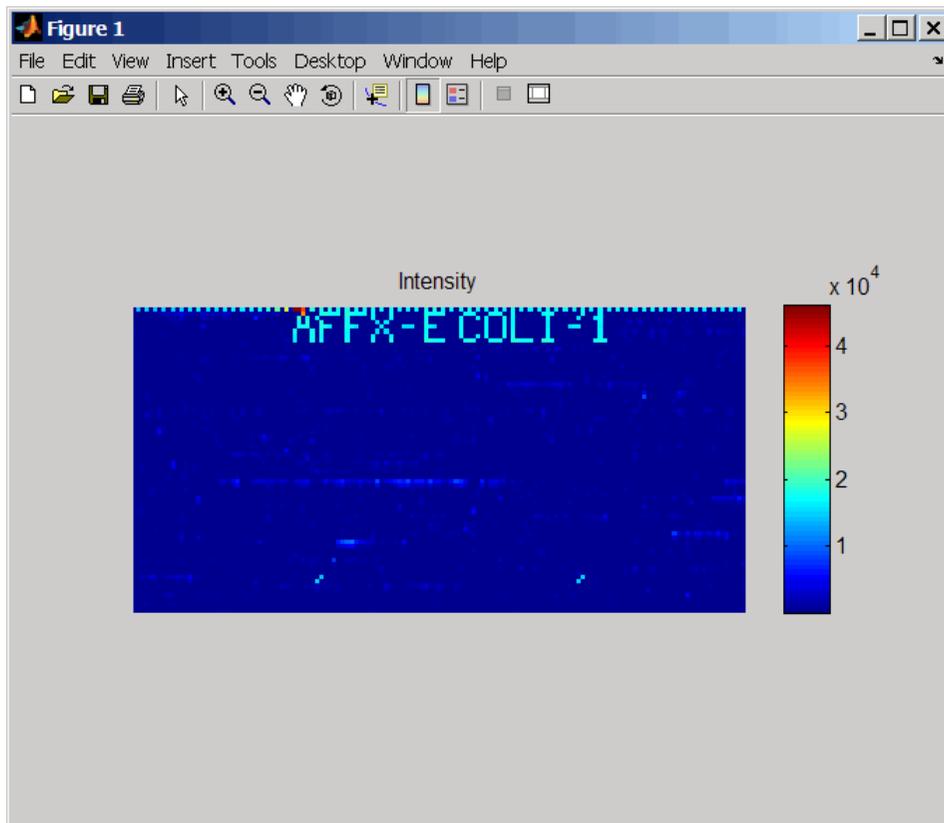
4 Read the contents of a DAT file into a MATLAB structure. Display the raw image data, and then use the `axis image` function to set the correct aspect ratio.

```
datStruct = affyread('Ecoli-antisense-121502.dat');
imagesc(datStruct.Image)
axis image
```
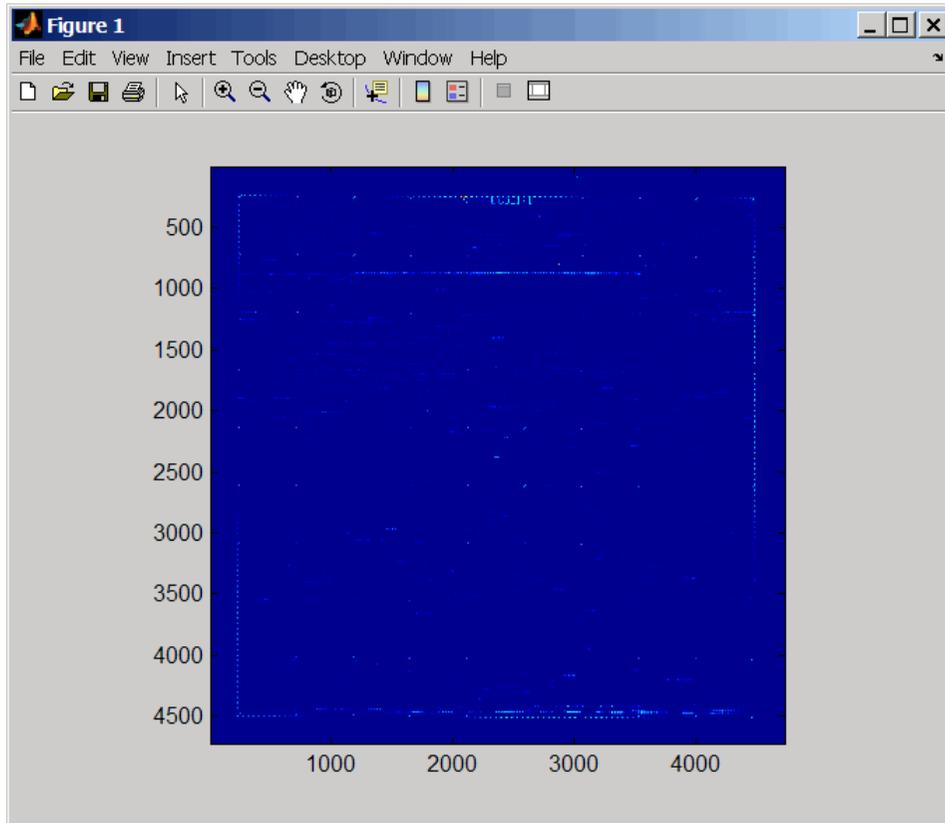
5 Zoom in on a specific region of the plot.

```
axis([1900 2800 160 650])
```
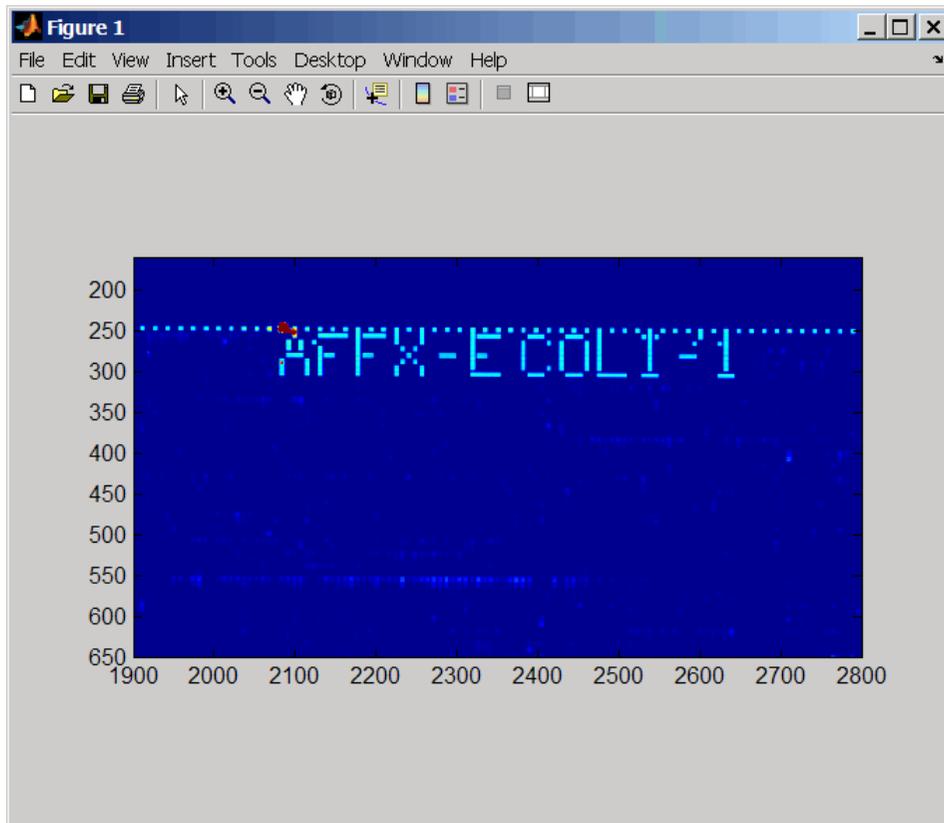
6 Read the contents of a CHP file into a MATLAB structure, specifying the location of the associated CDF library file. Then extract information for probe set 3315278.

```
chpStruct = affyread('Ecoli-antisense-121502.chp',...
            'D:\Affymetrix\LibFiles\Ecoli');
geneName = probesetlookup(chpStruct,'3315278')

geneName =

       Identifier: '3315278'
     ProbeSetName: 'argG_b3172_at'
         CDFIndex: 5213
         GINIndex: 3074
      Description: [1x82 char]
```

```
       Source: 'NCBI EColi Genome'
    SourceURL: [1x74 char]
```

More About

- http://www.affymetrix.com/support/technical/sample_data/demo_data.affx

- http://www.affymetrix.com/products_services/software/specific/dtt.affx

See Also

affyrma | affygcrma | affysnpannotread | affysnpintensitysplit |
agferead | celintensityread | geoseriesread | gprread | ilmnbsread
| probelibraryinfo | probesetlink | probesetlookup | probesetplot |
probesetvalues | sptread

affyrma

Perform Robust Multi-array Average (RMA) procedure on Affymetrix microarray probe-level data

Syntax

```
Expression = affyrma(CELFiles, CDFFile)
Expression = affyrma(ProbeStructure)
Expression = affyrma(CELFiles, CDFFile, ...'CELPath',
CELPathValue, ...)
Expression = affyrma(CELFiles, CDFFile, ...'CDFPath',
CDFPathValue, ...)
Expression = affyrma(..., 'Method', MethodValue, ...)
Expression = affyrma(..., 'Truncate', TruncateValue, ...)
Expression = affyrma(..., 'Median', MedianValue, ...)
Expression = affyrma(..., 'Output', OutputValue, ...)
Expression = affyrma(..., 'Showplot', ShowplotValue, ...)
Expression = affyrma(..., 'Verbose', VerboseValue, ...)
```

Input Arguments

CELFiles	Any of the following:
	• String specifying a single CEL file name.
	• '*', which reads all CEL files in the current folder.
	• ' ', which opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold **Ctrl** or **Shift** while clicking to select multiple CEL files.
	• Cell array of CEL file names.
CDFFile	Either of the following:
	• String specifying a CDF file name.
	• ' ', which opens the Select CDF File dialog box from which you select the CDF file.

ProbeStructure	MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs, returned by the `celintensityread` function.
CELPathValue	String specifying the path and folder where the files specified in *CELFiles* are stored.
CDFPathValue	String specifying the path and folder where the file specified in *CDFFile* is stored.
MethodValue	Specifies the estimation method for the background adjustment model parameters. Choices are `'RMA'` (to use estimation method described by Bolstad, 2005) or `'MLE'` (to estimate the parameters using maximum likelihood). Default is `'RMA'`.
TruncateValue	Specifies the background noise model. Choices are `true` (use a truncated Gaussian distribution) or `false` (use a nontruncated Gaussian distribution). Default is `true`.
MedianValue	Specifies the use of the median of the ranked values instead of the mean for normalization. Choices are `true` or `false` (default).
OutputValue	Specifies the scale of the returned gene expression values. Choices are: • `'log'` • `'log2'` • `'log10'` • `'linear'` • `@functionname` In the last instance, the data is transformed as defined by the function *functionname*. Default is `'log2'`.

ShowplotValue	Controls the plotting of a histogram showing the distribution of PM probe intensity values (blue) and the convoluted probability distribution function (red), with estimated parameters mu, sigma and alpha. Enter either `'all'` (plot a histogram for each column or chip) or specify a subset of columns (chips) by entering the column number, list of numbers, or range of numbers. For example: • (..., `'Showplot'`, 3, ...) plots the intensity values in column 3. • (..., `'Showplot'`, [3,5,7], ...) plots the intensity values in columns 3, 5, and 7. • (..., `'Showplot'`, 3:9, ...) plots the intensity values in columns 3 to 9.
VerboseValue	Controls the display of the status of the reading of files and RMA processing. Choices are `true` (default) or `false`.

Output Arguments

Expression	DataMatrix object containing the $\log_2$ based gene expression values that have been background adjusted, normalized, and summarized using the Robust Multi-array Average (RMA) procedure. Each row in *Expression* corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file.

Description

Expression = affyrma(*CELFiles*, *CDFFile*) reads the specified Affymetrix CEL files and the associated CDF library file (created from Affymetrix GeneChip arrays for expression or genotyping assays), processes the probe intensity values using RMA background adjustment, quantile normalization, and summarization procedures, then

returns *Expression*, a DataMatrix object containing the $\log_2$ based gene expression values in a matrix, the probe set IDs as row names, and the CEL file names as column names. Note that each row in *Expression* corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)

CELFiles is a string or cell array of CEL file names. *CDFFile* is a string specifying a CDF file name. If you set *CELFiles* to '*', then it reads all CEL files in the current folder. If you set *CELFiles* to ' ', then it opens the Select CEL Files dialog box from which you select the CEL files.

Note: For details on the reading of files and RMA processing, see `celintensityread`, `rmabackadj`, `quantilenorm`, and `rmasummary`.

Expression = affyrma(*ProbeStructure*) uses RMA background adjustment, quantile normalization, and summarization procedures to process the probe intensity values in *ProbeStructure*, a MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs, returned by the `celintensityread` function, and returns *Expression*.

Expression = affyrma(..., '*PropertyName*', *PropertyValue*, ...) calls `affyrma` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Expression = affyrma(*CELFiles*, *CDFFile*, ...'CELPath', *CELPathValue*, ...) specifies a path and folder where the files specified by *CELFiles* are stored.

Expression = affyrma(*CELFiles*, *CDFFile*, ...'CDFPath', *CDFPathValue*, ...) specifies a path and folder where the file specified by *CDFFile* is stored.

Expression = affyrma(..., 'Method', *MethodValue*, ...) specifies the estimation method for the background adjustment model parameters. When *MethodValue* is 'RMA', `affyrma` implements the estimation method described by Bolstad, 2005. When *MethodValue* is 'MLE', `affyrma` estimates the parameters using maximum likelihood. Default is 'RMA'.

Expression = affyrma(..., 'Truncate', *TruncateValue*, ...) specifies the background noise model used. When *TruncateValue* is false, affyrma uses nontruncated Gaussian as the background noise model. Default is true.

Expression = affyrma(..., 'Median', *MedianValue*, ...) specifies the use of the median of the ranked values instead of the mean for normalization. Choices are true or false (default).

Expression = affyrma(..., 'Output', *OutputValue*, ...) specifies the scale of the returned gene expression values. *OutputValue* can be:

- 'log'
- 'log2'
- 'log10'
- 'linear'
- @*functionname*

In the last instance, the data is transformed as defined by the function *functionname*. Default is 'log2'.

Expression = affyrma(..., 'Showplot', *ShowplotValue*, ...) lets you plot a histogram showing the distribution of PM probe intensity values (blue) and the convoluted probability distribution function (red), with estimated parameters mu, sigma and alpha. When *ShowplotValue* is 'all', rmabackadj plots a histogram for each column or chip. When *ShowplotValue* is a number, list of numbers, or range of numbers, rmabackadj plots a histogram for the indicated column number (chip).

For example:

- (..., 'Showplot', 3,...) plots the intensity values in column 3.
- (..., 'Showplot', [3,5,7],...) plots the intensity values in columns 3, 5, and 7.
- (..., 'Showplot', 3:9,...) plots the intensity values in columns 3 to 9.

Expression = affyrma(..., 'Verbose', *VerboseValue*, ...) controls the display of the status of the reading of files and RMA processing. Choices are true (default) or false.

Examples

The following example assumes that you have the HG_U95Av2.CDF library file stored at D:\Affymetrix\LibFiles\HGGenome, and that your current folder points to a location containing CEL files associated with this CDF library file. In this example, the affyrma function reads all the CEL files in the current folder and a CDF file in a specified folder. It also performs RMA background adjustment, quantile normalization, and summarization procedures on the PM probe intensity values, and returns a DataMatrix object, containing the metadata and processed data.

```
Expression = affyrma('*', 'HG_U95Av2.CDF',...
                     'CDFPath', 'D:\Affymetrix\LibFiles\HGGenome');
```

References

[1] Irizarry, R.A., Hobbs, B., Collin, F., Beazer-Barclay, Y.D., Antonellis, K.J., Scherf, U., Speed, T.P. (2003). Exploration, Normalization, and Summaries of High Density Oligonucleotide Array Probe Level Data. Biostatistics. *4*, 249–264.

[2] Mosteller, F., and Tukey, J. (1977). Data Analysis and Regression (Reading, Massachusetts: Addison-Wesley Publishing Company), pp. 165–202.

[3] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

[4] Bolstad, B. (2005). "affy: Built-in Processing Methods" http://www.bioconductor.org/packages/2.1/bioc/vignettes/affy/ inst/doc/builtinMethods.pdf

See Also
affygcrma | gcrma | mafdr | celintensityread | mattest | quantilenorm | rmabackadj | rmasummary

affysnpannotread

Read Affymetrix Mapping DNA array data from CSV-format annotation file

Syntax

AnnotStruct = affysnpannotread(*File*, *PID*)
AnnotStruct = affysnpannotread(*File*, *PID*, 'LookUpField',
LookUpFieldValue)

Input Arguments

File	String specifying a file name or a path and file name of an Affymetrix CSV annotation file for a Mapping 10K array set, Mapping 100K array set, or Mapping 500K array set.
	If you specify only a file name, that file must be on the MATLAB search path or in the current folder.
PID	String or cell array of strings specifying one or more probe set IDs on an Affymetrix mapping array.
LookUpFieldValue	String or cell array of strings specifying one or more column headers in an Affymetrix CSV annotation file. Default are the fields shown in the following table.

Output Arguments

AnnotStruct	MATLAB structure containing information for one or more probe sets from *File*, an Affymetrix CSV annotation file.
	AnnotStruct contains a subset of the fields in *File*. The fields are described in the table below.

Description

AnnotStruct = affysnpannotread(*File, PID*) reads *File*, an Affymetrix CSV annotation file for a Mapping 10K array set, Mapping 100K array set, or Mapping 500K array set, and returns *AnnotStruct*, a MATLAB structure containing annotation information for one or more probe sets specified by *PID*, a string or cell array of strings specifying one or more probe set IDs. *AnnotStruct* contains a subset of the fields in *File*. The fields are described in the following table.

Structure Created from an Affymetrix CSV Annotation File

Field	Description
ProbeSetIDs	Cell array containing the unique probe set IDs specified by the *PID* input.
Chromosome	Cell array containing the chromosome number on which each probe set is located.
ChromPosition	Cell array containing the SNP genomic position on the chromosome for each probe set.
Cytoband	Cell array containing the cytogenetic banding region of the chromosome on which each probe set is located.
Sequence	Cell array containing the sequence of each probe set.
AlleleA	Cell array containing the base that is allele A for each probe set.
AlleleB	Cell array containing the base that is allele B for each probe set.
Accession	Cell array containing the GenBank® accession number for each probe set.
FragmentLength	Cell array containing the length of each probe set.

AnnotStruct = affysnpannotread(*File, PID*, 'LookUpField', *LookUpFieldValue*) returns annotation information from only the field (column) specified by *LookUpFieldValue*, a string or cell array of strings specifying one or more column headers in an Affymetrix CSV annotation file. Default are the fields shown in the previous table.

Note: You can download Affymetrix CSV annotation files such as Mapping50K_Xba240.na25.annot.csv from:

http://www.affymetrix.com/support/technical/annotationfilesmain.affx

---

Examples

The following example assumes that you have the `Mapping50K_Xba240.CDF` file stored at `C:\AffyLibFiles\`, and that your current folder points to a location containing the `Mapping50K_Xba240.na25.annot.csv` annotation file.

1   Use the `affyread` function to create a structure containing information from the `Mapping50K_Xba240.CDF` library file.

```
cdf = affyread('C:\AffyLibFiles\Mapping50K_Xba240.CDF');
```

2   Create a variable containing a cell array of the names of the probe sets, which are stored in the `Name` field of the `ProbeSets` field of the `cdf` structure.

```
probesetIDs = {cdf.ProbeSets.Name}';
```

3   Return a structure containing annotation information for all the probe sets in the `Mapping50K_Xba240.na25.annot.csv` annotation file.

```
snpInfo = affysnpannotread('Mapping50K_Xba240.na25.annot.csv',probesetIDs)

snpInfo =

       ProbeSetIDs: {59024x1 cell}
        Chromosome: [59024x1 int8]
      ChromPosition: [59024x1 double]
          Cytoband: {59024x1 cell}
          Sequence: {59024x1 cell}
           AlleleA: {59024x1 cell}
           AlleleB: {59024x1 cell}
         Accession: {59024x1 cell}
    FragmentLength: [59024x1 double]
```

See Also
affysnpintensitysplit | affyread

affysnpintensitysplit

Split Affymetrix SNP probe intensity information for alleles A and B

Syntax

```
ProbeStructSplit = affysnpintensitysplit(ProbeStruct)
ProbeStructSplit = affysnpintensitysplit(ProbeStruct, 'Controls',
ControlsValue)
```

Input Arguments

ProbeStruct	MATLAB structure containing probe intensity information from an Affymetrix Mapping DNA array, such as returned by `celintensityread`.
ControlsValue	Controls the inclusion of control probes in *ProbeStructSplit*. Choices are `true` or `false` (default).

Output Arguments

ProbeStructSplit	MATLAB structure containing probe intensity information from an Affymetrix Mapping DNA array, split into information for alleles A and B.

Description

ProbeStructSplit = affysnpintensitysplit(*ProbeStruct*) splits *ProbeStruct*, a structure containing probe intensity information from an Affymetrix Mapping DNA array, into *ProbeStructSplit*, a structure containing probe intensity information from an Affymetrix Mapping DNA array, split into information for alleles A and B.

ProbeStructSplit contains the following fields.

Field	Description
CDFName	File name of the Affymetrix CDF library file.
CELNames	Cell array of names of the Affymetrix CEL files.
NumChips	Number of CEL files read into the input structure.
NumProbeSets	Number of probe sets in each CEL file.
NumProbes	Maximum number of probes for just one allele in each CEL file. **Note:** If the number of probes for allele A is not the same as for allele B, the larger number is used.
ProbeSetIDs	Cell array of the probe set IDs from the Affymetrix CDF library file.
ProbeIndices	Column vector containing probe indexing information for just one allele in each cell file. Probes within a probe set are numbered 0 through $N - 1$, where N is the number of probes for one allele in the probe set. **Note:** ProbeIndices has the same number of elements as NumProbes.
PMAIntensities	Matrix containing perfect match (PM) probe intensity values for allele A. Each row corresponds to an allele A probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the *CELFiles* input argument to the celintensityread function.
PMBIntensities	Matrix containing perfect match (PM) probe intensity values for allele B. Each row corresponds to an allele B probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the *CELFiles* input argument to the celintensityread function.

Field	Description
MMAIntensities (optional)	Matrix containing mismatch (MM) probe intensity values for allele A. Each row corresponds to an allele A probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the *CELFiles* input argument to the celintensityread function.
MMBIntensities (optional)	Matrix containing mismatch (MM) probe intensity values for allele B. Each row corresponds to an allele B probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the *CELFiles* input argument to the celintensityread function.

ProbeStructSplit = affysnpintensitysplit(*ProbeStruct*, 'Controls', *ControlsValue*) controls the return of control probe intensities. Choices are true or false (default).

Note: Control probes sometimes contain information for only one allele. In this case, the value for the corresponding allele (A or B) that is not present is set to NaN.

Examples

The following example assumes that your current folder points to a location containing the Mapping50K_Hind240.CDF library file and 18 CEL files associated with this CDF library file. These files are associated with an Affymetrix Mapping DNA array.

1   Use the celintensityread function to read the Mapping50K_Hind240.CDF library file and 18 CEL files associated with it into a MATLAB structure.

```
ps = celintensityread('*','Mapping50K_Hind240.CDF')

ps =

          CDFName: 'Mapping50K_Hind240.CDF'
         CELNames: {18x1 cell}
         NumChips: 18
     NumProbeSets: 57299
```

```
         NumProbes: 1145780
       ProbeSetIDs: {57299x1 cell}
      ProbeIndices: [1145780x1 uint8]
      GroupNumbers: [1145780x1 uint8]
      PMIntensities: [1145780x18 single]
```

2 Extract the PM probe intensities for allele A and allele B into another MATLAB structure, without including intensity information for the control probes.

```
ps_split = affysnpintensitysplit(ps)

ps_split =

           CDFName: 'Mapping50K_Hind240.CDF'
          CELNames: {18x1 cell}
          NumChips: 18
      NumProbeSets: 57275
         NumProbes: 572750
       ProbeSetIDs: {57275x1 cell}
      ProbeIndices: [572750x1 uint8]
     PMAIntensities: [572750x18 single]
     PMBIntensities: [572750x18 single]
```

See Also

affysnpannotread | affyread | celintensityread

affysnpquartets

Create table of SNP probe quartet results for Affymetrix probe set

Syntax

SNPQStruct = affysnpquartets(*CELStruct*, *CDFStruct*, *PS*)

Input Arguments

CELStruct	Structure created by the affyread function from an Affymetrix CEL file, which contains information about the intensity values of the individual probes.
CDFStruct	Structure created by the affyread function from an Affymetrix CDF library file associated with the CEL file. The CDF library file contains information about which probes belong to which probe set.
PS	Probe set index or the probe set ID/name.

Output Arguments

SNPQStruct	Structure containing probe quartet results for a specific SNP probe set from the data in a CEL file and associated CDF library file.

Description

SNPQStruct = affysnpquartets(*CELStruct*, *CDFStruct*, *PS*) creates *SNPQStruct*, a structure containing probe quartet results for a specific SNP probe set, specified by *PS*, from the probe-level data in a CEL file and associated CDF library file. *CELStruct* is a structure created by the affyread function from an Affymetrix CEL file. *PS* is a probe set index or probe set ID/name from *CDFStruct*, a structure created by the affyread function from an Affymetrix CDF library file associated with the CEL file. *SNPQStruct* is a structure containing the following fields.

Field	Description
`'ProbeSet'`	Identifier for the probe set.
`'AlleleA'`	String specifying the base that is allele A for the probe set.
`'AlleleB'`	String specifying the base that is allele B for the probe set.
`'Quartet'`	Structure array containing intensity values for PM (perfect match) and MM (mismatch) probe pairs, including the sense and antisense probes for alleles A and B. Each structure in the array corresponds to a probe pair in the probe set.

Examples

The following example uses the NA06985_Hind_B5_3005533.CEL file. You can download this and other sample CEL files from:

```
http://www.affymetrix.com/support/technical/sample_data/hapmap_trio_data.affx
```

The NA06985_Hind_B5_3005533.CEL file is included in the 100K_trios.hind.1.zip file.

The following example uses the CDF library file for the Mapping 50K Hind 240 array, which you can download from:

```
http://www.affymetrix.com/support/technical/byproduct.affx?product=100k
```

The following example assumes that the NA06985_Hind_B5_3005533.CEL file is stored on the MATLAB search path or in the current folder. It also assumes that the associated CDF library file, Mapping50K_Hind240.cdf, is stored at D:\Affymetrix\LibFiles\.

1 Read the contents of a CEL file into a MATLAB structure.

```
celStruct = affyread('NA06985_Hind_B5_3005533.CEL');
```

2 Read the contents of a CDF file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Mapping50K_Hind240.cdf');
```

3 Create a structure containing SNP probe quartet results for the SNP_A-1684395 probe set.

```
SNPQStruct = affysnpquartets(celStruct,cdfStruct,'SNP_A-1684395')

SNPQStruct =

    ProbeSet: 'SNP_A-1684395'
     AlleleA: 'A'
     AlleleB: 'G'
     Quartet: [1x5 struct]
```

4 View the intensity values of the first probe pair in the probe set.

```
SNPQStruct.Quartet(1)

ans =

        A_Sense_PM: 5013
        B_Sense_PM: 1290
        A_Sense_MM: 1485
        B_Sense_MM: 686
    A_Antisense_PM: 3746
    B_Antisense_PM: 1406
    A_Antisense_MM: 1527
    B_Antisense_MM: 958
```

See Also

affyread | probesetvalues

agferead

Read Agilent Feature Extraction Software file

Syntax

AGFEData = agferead(*File*)

Arguments

File	Microarray data file generated with the Agilent® Feature Extraction Software.

Description

AGFEData = agferead(*File*) reads files generated with the Feature Extraction Software from Agilent microarray scanners and creates a structure (*AGFEData*) containing the following fields:

- Header
- Stats
- Columns
- Rows
- Names
- IDs
- Data
- ColumnNames
- TextData
- TextColumnNames

The Feature Extraction Software takes an image from an Agilent microarray scanner and generates raw intensity data for each spot on the plate.

Examples

1   Read in a sample Agilent Feature Extraction Software file. Note that the file `fe_sample.txt` is not provided with the Bioinformatics Toolbox™ software.

```
agfeStruct = agferead('fe_sample.txt')
```

2   Plot the median foreground.

```
maimage(agfeStruct,'gMedianSignal');
maboxplot(agfeStruct,'gMedianSignal');
```

See Also

affyread | celintensityread | galread | geoseriesread | geosoftread | gprread | ilmnbsread | imageneread | magetfield | sptread

align2cigar

Convert aligned sequences to corresponding Compact Idiosyncratic Gapped Alignment Report (CIGAR) format strings

Syntax

```
[Cigars,Starts] = align2cigar(Alignment,Ref)
```

Description

`[Cigars,Starts] = align2cigar(Alignment,Ref)` converts aligned sequences represented in Alignment, a cell array of aligned strings or a character array, into Cigars, a cell array of corresponding CIGAR strings, using the reference sequence specified by Ref, a string. It also returns Starts, a vector of integers indicating the start position of each aligned sequence with respect to the ungapped reference sequence.

Input Arguments

Alignment

Cell array of aligned sequence strings or a character array representing aligned sequences. Soft clippings are assumed to be represented by lowercase letters in the aligned sequences. Skipped positions are assumed to be represented by . in the aligned sequences.

Default:

Ref

String specifying an aligned reference sequence. The length of Ref must equal the number of columns in Alignment.

Default:

Output Arguments

Cigars

Cell array of CIGAR strings corresponding to each aligned sequence in Alignment.

Starts

Vector of integers indicating the start position of each aligned sequence with respect to the ungapped reference sequence.

Examples

Convert aligned sequences to CIGAR strings

This example shows how to convert aligned strings to CIGAR strings

Create a cell array of aligned strings, create a string specifying a reference sequence, and then convert the alignment to CIGAR strings:

```
aln = ['ACG-ATGC'; 'ACGT-TGC'; '  GTAT-C']


aln =

ACG-ATGC
ACGT-TGC
  GTAT-C


ref =  'ACGTATGC';
[cigar, start] = align2cigar(aln, ref)


cigar =

    '3M1D4M'     '4M1D3M'     '4M1D1M'


start =

    1     1     3
```

More About

- "Manage Short-Read Sequence Data in Objects"
- Sequence Read Archive
- SAM format specification

References

[1] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Goncalo, A., and Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics *25, 16*, 2078–2079.

See Also

getAlignment | cigar2align | multialign | getBaseCoverage | getCompactAlignment | BioMap

allshortestpaths (biograph)

Find all shortest paths in biograph object

Syntax

```
[dist] = allshortestpaths(BGObj)
[dist] = allshortestpaths(BGObj, ...'Directed', DirectedValue, ...)
[dist] = allshortestpaths(BGObj, ...'Weights', WeightsValue, ...)
```

Arguments

BGObj	Biograph object created by `biograph` (object constructor).
DirectedValue	Property that indicates whether the graph is directed or undirected. Enter `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`.
WeightsValue	Column vector that specifies custom weights for the edges in the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. It must have one entry for every nonzero value (edge) in the matrix. The order of the custom weights in the vector must match the order of the nonzero values in the matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, `allshortestpaths` gets weight information from the nonzero entries in the matrix.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

`[dist] = allshortestpaths(BGObj)` finds the shortest paths between every pair of nodes in a graph represented by an N-by-N adjacency matrix extracted from a biograph

object, *BGObj*, using Johnson's algorithm. Nonzero entries in the matrix represent the weights of the edges.

Output *dist* is an N-by-N matrix where *dist*(S,T) is the distance of the shortest path from source node S to target node T. Elements in the diagonal of this matrix are always 0, indicating the source node and target node are the same. A 0 not in the diagonal indicates that the distance between the source node and target node is 0. An Inf indicates there is no path between the source node and the target node.

Johnson's algorithm has a time complexity of O(N*log(N)+N*E), where N and E are the number of nodes and edges respectively.

[...] = allshortestpaths (*BGObj*, '*PropertyName*', *PropertyValue*, ...) calls allshortestpaths with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[*dist*] = allshortestpaths(*BGObj*, ...'Directed', *DirectedValue*, ...) indicates whether the graph is directed or undirected. Set *DirectedValue* to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

[*dist*] = allshortestpaths(*BGObj*, ...'Weights', *WeightsValue*, ...) lets you specify custom weights for the edges. *WeightsValue* is a column vector having one entry for every nonzero value (edge) in the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. The order of the custom weights in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, allshortestpaths gets weight information from the nonzero entries in the N-by-N adjacency matrix.

More About

·   biograph object

References

[1] Johnson, D.B. (1977). Efficient algorithms for shortest paths in sparse networks. Journal of the ACM *24(1)*, 1-13.

[2] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

biograph | conncomp | isomorphism | maxflow | shortestpath | traverse | graphallshortestpaths | isdag | isspantree | minspantree | topoorder

aminolookup

Find amino acid codes, integers, abbreviations, names, and codons

Syntax

```
aminolookup
aminolookup(SeqAA)
aminolookup('Code', CodeValue)
aminolookup('Integer', IntegerValue)
aminolookup('Abbreviation', AbbreviationValue)
aminolookup('Name', NameValue)
```

Arguments

SeqAA	String of single-letter codes or three-letter abbreviations representing an amino acid sequence. For valid codes and abbreviations, see the table Amino Acid Lookup.
CodeValue	String specifying a single-letter code representing an amino acid. For valid single-letter codes, see the table Amino Acid Lookup.
IntegerValue	Single integer representing an amino acid. For valid integers, see the table Amino Acid Lookup.
AbbreviationValue	String specifying a three-letter abbreviation representing an amino acid. For valid three-letter abbreviations, see the table Amino Acid Lookup.
NameValue	String specifying an amino acid name. For valid amino acid names, see the table Amino Acid Lookup.

Description

aminolookup displays a table of amino acid codes, integers, abbreviations, names, and codons.

Amino Acid Lookup

Code	Integer	Abbreviation	Amino Acid Name	Codons
A	1	Ala	Alanine	GCU GCC GCA GCG
R	2	Arg	Arginine	CGU CGC CGA CGG AGA AGG
N	3	Asn	Asparagine	AAU AAC
D	4	Asp	Aspartic acid (Aspartate)	GAU GAC
C	5	Cys	Cysteine	UGU UGC
Q	6	Gln	Glutamine	CAA CAG
E	7	Glu	Glutamic acid (Glutamate)	GAA GAG
G	8	Gly	Glycine	GGU GGC GGA GGG
H	9	His	Histidine	CAU CAC
I	10	Ile	Isoleucine	AUU AUC AUA
L	11	Leu	Leucine	UUA UUG CUU CUC CUA CUG
K	12	Lys	Lysine	AAA AAG
M	13	Met	Methionine	AUG
F	14	Phe	Phenylalanine	UUU UUC
P	15	Pro	Proline	CCU CCC CCA CCG
S	16	Ser	Serine	UCU UCC UCA UCG AGU AGC
T	17	Thr	Threonine	ACU ACC ACA ACG
W	18	Trp	Tryptophan	UGG
Y	19	Tyr	Tyrosine	UAU UAC
V	20	Val	Valine	GUU GUC GUA GUG
B	21	Asx	Asparagine or Aspartic acid (Aspartate)	AAU AAC GAU GAC

Code	Integer	Abbreviation	Amino Acid Name	Codons
Z	22	Glx	Glutamine or Glutamic acid (Glutamate)	CAA CAG GAA GAG
X	23	Xaa	Any amino acid	All codons
*	24	END	Termination codon (translation stop)	UAA UAG UGA
-	25	GAP	Gap of unknown length	NA

aminolookup(*SeqAA*) converts between single-letter codes and three-letter abbreviations for an amino acid sequence. If the input is a string of single-letter codes, then the output is a character string of three-letter abbreviations. If the input is a string of three-letter abbreviations, then the output is a string of the corresponding single-letter codes.

If you enter one of the ambiguous single-letter codes B, Z, or X, this function displays the corresponding abbreviation for the ambiguous amino acid character.

```
aminolookup('abc')

ans =

AlaAsxCys
```

aminolookup('Code', *CodeValue*) displays the corresponding amino acid three-letter abbreviation and name.

aminolookup('Integer', *IntegerValue*) displays the corresponding amino acid single-letter code, three-letter abbreviation, and name.

aminolookup('Abbreviation', *AbbreviationValue*) displays the corresponding amino acid single-letter code and name.

aminolookup('Name', *NameValue*) displays the corresponding amino acid single-letter code and three-letter abbreviation.

Examples

- Convert an amino acid sequence in single-letter codes to the corresponding three-letter abbreviations.

```
aminolookup('MWKQAEDIRDIYDF')

ans =

MetTrpLysGlnAlaGluAspIleArgAspIleTyrAspPhe
```

- Convert an amino acid sequence in three-letter abbreviations to the corresponding single-letter codes.

```
aminolookup('MetTrpLysGlnAlaGluAspIleArgAspIleTyrAspPhe')

ans =

MWKQAEDIRDIYDF
```

- Display the three-letter abbreviation and name for the amino acid corresponding to the single-letter code R.

```
aminolookup('Code', 'R')

ans =

Arg Arginine
```

- Display the single-letter code, three-letter abbreviation, and name for the amino acid corresponding to the integer 1.

```
aminolookup('Integer', 1)

ans =

A Ala Alanine
```

- Display the single-letter code and name for the amino acid corresponding to the three-letter abbreviation asn.

```
aminolookup('Abbreviation', 'asn')

ans =

N Asparagine
```

• Display the single-letter code and three-letter abbreviation for the amino acid proline.

```
aminolookup('Name','proline')

ans =

P Pro
```

See Also
aa2int | aa2nt | aacount | geneticcode | int2aa | isotopicdist | nt2aa | revgeneticcode

atomiccomp

Calculate atomic composition of protein

Syntax

NumberAtoms = atomiccomp(*SeqAA*)

Arguments

SeqAA	Amino acid sequence. Enter a character string or vector of integers from the table Mapping Amino Acid Letter Codes to Integers. You can also enter a structure with the field `Sequence`.

Description

NumberAtoms = atomiccomp(*SeqAA*) counts the type and number of atoms in an amino acid sequence (*SeqAA*) and returns the counts in a 1-by-1 structure (*NumberAtoms*) with fields C, H, N, O, and S.

Examples

1 Retrieve an amino acid sequence from the NCBI GenPept database.

```
rhodopsin = getgenpept('NP_000530');
```

2 Count the atoms in the sequence.

```
rhodopsinAC = atomiccomp(rhodopsin)

rhodopsinAC =

    C: 1814
    H: 2725
    N: 423
    O: 477
```

```
        S: 25
```

3 Count the number of carbon atoms in the sequence.

```
rhodopsinAC.C
```

```
ans =
```

```
        1814
```

See Also
aacount | molweight | proteinplot

bamindexread

Read Binary Sequence Alignment/Map Index (BAI) file

Syntax

```
Index = bamindexread(File)
```

Description

`Index = bamindexread(File)` reads File, a BAI file, and returns Index, a MATLAB structure that specifies the offsets into the compressed Binary Sequence Alignment/ Map (BAM) file and decompressed data block for each reference sequence and range of positions (bins) on each reference sequence.

Input Arguments

File

String specifying a file name, or a path and a file name, of a BAM file or a BAI file. If File is a BAM file, `bamindexread` reads the corresponding BAI file, that is, the BAI file with the same root name and stored in the same folder as the BAM file. If you specify only a file name, that file must be on the MATLAB search path or in the Current Folder.

Default:

Output Arguments

Index

MATLAB array of structures that specifies the offsets into the compressed Binary Sequence Alignment/Map (BAM) file and decompressed data block for each reference sequence and range of positions (bins) on the reference sequence. Index contains the following fields.

Field	Description
Filename	Name of the BAM file or BAI file used to create the Index array of structures.
Index	A 1-by-*N* array of structures, where *N* is the number of reference sequences in the corresponding BAM file. Each structure contains the following fields: • `BinID` — Array of bin IDs for one reference sequence. • `BGZFOffsetStart` — Offset in the BAM file to the start of the first BGZF block where alignment records associated with the corresponding `BinID` are stored. • `BGZFOffsetEnd` — Offset in the BAM file to the start of the last BGZF block where alignment records associated with the corresponding `BinID` are stored. • `DataOffsetStart` — Offset in the decompressed data block to the start of where alignment records associated with the corresponding `BinID` are stored. • `DataOffsetEnd` — Offset in the decompressed data block to the end of where alignment records associated with the corresponding `BinID` are stored. • `LinearBGZFOffset` — Offset in the BAM file to the first alignment in the corresponding 16384 bp interval. • `LinearDataOffset` — Offset in the decompressed data file to the first alignment in the corresponding 16384 bp interval.

Examples

Generate an index structure from a BAM file

This example shows how to generate an index structure from a BAM index file.

```
ind = bamindexread('ex1.bam')


ind =
```

```
Filename: 'ex1.bam.bai'
   Index: [1x2 struct]
```

More About

*   "Manage Short-Read Sequence Data in Objects"
*   Sequence Read Archive
*   SAM format specification

References

[1] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Goncalo, A., and Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics *25, 16*, 2078–2079.

See Also
baminfo | bamread

baminfo

Return information about Binary Sequence Alignment/Map (BAM) file

Syntax

```
InfoStruct = baminfo(File)
InfoStruct = baminfo(File,Name,Value)
```

Description

`InfoStruct = baminfo(File)` returns a MATLAB structure containing summary information about a BAM-formatted file.

`InfoStruct = baminfo(File,Name,Value)` returns a MATLAB structure with additional options specified by one or more Name,Value pair arguments.

Input Arguments

File

String specifying a file name or path and file name of a BAM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the Current Folder.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`'ScanDictionary'`

Logical that controls the scanning of the BAM-formatted file to determine the reference names and the number of reads aligned to each reference. If `true`, the `ScannedDictionary` and `ScannedDictionaryCount` fields contain this information.

Default: `false`

`'NumOfReads'`

Logical that controls the scanning of a BAM-formatted file to determine the number of alignment records in the file. If `true`, the `NumReads` field contains this information.

Default: `false`

Output Arguments

`InfoStruct`

MATLAB structure containing summary information about a BAM-formatted file. The structure contains these fields.

Field	Description
Filename	Name of the BAM-formatted file.
FilePath	Path to the file.
FileSize	Size of the file in bytes.
FileModDate	Modification date of the file.
Header**	Structure containing the file format version, sort order, and group order.
ReadGroup**	Structure containing the: • Read group identifier • Sample • Library • Description • Platform unit • Predicted median insert size • Sequencing center • Date • Platform
SequenceDictionary**	Structure containing the:

Field	Description
	• Sequence name
	• Sequence length
	• Genome assembly identifier
	• MD5 checksum of sequence
	• URI of sequence
	• Species
Program**	Structure containing the:
	• Program name
	• Version
	• Command line
NumReads	Number of reference sequences in the BAM-formatted file.
ScannedDictionary*	Cell array of strings specifying the names of the reference sequences in the BAM-formatted file.
ScannedDictionaryCount*	Cell array specifying the number of reads aligned to each reference sequence.

* — The ScannedDictionary and ScannedDictionaryCount fields are empty if you do not set the ScanDictionary name-value pair argument to true.

** — These structures and their fields appear in the output structure only if they are in the BAM file. The information in these structures depends on the information in the BAM file.

Examples

Retrieve information about a BAM file

This example shows how to retrieve information about the ex1.bam file included with the Bioinformatics Toolbox™.

```
info = baminfo('ex1.bam','ScanDictionary',true,'numofreads',true)
```

```
info =

                 Filename: 'ex1.bam'
                 FilePath: 'B:\matlab\toolbox\bioinfo\bioinfodata'
                 FileSize: 126692
              FileModDate: '07-May-2010 16:12:04'
                   Header: [1x1 struct]
                ReadGroup: [1x2 struct]
       SequenceDictionary: [1x2 struct]
                 NumReads: 3307
         ScannedDictionary: {2x1 cell}
    ScannedDictionaryCount: [2x1 uint64]
```

List the number of references found in the BAM file.

```
numel(info.ScannedDictionary)
```

```
ans =

    2
```

Alternatively, you can use the available header information from a BAM file to find out the number of references, thus avoiding the whole traversal of the source file.

```
info = baminfo('ex1.bam');
NRefs = numel(info.SequenceDictionary)
```

```
NRefs =

    2
```

More About

Tips
Use `baminfo` to investigate the size and content of a BAM-formatted file, including reference sequence names, before using the `bamread` function to read the file contents into a MATLAB structure.

- "Manage Short-Read Sequence Data in Objects"
- Sequence Read Archive
- SAM format specification

References

[1] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Goncalo, A., and Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics *25, 16*, 2078–2079.

See Also
bamindexread | bamread

bamread

Read data from Binary Sequence Alignment/Map (BAM) file

Syntax

```
BAMStruct = bamread(File,RefSeq,Range)
[BAMStruct,HeaderStruct] = bamread(File,RefSeq,Range)
... = bamread(File,RefSeq,Range,Name,Value)
```

Description

`BAMStruct = bamread(File,RefSeq,Range)` reads the alignment records in File, a BAM-formatted file, that align to RefSeq, a reference sequence, in the range specified by Range. It returns the alignment data in BAMStruct, a MATLAB array of structures.

`[BAMStruct,HeaderStruct] = bamread(File,RefSeq,Range)` also returns the header information in HeaderStruct, a MATLAB structure.

`... = bamread(File,RefSeq,Range,Name,Value)` reads the alignment records with additional options specified by one or more Name,Value pair arguments.

Input Arguments

File

String specifying a file name or path and file name of a BAM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the Current Folder.

Default:

RefSeq

Either of the following:

- String specifying the name of a reference sequence in the BAM file.
- Positive integer specifying the index of a reference sequence in the BAM file. This number is also the index of the reference sequence in the `Reference` field of the `InfoStruct` structure returned by `baminfo`.

Default:

`Range`

Two-element vector specifying the begin and end range positions on the reference sequence, RefSeq. Both values must be positive, and are one-based. The second value must be ≥ to the first value.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`'Full'`

Controls the return of only alignment records that are fully contained within the range specified by Range. Choices are `true` or `false` (default).

Default: `false`

`'Tags'`

Controls the reading of the optional tags in addition to the first 11 fields for each alignment in the BAM-formatted file. Choices are `true` (default) or `false`.

Default: `true`

`'ToFile'`

String specifying a nonexisting file name or a path and file name for saving the alignment records in the specified range of a specific reference sequence. The `ToFile` name-value pair argument creates a SAM-formatted file. If you specify only a file name, the file is saved to the MATLAB Current Folder.

The SAM-formatted file is always one-based, even if you set the `ZeroBased` name-value pair argument to `true`. You can use the SAM-formatted file as input when creating a `BioMap` object.

Default:

`'ZeroBased'`

Logical specifying whether `bamread` uses zero-based indexing when reading a file. The logical controls the return of zero-based or one-based positions in the `Position` and `MatePosition` fields in BAMStruct. Choices are `true` or `false` (default), which returns one-based positions.

This name-value pair argument affects the `Position` and `MatePosition` fields of BAMStruct. It does not affect the Range input argument or the SAM file created when using the `ToFile` name-value pair argument. SAM files are always one-based.

Caution If you plan to use the BAMStruct output argument to construct a `BioMap` object, make sure the `ZeroBased` name-value pair argument is `false`.

Default: `false`

Output Arguments

BAMStruct

An *N*-by-1 array of structures containing sequence alignment and mapping information from a BAM-formatted file, where *N* is the number of alignment records stored in the specified range. Each structure contains the following fields.

Field	Description
QueryName	Name of the read sequence (if unpaired) or the name of sequence pair (if paired).
Flag	Integer indicating the bit-wise information that specifies the status of each of 11 flags described by the SAM format specification. **Tip** You can use the `bitget` function to determine the status of a specific SAM flag.
ReferenceIndex	Index of the reference sequence.

Field	Description
	Tip To convert this index to a reference name, see the Reference field in the HeaderStruct output argument
Position	Position of the forward reference sequence where the leftmost base of the alignment of the read sequence starts. This position is zero-based or one-based, depending on the ZeroBased name-value pair argument.
MappingQuality	Integer specifying the mapping quality score for the read sequence.
CigarString	CIGAR-formatted string representing how the read sequence aligns with the reference sequence.
MateReferenceIndex	Index of the reference sequence associated with the mate. If there is no mate, then this value is 0.
MatePosition	Position of the forward reference sequence where the leftmost base of the alignment of the mate of the read sequence starts. This position is zero-based or one-based, depending on the ZeroBased name-value pair argument.
InsertSize	The number of base positions between the read sequence and its mate, when both are mapped to the same reference sequence. Otherwise, this value is 0.
Sequence	String containing the letter representations of the read sequence. It is the reverse complement if the read sequence aligns to the reverse strand of the reference sequence.
Quality	String containing the ASCII representation of the per-base quality score for the read sequence. The quality score is reversed if the read sequence aligns to the reverse strand of the reference sequence.

Field	Description
Tags	List of applicable SAM tags and their values.

HeaderStruct

MATLAB structure containing header information for the BAM-formatted file in the following fields.

Field	Description
NRefs	Number of reference sequences in the BAM-formatted file.
Reference	1-by-NRefs array of structures containing these fields: • Name — Name of the reference sequence. • Length — Length of the reference sequence.
Header*	Structure containing the file format version, sort order, and group order.
SequenceDictionary*	Structure containing the: • Sequence name • Sequence length • Genome assembly identifier • MD5 checksum of sequence • URI of sequence • Species
ReadGroup*	Structure containing the: • Read group identifier • Sample • Library • Description • Platform unit • Predicted median insert size

Field	Description
	• Sequencing center
	• Date
	• Platform
Program*	Structure containing the:
	• Program name
	• Version
	• Command line

* These structures and their fields appear in the output structure only if they are present in the BAM file. The information in these structures depends on the information present in the BAM file.

Examples

Retrieve alignment records that align to reference sequences

Read multiple alignment records from the ex1.bam file that align to two different reference sequences.

```
data1 = bamread('ex1.bam', 'seq1', [100 200])
data2 = bamread('ex1.bam', 'seq2', [100 200])


data1 =

59x1 struct array with fields:

    QueryName
    Flag
    Position
    MappingQuality
    CigarString
    MatePosition
    InsertSize
    Sequence
    Quality
    Tags
```

```
    ReferenceIndex
    MateReferenceIndex


data2 =

79x1 struct array with fields:

    QueryName
    Flag
    Position
    MappingQuality
    CigarString
    MatePosition
    InsertSize
    Sequence
    Quality
    Tags
    ReferenceIndex
    MateReferenceIndex
```

Read alignments from the ex1.bam file that are fully contained in the 100 to 200 bp range of the seq1 reference sequence.

```
data3 = bamread('ex1.bam', 'seq1', [100 200], 'full', true)


data3 =

31x1 struct array with fields:

    QueryName
    Flag
    Position
    MappingQuality
    CigarString
    MatePosition
    InsertSize
    Sequence
    Quality
    Tags
    ReferenceIndex
    MateReferenceIndex
```

Read alignments from the ex1.bam file that align to the 100 to 300 bp range of the seq1 reference sequence. Read the same alignments using zero-based indexing. Compare the position of the 27th record in the two outputs.

```
data_one = bamread('ex1.bam','seq1', [100 300]);
data_zero = bamread('ex1.bam','seq1', [100 300], 'zerobased', true);
data_one(27).Position


ans =

        135


data_zero(27).Position

ans =

        134
```

More About

Tips

- The `bamread` function requires a BAM file.
- Use the `baminfo` function to investigate the size and content, including reference sequence names, of a BAM-formatted file before using the `bamread` function to read the file contents into a MATLAB array of structures.
- If your BAM-formatted file is too large to read using available memory, try either of the following:

  - Use a smaller range.
  - Use `bamread` without specifying outputs, but using the `ToFile` Name,Value pair arguments to create a SAM-formatted file. You can then use `samread` with the `BlockRead` Name,Value pair arguments to read the SAM-formatted file. Or you can pass the SAM-formatted file to the `BioIndexedFile` constructor function to construct a `BioIndexedFile` object, which you can use to create a `BioMap` object.

- Use the BAMStruct output argument that `bamread` returns to construct a `BioMap` object, which lets you explore, access, filter, and manipulate all or a subset of the data, before doing subsequent analyses or viewing the data.

- "Manage Short-Read Sequence Data in Objects"

- "Work with Large Multi-Entry Text Files"

- Sequence Read Archive

- SAM format specification

References

[1] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Goncalo, A., and Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics *25, 16*, 2078–2079.

See Also

`BioIndexedFile` | `BioMap` | `bamindexread` | `baminfo` | `samread` | `saminfo` | `soapread` | `fastqwrite` | `fastqinfo` | `fastainfo` | `fastaread` | `fastawrite` | `sffinfo` | `sffread` | `fastqread`

basecount

Count nucleotides in sequence

Syntax

```
NTStruct = basecount(SeqNT)
NTStruct = basecount(SeqNT, ...'Ambiguous', AmbiguousValue, ...)
NTStruct = basecount(SeqNT, ...'Gaps', GapsValue, ...)
NTStruct = basecount(SeqNT, ...'Chart', ChartValue, ...)
```

Input Arguments

SeqNT	One of the following: • String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers • Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes • MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, or `getgenbank`.
AmbiguousValue	String specifying how to treat ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N). Choices are: • `'ignore'` (default) — Skips ambiguous characters • `'bundle'` — Counts ambiguous characters and reports the total count in the `Ambiguous` field. • `'prorate'` — Counts ambiguous characters and distributes them proportionately in the appropriate fields. For example, the counts for the character R are distributed evenly between the A and G fields.

	• `'individual'` — Counts ambiguous characters and reports them in individual fields. • `'warn'` — Skips ambiguous characters and displays a warning.
GapsValue	Specifies whether gaps, indicated by a hyphen (-), are counted or ignored. Choices are `true` or `false` (default).
ChartValue	String specifying a chart type. Choices are `'pie'` or `'bar'`.

Output Arguments

NTStruct	1-by-1 MATLAB structure containing the fields A, C, G, and T.

Description

NTStruct = basecount(*SeqNT*) counts the number of each type of base in SeqNT, a nucleotide sequence, and returns the counts in *NTStruct*, a 1-by-1 MATLAB structure containing the fields A, C, G, and T.

• The character U is added to the T field.

• Ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N), and gaps, indicated by a hyphen (-), are ignored by default.

• Unrecognized characters are ignored and cause the following warning message.

  Warning: Unknown symbols appear in the sequence. These will be ignored.

NTStruct = basecount(*SeqNT*, ...'*PropertyName*', *PropertyValue*, ...) calls basecount with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

NTStruct = basecount(*SeqNT*, ...'Ambiguous', *AmbiguousValue*, ...) specifies how to treat ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N). Choices are:

• `'ignore'` (default)

- `'bundle'`
- `'prorate'`
- `'individual'`
- `'warn'`

NTStruct = basecount(*SeqNT*, ...'Gaps', *GapsValue*, ...) specifies whether gaps, indicated by a hyphen (-), are counted or ignored. Choices are `true` or `false` (default).

NTStruct = basecount(*SeqNT*, ...'Chart', *ChartValue*, ...) creates a chart showing the relative proportions of the nucleotides. *ChartValue* can be `'pie'` or `'bar'`.

Examples

1   Count the bases in a DNA sequence and return the results in a structure.

```
Bases = basecount('TAGCTGGCCAAGCGAGCTTG')

Bases =

    A: 4
    C: 5
    G: 7
    T: 4
```

2   Get the count for adenosine (A) bases.

```
Bases.A

ans =

    4
```

3   Count the bases in a DNA sequence containing ambiguous characters, listing the ambiguous characters in separate fields.

```
basecount('ABCDGGCCAAGCGAGCTTG','Ambiguous','individual')

ans =

    A: 4
    C: 5
```

```
G: 6
T: 2
R: 0
Y: 0
K: 0
M: 0
S: 0
W: 0
B: 1
D: 1
H: 0
V: 0
N: 0
```

See Also

aacount | baselookup | codoncount | cpgisland | dimercount | nmercount |
ntdensity | seqviewer

baselookup

Find nucleotide codes, integers, names, and complements

Syntax

```
baselookup
baselookup('Complement', SeqNT)
baselookup('Code', CodeValue)
baselookup('Integer', IntegerValue)
baselookup('Name', NameValue)
```

Arguments

SeqNT	Nucleotide sequence(s) represented by one of the following: • String of single-letter codes from the table Nucleotide Lookup • Cell array of sequences • Two-dimensional character array of sequences **Note:** If the input is multiple sequences, the complement for each sequence is determined independently.
CodeValue	Nucleotide letter code represented by one of the following: • String specifying a single-letter code representing a nucleotide. For valid single-letter codes, see the table Nucleotide Lookup. • Cell array of letter codes. • Two-dimensional character array of letter codes.
IntegerValue	Single integer representing a nucleotide. For valid integers, see the table Nucleotide Lookup.
NameValue	Nucleotide name represented by one of the following:

	• String specifying a nucleotide name. For valid nucleotide names, see the table Nucleotide Lookup.
	• Cell array of names.
	• Two-dimensional character array of names.

Description

baselookup displays a table of nucleotide codes, integers, names, and complements.

Nucleotide Lookup

Code	Integer	Nucleotide Name	Meaning	Complement
A	1	Adenine	A	T
C	2	Cytosine	C	G
G	3	Guanine	G	C
T	4	Thymine	T	A
U	4	Uracil	U	A
R	5	Purine	A or G	Y
Y	6	Pyrimidine	C or T	R
K	7	Keto	G or T	M
M	8	Amino	A or C	K
S	9	Strong interaction (3 H bonds)	C or G	S
W	10	Weak interaction (2 H bonds)	A or T	W
B	11	Not A	C or G or T	V
D	12	Not C	A or G or T	H
H	13	Not G	A or C or T	D
V	14	Not T or U	A or C or G	B
N, X	15	Any nucleotide	A or C or G or T or U	N
-	16	Gap of indeterminate length	Gap	-

`baselookup('Complement', `*`SeqNT`*`)` displays the complementary nucleotide sequence.

`baselookup('Code', `*`CodeValue`*`)` displays the corresponding meaning and nucleotide name. For ambiguous nucleotide codes (R, Y, K, M, S, W, B, D, H, V, N, and X), the nucleotide name is a descriptive name.

`baselookup('Integer', `*`IntegerValue`*`)` displays the corresponding letter code, meaning, and nucleotide name.

`baselookup('Name', `*`NameValue`*`)` displays the corresponding letter code, meaning, and nucleotide name or descriptive name.

Examples

- Convert a nucleotide sequence to its complementary sequence.

  ```
  baselookup('Complement', 'TAGCTGRCCAAGGCCAAGCGAGCTTN')

  ans =

  ATCGACYGGTTCCGGTTCGCTCGAAN
  ```
- Display the meaning and nucleotide name or descriptive name for the nucleotide codes G and Y.

  ```
  baselookup('Code', 'G')

  ans =

  G  Guanine


  baselookup('Code', 'Y')

  ans =

  T|C  pYrimidine
  ```
- Display the nucleotide letter code, meaning, and nucleotide name or descriptive name for the integers 1 and 7.

  ```
  baselookup('Integer', 1)
  ```

```
ans =

A  A - Adenine


baselookup('Integer', 7)

ans =

K  G|T - Keto
```

- Display the corresponding nucleotide letter code, meaning, and name for cytosine and purine.

```
baselookup('Name','cytosine')

ans =

C C - Cytosine


baselookup('Name','purine')

ans =

R G|A - puRine
```

See Also

aa2nt | basecount | codoncount | dimercount | geneticcode | int2nt | nt2aa | nt2int | revgeneticcode | seqviewer

biograph object

Data structure containing generic interconnected data used to implement directed graph

Description

A biograph object is a data structure containing generic interconnected data used to implement a directed graph. Nodes represent proteins, genes, or any other biological entity, and edges represent interactions, dependences, or any other relationship between the nodes. A biograph object also stores information, such as color properties and text label characteristics, used to create a 2-D visualization of the graph.

You create a biograph object using the object constructor function `biograph`. You can view a graphical representation of a biograph object using the `view` method.

Method Summary

Following are methods of a biograph object:

allshortestpaths (biograph)

Find all shortest paths in biograph object

conncomp (biograph)

Find strongly or weakly connected components in biograph object

dolayout (biograph)

Calculate node positions and edge trajectories

get (biograph)

Retrieve information about biograph object

getancestors (biograph)

Find ancestors in biograph object

getdescendants (biograph)

Find descendants in biograph object

getedgesbynodeid (biograph)

Get handles to edges in biograph object

getmatrix (biograph)

Get connection matrix from biograph object

getnodesbyid (biograph)

Get handles to nodes

getrelatives (biograph)

Find relatives in biograph object

isdag (biograph)

Test for cycles in biograph object

isomorphism (biograph)

Find isomorphism between two biograph objects

isspantree (biograph)

Determine if tree created from biograph object is spanning tree

maxflow (biograph)

Calculate maximum flow in biograph object

minspantree (biograph)

Find minimal spanning tree in biograph object

set (biograph)

Set property of biograph object

shortestpath (biograph)

Solve shortest path problem in biograph object

topoorder (biograph)

Perform topological sort of directed acyclic graph extracted from biograph object

traverse (biograph)

Traverse biograph object by following adjacent nodes

view (biograph)

Draw figure from biograph object

Following are methods of a node object:

getancestors (biograph)

Find ancestors in biograph object

getdescendants (biograph)

Find descendants in biograph object

getrelatives (biograph)

Find relatives in biograph object

Property Summary

A biograph object contains two kinds of objects, node objects and edge objects, that have their own properties. For a list of the properties of node objects and edge objects, see the following tables.

Properties of a Biograph Object

Property	Description
ID	String to identify the biograph object. Default is ''.
Label	String to label the biograph object. Default is ''.
Description	String that describes the biograph object. Default is ''.
LayoutType	String that specifies the algorithm for the layout engine. Choices are: • `'hierarchical'` (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges. • `'radial'` — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges. • `'equilibrium'` — Calculates layout by minimizing the energy in a dynamic spring system.
EdgeType	String that specifies how edges display. Choices are: • `'straight'` • `'curved'` (default) • `'segmented'`

Property	Description
	Note: Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with `LayoutType` equal to `'equilibrium'` or `'radial'` cannot produce curved or segmented edges.
`Scale`	Positive number that post-scales the node coordinates. Default is `1`.
`LayoutScale`	Positive number that scales the size of the nodes before calling the layout engine. Default is `1`.
`EdgeTextColor`	Three-element numeric vector of RGB values. Default is `[0, 0, 0]`, which defines black.
`EdgeFontSize`	Positive number that sets the size of the edge font in points. Default is `8`.
`ShowArrows`	Controls the display of arrows with the edges. Choices are `'on'` (default) or `'off'`.
`ArrowSize`	Positive number that sets the size of the arrows in points. Default is `8`.
`ShowWeights`	Controls the display of text indicating the weight of the edges. Choices are `'on'` or `'off'` (default).
`ShowTextInNodes`	String that specifies the node property used to label nodes when you display a biograph object using the `view` method. Choices are: • `'Label'` — Uses the `Label` property of the node object (default). • `'ID'` — Uses the `ID` property of the node object. • `'None'`
`NodeAutoSize`	Controls precalculating the node size before calling the layout engine. Choices are `'on'` (default) or `'off'`. **Note:** Set it to `off` if you want to apply different node sizes by changing the `Size` property.
`NodeCallback`	User-defined callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple

Property	Description
	function handles. After using the `view` function to display the biograph object in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, `@(node) inspect(node)`, which displays the Property Inspector dialog box.
EdgeCallback	User-defined callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the `view` function to display the biograph object in the Biograph Viewer, you can double-click an edge to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, `@(edge) inspect(edge)`, which displays the Property Inspector dialog box.
CustomNodeDrawFcn	Function handle to a customized function to draw nodes. Default is `[]`.
Nodes	Read-only column vector with handles to node objects of a biograph object. The size of the vector is the number of nodes. For properties of node objects, see Properties of a Node Object.
Edges	Read-only column vector with handles to edge objects of a biograph object. The size of the vector is the number of edges. For properties of edge objects, see Properties of an Edge Object.

Properties of a Node Object

Property	Description
ID	Character string defined when the biograph object is created, either by the *NodeIDs* input argument or internally by the `biograph` constructor function. You can modify this property using the `set` method, but each node object's `ID` must be unique.
Label	String for labeling a node when you display a biograph object using the `view` method. Default is `''`.
Description	String that describes the node. Default is `''`.

Property	Description
Position	Two-element numeric vector of *x*- and *y*-coordinates, for example, [150, 150]. If you do not specify this property, default is initially [], then when the layout algorithms are executed, it becomes a two-element numeric vector of *x*- and *y*-coordinates computed by the layout engine.
Shape	String that specifies the shape of the nodes. Choices are: • 'box'(default) • 'ellipse' • 'circle' • 'rect' • 'rectangle' • 'diamond' • 'trapezium' • 'invtrapezium' • 'house' • 'invhouse' • 'parallelogram'
Size	Two-element numeric vector calculated before calling the layout engine using the actual font size and shape of the node. Default is [10, 10].
Color	Three-element numeric vector of RGB values that specifies the fill color of the node. Default is [1, 1, 0.7], which defines yellow.
LineWidth	Positive number. Default is 1.
LineColor	Three-element numeric vector of RGB values that specifies the outline color of the node. Default is [0.3, 0.3, 1], which defines blue.
FontSize	Positive number that sets the size of the node font in points. Default is 8.
TextColor	Three-element numeric vector of RGB values that specifies the color of the node labels. Default is [0, 0, 0], which defines black.

Property	Description
UserData	Miscellaneous, user-defined data that you want to associate with the node. The node does not use this property, but you can access and specify it using the get and set functions. Default is [].

Properties of an Edge Object

Property	Description
ID	Character string automatically generated from the node IDs when the biograph object is created by the biograph constructor function. You can modify this property using the set method, but each edge object's ID must be unique.
Label	String for labeling an edge. Default is ''.
Description	String that describes the edge. Default is ''.
Weight	Value that represents the weight (cost, distance, length, or capacity) associated with the edge. Default is 1.
LineWidth	Positive number. Default is 1.
LineColor	Three-element numeric vector of RGB values that specifies the color of the edge. Default is [0.5, 0.5, 0.5], which defines gray.
UserData	Miscellaneous, user-defined data that you want to associate with the edge. The edge does not use this property, but you can access and specify it using the get and set functions. Default is [].

Examples

Create a Biograph object and specify its properties

This example shows how to create a biograph object, access, and update its properties.

Create a biograph object with custom node IDs.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
ids = {'M30931','L07625','K03454','M27323','M15390'};
bg1 = biograph(cm,ids)

Biograph object with 5 nodes and 9 edges.
```

Specify the **ID** property of the object.

```
bg1.ID = 'mybg';
```

Use the **get** function to display the node IDs.

```
get(bg1.nodes,'ID')


ans =

    'M30931'
    'L07625'
    'K03454'
    'M27323'
    'M15390'
```

Display all properties and their current values of the 5th node and 5th edge of the object.

```
bg1.nodes(5)
             ID: 'M15390'
          Label: ''
    Description: ''
       Position: []
          Shape: 'box'
           Size: [10 10]
          Color: [1 1 0.7000]
      LineWidth: 1
      LineColor: [0.3000 0.3000 1]
       FontSize: 9
      TextColor: [0 0 0]
       UserData: []


bg1.edges(5)
             ID: 'L07625 -> M15390'
          Label: ''
    Description: ''
         Weight: 1
      LineWidth: 0.5000
      LineColor: [0.5000 0.5000 0.5000]
       UserData: []
```

Set the `LineWidth` property of the 5th node to 2.

```
bg1.nodes(5).LineWidth = 2.0;
bg1.nodes(5)

            ID: 'M15390'
         Label: ''
   Description: ''
      Position: []
         Shape: 'box'
          Size: [10 10]
         Color: [1 1 0.7000]
     LineWidth: 2
     LineColor: [0.3000 0.3000 1]
      FontSize: 9
     TextColor: [0 0 0]
      UserData: []
```

Alternatively use `getnodesbyid` function to create a handle for the 5th node, and set its `Shape` property to 'circle'.

```
nh1 = getnodesbyid(bg1,'M15390')

            ID: 'M15390'
         Label: ''
   Description: ''
      Position: []
         Shape: 'box'
          Size: [10 10]
         Color: [1 1 0.7000]
     LineWidth: 2
     LineColor: [0.3000 0.3000 1]
      FontSize: 9
     TextColor: [0 0 0]
      UserData: []
```

```
nh1.Shape = 'circle';
```

Specify the `LineColor` property of the 5th edge.

```
bg1.edges(5).LineColor = [0.7 0.0 0.1];
```

Alternatively use `getedgesbynodeid` to retrieve the handel to the edge by providing a source node id and a sink node id.

```
eh1 = getedgesbynodeid(bg1,'L07625','M15390')

           ID: 'L07625 -> M15390'
        Label: ''
  Description: ''
       Weight: 1
    LineWidth: 0.5000
    LineColor: [0.7000 0 0.1000]
     UserData: []
```

Use the handle to specify the `LineWidth` property or any other properties of the edge.

```
eh1.LineWidth = 2.0;
```

View the biograph object.

```
view(bg1)
```
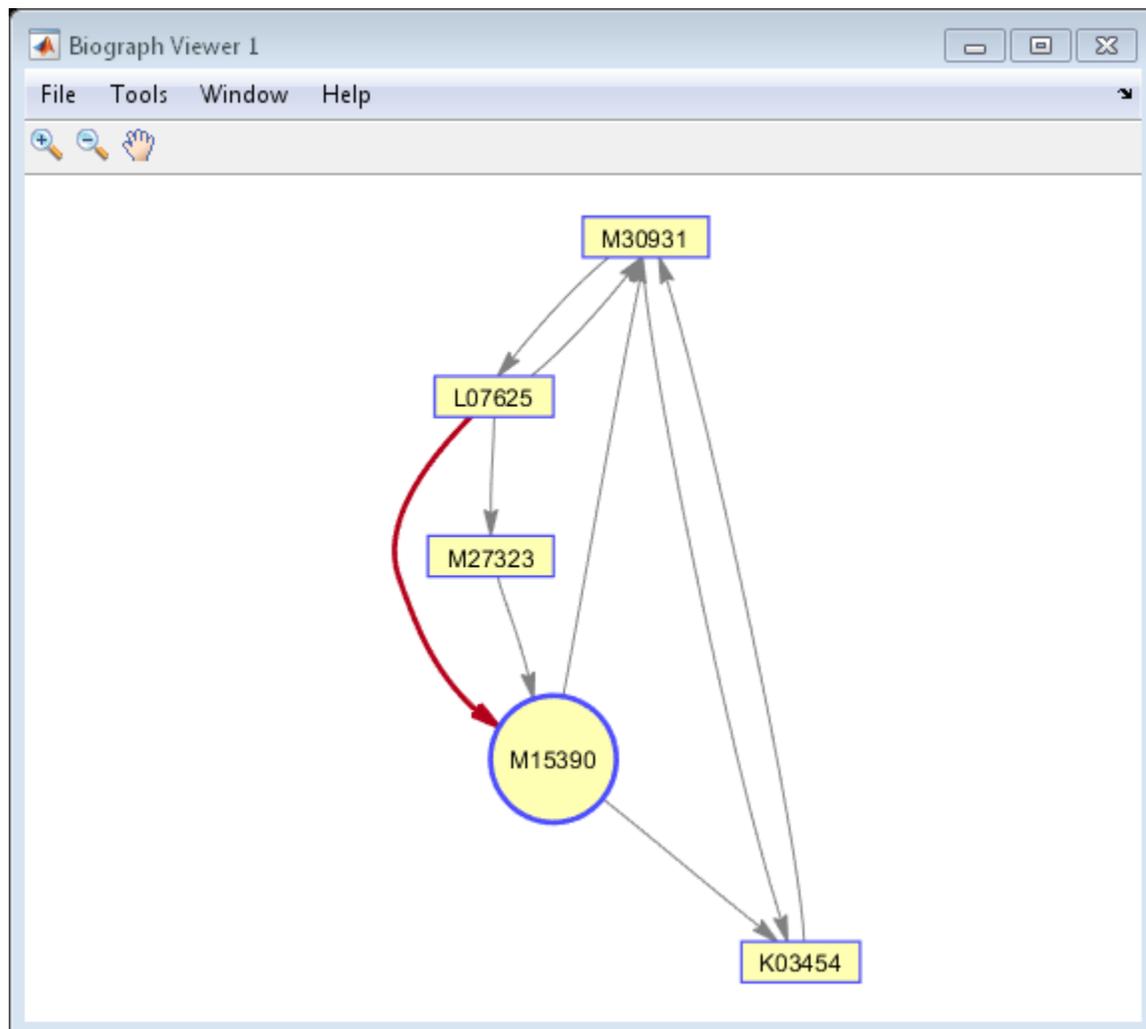
See Also

biograph | conncomp | get | getdescendants | getmatrix | getrelatives
| isomorphism | maxflow | set | topoorder | view | allshortestpaths |
dolayout | getancestors | getedgesbynodeid | getnodesbyid | isdag |
isspantree | minspantree | shortestpath | traverse

biograph

Create biograph object

Syntax

```
BGobj = biograph(CMatrix)
BGobj = biograph(CMatrix, NodeIDs)
BGobj = biograph(CMatrix, NodeIDs, ...'ID', IDValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'Label', LabelValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'Description',
DescriptionValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'LayoutType',
LayoutTypeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeType',
EdgeTypeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'Scale', ScaleValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'LayoutScale',
LayoutScaleValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeTextColor',
EdgeTextColorValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeFontSize',
EdgeFontSizeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ShowArrows',
ShowArrowsValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ArrowSize',
ArrowSizeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ShowWeights',
ShowWeightsValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ShowTextInNodes',
ShowTextInNodesValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'NodeAutoSize',
NodeAutoSizeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'NodeCallback',
NodeCallbackValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeCallback',
EdgeCallbackValue, ...)
```

```
BGobj = biograph(CMatrix, NodeIDs, ...'CustomNodeDrawFcn',
CustomNodeDrawFcnValue, ...)
```

Arguments

CMatrix	Full or sparse square matrix that acts as a connection matrix. That is, a value of 1 indicates a connection between nodes while a 0 indicates no connection. The number of rows/columns is equal to the number of nodes.
NodeIDs	Node identification strings. Enter any of the following: • Cell array of strings with the number of strings equal to the number of rows or columns in the connection matrix *CMatrix*. Each string must be unique. • Character array with the number of rows equal to the number of nodes. Each row in the array must be unique. • String with the number of characters equal to the number of nodes. Each character must be unique. Default values are the row or column numbers. **Note:** You must specify *NodeIDs* if you want to specify property name/value pairs. Set *NodeIDs* to [ ] to use the default values of the row/column numbers.
IDValue	String to identify the biograph object. Default is ' '.
LabelValue	String to label the biograph object. Default is ' '.
DescriptionValue	String that describes the biograph object. Default is ' '.
LayoutTypeValue	String that specifies the algorithm for the layout engine. Choices are: • 'hierarchical' (default) — Uses a topological order of the graph to assign levels, and then

	arranges the nodes from top to bottom, while minimizing crossing edges.
	• `'radial'` — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges.
	• `'equilibrium'` — Calculates layout by minimizing the energy in a dynamic spring system.
EdgeTypeValue	String that specifies how edges display. Choices are:
	• `'straight'`
	• `'curved'` (default)
	• `'segmented'`
	Note: Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with `LayoutType` equal to `'equilibrium'` or `'radial'` cannot produce curved or segmented edges.
ScaleValue	Positive number that post-scales the node coordinates. Default is `1`.
LayoutScaleValue	Positive number that scales the size of the nodes before calling the layout engine. Default is `1`.
EdgeTextColorValue	Three-element numeric vector of RGB values. Default is `[0, 0, 0]`, which defines black.
EdgeFontSizeValue	Positive number that sets the size of the edge font in points. Default is `8`.
ShowArrowsValue	Controls the display of arrows for the edges. Choices are `'on'` (default) or `'off'`.
ArrowSizeValue	Positive number that sets the size of the arrows in points. Default is `8`.
ShowWeightsValue	Controls the display of text indicating the weight of the edges. Choices are `'on'` or `'off'` (default).

ShowTextInNodesValue	String that specifies the node property used to label nodes when you display a biograph object using the view method. Choices are: • `'Label'` — Uses the Label property of the node object (default). • `'ID'` — Uses the ID property of the node object. • `'None'`
NodeAutoSizeValue	Controls precalculating the node size before calling the layout engine. Choices are `'on'` (default) or `'off'`. **Note:** Set it to off if you want to apply different node sizes by changing the Size property.
NodeCallbackValue	User callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is `@(node) inspect(node)`, which displays the Property Inspector dialog box.
EdgeCallbackValue	User callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph in the Biograph Viewer, you can double-click an edge to activate the first callback, or right-click and select a callback to activate. Default is `@(edge) inspect(edge)`, which displays the Property Inspector dialog box.
CustomNodeDrawFcnValue	Function handle to a customized function to draw nodes. Default is `[]`.

Description

BGobj = biograph(*CMatrix*) creates a biograph object, *BGobj*, using a connection matrix, *CMatrix*. All nondiagonal and positive entries in the connection matrix, *CMatrix*, indicate connected nodes, rows represent the source nodes, and columns represent the sink nodes.

BGobj = biograph(*CMatrix, NodeIDs*) specifies the node identification strings. *NodeIDs* can be:

- Cell array of strings with the number of strings equal to the number of rows or columns in the connection matrix *CMatrix*. Each string must be unique.
- Character array with the number of rows equal to the number of nodes. Each row in the array must be unique.
- String with the number of characters equal to the number of nodes. Each character must be unique.

Default values are the row or column numbers.

Note: If you want to specify property name/value pairs, you must specify *NodeIDs*. Set *NodeIDs* to [] to use the default values of the row/column numbers.

BGobj = biograph(..., '*PropertyName*', *PropertyValue*, ...) calls biograph with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

BGobj = biograph(*CMatrix, NodeIDs*, ...'ID', *IDValue*, ...) specifies an ID for the biograph object. Default is ''.

BGobj = biograph(*CMatrix, NodeIDs*, ...'Label', *LabelValue*, ...) specifies a label for the biograph object. Default is ''.

BGobj = biograph(*CMatrix, NodeIDs*, ...'Description', *DescriptionValue*, ...) specifies a description of the biograph object. Default is ''.

BGobj = biograph(*CMatrix, NodeIDs*, ...'LayoutType', *LayoutTypeValue*, ...) specifies the algorithm for the layout engine.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'EdgeType',
EdgeTypeValue, ...) specifies how edges display.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'Scale', *ScaleValue*, ...) post-scales the node coordinates. Default is 1.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'LayoutScale',
LayoutScaleValue, ...) scales the size of the nodes before calling the layout engine. Default is 1.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'EdgeTextColor',
EdgeTextColorValue, ...) specifies a three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'EdgeFontSize',
EdgeFontSizeValue, ...) sets the size of the edge font in points. Default is 8.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'ShowArrows',
ShowArrowsValue, ...) controls the display of arrows for the edges. Choices are 'on' (default) or 'off'.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'ArrowSize',
ArrowSizeValue, ...) sets the size of the arrows in points. Default is 8.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'ShowWeights',
ShowWeightsValue, ...) controls the display of text indicating the weight of the edges. Choices are 'on' (default) or 'off'.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'ShowTextInNodes',
ShowTextInNodesValue, ...) specifies the node property used to label nodes when you display a biograph object using the view method.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'NodeAutoSize',
NodeAutoSizeValue, ...) controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'NodeCallback',
NodeCallbackValue, ...) specifies user callback for all nodes.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'EdgeCallback',
EdgeCallbackValue, ...) specifies user callback for all edges.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'CustomNodeDrawFcn', *CustomNodeDrawFcnValue*, ...) specifies function handle to customized function to draw nodes. Default is [].

Examples

Create a biograph object

This example shows how to create a biograph object.

Create a biograph object with default node IDs, and then use the `get` function to display the node IDs.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg1 = biograph(cm)

Biograph object with 5 nodes and 9 edges.

get(bg1.nodes,'ID')


ans =

    'Node 1'
    'Node 2'
    'Node 3'
    'Node 4'
    'Node 5'
```

Create a biograph object, assign the node IDs, and then use the `get` function to display the node IDs.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
ids = {'M30931','L07625','K03454','M27323','M15390'};
bg2 = biograph(cm,ids);
get(bg2.nodes,'ID')


ans =

    'M30931'
```

```
'L07625'
'K03454'
'M27323'
'M15390'
```

Display the biograph object.

```
view(bg2)
```
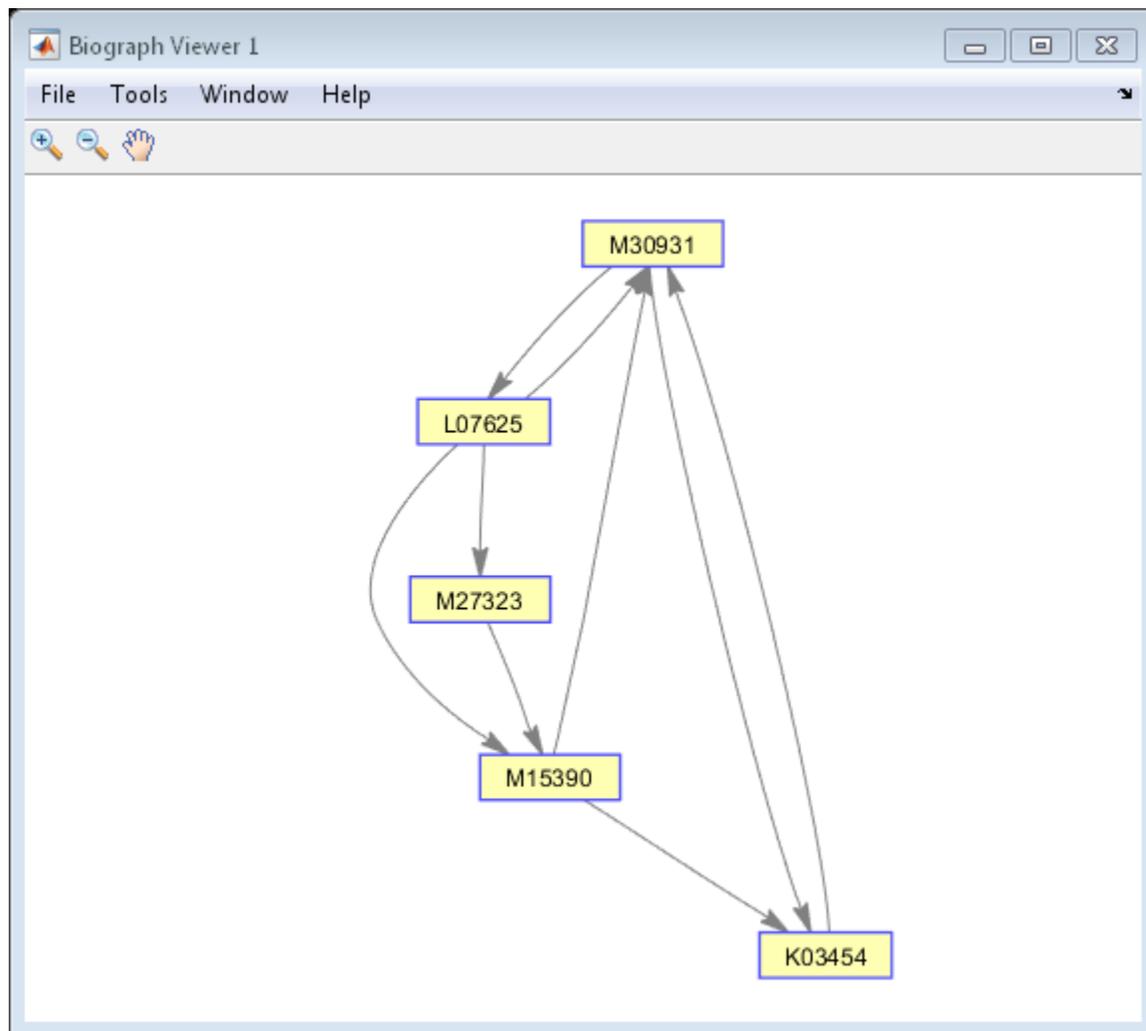
More About

- biograph object

See Also

conncomp | get | getdescendants | getmatrix | getrelatives | isomorphism | maxflow | set | topoorder | view | allshortestpaths | dolayout | getancestors | getedgesbynodeid | getnodesbyid | isdag | isspantree | minspantree | shortestpath | traverse

BioIndexedFile class

Allow quick and efficient access to large text file with nonuniform-size entries

Description

The `BioIndexedFile` class allows access to text files with nonuniform-size entries, such as sequences, annotations, and cross-references to data sets. It lets you quickly and efficiently access this data without loading the source file into memory.

This class lets you access individual entries or a subset of entries when the source file is too big to fit into memory. You can access entries using indices or keys. You can read and parse one or more entries using provided interpreters or a custom interpreter function.

Construction

`BioIFobj = BioIndexedFile(Format,SourceFile)` returns a `BioIndexedFile` object BioIFobj that indexes the contents of SourceFile following the parsing rules defined by Format, where SourceFile and Format are strings specifying a text file and a file format, respectively. It also constructs an auxiliary index file to store information that allows efficient, direct access to SourceFile. The index file by default is stored in the same location as the source file and has the same name as the source file, but with an IDX extension. The `BioIndexedFile` constructor uses the index file to construct subsequent objects from SourceFile, which saves time.

`BioIFobj = BioIndexedFile(Format,SourceFile,IndexDir)` returns a `BioIndexedFile` object BioIFobj by specifying the relative or absolute path to a folder to use when searching for or saving the index file.

`BioIFobj = BioIndexedFile(Format,SourceFile,IndexFile)` returns a `BioIndexedFile` object BioIFobj by specifying a file name, optionally including a relative or absolute path, to use when searching for or saving the index file.

`BioIFobj = BioIndexedFile( ___ ,Name,Value)` returns a `BioIndexedFile` object BioIFobj by using any input arguments from the previous syntaxes and additional options, specified as one or more `Name,Value` pair arguments.

Input Arguments

Format

String specifying a file format. Choices are:

- `'SAM'` — SAM-formatted file
- `'FASTQ'` — FASTQ-formatted file
- `'FASTA'` — FASTA-formatted file
- `'TABLE'` — Tab-delimited table with multiple columns. Keys can be in any column. Rows with the same key are considered separate entries.
- `'MRTAB'` — Tab-delimited table with multiple columns. Keys can be in any column. Contiguous rows with the same key are considered a single entry. Noncontiguous rows with the same key are considered separate entries.
- `'FLAT'` — Flat file with concatenated entries separated by a string, typically `'//'`. Within an entry, the key is separated from the rest of the entry by a white space.

Default:

SourceFile

String specifying a text file. The string can include a relative or absolute path.

Default:

IndexDir

String specifying the relative or absolute path to a folder to use when searching for or saving the index file.

Default:

IndexFile

String specifying a file name, optionally including a relative or absolute path, to use when searching for or saving the index file.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single

quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'IndexedByKeys'

Specifies if you can access the object BioIFobj using keys. Choices are `true` or `false`.

Tip Set the value to `false` if you do not need to access entries in the object using keys. Doing so saves time and space when creating the object.

Default: `true`

'MemoryMappedIndex'

Specifies whether the constructor stores the indices in the auxiliary index file and accesses them via memory maps (`true`) or loads the indices into memory at construction time (`false`).

Tip If memory is not an issue and you want to maximize performance when accessing entries in the object, set the value to `false`.

Default: `true`

'Interpreter'

Handle to a function that the `read` method uses when parsing entries from the source file. The interpreter function must accept a single string of one or more concatenated entries and return a structure or an array of structures containing the interpreted data.

When `Format` is a general-purpose format such as `'TABLE'`, `'MRTAB'`, or `'FLAT'`, then the default is `[]`, which means the function is an anonymous function in which the output is equivalent to the input.

When `Format` is an application-specific format such as `'SAM'`, `'FASTQ'`, or `'FASTA'`, then the default is a function handle appropriate for that file type and typically does not require you to change it.

Default:

'Verbose'

Controls the display of the status of the object construction. Choices are `true` or `false`.

Default: `true`

Note: The following name-value pair arguments apply only when both of the following are true:

- There is no pre-existing index file associated with your source file.
- Your source file has a general-purpose format such as `'TABLE'`, `'MRTAB'`, or `'FLAT'`.

For source files with application-specific formats, the following name-value pairs are pre-defined and you cannot change them.

'KeyColumn'

Positive integer specifying the column in the `'TABLE'` or `'MRTAB'` file that contains the keys.

Default: 1

'KeyToken'

String that occurs in each entry before the key, for `'FLAT'` files that contain keys. If the value is `' '`, it indicates the key is the first string in each entry and is delimited by blank spaces.

Default: `' '`

'HeaderPrefix'

String specifying a prefix that denotes header lines in the source file so the constructor ignores them when creating the object. If the value is `[]`, it means the constructor does not check for header lines in the source file.

Default: `[]`

'CommentPrefix'

String specifying a prefix that denotes comment lines in the source file so the constructor ignores them when creating the object. If the value is `[]`, it means the constructor does not check for comment lines in the source file.

Default: `[]`

'ContiguousEntries'

Specifies whether entries are on contiguous lines, which means they are not separated by empty lines or comment lines, in the source file or not. Choices are `true` or `false`.

Tip Set the value to `true` when entries are not separated by empty lines or comment lines. Doing so saves time and space when creating the object.

Default: `false`

'TableDelimiter'

String specifying a delimiter symbol to use as a column separator for SourceFile when Format is `'TABLE'` or `'MRTAB'`. Choices are `'\t'` (horizontal tab), `' '` (blank space), or `','`, (comma).

Default: `'\t'`

'EntryDelimiter'

String specifying a delimiter symbol to use as an entry separator for SourceFile when Format is `'FLAT'`.

Default: `'//'`

Properties

FileFormat

File format of the source file

This information is read only. Possible values are:

- · 'SAM' — SAM-formatted file
- · 'FASTQ' — FASTQ-formatted file
- · 'FASTA' — FASTA-formatted file
- · 'TABLE' — Tab-delimited table with multiple columns. Keys can be in any column. Rows with the same key are considered separate entries.
- · 'MRTAB' — Tab-delimited table with multiple columns. Keys can be in any column. Contiguous rows with the same key are considered a single entry. Noncontiguous rows with the same key are considered separate entries.
- · 'FLAT' — Flat file with concatenated entries separated by a string, typically '//'. Within an entry, the key is separated from the rest of the entry by a white space.

IndexedByKeys

Whether or not the entries in the source file can be indexed by an alphanumeric key.

This information is read only.

IndexFile

Path and file name of the auxiliary index file.

This information is read only. Use this property to confirm the name and location of the index file associated with the object.

InputFile

Path and file name of the source file.

This information is read only. Use this property to confirm the name and location of the source file from which the object was constructed.

Interpreter

Handle to a function used by the read method to parse entries in the source file.

This interpreter function must accept a single string of one or more concatenated entries and return a structure or an array of structures containing the interpreted data. Set this property when your source file has a 'TABLE', 'MRTAB', or 'FLAT' format. When your source file is an application-specific format such as 'SAM', 'FASTQ', or 'FASTA', then the default is a function handle appropriate for that file type and typically does not require you to change it.

MemoryMappedIndex

Whether the indices to the source file are stored in a memory-mapped file or in memory.

NumEntries

Number of entries indexed by the object.

This information is read only.

Methods

getDictionary	Retrieve reference sequence names from SAM-formatted source file associated with BioIndexedFile object
getEntryByIndex	Retrieve entries from source file associated with BioIndexedFile object using numeric index
getEntryByKey	Retrieve entries from source file associated with BioIndexedFile object using alphanumeric key
getIndexByKey	Retrieve indices from source file associated with BioIndexedFile object using alphanumeric key
getKeys	Retrieve alphanumeric keys from source file associated with BioIndexedFile object
getSubset	Create object containing subset of elements from BioIndexedFile object
read	Read one or more entries from source file associated with BioIndexedFile object

Copy Semantics

Value. To learn how value classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Examples

Construct a BioIndexedFile object and access its gene ontology (GO) terms

This example shows how to construct a BioIndexedFile object and access its gene ontology (GO) terms.

Create a variable containing full absolute path of source file.

```
sourcefile = which('yeastgenes.sgd');
```

Copy the file to the current working directory.

```
copyfile(sourcefile,'yeastgenes_copy.sgd');
```

Construct a BioIndexedFile object from the source file that is a tab-delimited file, considering contiguous rows with the same key as a single entry. Indicate that keys are located in column 3 and that header lines are prefaced with '!'.

```
gene2goObj = BioIndexedFile('mrtab','yeastgenes_copy.sgd','KeyColumn',3,'HeaderPrefix',
Source File: yeastgenes_copy.sgd
   Path: C:\TEMP\Bdoc15a_200152_5488\tp41c9768c_1396_4eb0_8d08_b644281b4c56
   Size: 16069425 bytes
   Date: 19-Jan-2013 22:45:16
Creating new index file ...
Indexer found 6381 entries after parsing 90171 text lines.
Index File: yeastgenes_copy.sgd.idx
   Path: C:\TEMP\Bdoc15a_200152_5488\tp41c9768c_1396_4eb0_8d08_b644281b4c56
   Size: 89578 bytes
   Date: 23-Feb-2015 09:45:27
Mapping object to yeastgenes_copy.sgd.idx ...
Done.
```

Return the GO term from all entries that are associated with the gene YAT2. Access entries that have a key of YAT2.

```
YAT2_entries = getEntryByKey(gene2goObj,'YAT2');
```

Adjust object interpreter to return only the column containing the GO term.

```
gene2goObj.Interpreter = @(x) regexp(x,'GO:\d+','match');
```

Parse the entries with a key of YAT2 and return all GO terms from those entries.

```
GO_YAT2_entries = read(gene2goObj, 'YAT2')

GO_YAT2_entries =

  Columns 1 through 4

    'GO:0006066'    'GO:0006810'    'GO:0004092'    'GO:0005737'

  Columns 5 through 8

    'GO:0005737'    'GO:0006629'    'GO:0009437'    'GO:0004092'

  Columns 9 through 12

    'GO:0016740'    'GO:0006631'    'GO:0005737'    'GO:0005829'

  Columns 13 through 15

    'GO:0016746'    'GO:0016746'    'GO:0006066'
```

See Also
| memmapfile | fastaread | fastqread | samread | genbankread

How To
•   "Work with Large Multi-Entry Text Files"

bioma.data.ExptData class

Package: bioma.data

Contain data values from microarray experiment

Description

The ExptData class is designed to contain data values, such as gene expression values, from a microarray experiment. It stores the data values in one or more DataMatrix objects, each having the same row names (feature names) and column names (sample names). It provides a convenient way to store related experiment data in a single data structure (object). It also lets you manage and subset the data.

The ExptData class includes properties and methods that let you access, retrieve, and change data values from a microarray experiment. These properties and methods are useful to view and analyze the data.

Construction

EDobj = bioma.data.ExptData(*Data1*, *Data2*, ...) creates an ExptData object, from one or more matrices of data. Each matrix can be a logical matrix, a numeric matrix, or a DataMatrix object.

EDobj = bioma.data.ExptData(..., {*DMobj1*, *Name1*}, {*DMobj2*, *Name2*}, ...) specifies an element name for each DataMatrix object. *Name#* is a string specifying a unique name. Default names are Elmt1, Elmt2, etc.

EDobj = bioma.data.ExptData({*Data1*, *Data2*, ...}) creates an ExptData object, from a cell array of matrices of data. Each matrix can be a logical matrix, a numeric matrix, or a DataMatrix object.

EDobj = bioma.data.ExptData(..., '*PropertyName*', *PropertyValue*) constructs the object using options, specified as property name/property value pairs.

EDobj = bioma.data.ExptData(..., 'ElementNames', *ElementNamesValue*) specifies element names for the matrix inputs. *ElementNamesValue* is a cell array of strings. Default names are Elmt1, Elmt2, etc.

EDobj = bioma.data.ExptData(..., 'FeatureNames', *FeatureNamesValue*) specifies feature names (row names) for the ExptData object. .

EDobj = bioma.data.ExptData(..., 'SampleNames', *SampleNamesValue*) specifies sample names (column names) for the ExptData object.

Input Arguments

`Data#`

Matrix of experimental data values specified by any of the following:

- Logical matrix
- Numeric matrix
- DataMatrix object

All inputs must have the same dimensions. All DataMatrix objects must also have the same row names and columns names. If you provide logical or numeric matrices, `bioma.data.ExptData` converts them to DataMatrix objects with either default row and column names, or the row and column names of DataMatrix inputs, if provided.

The rows must correspond to features and the columns must correspond to samples.

Default:

`DMobj#`

Variable name of a DataMatrix object in the MATLAB Workspace.

Default:

`Name#`

String specifying an element name for the corresponding DataMatrix object

Default:

`ElementNamesValue`

Cell array of strings that specifies unique element names for the matrix inputs. The number of elements in *ElementNamesValue* must equal the number input matrices.

Default: {Elmt1, Elmt2, ...}

FeatureNamesValue

Feature names (row names) for the ExptData object, specified by one of the following:

- Cell array of strings
- Character array
- Numeric or logical vector
- Single string, which is used as a prefix for the feature names, with feature numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `bioma.data.ExptData` assigns unique feature names using the format `Feature1`, `Feature2`, etc.

If you use a cell array of strings, character array, or vector, then the number of elements must be equal in number to the number of rows in *Data1*.

Default:

SampleNamesValue

Sample names (column names) for the ExptData object, specified by one of the following:

- Cell array of strings
- Character array
- Numeric or logical vector
- Single string, which is used as a prefix for the sample names, with sample numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `bioma.data.ExptData` assigns unique sample names using the format `Sample1`, `Sample2`, etc.

If you use a cell array of strings, character array, or vector, then the number of elements must be equal in number to the number of columns in *Data1*. If the ExptData object is part of an ExpressionSet object that contains a MetaData object, the sample names (column names) in the ExptData object must match the sample names (row names) in a MetaData object.

Default:

Properties

ElementClass

Class type of the DataMatrix objects in the experiment

Cell array of strings specifying the class type of each DataMatrix object in the ExptData object. Possible values are MATLAB classes, such as `single`, `double`, and `logical`. This information is read-only.

Attributes:

SetAccess	private

Name

Name of the ExptData object.

String specifying the name of the ExptData object. Default is `[]`.

NElements

Number of elements in the experiment

Positive integer specifying the number of elements (DataMatrix objects) in the experiment data. This value is equivalent to the number of DataMatrix objects in the ExptData object. This information is read-only.

Attributes:

SetAccess	private

NFeatures

Number of features in the experiment

Positive integer specifying the number of features in the experiment. This value is equivalent to the number of rows in each DataMatrix object in the ExptData object. This information is read-only.

Attributes:

SetAccess   private

NSamples

Number of samples in the experiment

Positive integer specifying the number of samples in the experiment. This value is equivalent to the number of columns in each DataMatrix object in the ExptData object. This information is read-only.

Attributes:

SetAccess   private

Methods

combine	
	Combine two ExptData objects
dmNames	
	Retrieve or set Name properties of DataMatrix objects in ExptData object
elementData	
	Retrieve or set data element (DataMatrix object) in ExptData object
elementNames	
	Retrieve or set element names of DataMatrix objects in ExptData object
featureNames	
	Retrieve or set feature names in ExptData object
isempty	
	Determine whether ExptData object is empty
sampleNames	
	Retrieve or set sample names in ExptData object

size

Return size of ExptData object

Instance Hierarchy

An ExpressionSet object contains an ExptData object. An ExptData object contains one or more DataMatrix objects.

Attributes

To learn about attributes of classes, see Class Attributes in the MATLAB Object-Oriented Programming documentation.

Copy Semantics

Value. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Indexing

ExptData objects support 1-D parenthesis ( ) indexing to extract, assign, and delete data.

ExptData objects do not support:

- Dot . indexing
- Curly brace { } indexing

Examples

Construct an ExptData object

This example shows how to construct an ExptData object containing one DataMatrix object.

Import bioma.data package to make constructor functions available.

```
import bioma.data.*
```

Create a DataMatrix object from .txt file containing expression values from microarray experiment.

```
dmObj = DataMatrix('File', 'mouseExprsData.txt');
```

Construct an ExptData object from the DataMatrix object.

```
EDObj = ExptData(dmObj)


EDObj =

Experiment Data:
  500 features,  26 samples
  1 elements
  Element names: Elmt1
```

- Working with Objects for Microarray Experiment Data
- Analyzing Illumina Bead Summary Gene Expression Data

References

[1] Hovatta, I., Tennant, R S., Helton, R., et al. (2005). Glyoxalase 1 and glutathione reductase 1 regulate anxiety in mice. Nature *438*, 662–666.

See Also
bioma.ExpressionSet | bioma.data.MetaData | bioma.data.MIAME

How To
- Class Attributes
- Property Attributes
- "Representing Expression Data Values in ExptData Objects"

bioma.data.MetaData class

Package: bioma.data

Contain metadata from microarray experiment

Description

The MetaData class is designed to contain metadata (variable values and descriptions) from a microarray experiment. It provides a convenient way to store related metadata in a single data structure (object). It also lets you manage and subset the data.

The metadata is a collection of variable names, for example related to samples or microarray features, along with descriptions and values for the variables. A MetaData object stores the metadata in two dataset arrays.

- **Values dataset array** — A dataset array containing the measured value of each variable per sample or feature. In this dataset array, the columns correspond to variables and rows correspond to either samples or features. The number and names of the columns in this dataset array must match the number and names of the rows in the Descriptions dataset array. If this dataset array contains *sample* metadata, then the number and names of the rows (samples) must match the number and names of the columns in the DataMatrix objects in the same ExpressionSet object. If this dataset array contains *feature* metadata, then the number and names of the rows (features) must match the number and names of the rows in the DataMatrix objects in the same ExpressionSet object.

- **Descriptions dataset array** — A dataset array containing a list of the variable names and their descriptions. In this dataset array, each row corresponds to a variable. The row names are the variable names, and a column, named `VariableDescription`, contains a description of the variable. The number and names of the rows in the Descriptions dataset array must match the number and names of the columns in the Values dataset array.

The MetaData class includes properties and methods that let you access, retrieve, and change metadata variables, and their values and descriptions. These properties and methods are useful to view and analyze the metadata.

Construction

MDobj = bioma.data.MetaData(*VarValues*) creates a MetaData object from one dataset array whose rows correspond to sample (observation) names and whose columns correspond to variables. The dataset array contains the measured value of each variable per sample.

MDobj = bioma.data.MetaData(*VarValues*, *VarDescriptions*) creates a MetaData object from two dataset arrays. *VarDescriptions* is a dataset array whose rows correspond to variables. The row names are the variable names, and another column, named VariableDescription, contains a description of each variable.

MDobj = bioma.data.MetaData(*VarValues*, *VarDesc*) creates a MetaData object from a dataset array and *VarDesc* a cell array of strings containing descriptions of the variables.

MDobj = bioma.data.MetaData(..., '*PropertyName*', *PropertyValue*) constructs the object using options, specified as property name/property value pairs.

MDobj = bioma.data.MetaData('File', *FileValue*) creates a MetaData object from a text file containing a table of metadata. The table row labels must be sample names, and its column headers must be variable names.

MDobj = bioma.data.MetaData('File', *FileValue*, ...'Path', *PathValue*) specifies a folder or path and folder where *FileValue* is stored.

MDobj = bioma.data.MetaData('File', *FileValue*, ...'Delimiter', *DelimiterValue*) specifies a delimiter symbol to use as a column separator for *FileValue*. Default is '\t'.

MDobj = bioma.data.MetaData('File', *FileValue*, ...'RowNames', *RowNamesValue*) specifies the row names (sample names) for the MetaData object. Default is the information in the first column of the table.

MDobj = bioma.data.MetaData('File', *FileValue*, ...'ColumnNames', *ColumnNamesValue*) specifies the columns of data to read from the table. *ColumnNamesValue* is a cell array of strings specifying the column header names. Default is to read all columns of data from the table, assuming the first row contains column headers.

MDobj = bioma.data.MetaData('File', *FileValue*, ...'VarDescChar', *VarDescCharValue*) specifies that lines in the table prefixed by *VarDescCharValue*

to be read as descriptions and used to create the *VarDescriptions* dataset array. By default, `bioma.data.MetaData` does not read variable description information, and does not create a Descriptions dataset array. These prefixed lines must appear at the top of the file, before the table of metadata values.

MDobj = bioma.data.MetaData(...'Name', *NameValue*) specifies a name for the MetaData object.

MDobj = bioma.data.MetaData('File', *FileValue*, ...'Description', *DescriptionValue*) specifies a description for the MetaData object.

MDobj = bioma.data.MetaData('File', *FileValue*, ...'SampleNames', *SampleNamesValue*) specifies sample names (row names) for the MetaData object.

MDobj = bioma.data.MetaData('File', *FileValue*, ...'VariableNames', *VariableNamesValue*) specifies variable names (column names) for the MetaData object.

Input Arguments

VarValues

Dataset array whose rows correspond to sample (observation) names and whose columns correspond to variables. The dataset array contains the measured value of each variable per sample or feature.

The number and names of the columns in the *VarValues* dataset array must match the number and names of the rows in the *VarDescriptions* dataset array. If *VarValues* contains *sample* metadata, then the number and names of the rows (samples) must match the number and names of the columns in the DataMatrix objects in the same ExpressionSet object. If *VarValues* contains *feature* metadata, then the number and names of the rows (features) must match the number and names of the rows in the DataMatrix objects in the same ExpressionSet object.

Default:

VarDescriptions

Dataset array whose rows correspond to variables. The row names are the variable names, and a column, named `VariableDescription`, contains a description of the variable. The number and names of the rows in the *VarDescriptions* dataset array must match the number and names of the columns in the *VarValues* dataset array.

Default:

VarDesc

Cell array of strings containing descriptions of the variables. The number of elements in *VarDesc* must equal the number of columns (variable names) in *VarValues*.

Default:

FileValue

String specifying a text file containing a table of metadata. The table row labels must be sample or feature names, and its column headers must be variable names. The text file must be on the MATLAB search path or in the Current Folder (unless you use the Path property).

Default:

PathValue

String specifying a folder or path and folder where *FileValue* is stored.

Default:

DelimiterValue

String specifying a delimiter symbol to use as a column separator for *FileValue*. Typical choices are:

- `' '`
- `'\t'` (default)
- `','`
- `';'`
- `'|'`

Default:

RowNamesValue

Row names (sample or feature names) for the MetaData object, specified by one of the following:

- Cell array of strings

- Single number indicating the column of the table containing the row names
- Character string indicating the column header of the table containing the row names

If you specify [ ] for *RowNamesValue*, then `bioma.data.MetaData` provides numbered row names, starting with 1.

Default: 1, which specifies the information in the first column of the table

ColumnNamesValue

Cell array of strings specifying the column header names to indicate which columns of data to read from the table. Default is to read all columns of data from the table, assuming the first row contains column headers. If the table does not have column headers, specify [ ] for *ColumnNamesValue* to read all columns of data and provide numbered column names, starting with 1.

Default:

VarDescCharValue

String specifying a character to prefix lines in the table that are to be read as descriptions and used to create the *VarDescriptions* dataset array. By default, `bioma.data.MetaData` does not read variable description information, and does not create a *VarDescriptions* dataset array. These prefixed lines must appear at the top of the file, before the table of metadata values.

Default:

NameValue

String specifying a name for the MetaData object.

Default:

DescriptionValue

String specifying a description for the MetaData object.

Default:

SampleNamesValue

Cell array of strings specifying sample names for the MetaData object. The number of elements in the cell array must equal the number of samples in the MetaData object.

This input overwrites sample names from the input file. Default are the sample names (row names) from the input file.

Default:

VariableNamesValue

Cell array of strings specifying variable names for the MetaData object. The number of elements in the cell array must equal the number of variables in the MetaData object. This input overwrites variable names from the input file. Default are the variable names (column names) from the input file.

Default:

Properties

Description

Description of the MetaData object.

String specifying a description of the MetaData object. Default is [].

DimensionLabels

Row and column labels for the MetaData object.

Two-element cell array containing strings specifying labels of the rows and columns respectively in the MetaData object. Default is {'Samples', 'Variables'}.

Name

Name of the MetaData object.

String specifying the name of the MetaData object. Default is [].

NSamples

Number of samples (observations) in the experiment

Positive integer specifying the number of samples in the experiment. This value is equivalent to the number of rows in the *VarValues* dataset array. This information is read-only

Attributes:

SetAccess	private

NVariables

Number of variables in the experiment

Positive integer specifying the number of variables in the experiment. This value is equivalent to the number of columns in the *VarValues* dataset array. This information is read-only

Attributes:

SetAccess	private

Methods

combine	Combine two MetaData objects
isempty	Determine whether MetaData object is empty
sampleNames	Retrieve or set sample names in MetaData object
size	Return size of MetaData object
variableDesc	Retrieve or set variable descriptions for samples in MetaData object
variableNames	Retrieve or set variable names for samples in MetaData object
variableValues	Retrieve or set variable values for samples in MetaData object

varValuesTable

Create 2-D graphic table GUI of variable values in MetaData object

Instance Hierarchy

An ExpressionSet object contains two MetaData objects, one for sample information and one for microarray feature information. A MetaData object contains two dataset arrays. One dataset array contains the measured value of each variable per sample or feature. The other dataset array contains a list of the variable names and their descriptions.

Attributes

To learn about attributes of classes, see Class Attributes in the MATLAB Object-Oriented Programming documentation.

Copy Semantics

Value. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Indexing

MetaData objects support 2-D parenthesis ( ) indexing and dot . indexing to extract, assign, and delete data.

MetaData objects do not support:

- Curly brace { } indexing
- Linear indexing

Examples

Construct a MetaData object containing sample variable information from a text file:

```
% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Display information about the MetaData object
MDObj2
% Supply a description for the MetaData object
MDObj2.Description = 'This MetaData Object contains sample variable info.'
```

See Also
bioma.ExpressionSet | bioma.data.ExptData | bioma.data.MIAME

How To

- Class Attributes
- Property Attributes
- "Representing Sample and Feature Metadata in MetaData Objects"

bioma.data.MIAME class

Package: bioma.data

Contain experiment information from microarray gene expression experiment

Description

The MIAME class is designed to contain information about experimental methods and conditions from a microarray gene expression experiment. It loosely follows the Minimum Information About a Microarray Experiment (MIAME) specification. It can include information about:

- Experiment design
- Microarrays used in the experiment
- Samples used
- Sample preparation and labeling
- Hybridization procedures and parameters
- Normalization controls
- Preprocessing information
- Data processing specifications

It provides a convenient way to store related information about a microarray experiment in a single data structure (object).

The MIAME class includes properties and methods that let you access, retrieve, and change experiment information related to a microarray experiment. These properties and methods are useful to view and analyze the information.

Construction

MIAMEobj = bioma.data.MIAME() creates an empty MIAME object for storing experiment information from a microarray gene expression experiment.

MIAMEobj = bioma.data.MIAME(*GeoSeriesStruct*) creates a MIAME object from a structure containing Gene Expression Omnibus (GEO) Series data.

MIAMEobj = bioma.data.MIAME(..., '*PropertyName*', *PropertyValue*) constructs the object using options, specified as property name/property value pairs.

MIAMEobj = bioma.data.MIAME(...,'Investigator', *InvestigatorValue*) specifies the name of the experiment investigator.

MIAMEobj = bioma.data.MIAME(...,'Lab', *LabValue*) specifies the laboratory that conducted the experiment.

MIAMEobj = bioma.data.MIAME(...,'Contact', *ContactValue*) specifies the contact information for the experiment investigator or laboratory.

MIAMEobj = bioma.data.MIAME(...,'URL', *URLValue*) specifies the experiment URL.

Input Arguments

GeoSeriesStruct

Gene Expression Omnibus (GEO) Series data specified by either:

- MATLAB structure returned by the getgeodata function
- Structure.Header.Series substructure returned by the getgeodata function

Default:

InvestigatorValue

String specifying the name of the experiment investigator.

Default:

LabValue

String specifying the laboratory that conducted the experiment.

Default:

ContactValue

String specifying the contact information for the experiment investigator or laboratory

Default:

URLValue

String specifying the experiment URL.

Default:

Properties

Abstract

Abstract describing the experiment

String containing an abstract describing the experiment.

Arrays

Information about the microarray chips used in the experiment

Cell array containing information about the microarray chips used in the experiment. Information can include array name, array platform, number of features on the array, and so on.

Contact

Contact information for the experiment investigator or laboratory

Character array containing contact information for the experiment investigator or laboratory.

ExptDesign

Brief description of the experiment design

Character array containing description of the experiment design.

Hybridization

Information about the experiment hybridization

Cell array containing information about the hybridization protocol used in the experiment. Information can include hybridization time, concentration, volume, temperature, and so on.

`Investigator`

Name of the experiment investigator

Character array containing the name of the experiment investigator.

`Laboratory`

Name of the laboratory where the experiment was conducted

Character array containing the name of laboratory.

`Other`

Other information about the experiment

Cell array containing other information about the experiment, not covered by the other properties.

`Preprocessing`

Information about the experiment preprocessing steps

Cell array containing information about the preprocessing steps used on the data from the experiment.

`PubMedID`

PubMed identifiers for relevant publications.

Character array containing PubMed identifiers for papers relevant to the data set used in the experiment.

`QualityControl`

Information about the experiment quality controls

Cell array containing information about the experiment quality control steps. Information can include replicates, dye swap, and so on.

`Samples`

Information about samples used in the experiment

Cell array containing information about the samples used in the experiment. Information can include sample source, sample organism, treatment type, compound, labeling protocol, external control, and so on.

Title

Experiment title

Character array containing a single sentence experiment title.

URL

URL for the experiment

Character array containing URL for the experiment.

Methods

combine
Combine two MIAME objects

isempty
Determine whether MIAME object is empty

Instance Hierarchy

An ExpressionSet object contains a MIAME object.

Attributes

To learn about attributes of classes, see Class Attributes in the MATLAB Object-Oriented Programming documentation.

Copy Semantics

Value. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Examples

Construct a MIAME object

Create a MATLAB structure containing Gene Expression Omnibus (GEO) series data.

```
geoStruct = getgeodata('GSE4616');
```

Import bioma.data package to make the constructor function available.

```
import bioma.data.*
```

Construct MIAME object from the structure.

```
MIAMEObj1 = MIAME(geoStruct)

MIAMEObj1 =

Experiment Description:
  Author name: Mika,,Silvennoinen
Riikka,,Kivelä
Maarit,,Lehti
Anna-Maria,,Touvras
Jyrki,,Komulainen
Veikko,,Vihko
Heikki,,Kainulainen
  Laboratory: LIKES - Research Center
  Contact information: Mika,,Silvennoinen
  URL:
  PubMedIDs: 17003243
  Abstract: A 90 word abstract is available. Use the Abstract property.
  Experiment Design: A 234 word summary is available. Use the ExptDesign property.
  Other notes:
    [1x84 char]
```

Supply a URL for the MIAME object.

```
MIAMEObj1.URL = 'www.nonexistinglab.com'

MIAMEObj1 =

Experiment Description:
  Author name: Mika,,Silvennoinen
Riikka,,Kivelä
```

```
Maarit,,Lehti
Anna-Maria,,Touvras
Jyrki,,Komulainen
Veikko,,Vihko
Heikki,,Kainulainen
  Laboratory: LIKES - Research Center
  Contact information: Mika,,Silvennoinen
  URL: www.nonexistinglab.com
  PubMedIDs: 17003243
  Abstract: A 90 word abstract is available. Use the Abstract property.
  Experiment Design: A 234 word summary is available. Use the ExptDesign property.
  Other notes:
    [1x84 char]
```

Alternatively you can construct a MIAME object using customized properties.

```
MIAMEObj2 = MIAME('investigator', 'Jane Researcher',...
                  'lab', 'One Bioinformatics Laboratory',...
                  'contact', 'jresearcher@lab.not.exist',...
                  'url', 'www.lab.not.exist',...
                  'title', 'Normal vs. Diseased Experiment',...
                  'abstract', 'Example of using expression data',...
                  'other', {'Notes:Created from a text file.'})

MIAMEObj2 =

Experiment Description:
  Author name: Jane Researcher
  Laboratory: One Bioinformatics Laboratory
  Contact information: jresearcher@lab.not.exist
  URL: www.lab.not.exist
  PubMedIDs:
  Abstract: A 4 word abstract is available. Use the Abstract property.
  No experiment design summary available.
  Other notes:
    'Notes:Created from a text file.'
```

- Working with Objects for Microarray Experiment Data
- Analyzing Illumina Bead Summary Gene Expression Data

See Also

bioma.ExpressionSet | bioma.data.ExptData | bioma.data.MetaData |
getgeodata

How To

- Class Attributes
- Property Attributes
- "Representing Experiment Information in a MIAME Object"

bioma.ExpressionSet class

Package: bioma

Contain data from microarray gene expression experiment

Description

The ExpressionSet class is designed to contain data from a microarray gene expression experiment, including expression values, sample and feature metadata, and information about experimental methods and conditions. It provides a convenient way to store related information about a microarray gene expression experiment in a single data structure (object). It also lets you manage and subset the data.

The ExpressionSet class includes properties and methods that let you access, retrieve, and change data, metadata, and other information about the microarray gene expression experiment. These properties and methods are useful for viewing and analyzing the data.

Construction

ExprSetobj = bioma.ExpressionSet(*Data*) creates an ExpressionSet object, from *Data*, a numeric matrix, a DataMatrix object, or an ExptData object, which contains one or more DataMatrix objects with the same dimensions, row names and column names.

ExprSetobj = bioma.ExpressionSet(*Data*, {*DMobj1*, *Name1*}, {*DMobj2*, *Name2*}, ...) creates an ExpressionSet object, from *Data*, and additional DataMatrix objects with specified element names. All DataMatrix objects must have the same dimensions, row names, and column names.

ExprSetobj = bioma.ExpressionSet(..., '*PropertyName*', *PropertyValue*) constructs the object using options, specified as property name/property value pairs.

ExprSetobj = bioma.ExpressionSet(..., 'SData', *SDataValue*) includes a MetaData object containing sample metadata in the ExpressionSet object.

ExprSetobj = bioma.ExpressionSet(..., 'FData', *FDataValue*) includes a MetaData object containing microarray feature metadata in the ExpressionSet object.

ExprSetobj = bioma.ExpressionSet(..., 'EInfo', *EInfoValue*) includes a MIAME object, which contains experiment information, in the ExpressionSet object.

Input Arguments

`Data`

Any of the following:

- Numeric matrix
- DataMatrix object
- ExptData object, which contains one or more DataMatrix objects having the same dimensions

If you provide a DataMatrix object, `bioma.ExpressionSet` creates an ExptData object from it and names the DataMatrix object `Expressions`. If you provide an ExptData object, `bioma.ExpressionSet` renames the first DataMatrix object in the ExptData object to `Expressions`, unless another DataMatrix object in the ExptData object is already named `Expressions`.

Default:

`DMobj#`

Variable name of a DataMatrix object. Each DataMatrix object must have the same dimensions as *Data*.

Default:

`Name#`

String specifying an element name for the corresponding DataMatrix object. Each DataMatrix object in an ExpressionSet object has an element name. At least one DataMatrix object in an ExpressionSet object has an element name of `Expressions`. By default, it is the first DataMatrix object.

Default:

`SDataValue`

Variable name of a MetaData object containing sample metadata for the experiment. The variable name must exist in the MATLAB Workspace.

Default:

FDataValue

Variable name of a MetaData object containing microarray feature metadata for the experiment. The variable name must exist in the MATLAB Workspace.

Default:

EInfoValue

Variable name of a MIAME object, which contains information about the experiment methods and conditions. The variable name must exist in the MATLAB Workspace.

Default:

Properties

NElements

Number of elements in the experiment

Positive integer specifying the number of elements (DataMatrix objects) in the experiment data. This value is equivalent to the number of DataMatrix objects in the ExperimentSet object. This information is read-only.

Attributes:

SetAccess	private

NFeatures

Number of features in the experiment

Positive integer specifying the number of features in the experiment. This value is equivalent to the number of rows in each DataMatrix object in the ExperimentSet object. This information is read-only.

Attributes:

SetAccess	private

NSamples

Number of samples in the experiment

Positive integer specifying the number of samples in the experiment. This value is equivalent to the number of columns in each DataMatrix object in the ExperimentSet object. This information is read-only.

Attributes:

SetAccess	private

Methods

abstract	Retrieve or set abstract describing experiment in ExpressionSet object
elementData	Retrieve or set data element (DataMatrix object) in ExpressionSet object
elementNames	Retrieve or set element names of DataMatrix objects in ExpressionSet object
expressions	Retrieve or set Expressions DataMatrix object from ExpressionSet object
exprWrite	Write expression values in ExpressionSet object to text file
exptData	Retrieve or set experiment data in ExpressionSet object
exptInfo	Retrieve or set experiment information in ExpressionSet object
featureData	Retrieve or set feature metadata in ExpressionSet object
featureNames	Retrieve or set feature names in ExpressionSet object

featureVarDesc

Retrieve or set feature variable descriptions in ExpressionSet object

featureVarNames

Retrieve or set feature variable names in ExpressionSet object

featureVarValues

Retrieve or set feature variable data values in ExpressionSet object

pubMedID

Retrieve or set PubMed IDs in ExpressionSet object

sampleData

Retrieve or set sample metadata in ExpressionSet object

sampleNames

Retrieve or set sample names in ExpressionSet object

sampleVarDesc

Retrieve or set sample variable descriptions in ExpressionSet object

sampleVarNames

Retrieve or set sample variable names in ExpressionSet object

sampleVarValues

Retrieve or set sample variable values in ExpressionSet object

size

Return size of ExpressionSet object

Instance Hierarchy

An ExpressionSet object contains an ExptData object, two MetaData objects, and a MIAME object. These objects can be empty.

Attributes

To learn about attributes of classes, see Class Attributes in the MATLAB Object-Oriented Programming documentation.

Copy Semantics

Value. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Indexing

ExpressionSet objects support 2-D parenthesis ( ) indexing to extract, assign, and delete data.

ExpressionSet objects do not support:

- Dot . indexing
- Curly brace { } indexing
- Linear indexing

Examples

Construct an ExpressionSet Object

This example shows how to construct an ExpressionSet object. The mouseExprsData.txt file used in this example contains data from Hovatta et al., 2005.

Import bioma.data package to make the constructor function available.

```
import bioma.data.*
```

Create a DataMatrix object from .txt file containing expression values from microarray experiment.

```
dmObj = DataMatrix('File', 'mouseExprsData.txt');
```

Construct an ExptData object.

```
EDObj = ExptData(dmObj)

EDObj =

Experiment Data:
  500 features,  26 samples
  1 elements
  Element names: Elmt1
```

Construct a MetaData object from .txt file.

```
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#')

MDObj2 =

Sample Names:
    A, B, ...,Z (26 total)
Variable Names and Meta Information:
               VariableDescription
    Gender     ' Gender of the mouse in study'
    Age        ' The number of weeks since mouse birth'
    Type       ' Genetic characters'
    Strain     ' The mouse strain'
    Source     ' The tissue source for RNA collection'
```

Create a MATLAB structure containing GEO Series data.

```
geoStruct = getgeodata('GSE4616');
```

Construct a MIAME object.

```
MIAMEObj = MIAME(geoStruct)

MIAMEObj =

Experiment Description:
  Author name: Mika,,Silvennoinen
Riikka,,Kivelä
Maarit,,Lehti
Anna-Maria,,Touvras
Jyrki,,Komulainen
Veikko,,Vihko
Heikki,,Kainulainen
  Laboratory: LIKES - Research Center
  Contact information: Mika,,Silvennoinen
```

```
URL:
PubMedIDs: 17003243
Abstract: A 90 word abstract is available. Use the Abstract property.
Experiment Design: A 234 word summary is available. Use the ExptDesign property.
Other notes:
  [1x84 char]
```

Import bioma package to make constructor function available.

```
import bioma.*
```

Construct an ExpressionSet object.

```
ESObj = ExpressionSet(EDObj, 'SData', MDObj2, 'EInfo', MIAMEObj)

ESObj =

ExpressionSet
Experiment Data: 500 features, 26 samples
  Element names: Expressions
Sample Data:
    Sample names:      A, B, ...,Z (26 total)
    Sample variable names and meta information:
        Gender:  Gender of the mouse in study
        Age:  The number of weeks since mouse birth
        Type:  Genetic characters
        Strain:  The mouse strain
        Source:  The tissue source for RNA collection
Feature Data: none
Experiment Information: use 'exptInfo(obj)'
```

- Working with Objects for Microarray Experiment Data
- Analyzing Illumina Bead Summary Gene Expression Data

References

[1] Hovatta, I., Tennant, R S., Helton, R., et al. (2005). Glyoxalase 1 and glutathione reductase 1 regulate anxiety in mice. Nature *438*, 662–666.

See Also

bioma.data.ExptData | bioma.data.MetaData | bioma.data.MIAME

How To

- Class Attributes
- Property Attributes
- "Managing Gene Expression Data in Objects"

BioRead class

Contain sequence and quality data

Description

The `BioRead` class contains data from short-read sequences, including sequence headers, nucleotide sequences, and the quality scores for the sequences. This data is typically obtained from a high-throughput sequencing instrument.

You construct a `BioRead` object from short-read sequence data. Each element in the object has a sequence, header, and quality score associated with it. Use the object properties and methods to explore, access, filter, and manipulate all or a subset of the data, before doing subsequent analyses or sequence alignment and mapping.

Construction

`BioReadobj = BioRead` constructs BioReadobj, an empty `BioRead` object.

`BioReadobj = BioRead(File)` constructs BioReadobj, a `BioRead` object, from File, a FASTQ- or SAM-formatted file. The data remains in the source file, and the `BioRead` object accesses it using an auxiliary index file. The index file must have the same name as the source file, but with an .IDX extension. If the index file is not present in the same folder as the source file, the BioRead constructor function creates the index file in that folder.

Note: Because the data remains in the source file:

- Do not delete the source file (FASTQ or SAM) or the auxiliary index file.

- You cannot modify *BioReadobj* properties.

`BioReadobj = BioRead(Struct)` constructs BioReadobj, a `BioRead` object, from Struct, a MATLAB structure containing `Header`, `Sequence`, and `Quality` fields, such as returned by the `fastqread` or the `samread` function. The data from *Struct* is kept in memory, which lets you modify the properties of *BioReadobj*.

`BioReadobj = BioRead(Seqs)` constructs BioReadobj, a `BioRead` object, from Seqs, a cell array of strings containing the letter representations of nucleotide sequences.

`BioReadobj = BioRead(Seqs,Quals)` constructs BioReadobj, a `BioRead` object, also from Quals, a cell array of strings containing the ASCII representation of per-base quality scores for nucleotide sequences.

`BioReadobj = BioRead(Seqs,Quals,Headers)` constructs BioReadobj, a `BioRead` object, also from Headers, a cell array of strings containing header text for nucleotide sequences.

`BioReadobj = BioRead( ___ ,'PropertyName',PropertyValue)` constructs a `BioRead` object using options, specified as name-value pair arguments.

`BioReadobj = BioRead(File,'InMemory',InMemoryValue)` specifies whether to place the data in memory or leave the data in the source file. Leaving the data in the source file and accessing it via an index file is more memory efficient, but does not let you modify properties of BioReadobj. Choices are `true` or `false` (default). If the first input argument is not a file name, then this name-value pair argument is ignored, and the data is automatically placed in memory.

Tip Set the `InMemory` name-value pair argument to `true` if you want to modify the properties of BioReadobj.

`BioReadobj = BioRead( ___ ,'IndexDir',IndexDirValue)` specifies the path to the folder where the index file either exists or will be created.

Tip Use the `IndexDir` name-value pair argument if you do not have write access to the folder where the source file is located.

`BioReadobj = BioRead( ___ ,'Sequence',SequenceValue)` constructs BioReadobj, a `BioRead` object, from SequenceValue, a cell array of strings containing the letter representations of nucleotide sequences. This name-value pair works only if the data is read into memory.

`BioReadobj = BioRead( ___ ,'Quality',QualityValue)` constructs BioReadobj, a `BioRead` object, from QualityValue, a cell array of strings containing the ASCII

representation of per-base quality scores for nucleotide sequences. This name-value pair works only if the data is read into memory.

`BioReadobj = BioRead( ___ ,'Header',HeaderValue)` constructs BioReadobj, a `BioRead` object, from HeaderValue, a cell array of strings containing header text for nucleotide sequences. This name-value pair works only if the data is read into memory.

`BioReadobj = BioRead( ___ ,'Name',NameValue)` constructs BioReadobj, a `BioRead` object, and then sets the `Name` property to NameValue, a string describing the object. Default is `' '`, an empty string.

Input Arguments

`File`

String specifying a FASTQ- or SAM-formatted file.

Default:

`Struct`

MATLAB structure containing `Header`, `Sequence`, and `Quality` fields, such as returned by the `fastqread` or the `samread` function.

Default:

`InMemoryValue`

Logical specifying whether to place the data in memory or leave the data in the source file. Leaving the data in the source file and accessing it via an index file is more memory efficient, but does not let you modify properties of the `BioRead` object. If the first input argument is not a file name, then this name-value pair argument is ignored, and the data is automatically placed in memory.

Default: `false`

`IndexDirValue`

String specifying the path to the folder where the index file either exists or will be created.

Default: Folder where *File* is located

Seqs

Cell array of strings containing the letter representations of nucleotide sequences. This information populates the `BioRead` object's `Sequence` property.

Default:

Quals

Cell array of strings containing the ASCII representation of per-base quality scores for nucleotide sequences. This information populates the `BioRead` object's `Quality` property.

Default:

Headers

Cell array of strings containing header text for nucleotide sequences. This information populates the `BioRead` object's `Header` property.

Default:

SequenceValue

Cell array of strings containing the letter representations of nucleotide sequences. This information populates the `BioRead` object's `Sequence` property. This name-value pair works only if the data is read into memory.

Default:

QualityValue

Cell array of strings containing the ASCII representation of per-base quality scores for nucleotide sequences. This information populates the `BioRead` object's `Quality` property. This name-value pair works only if the data is read into memory.

Default: Empty cell array

HeaderValue

Cell array of strings containing header text for nucleotide sequences. This information populates the `BioRead` object's `Header` property. This name-value pair works only if the data is read into memory.

Default: Empty cell array

`NameValue`

String describing the `BioRead` object. This information populates the object's `Name` property.

Default: ' ', an empty string

Properties

`Header`

Headers associated with all sequences represented in the `BioRead` object.

Cell array of strings, such that there is a header for each sequence in the object. Header strings can be empty. There is a one-to-one relationship between the number and order of elements in `Header` and `Sequence`, unless `Header` is an empty cell array.

`Name`

Description of the `BioRead` object.

Single string describing the `BioRead` object.

Default: ' ', an empty string

`NSeqs`

Number of sequences in the `BioRead` object.

This information is read only.

`Quality`

Per-base quality scores associated with all sequences represented in the `BioRead` object.

Cell array of strings, such that there is a quality string for each sequence in the object. Each quality string is an ASCII representation of per-base quality scores for a nucleotide sequence or an empty string. A one-to-one relationship exists between the number and order of elements in `Quality` and `Sequence`, unless `Quality` is an empty cell array.

`Sequence`

Nucleotide sequences in the `BioRead` object.

Cell array of strings containing the letter representations of the nucleotide sequences.

Methods

combine	Combine two objects
get	Retrieve property of object
getHeader	Retrieve sequence headers from object
getQuality	Retrieve sequence quality scores from object
getSequence	Retrieve sequences from object
getSubsequence	Retrieve partial sequences from object
getSubset	Create object containing subset of elements from object
plotSummary	Plot summary statistics of BioRead object
set	Set property of object
setHeader	Set sequence headers for object
setQuality	Set sequence quality scores for object
setSequence	Set sequences for object
setSubsequence	Set partial sequences for object
setSubset	Set elements for object

write

Write contents of BioRead or BioMap object to file

Copy Semantics

Value. To learn how value classes affect copy operations, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Indexing

`BioRead` objects support dot . indexing to extract, assign, and delete data.

Examples

Construct BioRead Object from FASTQ File

Construct a `BioRead` object from a FASTQ-formatted file that is provided with Bioinformatics Toolbox.

```
BRObj1 = BioRead('SRR005164_1_50.fastq','Name','MyObject')

BRObj1 =

  BioRead with properties:

     Quality: [50x1 File indexed property]
    Sequence: [50x1 File indexed property]
      Header: [50x1 File indexed property]
       NSeqs: 50
        Name: 'MyObject'
```

Construct BioRead Object from MATLAB Workspace Variables

Create variables containing sequences, quality scores, and headers.

```
seqs = {randseq(10);randseq(15);randseq(20)};
quals = {repmat('!', 1, 10);repmat('%', 1, 15);repmat('&', 1, 20)};
```

```
headers = {'H1';'H2';'H3'};
```

Construct a BioRead object from these three variables.

```
BRObj2 = BioRead(seqs,quals,headers)

BRObj2 =

  BioRead with properties:

     Quality: {3x1 cell}
    Sequence: {3x1 cell}
      Header: {3x1 cell}
       NSeqs: 3
        Name: ''
```

Construct BioRead Object from MATLAB Structure

Create variables containing sequences, quality scores, and headers.

```
seqs = {randseq(10);randseq(15);randseq(20)};
quals = {repmat('!',1,10); repmat('%',1,15);repmat('&',1,20)};
headers = {'H1';'H2';'H3'};
```

Construct a structure containing Header, Sequence, and Quality fields.

```
BRStruct = struct('Header',headers,'Sequence',seqs,'Quality',quals);
```

Construct a BioRead object from this structure.

```
BRObj3 = BioRead(BRStruct)

BRObj3 =

  BioRead with properties:

     Quality: {3x1 cell}
    Sequence: {3x1 cell}
      Header: {3x1 cell}
       NSeqs: 3
        Name: ''
```

See Also
BioIndexedFile | fastqinfo | BioMap | fastqread | saminfo | samread

How To

- "Manage Short-Read Sequence Data in Objects"

Related Links

- Sequence Read Archive

BioReadQualityStatistics class

Quality statistics from a short-read sequence file

Description

The `BioReadQualityStatistics` class contains quality statistics data from short-read sequences and provides a standard set of quality control plots for such data.

Construct a `BioReadQualityStatistics` object from short-read sequence data stored in FASTQ, SAM, or BAM files. Perform data quality analyses using the object's methods to generate several quality control plots regarding average quality score for each base position, average quality score distribution, read count percentage for each base position, percentage of G and C nucleotides for each base position, G and C content distribution, and all nucleotide distribution. The object lets parse a sequence file without creating a `BioRead` object and interact with the quality data in order to compare different data sets or filtering options and create customized plots.

Construction

`QSObj = BioReadQualityStatistics(File)` constructs `QSObj`, a `BioReadQualityStatistics` object, from the data stored in `File`, a FASTQ-, SAM-, or BAM-formatted file.

`QSObj = BioReadQualityStatistics(Obj)` constructs `QSObj`, a `BioReadQualityStatistics` object, from the data stored in `Obj`, a `BioRead` or `BioMap` object.

`QSObj = BioReadQualityStatistics( ___ ,Name,Value)` constructs a `BioReadQualityStatistics` object using options specified by one or more name-value pair arguments.

Note: Once created, you cannot modify the properties of *QSObj* since it is an immutable object.

Input Arguments

`File`

String specifying a FASTQ file. The string can contain the path or folder location of the file.

`Obj`

A `BioRead` or `BioMap` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`'Encoding'` — Encoding format
`'Illumina18'` (default) | `'Sanger'` | `'Illumina13'` | `'Illumina15'` | `'Solexa'`

Encoding format, specified as `'Sanger'`, `'Illumina13'`, `'Illumina15'`, `'Illumina18'`, or `'Solexa'`. It is the format that is used for characters encoding sequence information and quality scores in a FASTQ file.

Example: `'Encoding','Sanger'`

`'FilterLength'` — Number of characters
`[]` (default) | positive integer

Number of characters, specified as a positive integer, from each read to be used. No filtering is applied if you use an empty array, which is the default value.

Example: `'FilterLength',40`

`'QualityScoreThreshold'` — Average quality threshold
`-Inf` (default) | real number

Average quality threshold, specified as a real number. Any read with an average score of less than the specified threshold is ignored.

Example: `'QualityScoreThreshold', 10`

Properties

FileName

Name of a file used to create `BioReadQualityStatistics` object.

FileType

Type of file from which a `BioReadQualityStatistics` object is created. Supported file types are FASTQ, SAM, and BAM formats.

Encoding

String specifying the format of the character encoding sequence information and quality scores in the file. Supported formats are: `'Sanger'`, `'Illumina13'`, `'Illumina15'`, `'Illumina18'`, and `'Solexa'`. The default format is `'Illumina18'`.

CharOffset

Integer specifying ASCII code where the quality score begins for a sequence.

NumberOfReads

Integer representing the number of short-read sequences `BioReadQualityStatistics` object contains.

MaxReadLength

Integer representing maximum length of a short-read sequence among all sequences of `BioReadQualityStatistics` object.

MinEncodingPhred

Integer specifying minimum Phred quality score [1] among all short-read sequences of a `BioReadQualityStatistics` object.

MaxEncodingPhred

Integer specifying maximum Phred quality score among all short-read sequences of a `BioReadQualityStatistics` object.

SkipPhred

Integer specifying the number of Phred scores that are not considered in the quality score range.

PerSeqAverageQualityDist

Vector of integers representing average quality distribution per sequence.

PerPosQualities

s-by-p matrix of integers that represent quality scores (s) per base positions (p).

PerSeqGCDist

Vector of integers representing the distribution of G and C nucleotides per sequence.

PerPosBaseDist

n-by-p matrix of integers that represents distribution of all nucleotides ($n = 5$) per base position (p).

Name

String describing the user-defined name for the object.

MaxScore

Integer representing maximum sequence quality score among all scores.

MinScore

Integer representing minimum sequence quality score among all scores.

FilterLength

Positive integer specifying the length of each read used in quality analysis.

QualityScoreThreshold

Scalar value specifying minimum average quality threshold for a read. Any read with an average score of less than the specified threshold is ignored. The default value is −Inf, which causes all reads to be considered.

Subset

Vector of integers specifying the index for subset of information from the original sequence data used in analysis.

Methods

plotPerPositionCountByQuality	
	Plot fractions of reads with Phred scores in ranges
plotPerPositionGC	
	Plot percentages of G or C nucleotides at each base position
plotPerPositionQuality	
	Plot Phred score distributions
plotPerSequenceGC	
	Plot G or C nucleotide distribution
plotPerSequenceQuality	
	Plot distribution of average quality scores
plotSummary	
	Plot summary statistics of a BioReadQualityStatistics object
plotTotalGC	
	Plot distribution of all nucleotides of short-read sequences

Examples

Create a BioReadQualityStatistics object and plot its summary statistics

This example shows how to create a BioReadQualityStatistics object and plot summary statistics of it.

Create a BioReadQualityStatistics object from a FASTQ file using only the first 40 characters of each read with a minimum average quality score of 5.

```
QSObj = BioReadQualityStatistics('SRR005164_1_50.fastq', 'FilterLength',...
                                 40, 'QualityScoreThreshold', 5)


QSObj =

  BioReadQualityStatistics with properties:

                     FileName: 'B:\matlab\toolbox\bioinfo\bioinfodata\SRR00...'
                     FileType: 'FASTQ'
                     Encoding: 'Illumina18'
                   CharOffset: 33
                NumberOfReads: 50
                MaxReadLength: 40
             MinEncodingPhred: 0
             MaxEncodingPhred: 62
                    SkipPhred: []
    PerSeqAverageQualityDist: [1x62 double]
             PerPosQualities: [63x40 double]
                  PerSeqGCDist: [0 0 0 0 3 3 8 5 9 7 6 5 2 2 0 0 0 0 0 0]
               PerPosBaseDist: [5x40 double]
                         Name: ''
                     MaxScore: 34
                     MinScore: 1
                 FilterLength: 40
        QualityScoreThreshold: 5
                       Subset: NaN
```

Plot the summary statistics of the object.

```
plotSummary(QSObj)


ans =

    0.0084
    1.0084
    2.0084
    3.0084
    4.0084
    5.0084
```

Quality Scores

Quality Stratification

GC Content

Quality Distribution   **GC Distribution**   **Nucleotide Distribution**

[1] Wikipedia. (2012). Phred quality score; http://en.wikipedia.org/wiki/Phred_quality_score

References
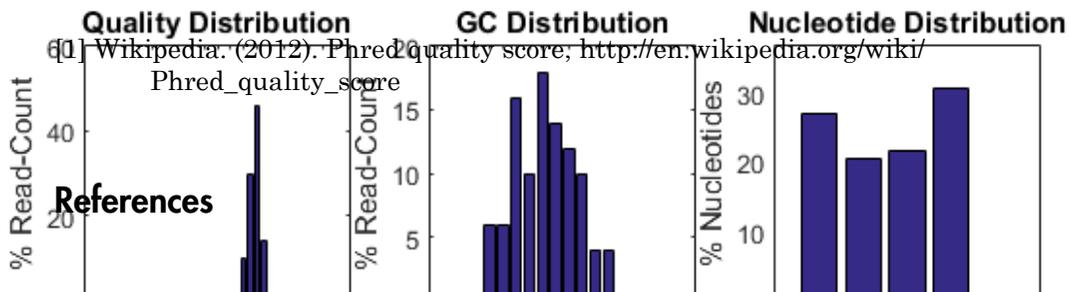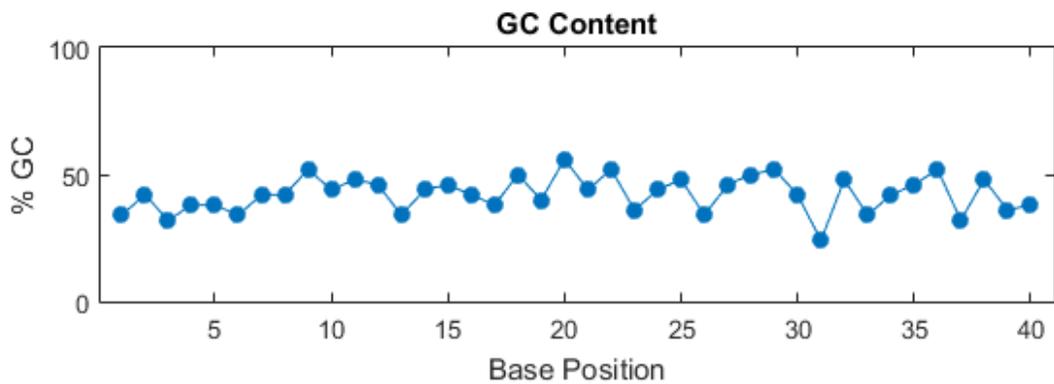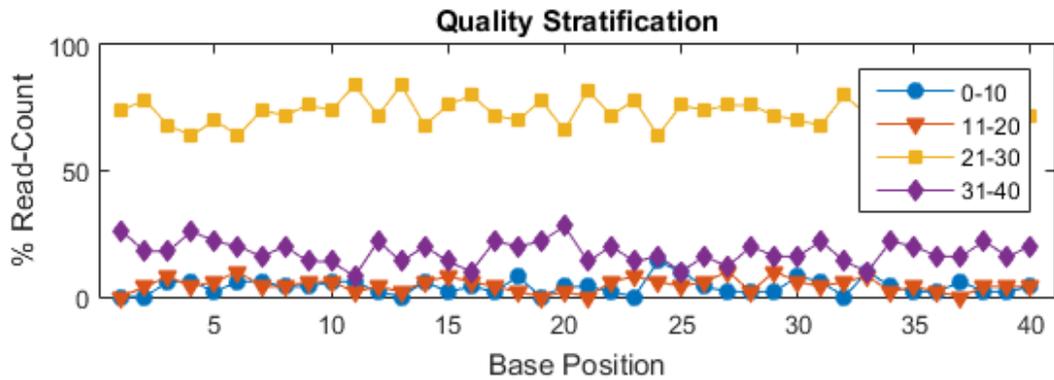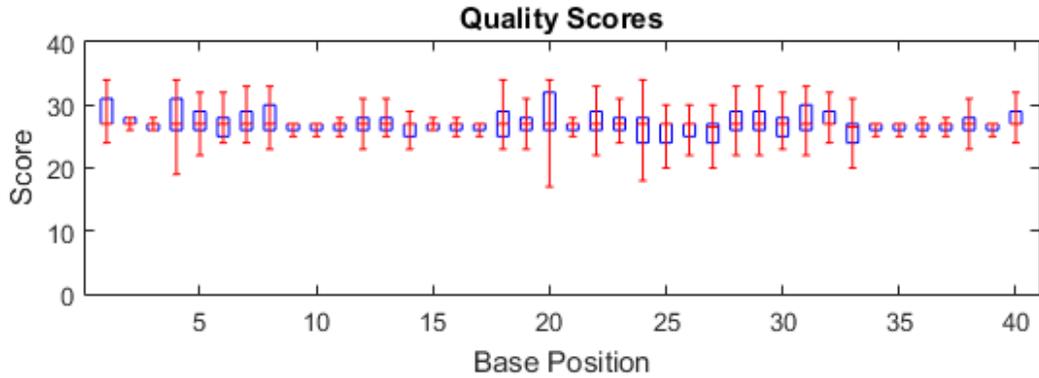
See Also
`BioMap` | `BioRead`

blastformat

Create local BLAST database

Syntax

```
blastformat('Inputdb', InputdbValue)
blastformat(..., 'FormatPath', FormatPathValue, ...)
blastformat(..., 'Title', TitleValue, ...)
blastformat(..., 'Log', LogValue, ...)
blastformat(..., 'Protein', ProteinValue, ...)
blastformat(..., 'FormatArgs', FormatArgsValue, ...)
```

Arguments

InputdbValue	String specifying a file name or path and file name of a FASTA file containing a set of sequences to be formatted as a blastable database. If you specify only a file name, that file must be on the MATLAB search path or in the current folder. (This corresponds to the formatdb option -i.)
FormatPathValue	String specifying the full path to the formatdb executable file, including the name and extension of the executable file. Default is the system path.
TitleValue	String specifying the title for the local database. Default is the input FASTA file name. (This corresponds to the formatdb option -t.)
LogValue	String specifying the file name or path and file name for the log file associated with the local database. Default is formatdb.log. (This corresponds to the formatdb option -l.)
ProteinValue	Specifies whether the sequences formatted as a local BLAST database are protein or not. Choices are true (default) or false. (This corresponds to the formatdb option -p.)
FormatArgsValue	NCBI formatdb command string, that is, a string containing one or more instances of -x and the option associated with it,

	used to specify input arguments. For an example, see Using blastformat with formatdb Syntax and Input Arguments.

Description

Note: To use the `blastformat` function, you must have a local copy of the NCBI `formatdb` executable file available from your system. You can download the `formatdb` executable file by accessing BLAST+ executables, then clicking the **download** link under the **blast** column for your platform. Run the downloaded executable and configure it for your system. .

For more information, see the readme file on the NCBI ftp site.

For convenience, consider placing the NCBI `formatdb` executable file on your system path.

`blastformat('Inputdb', ` *InputdbValue*`)` calls a local version of the NCBI `formatdb` executable file with *InputdbValue*, a file name or path and file name of a FASTA file containing a set of sequences. If you specify only a file name, that file must be on the MATLAB search path or in the current folder. (This corresponds to the `formatdb` option `-i`.)

It then formats the sequences as a local, blastable database, by creating multiple files, each with the same name as the *InputdbValue* FASTA file, but with different extensions. The database files are placed in the same location as the FASTA file.

Note: If you rename the database files, make sure they all have the same name.

`blastformat(..., '`*PropertyName*`', `*PropertyValue*`, ...)` calls `blastformat` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows.

blastformat(..., 'FormatPath', *FormatPathValue*, ...) specifies the full path to the formatdb executable file, including the name and extension of the executable file. Default is the system path.

blastformat(..., 'Title', *TitleValue*, ...) specifies the title for the local database. Default is the input FASTA file name. (This corresponds to the formatdb option -t.)

Note: The 'Title' property does not change the file name of the database files. This title is used internally only, and appears in the report structure returned by the blastlocal function.

---

blastformat(..., 'Log', *LogValue*, ...) specifies the file name or path and file name for the log file associated with the local database. Default is formatdb.log. The log file captures the progress of the database creation and formatting. (This corresponds to the formatdb option -l.)

blastformat(..., 'Protein', *ProteinValue*, ...) specifies whether the sequences formatted as a local BLAST database are protein or not. Choices are true (default) or false. (This corresponds to the formatdb option -p.)

blastformat(..., 'FormatArgs', *FormatArgsValue*, ...) specifies options using the input arguments for the NCBI formatdb function. *FormatArgsValue* is a string containing one or more instances of -*x* and the option associated with it. For example, to specify that the input is a database in ASN.1 format, instead of a FASTA file, you would use the following syntax:

blastformat('Inputdb', 'ecoli.asn', 'FormatArgs', '-a T')

Tip Use the 'FormatArgs' property to specify formatdb options for which there are no corresponding property name/property value pairs.

Note: For a complete list of valid input arguments for the NCBI formatdb function, make sure that the formatdb executable file is located on your system path or current folder, then type the following at your system's command prompt.

formatdb -

Using formatdb Syntax

You can also use the syntax and input arguments accepted by the NCBI `formatdb` function, instead of the property name/property value pairs listed previously. To do so, supply a single string containing multiple options using the *-x option* syntax. For example, you can specify the `ecoli.nt` FASTA file, a title of `myecoli`, and that the sequences are not protein by using

```
blastformat('-i ecoli.nt -t myecoli -p F')
```

Note: For a complete list of valid input arguments for the NCBI `formatdb` function, make sure that the `formatdb` executable file is located on your system path or current folder, then type the following at your system's command prompt.

```
formatdb -
```

Examples

Using blastformat with Property Name/Value Pairs

The following example assumes you have a FASTA nucleotide file, such as the *E. coli* file `NC_004431.fna`, which you can download from ftp://ftp.ncbi.nih.gov/genomes/Bacteria/, saved to your MATLAB current folder.

Create a local blastable database from the `NC_004431.fna` FASTA file and give it a title using the `'title'` property.

```
blastformat('inputdb', 'NC_004431.fna', 'protein', 'false',...
            'title', 'myecoli_nt');
```

Using blastformat with formatdb Syntax and Input Arguments

The following example assumes you have a FASTA amino acid file, such as the *E. coli* file `NC_004431.faa`, which you can download from ftp://ftp.ncbi.nih.gov/genomes/Bacteria/, saved to your MATLAB current folder.

Create a local blastable database from the `NC_004431.faa` FASTA file and rename the title and log file using `formatdb` syntax and input arguments.

```
blastformat('inputdb', 'NC_004431.faa',...
```

```
'formatargs', '-t myecoli_aa -l ecoli_aa.log');
```

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. (1990). Basic local alignment search tool. J. Mol. Biol. *215*, 403–410.

[2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. *25*, 3389–3402.

For more information on the NCBI `formatdb` function, see:

`http://blast.ncbi.nlm.nih.gov/docs/formatdb.html`

See Also
blastlocal | blastncbi | blastread | blastreadlocal | getblast

blastlocal

Perform search on local BLAST database to create BLAST report

Syntax

```
blastlocal('InputQuery', InputQueryValue)
Data = blastlocal('InputQuery', InputQueryValue)
... blastlocal(..., 'Program', ProgramValue, ...)
... blastlocal(..., 'Database', DatabaseValue, ...)
... blastlocal(..., 'BlastPath', BlastPathValue, ...)
... blastlocal(..., 'Expect', ExpectValue, ...)
... blastlocal(..., 'Format', FormatValue, ...)
... blastlocal(..., 'ToFile', ToFileValue, ...)
... blastlocal(..., 'Filter', FilterValue, ...)
... blastlocal(..., 'GapOpen', GapOpenValue, ...)
... blastlocal(..., 'GapExtend', GapExtendValue, ...)
... blastlocal(..., 'BLASTArgs', BLASTArgsValue, ...)
```

Input Arguments

InputQueryValue	String specifying the file name or path and file name of a FASTA file containing query nucleotide or amino acid sequence(s). (This corresponds to the `blastall` option `-i`.)
ProgramValue	String specifying a BLAST program. Choices are: • `'blastp'` (default) — Search protein query versus protein database. • `'blastn'` — Search nucleotide query versus nucleotide database. • `'blastx'` — Search translated nucleotide query versus protein database. • `'tblastn'` — Search protein query versus translated nucleotide database. • `'tblastx'` — Search translated nucleotide query versus translated nucleotide database.

	(The *ProgramValue* argument corresponds to the blastall option -p.)
DatabaseValue	String specifying a file name or path and file name of a local BLAST database (formatted using the NCBI formatdb function) to search. Default is a local version of the nr database in the MATLAB current folder. (This corresponds to the blastall option -d.)
BlastPathValue	String specifying the full path to the blastall executable file, including the name and extension of the executable file. Default is the system path.
ExpectValue	Value specifying the statistical significance threshold for matches against database sequences. Choices are any real number. Default is 10. (This corresponds to the blastall option -e.)
FormatValue	Integer specifying the alignment format of the BLAST search results. Choices are: • 0 (default) — Pairwise • 1 — Query-anchored, showing identities • 2 — Query-anchored, no identities • 3 — Flat query-anchored, showing identities • 4 — Flat query-anchored, no identities • 5 — Query-anchored, no identities and blunt ends • 6 — Flat query-anchored, no identities and blunt ends • 8 — Tabular • 9 — Tabular with comment lines (This corresponds to the blastall option -m.)
ToFileValue	String specifying a file name or path and file name in which to save the contents of the BLAST report. (This corresponds to the blastall option -o.)
FilterValue	Controls the application of a filter (DUST filter for the blastn program or SEG filter for other programs) to the query sequence(s). Choices are true (default) or false. (This corresponds to the blastall option -F.)

GapOpenValue	Integer that specifies the penalty for opening a gap in the alignment of sequences. Default is -1. (This corresponds to the `blastall` option -G.)
GapExtendValue	Integer that specifies the penalty for extending a gap in the alignment of sequences. Default is -1. (This corresponds to the `blastall` option -E.)
BLASTArgsValue	NCBI `blastall` command string, that is a string containing one or more instances of -*x* and the option associated with it, used to specify input arguments. For an example, see step 2 in "Examples" on page 1-317.

Output Arguments

Data	MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report.

Description

This function assumes that

The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of protein and nucleotide sequences against known sequences in online or local databases.

Note: To use the `blastlocal` function, you must have a local copy of the NCBI `blastall` executable file (version 2.2.17) available from your system. You can download the `blastall` executable file by accessing BLAST+ executables, then clicking the **download** link under the **blast** column for your platform. Run the downloaded executable and configure it for your system.

For more information, see the readme file on the NCBI ftp site.

For convenience, consider placing the NCBI `blastall` executable file on your system path.

---

blastlocal('InputQuery', *InputQueryValue*) submits query sequence(s) specified by *InputQueryValue*, a FASTA file containing nucleotide or amino acid sequence(s), for a BLAST search of a local BLAST database, by calling a local version of the NCBI blastall executable file. The BLAST search results are displayed in the MATLAB Command Window. (This corresponds to the blastall option -i.)

Data = blastlocal('InputQuery', *InputQueryValue*) returns the BLAST search results in *Data*, a MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report.

Data contains a subset of the following fields, based on the specified alignment format.

Field	Description
Algorithm	NCBI algorithm used to do a BLAST search.
Query	Identifier of the query sequence submitted to a BLAST search.
Length	Length of the query sequence.
Database	All databases searched.
Hits.Name	Name of a database sequence (subject sequence) that matched the query sequence.
Hits.Score	Alignment score between the query sequence and the subject sequence.
Hits.Expect	Expectation value for the alignment between the query sequence and the subject sequence.
Hits.Length	Length of a subject sequence.
Hits.HSPs.Score	Pairwise alignment score for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Expect	Expectation value for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Identities	Identities (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Positives	Identical or similar residues (match, possible, and percent) for a high-scoring sequence pair

Field	Description
	between the query sequence and a subject amino acid sequence. **Note:** This field applies only to translated nucleotide or amino acid query sequences and/or databases.
`Hits.HSPs.Gaps`	Nonaligned residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.Mismatches`	Residues that are not similar to each other (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.Frame`	Reading frame of the translated nucleotide sequence for a high-scoring sequence pair between the query sequence and a subject sequence. **Note:** This field applies only when performing translated searches, that is, when using `tblastx`, `tblastn`, and `blastx`.
`Hits.HSPs.Strand`	Sense (`Plus` = 5' to 3' and `Minus` = 3' to 5') of the DNA strands for a high-scoring sequence pair between the query sequence and a subject sequence. **Note:** This field applies only when using a nucleotide query sequence and database.
`Hits.HSPs.Alignment`	Three-row matrix showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.QueryIndices`	Indices of the query sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.

Field	Description
`Hits.HSPs.SubjectIndices`	Indices of the subject sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.AlignmentLength`	Length of the pairwise alignment for a high-scoring sequence pair between the query sequence and a subject sequence.
`Alignment`	Entire alignment for the query sequence and the subject sequence(s).
`Statistics`	Summary of statistical details about the performed search, such as lambda values, gap penalties, number of sequences searched, and number of hits.

`... blastlocal(..., '`*PropertyName*`', `*PropertyValue*`, ...)` calls `blastlocal` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows.

`... blastlocal(..., 'Program', `*ProgramValue*`, ...)` specifies the BLAST program. Choices are `'blastp'` (default), `'blastn'`, `'blastx'`, `'tblastn'`, and `'tblastx'`. (This corresponds to the `blastall` option `-p`.) For help in selecting an appropriate BLAST program, visit:

http://blast.ncbi.nlm.nih.gov/producttable.shtml

`... blastlocal(..., 'Database', `*DatabaseValue*`, ...)` specifies the local BLAST database (formatted using the NCBI `formatdb` function) to search. Default is a local version of the `nr` database in the MATLAB current folder. (This corresponds to the `blastall` option `-d`.)

`... blastlocal(..., 'BlastPath', `*BlastPathValue*`, ...)` specifies the full path to the `blastall` executable file, including the name and extension of the executable file. Default is the system path.

`... blastlocal(..., 'Expect', `*ExpectValue*`, ...)` specifies a statistical significance threshold for matches against database sequences. Choices are any real number. Default is `10`. (This corresponds to the `blastall` option `-e`.) You can learn more about the statistics of local sequence comparison at:

http://blast.ncbi.nlm.nih.gov/tutorial/Altschul-1.html#head2

... `blastlocal(..., 'Format', `*`FormatValue`*`, ...)` specifies the alignment format of the BLAST search results. Choices are:

- `0` (default) — Pairwise
- `1` — Query-anchored, showing identities
- `2` — Query-anchored, no identities
- `3` — Flat query-anchored, showing identities
- `4` — Flat query-anchored, no identities
- `5` — Query-anchored, no identities and blunt ends
- `6` — Flat query-anchored, no identities and blunt ends
- `7` — Not used
- `8` — Tabular
- `9` — Tabular with comment lines

(This corresponds to the `blastall` option `-m`.)

... `blastlocal(..., 'ToFile', `*`ToFileValue`*`, ...)` saves the contents of the BLAST report to the specified file. (This corresponds to the `blastall` option `-o`.)

... `blastlocal(..., 'Filter', `*`FilterValue`*`, ...)` specifies whether a filter (DUST filter for the blastn program or SEG filter for other programs) is applied to the query sequence(s). Choices are `true` (default) or `false`. (This corresponds to the `blastall` option `-F`.)

... `blastlocal(..., 'GapOpen', `*`GapOpenValue`*`, ...)` specifies the penalty for opening a gap in the alignment of sequences. Default is `-1`. (This corresponds to the `blastall` option `-G`.)

... `blastlocal(..., 'GapExtend', `*`GapExtendValue`*`, ...)` specifies the penalty for extending a gap in the alignment of sequences. Default is `-1`. (This corresponds to the `blastall` option `-E`.)

... `blastlocal(..., 'BLASTArgs', `*`BLASTArgsValue`*`, ...)` specifies options using the input arguments for the NCBI `blastall` function. *BLASTArgsValue* is a string containing one or more instances or `-x` and the option associated with it. For example, to specify the `BLOSUM 45` matrix, you would use the following syntax:

```
blastlocal('InputQuery', ecoliquery.txt, 'BLASTArgs', '-M BLOSUM45')
```

Tip Use the `'BlastArgs'` property to specify `blastall` options for which there are no corresponding property name/property value pairs.

Note: For a complete list of valid input arguments for the NCBI `blastall` function, make sure that the `blastall` executable file is located on your system path or current folder, then type the following at your system's command prompt.

```
blastall -
```

Using blastall Syntax

You can also use the syntax and input arguments accepted by the NCBI `blastall` function, instead of the property name/property value pairs listed previously. To do so, supply a single string containing multiple options using the *-x option* syntax. For example, you can specify the `ecoliquery.txt` FASTA file as your query sequences, the `blastp` program, and the `ecoli` local database, by using

```
blastlocal('-i ecoliquery.txt -p blastp -d ecoli')
```

Note: For a complete list of valid input arguments for the NCBI `blastall` function, make sure that the `blastall` executable file is located on your system path or current folder, then type the following at your system's command prompt.

```
blastall -
```

Examples

The following examples assume you have a FASTA nucleotide file and a FASTA amino acid file for *E. coli*, such as the files `NC_004431.fna` and `NC_004431.faa`, which you can download from ftp://ftp.ncbi.nih.gov/genomes/Bacteria/, saved to your MATLAB current folder.

Performing a Nucleotide Translated Search

1   Create local blastable databases from the `NC_004431.fna` and `NC_004431.faa` FASTA files by using the `blastformat` function.

```
blastformat('inputdb', 'NC_004431.fna', 'protein', 'false');
```

1-317

```
blastformat('inputdb', 'NC_004431.faa');
```

2 Use the `getgenbank` function to retrieve sequence information for the *E. coli* threonine operon from the GenBank database.

```
S = getgenbank('M28570');
```

3 Create a query file by using the `fastawrite` function to create a FASTA file named `query_nt.fa` from this sequence information, using only the accession number as the header.

```
S.Header = S.Accession;
fastawrite('query_nt.fa', S);
```

4 Use MATLAB syntax to submit the query sequence in the `query_nt.fa` FASTA file for a BLAST search of the local amino acid database `NC_004431.faa`. Specify the BLAST program `blastx`. Return the BLAST search results in `results`, a MATLAB structure.

```
results = blastlocal('inputquery', 'query_nt.fa',...
                     'database', 'NC_004431.faa',...
                     'program', 'blastx');
```

Performing a Nucleotide Search Using blastall Syntax

1 If you have not already done so, create local blastable databases and a query file as described in steps 1 through 3 in Performing a Nucleotide Translated Search.

2 Use `blastall` syntax to submit the query sequence in the `query_nt.fa` FASTA file for a BLAST search of the local nucleotide database `NC_004431.fna`. Specify the BLAST program `blastn` and an expectation value of `0.0001`. Return the BLAST search results in `results`, a MATLAB structure.

```
results = blastlocal('-i query_nt.fa -d NC_004431.fna ...
                     -p blastn -e 0.0001');
```

Performing a Nucleotide Search and Creating a Formatted Report

1 If you have not already done so, create local blastable databases and a query file as described in steps 1 through 3 in Performing a Nucleotide Translated Search.

2 Submit the query sequence in the `query_nt.fa` FASTA file for a BLAST search of the local nucleotide database `NC_004431.fna`. Specify the BLAST program `blastn` and a tabular alignment format. Save the contents of the BLAST report to a file named `myecoli_nt.txt`.

```
blastlocal('inputquery', 'query_nt.fa',...
```

```
'database', 'NC_OO4431.fna', 'tofile',...
'myecoli_nt.txt', 'blastargs', '-p blastn -m 8');
```

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. (1990). Basic local alignment search tool. J. Mol. Biol. *215*, 403–410.

[2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. *25*, 3389–3402.

For more information on the NCBI `blastall` function, see:

`http://blast.ncbi.nlm.nih.gov/docs/blastall.html`

See Also
`blastformat` | `blastncbi` | `blastread` | `blastreadlocal` | `getblast`

blastncbi

Create remote NCBI BLAST report request ID or link to NCBI BLAST report

Syntax

```
blastncbi(Seq, Program)
RID = blastncbi(Seq, Program)
[RID, RTOE] = blastncbi(Seq, Program)
... blastncbi(Seq, Program, ...'Database', DatabaseValue, ...)
... blastncbi(Seq, Program, ...'Descriptions',
DescriptionsValue, ...)
... blastncbi(Seq, Program, ...'Alignments', AlignmentsValue, ...)
... blastncbi(Seq, Program, ...'Filter', FilterValue, ...)
... blastncbi(Seq, Program, ...'Expect', ExpectValue, ...)
... blastncbi(Seq, Program, ...'Word', WordValue, ...)
... blastncbi(Seq, Program, ...'Matrix', MatrixValue, ...)
... blastncbi(Seq, Program, ...'GapOpen', GapOpenValue, ...)
... blastncbi(Seq, Program, ...'ExtendGap', ExtendGapValue, ...)
... blastncbi(Seq, Program, ...'GapCosts', GapCostsValue, ...)
... blastncbi(Seq, Program, ...'Inclusion', InclusionValue, ...)
... blastncbi(Seq, Program, ...'Pct', PctValue, ...)
... blastncbi(Seq, Program, ...'Entrez', EntrezValue, ...)
```

Input Arguments

Seq	Nucleotide or amino acid sequence specified by any of the following: • GenBank, GenPept, or RefSeq accession number • GI sequence identifier • FASTA file • URL pointing to a sequence file • String • Character array

	• MATLAB structure containing a `Sequence` field
Program	String specifying a BLAST program. Choices are:
	• `'blastn'` — Search nucleotide query versus nucleotide database.
	• `'blastp'` — Search protein query versus protein database.
	• `'blastx'` — Search translated nucleotide query versus protein database.
	• `'megablast'` — Quickly search for highly similar nucleotide sequences.
	• `'psiblast'` — Search protein query using position-specific iterative BLAST.
	• `'tblastn'` — Search protein query versus translated nucleotide database.
	• `'tblastx'` — Search translated nucleotide query versus translated nucleotide database.
DatabaseValue	String specifying a database. Compatible databases depend on the type of sequence specified by *Seq*, and the program specified by *Program*.
	For a list of database choices for nucleotide sequences and amino acid sequences, see the lists in the section "Description" on page 1-326.
DescriptionsValue	Value specifying the number of short descriptions to include in the report. Default is `100`, unless *Program* = `'psiblast'`, then default is `500`.
AlignmentsValue	Value specifying the number of sequences for which high-scoring segment pairs (HSPs) are reported. Default is `100`, unless *Program* = `'psiblast'`, then default is `500`.

FilterValue	String specifying a filter. Possible choices are: • `'L'` (default) — Low complexity. • `'R'` — Human repeats. • `'m'` — Mask for lookup table. • `'lcase'` — Turn on the lowercase mask. Choices vary depending on the selected *Program*. For more information, see the table Choices for Optional Properties by BLAST Program.
ExpectValue	Value specifying the statistical significance threshold for matches against database sequences. Choices are any real number. Default is `10`.

WordValue	Value specifying a word length for the query sequence.
	Choices for amino acid sequences are:
	• `2`
	• `3` (default)
	Choices for nucleotide sequences are:
	• `7`
	• `11` (default)
	• `15`
	Choices when *Program* = `'megablast'` are:
	• `11`
	• `12`
	• `16`
	• `20`
	• `24`
	• `28` (default)
	• `32`
	• `48`
	• `64`
MatrixValue	String specifying the substitution matrix for amino acid sequences only. The matrix assigns the score for a possible alignment of any two amino acid residues. Choices are:
	• `'PAM30'`
	• `'PAM70'`
	• `'BLOSUM45'`
	• `'BLOSUM62'` (default)
	• `'BLOSUM80'`

GapOpenValue	Integer that specifies the penalty for opening a gap in the alignment of amino acid sequences.
	Choices and default depend on the substitution matrix specified by the `'Matrix'` property. For more information, see the table Choices for the GapCosts Property by Matrix.
ExtendGapValue	Integer that specifies the penalty for extending a gap in the alignment of amino acid sequences.
	Choices and default depend on the substitution matrix specified by the `'Matrix'` property. For more information, see the table Choices for the GapCosts Property by Matrix.
GapCostsValue	Vector containing two integers: the first is the penalty for opening a gap, and the second is the penalty for extending the gap, in the alignment of amino acid sequences.
	Choices and default depend on the substitution matrix specified by the `'Matrix'` property. For more information, see the table Choices for the GapCosts Property by Matrix.
InclusionValue	Value specifying the statistical significance threshold for including a sequence in the Position-Specific Scoring Matrix (PSSM) created by PSI-BLAST for the subsequent iteration. Default is `0.005`.
	Note: Specify an *InclusionValue* only when *Program* = `'psiblast'`.

PctValue	Value specifying the percent identity and the corresponding match and mismatch score for matching existing sequences in a public database. Choices are:
	• None
	• 99 (default) — 99, 1, -3
	• 98 — 98, 1, -3
	• 95 — 95, 1, -3
	• 90 — 90, 1, -2
	• 85 — 85, 1, -2
	• 80 — 80, 2, -3
	• 75 — 75, 4, -5
	• 60 — 60, 1, -1
	Note: Specify a *PctValue* only when *Program* = 'megablast'.
EntrezValue	String specifying Entrez query syntax to search a subset of the selected database.
	Tip Use this property to limit searches based on molecule types, sequence lengths, organisms, and so on.

Output Arguments

RID	Request ID for the NCBI BLAST report.
RTOE	Request Time Of Execution, which is an estimate of the time (in minutes) until completion.
	Tip Use this time estimate with the 'WaitTime' property when using the getblast function.

Description

The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of protein and nucleotide sequences against known sequences in online databases.

blastncbi(*Seq*, *Program*) sends a BLAST request to NCBI against a *Seq*, a nucleotide or amino acid sequence, using *Program*, a specified BLAST program, and then returns a command window link to the NCBI BLAST report. For help in selecting an appropriate BLAST program, visit:

http://blast.ncbi.nlm.nih.gov/producttable.shtml

RID = blastncbi(*Seq*, *Program*) returns *RID*, the Request ID for the report.

[*RID*, *RTOE*] = blastncbi(*Seq*, *Program*) returns both *RID*, the Request ID for the NCBI BLAST report, and *RTOE*, the Request Time Of Execution, which is an estimate of the time until completion.

Tip Use *RTOE* with the 'WaitTime' property when using the getblast function.

... blastncbi(..., '*PropertyName*', *PropertyValue*,...) calls blastncbi with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are explained below. Additional information on these optional properties can be found at:

http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml

... blastncbi(*Seq*, *Program*, ...'Database', *DatabaseValue*, ...) specifies a database for the alignment search. Compatible databases depend on the type of sequence specified by *Seq*, and the program specified by *Program*.

Database choices for nucleotide sequences are:

- 'nr' (default)
- 'refseq_rna'
- 'refseq_genomic'
- 'est'

- `'est_human'`
- `'est_mouse'`
- `'est_others'`
- `'gss'`
- `'htgs'`
- `'pat'`
- `'pdb'`
- `'month'`
- `'alu_repeats'`
- `'dbsts'`
- `'chromosome'`
- `'wgs'`
- `'env_nt'`

Database choices for amino acid sequences are:

- `'nr'` (default)
- `'refseq_protein'`
- `'swissprot'`
- `'pat'`
- `'month'`
- `'pdb'`
- `'env_nr'`

For help in selecting an appropriate database, visit:

`http://blast.ncbi.nlm.nih.gov/producttable.shtml`

`... blastncbi(`*Seq*`, `*Program*`, ...'Descriptions',` *DescriptionsValue*`, ...)` specifies the number of short descriptions to include in the report, when you do not specify return values.

`... blastncbi(`*Seq*`, `*Program*`, ...'Alignments', `*AlignmentsValue*`, ...)` specifies the number of sequences for which high-scoring segment pairs (HSPs) are reported, when you do not specify return values.

`... blastncbi(`*Seq*`, `*Program*`, ...'Filter', `*FilterValue*`, ...)` specifies the filter to apply to the query sequence.

`... blastncbi(`*Seq*`, `*Program*`, ...'Expect', `*ExpectValue*`, ...)` specifies a statistical significance threshold for matches against database sequences. Choices are any real number. Default is `10`. You can learn more about the statistics of local sequence comparison at:

`http://blast.ncbi.nlm.nih.gov/tutorial/Altschul-1.html#head2`

`... blastncbi(`*Seq*`, `*Program*`, ...'Word', `*WordValue*`, ...)` specifies a word size for the query sequence.

`... blastncbi(`*Seq*`, `*Program*`, ...'Matrix', `*MatrixValue*`, ...)` specifies the substitution matrix for amino acid sequences only. This matrix assigns the score for a possible alignment of two amino acid residues.

`... blastncbi(`*Seq*`, `*Program*`, ...'GapOpen', `*GapOpenValue*`, ...)` specifies the penalty for opening a gap in the alignment of amino acid sequences. Choices and default depend on the substitution matrix specified by the `'Matrix'` property. For more information, see the table Choices for the GapCosts Property by Matrix.

For more information about allowed gap penalties for various matrices, see:

`http://blast.ncbi.nlm.nih.gov/html/sub_matrix.html`

`... blastncbi(`*Seq*`, `*Program*`, ...'ExtendGap', `*ExtendGapValue*`, ...)` specifies the penalty for extending a gap greater than one space in the alignment of amino acid sequences. Choices and default depend on the substitution matrix specified by the `'Matrix'` property. For more information, see the table Choices for the GapCosts Property by Matrix.

`... blastncbi(`*Seq*`, `*Program*`, ...'GapCosts', `*GapCostsValue*`, ...)` specifies the penalty for opening and extending a gap in the alignment of amino acid sequences. *GapCostsValue* is a vector containing two integers: the first is the penalty for opening a gap, and the second is the penalty for extending the gap. Choices and default depend on the substitution matrix specified by the `'Matrix'` property. For more information, see the table Choices for the GapCosts Property by Matrix.

`... blastncbi(`*Seq*`, `*Program*`, ...'Inclusion', `*InclusionValue*`, ...)` specifies the statistical significance threshold for including a sequence in the Position-

Specific Scoring Matrix (PSSM) created by PSI-BLAST for the subsequent iteration. Default is 0.005.

Note: Specify an *InclusionValue* only when *Program* = 'psiblast'.

... blastncbi(*Seq*, *Program*, ...'Pct', *PctValue*, ...) specifies the percent identity and the corresponding match and mismatch score for matching existing sequences in a public database. Default is 99.

Note: Specify a *PctValue* only when *Program* = 'megablast'.

... blastncbi(*Seq*, *Program*, ...'Entrez', *EntrezValue*, ...) specifies Entrez query syntax to search a subset of the selected database.

Note: For more information about Entrez query syntax, see:

http://www.ncbi.nlm.nih.gov/books/NBK3837/

Tip Use this property to limit searches based on molecule types, sequence lengths, organisms, and so on. For more information on limiting searches, see:

http://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml#entrez_query

---

Choices for Optional Properties by BLAST Program

336

When BLAST program is...	Then choices for the following properties are...					
	Database	**Filter**	**Word**	**Matrix**	**GapCosts**	**Pct**
'blastn'	'nr' (default) 'est' 'est_human' 'est_mouse'	'L' (default) 'R' 'm' 'lcase'	7 11 (default) 15	—	—	—
'megablast'	'est_others' 'gss' 'htgs' 'pat' 'pdb' 'month' 'alu_repeats' 'dbsts' 'chromosome' 'wgs' 'refseq_rna' 'refseq_genomic' 'env_nt'	'L'	11 12 16 20 24 28 (default) 32 48 64			None 99 (default) 98 95 90 85 80 75 60
'tblastn'		'L' (default) 'm' 'lcase'	2 3 (default)	'PAM30' 'PAM70' 'BLOSUM45' 'BLOSUM62' (default) 'BLOSUM80'	See the next table.	—
'tblastx'		'L' (default) 'R' 'm' 'lcase'				
'blastp'	'nr' (default) 'swissprot' 'pat' 'pdb' 'month' 'refseq_protein' 'env_nr'	'L' (default) 'm' 'lcase'				
'blastx'						
'psiblast'						

Choices for the GapCosts Property by Matrix

When substitution matrix is...	Then choices for GapCosts are...

`'PAM30'`	`[7 2]`
	`[6 2]`
	`[5 2]`
	`[10 1]`
	`[9 1]` (default)
	`[8 1]`

`'PAM70'`
 	
`'BLOSUM80'`
	`[8 2]`
	`[7 2]`
	`[6 2]`
	`[11 1]`
	`[10 1]` (default)
	`[9 1]`

`'BLOSUM45'`	`[13 3]`
	`[12 3]`
	`[11 3]`
	`[10 3]`
	`[15 2]` (default)
	`[14 2]`
	`[13 2]`
	`[12 2]`
	`[19 1]`
	`[18 1]`
	`[17 1]`
	`[16 1]`

`'BLOSUM62'`	`[9 2]`
	`[8 2]`
	`[7 2]`
	`[12 1]`
	`[11 1]` (default)
	`[10 1]`

Examples

```
% Get a sequence from the Protein Data Bank and create
```

```
% a MATLAB structure.
S = getpdb('1CIV')

% Use the structure as input for a BLAST search with an
% expectation of 1e-10.
blastncbi(S,'blastp','expect',1e-10)

% Click the URL link (Link to NCBI BLAST Request) to go
% directly to the NCBI request.

% You can also perform a typical BLAST protein search directly
% with an accession number and an alternative scoring matrix.
RID = blastncbi('AAA59174','blastp','matrix','PAM70',...
                            'expect',1e-10)

% You can pass the RID to GETBLAST to parse the report and
% load it into a MATLAB structure.
Struct = getblast(RID)
```

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990). Basic local alignment search tool. J. Mol. Biol. *215*, 403–410.

[2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. *25*, 3389–3402.

See Also
blastformat | blastlocal | blastread | blastreadlocal | getblast

blastread

Read data from NCBI BLAST report file

Syntax

Data = blastread(*BLASTReport*)

Input Arguments

BLASTReport	NCBI BLAST-formatted report specified by any of the following: • File name or path and file name, such as returned by the `getblast` function with the `'ToFile'` property. • URL pointing to an NCBI BLAST report. • MATLAB character array that contains the text for an NCBI BLAST report. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.

Output Arguments

Data	MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from an NCBI BLAST report.

Description

The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of protein and nucleotide sequences against known sequences in online databases. BLAST reports can be lengthy, and parsing the data from the various formats can be cumbersome.

Data = blastread(*BLASTReport*) reads a BLAST report from *BLASTReport*, an NCBI-formatted report, and returns *Data*, a MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to the BLAST keywords. blastread parses the basic BLAST reports BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX.

Data contains the following fields.

Field	Description
RID	Request ID for retrieving results for a specific NCBI BLAST search.
Algorithm	NCBI algorithm used to do a BLAST search.
Query	Identifier of the query sequence submitted to a BLAST search.
Database	All databases searched.
Hits.Name	Name of a database sequence (subject sequence) that matched the query sequence.
Hits.Length	Length of a subject sequence.
Hits.HSPs.Score	Pairwise alignment score for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Expect	Expectation value for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Identities	Identities (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Positives	Identical or similar residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject amino acid sequence. **Note:** This field applies only to translated nucleotide or amino acid query sequences and/or databases.

Field	Description
`Hits.HSPs.Gaps`	Nonaligned residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.Frame`	Reading frame of the translated nucleotide sequence for a high-scoring sequence pair between the query sequence and a subject sequence. **Note:** This field applies only when performing translated searches, that is, when using `tblastx`, `tblastn`, and `blastx`.
`Hits.HSPs.Strand`	Sense (`Plus` = 5' to 3' and `Minus` = 3' to 5') of the DNA strands for a high-scoring sequence pair between the query sequence and a subject sequence. **Note:** This field applies only when using a nucleotide query sequence and database.
`Hits.HSPs.Alignment`	Three-row matrix showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.QueryIndices`	Indices of the query sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.SubjectIndices`	Indices of the subject sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
`Statistics`	Summary of statistical details about the performed search, such as lambda values, gap penalties, number of sequences searched, and number of hits.

Examples

1 Create an NCBI BLAST report request using a GenPept accession number.

```
RID = blastncbi('AAA59174', 'blastp', 'expect', 1e-10)

RID =

    '1175088155-31624-126008617054.BLASTQ3'
```

2 Pass the Request ID for the report to the `getblast` function, and save the report data to a text file.

```
getblast(RID, 'ToFile' ,'AAA59174_BLAST.rpt');
```

Note: You may need to wait for the report to become available on the NCBI Web site before you can run the preceding command.

---

3 Using the saved file, read the results into a MATLAB structure.

```
resultsStruct = blastread('AAA59174_BLAST.rpt')

resultsStruct =

          RID: '1175093446-29831-201366571074.BLASTQ2'
    Algorithm: 'BLASTP 2.2.16 [Mar-11-2007]'
        Query: [1x63 char]
     Database: [1x96 char]
         Hits: [1x50 struct]
   Statistics: [1x1034 char]
```

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990). Basic local alignment search tool. J. Mol. Biol. *215*, 403–410.

[2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. *25*, 3389–3402.

For more information about reading and interpreting NCBI BLAST reports, see:

http://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastDocs

See Also
blastformat | blastlocal | blastncbi | blastreadlocal | getblast

blastreadlocal

Read data from local BLAST report

Syntax

Data = blastreadlocal(*BLASTReport*, *Format*)

Input Arguments

BLASTReport	BLAST report specified by any of the following: • File name or path and file name of a locally created BLAST report file, such as returned by the blastlocal function with the 'ToFile' property. • MATLAB character array that contains the text for a local BLAST report. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.
Format	Integer specifying the alignment format used to create *BLASTReport*. Choices are: • 0 — Pairwise • 1 — Query-anchored, showing identities • 2 — Query-anchored, no identities • 3 — Flat query-anchored, showing identities • 4 — Flat query-anchored, no identities • 5 — Query-anchored, no identities and blunt ends • 6 — Flat query-anchored, no identities and blunt ends • 7 — Not used • 8 — Tabular • 9 — Tabular with comment lines

Output Arguments

Data	MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report.

Description

The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of protein and nucleotide sequences against known sequences in online and local databases. BLAST reports can be lengthy, and parsing the data from the various formats can be cumbersome.

Data = blastreadlocal(*BLASTReport*, *Format*) reads *BLASTReport*, a locally created BLAST report file, and returns *Data*, a MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report. *Format* is an integer specifying the alignment format used to create *BLASTReport*.

Note: The function assumes the BLAST report was produced using version 2.2.17 of the blastall executable.

Data contains a subset of the following fields, based on the specified alignment format.

Field	Description
Algorithm	NCBI algorithm used to do a BLAST search.
Query	Identifier of the query sequence submitted to a BLAST search.
Length	Length of the query sequence.
Database	All databases searched.
Hits.Name	Name of a database sequence (subject sequence) that matched the query sequence.

Field	Description
`Hits.Score`	Alignment score between the query sequence and the subject sequence.
`Hits.Expect`	Expectation value for the alignment between the query sequence and the subject sequence.
`Hits.Length`	Length of a subject sequence.
`Hits.HSPs.Score`	Pairwise alignment score for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.Expect`	Expectation value for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.Identities`	Identities (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.Positives`	Identical or similar residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject amino acid sequence. **Note:** This field applies only to translated nucleotide or amino acid query sequences and/or databases.
`Hits.HSPs.Gaps`	Nonaligned residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.Mismatches`	Residues that are not similar to each other (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.

Field	Description
Hits.HSPs.Frame	Reading frame of the translated nucleotide sequence for a high-scoring sequence pair between the query sequence and a subject sequence.
	Note: This field applies only when performing translated searches, that is, when using tblastx, tblastn, and blastx.
Hits.HSPs.Strand	Sense (Plus = 5' to 3' and Minus = 3' to 5') of the DNA strands for a high-scoring sequence pair between the query sequence and a subject sequence.
	Note: This field applies only when using a nucleotide query sequence and database.
Hits.HSPs.Alignment	Three-row matrix showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.QueryIndices	Indices of the query sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.SubjectIndices	Indices of the subject sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.AlignmentLength	Length of the pairwise alignment for a high-scoring sequence pair between the query sequence and a subject sequence.
Alignment	Entire alignment for the query sequence and the subject sequence(s).

Field	Description
Statistics	Summary of statistical details about the performed search, such as lambda values, gap penalties, number of sequences searched, and number of hits.

Examples

The following examples assume you have a FASTA nucleotide file for *E. coli*, such as the file NC_004431.fna, which you can download from ftp://ftp.ncbi.nih.gov/genomes/Bacteria/, saved to your MATLAB current folder.

Reading Data Using a Tabular Alignment Format

1   Create a local blastable database from the NC_004431.fna FASTA file.

```
blastformat('inputdb', 'NC_004431.fna', 'protein', 'false');
```

2   Use the getgenbank function to retrieve two sequences from the GenBank database.

```
S1 = getgenbank('M28570.1');
S2 = getgenbank('M12565');
```

3   Create a query file by using the fastawrite function to create a FASTA file named query_multi_nt.fa from these two sequences, using the only accession number as the header.

```
Seqs(1).Header = S1.Accession;
Seqs(1).Sequence = S1.Sequence;
Seqs(2).Header = S2.Accession;
Seqs(2).Sequence = S2.Sequence;
fastawrite('query_multi_nt.fa', Seqs);
```

4   Submit the query sequences in the query_multi_nt.fa FASTA file for a BLAST search of the local nucleotide database NC_004431.fna. Specify the BLAST program blastn and a tabular alignment format. Save the contents of the BLAST report to a file named myecoli_nt8.txt, and then read the local BLAST report.

```
blastlocal('inputquery', 'query_multi_nt.fa',...
           'database', 'NC_004431.fna',...
           'tofile', 'myecoli_nt8.txt', 'program', 'blastn',...
           'format', 8);
```

```
blastreadlocal('myecoli_nt8.txt', 8);
```

Reading Data Using a Query Anchored Format

1 If you have not already done so, create a local blastable database and a query file as described in steps 1 through 3 in Reading Data Using a Tabular Alignment Format.

2 Submit the query sequences in the `query_multi_nt.fa` FASTA file for a BLAST search of the local nucleotide database `NC_004431.fna`. Specify the BLAST program `blastn` and a query-anchored format. Save the contents of the BLAST report to a file named `myecoli_nt1.txt`, and then read the local BLAST report, saving the results in `results`, an array of structures.

```
blastlocal('inputquery', 'query_multi_nt.fa',...
           'database', 'NC_004431.fna',...
           'tofile', 'myecoli_nt1.txt', 'program', 'blastn',...
           'format', 1);
results = blastreadlocal('myecoli_nt1.txt', 1);
```

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. (1990). Basic local alignment search tool. J. Mol. Biol. *215*, 403–410.

[2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. *25*, 3389–3402.

For more information about reading and interpreting BLAST reports, see:

```
http://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastDocs
```

See Also
blastformat | blastlocal | blastncbi | blastread | getblast

blosum

Return BLOSUM scoring matrix

Syntax

```
Matrix = blosum(Identity)
[Matrix, MatrixInfo] = blosum(Identity)
... = blosum(Identity, ...'Extended', ExtendedValue, ...)
... = blosum(Identity, ...'Order', OrderValue, ...)
```

Input Arguments

Identity	Scalar specifying a percent identity level. Choices are: • Values from 30 to 90 in increments of 5 • 62 • 100
ExtendedValue	Controls the listing of extended amino acid codes. Choices are true (default) or false.
OrderValue	Character string of legal amino acid characters that specifies the order amino acids are listed in the matrix. The length of the character string must be 20 or 24.

Output Arguments

Matrix	BLOSUM (Blocks Substitution Matrix) scoring matrix with a specified percent identity.
MatrixInfo	Structure of information about *Matrix* containing the following fields: • Name • Scale

	• Entropy
	• ExpectedScore
	• HighestScore
	• LowestScore
	• Order

Description

Matrix = blosum(*Identity*) returns a BLOSUM (Blocks Substitution Matrix) scoring matrix with a specified percent identity. The default ordering of the output includes the extended characters B, Z, X, and *.

A R N D C Q E G H I L K M F P S T W Y V B Z X *

[*Matrix*, *MatrixInfo*] = blosum(*Identity*) returns *MatrixInfo*, a structure of information about *Matrix*, a BLOSUM matrix. *MatrixInfo* contains the following fields:

- Name
- Scale
- Entropy
- ExpectedScore
- HighestScore
- LowestScore
- Order

... = blosum(*Identity*, ...'*PropertyName*', *PropertyValue*, ...) calls blosum with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = blosum(*Identity*, ...'Extended', *ExtendedValue*, ...) controls the listing of extended amino acid codes. Choices are true (default) or false. If *ExtendedValue* is false, returns the scoring matrix for the standard 20 amino acids. Ordering of the output when *ExtendedValue* is false is

```
A R N D C Q E G H I L K M F P S T W Y V
```

`... = blosum(`*Identity*`, ...`'Order'`, `*OrderValue*`, ...)` returns a BLOSUM matrix ordered by *OrderValue*, a character string of legal amino acid characters that specifies the order amino acids are listed in the matrix. The length of the character string must be 20 or 24.

Examples

Return a BLOSUM matrix with a percent identity level of 50.

```
B50 = blosum(50)
```

Return a BLOSUM matrix with the amino acids in a specific order.

```
B75 = blosum(75,'Order','CSTPAGNDEQHRKMILVFYW')
```

See Also

dayhoff | gonnet | localalign | nuc44 | nwalign | pam | swalign

bowtie

Map short reads to reference sequence using Burrows-Wheeler transform

Syntax

```
bowtie(indexBaseName,reads,outputFileName)
bowtie(indexBaseName,reads,outputFileName,Name,Value)
```

Description

bowtie(indexBaseName,reads,outputFileName) aligns the reads specified in reads to the indexed reference specified by indexBaseName, and writes the results to the BAM-formatted file outputFileName.

Note: bowtie runs on Mac and UNIX® platforms only.

---

bowtie(indexBaseName,reads,outputFileName,Name,Value) aligns reads using additional options specified by one or more name-value pair arguments.

Examples

Align Short Reads

Download the *E. coli* genome from NCBI.

```
getgenbank('NC_008253','tofile','NC_008253.fna','SequenceOnly',true)
```

Built a Bowtie index with the base name ECOLI.

```
bowtiebuild('NC_008253.fna','ECOLI')
```

Find the path to the example FASTQ file ecoli100.fq, which has *E. Coli* short reads.

```
fastqfile = which('ecoli100.fq')
```

Align the short reads in `ecoli100.fq` to the built index with base name `ECOLI`.

```
bowtie('ECOLI',fastqfile,'ecoli100.bam')
```

Access the mapped reads using `BioMap`.

```
bm = BioMap('ecoli100.bam')

bm =

BioMap with properties:

    SequenceDictionary: {'gi|110640213|ref|NC_008253.1|'}
             Reference: [73x1 File indexed property]
             Signature: [73x1 File indexed property]
                 Start: [73x1 File indexed property]
        MappingQuality: [73x1 File indexed property]
                  Flag: [73x1 File indexed property]
          MatePosition: [73x1 File indexed property]
               Quality: [73x1 File indexed property]
              Sequence: [73x1 File indexed property]
                Header: [73x1 File indexed property]
                 NSeqs: 73
                  Name: ''
```

Input Arguments

`indexBaseName` — Name of indexed reference file
string

Name of indexed reference file for short read alignment, specified as a string containing the path and base name of the Bowtie index file.

`reads` — Short reads to align
string | cell array of strings

Short reads to align to the indexed reference, specified as a string or cell array of strings indicating one or more FASTQ formatted files with the input reads.

`outputFileName` — Name for output file
string

Name for output file containing the results of the short read alignment, specified as a string. By default, the output file is BAM-formatted, and `bowtie` automatically adds the `.bam` extension if it is missing from the file name.

To specify a SAM-formatted output file, use the name-value pair argument `BamFileOutput,false`. In this case, `bowtie` automatically adds the `.sam` extension if it is missing from the file name.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'BamFileOutput',false,'Paired',true` specifies the output file is SAM-formatted, and `bowtie` performs pair-read alignment.

`'BamFileOutput'` — Indicator for output file format
`true` (default) | `false`

Indicator for the output file format, specified as the comma-separated pair consisting of `'BamFileOutput'` and either `true` or `false`.

- If `true` (the default), then the output file is BAM-formatted, with a `.bam` extension.
- If `false`, then the output file is SAM-formatted, with a `.sam` extension.

`bowtie` automatically adds the corresponding file extension if it is missing from the input argument `outputFileName`.

Example: `'BamFileOutput',false`

Data Types: `logical`

`'Paired'` — Indicator for paired-read alignment performance
`false` (default) | `true`

Indicator for paired-read alignment performance, specified as the comma-separated pair consisting of `'Paired'` and either `true` or `false` (the default). If `false`, then `bowtie` performs paired-read alignment using the odd elements in reads as the upstream mates and the even elements in reads as the downstream mates.

Example: `'Paired',true`

Data Types: `logical`

`'BowtieOptions'` — Additional `bowtie` options
valid `bowtie` option

Additional `bowtie` options, specified as the comma-separated pair consisting of `'BowtieOptions'` and any valid `bowtie` option. Type `bowtie('--help')` for available options.

Example: `'BowtieOptions','-k 5'`

More About

Tips

- More information on the Bowtie algorithm (Version 0.12.7) can be found at http://bowtie-bio.sourceforge.net/index.shtml.
- Some prebuilt index files for model organisms can be downloaded directly from the Bowtie repository.

See Also

baminfo | BioMap | bowtiebuild | fastainfo | fastqinfo | saminfo | samread

bowtiebuild

Generate index using Burrows-Wheeler transform

Syntax

```
bowtiebuild(input,indexBaseName)
bowtiebuild(input,indexBaseName,'BowtieBuildOptions',options)
```

Description

bowtiebuild(input,indexBaseName) builds an index using the reference sequence(s) in input, and saves it to the index file indexBaseName.

Note: bowtiebuild runs on Mac and UNIX platforms only.

---

bowtiebuild(input,indexBaseName,'BowtieBuildOptions',options) specifies additional options.

Examples

Build a Bowtie Index

Download the *E. coli* genome from NCBI.

```
getgenbank('NC_008253','tofile','NC_008253.fna','SequenceOnly',true)
```

Built a Bowtie index with the base name ECOLI.

```
bowtiebuild('NC_008253.fna','ECOLI')
```

Input Arguments

input — FASTA-formatted files
string | cell array of strings

FASTA-formatted files with the reference sequences to be indexed, specified as a string or cell array of strings. Use a cell array of strings to specify multiple files.

`indexBaseName` — Name for indexed reference file
string

Name for indexed reference file, specified as a string containing the path and base name for the resulting Bowtie index file.

`options` — Additional `bowtiebuild` options
valid `bowtiebuild` option

Additional `bowtiebuild` options, specified as any valid `bowtiebuild` option. Type `bowtiebuild('--help')` for available options.

More About

Tips

- More information on the Bowtie algorithm (Version 0.12.7) can be found at http://bowtie-bio.sourceforge.net/index.shtml.
- Some prebuilt index files for model organisms can be downloaded directly from the Bowtie repository.

See Also
baminfo | BioMap | bowtie | fastainfo | fastqinfo | saminfo | samread

celintensityread

Read probe intensities from Affymetrix CEL files

Syntax

ProbeStructure = celintensityread(*CELFiles, CDFFile*)
ProbeStructure = celintensityread(..., 'CELPath', *CELPathValue*, ...)
ProbeStructure = celintensityread(..., 'CDFPath', *CDFPathValue*, ...)
ProbeStructure = celintensityread(..., 'PMOnly', *PMOnlyValue*, ...)
ProbeStructure = celintensityread(..., 'Verbose', *VerboseValue*, ...)

Input Arguments

CELFiles	Any of the following:
	• String specifying a single CEL file name.
	• `'*'`, which reads all CEL files in the current folder.
	• `' '`, which opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold **Ctrl** or **Shift** while clicking to select multiple CEL files.
	• Cell array of CEL file names.
CDFFile	Either of the following:
	• String specifying a CDF file name.
	• `' '`, which opens the Select CDF File dialog box from which you select the CDF file.
CELPathValue	String specifying the path and folder where the files specified in *CELFiles* are stored.
CDFPathValue	String specifying the path and folder where the file specified in *CDFFile* is stored.
PMOnlyValue	Property to include or exclude the mismatch (MM) probe intensity values in the returned structure. Enter `true` to return

	only perfect match (PM) probe intensities. Enter `false` to return both PM and MM probe intensities. Default is `true`.
VerboseValue	Controls the display of a progress report showing the name of each CEL file as it is read. When *VerboseValue* is `false`, no progress report is displayed. Default is `true`.

Output Arguments

ProbeStructure	MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs.

Description

ProbeStructure = celintensityread(*CELFiles, CDFFile*) reads the specified Affymetrix CEL files and the associated CDF library file (created from Affymetrix GeneChip arrays for expression or genotyping assays), and then creates *ProbeStructure*, a structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs. *CELFiles* is a string or cell array of CEL file names. *CDFFile* is a string specifying a CDF file name.

If you set *CELFiles* to '*', then it reads all CEL files in the current folder. If you set *CELFiles* to ' ', then it opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold **Ctrl** or **Shift** while clicking to select multiple CEL files.

If you set *CDFFile* to ' ', then it opens the Select CDF File dialog box from which you select the CDF file.

ProbeStructure = celintensityread(..., '*PropertyName*', *PropertyValue*, ...) calls celintensityread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

ProbeStructure = celintensityread(..., 'CELPath', *CELPathValue*, ...) specifies a path and folder where the files specified by *CELFiles* are stored.

ProbeStructure = celintensityread(..., 'CDFPath', *CDFPathValue*, ...) specifies a path and folder where the file specified by *CDFFile* is stored.

ProbeStructure = celintensityread(..., 'PMOnly', *PMOnlyValue*, ...) includes or excludes the mismatch (MM) probe intensity values. When *PMOnlyValue* is true, celintensityread returns only perfect match (PM) probe intensities. When *PMOnlyValue* is false, celintensityread returns both PM and MM probe intensities. Default is true.

You can learn more about the Affymetrix CEL files and download sample files from:

http://www.affymetrix.com/support/technical/sample_data/demo_data.affx

---

Note: Some Affymetrix CEL files are combined with other data files in a DTT or CAB file. You must download and use the Affymetrix Data Transfer Tool to extract these files from the DTT or CAB file. You can download the Affymetrix Data Transfer Tool from:

http://www.affymetrix.com/browse/products.jsp?productId=131431&navMode=34000&navAction=

You will have to register and log in at the Affymetrix Web site to download the Affymetrix Data Transfer Tool.

Tip Reading a large number of CEL files and/or a large CEL file can require extended amounts of memory from the operating system. If you receive any errors related to memory or have trouble reading CEL files, try the following:

- Increase the virtual memory (swap space) for your operating system (with a recommended initial size of 3,069 and a maximum size of 16,368) as described in "Memory Usage".

- Set the 3 GB switch (32-bit Windows® XP only) as described in "Memory Usage".

ProbeStructure contains the following fields.

Field	Description
CDFName	File name of the Affymetrix CDF library file.
CELNames	Cell array of names of the Affymetrix CEL files.
NumChips	Number of CEL files read into the structure.
NumProbeSets	Number of probe sets in each CEL file.
NumProbes	Number of probes in each CEL file.

Field	Description
ProbeSetIDs	Cell array of the probe set IDs from the Affymetrix CDF library file.
ProbeIndices	Column vector containing probe indexing information. Probes within a probe set are numbered 0 through N - 1, where N is the number of probes in the probe set.
GroupNumbers	Column vector containing group numbers for probes within the probe set. For gene expression data, the group number for all probes is 1. For SNP (genotyping) data, the group numbers for probes are: • 1 — Allele A – (sense) • 2 — Allele B – (sense) • 3 — Allele A + (antisense) • 4 — Allele B + (antisense)
PMIntensities	Matrix containing perfect match (PM) probe intensity values. Each row corresponds to a probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the *CELFiles* input argument.
MMIntensities (optional)	Matrix containing mismatch (MM) probe intensity values. Each row corresponds to a probe, and each column corresponds to a CEL file. The rows are ordered the same way as in ProbeIndices, and the columns are ordered the same way as in the *CELFiles* input argument.

ProbeStructure = celintensityread(..., 'Verbose', *VerboseValue*, ...)
controls the display of a progress report showing the name of each CEL file as it is read.
When *VerboseValue* is false, no progress report is displayed. Default is true.

Examples

The following example assumes that you have the HG_U95Av2.CDF library file stored
at D:\Affymetrix\LibFiles\HGGenome, and that your current folder points to a
location containing CEL files associated with this CDF library file. In this example, the
celintensityread function reads all the CEL files in the current folder and a CDF file

in a specified folder. The next command line uses the `rmabackadj` function to perform background adjustment on the PM probe intensities in the `PMIntensities` field of `PMProbeStructure`.

```
PMProbeStructure = celintensityread('*', 'HG_U95Av2.CDF',...
                    'CDFPath', 'D:\Affymetrix\LibFiles\HGGenome');
BackAdjustedMatrix = rmabackadj(PMProbeStructure.PMIntensities);
```

The following example lets you select CEL files and a CDF file to read using Open File dialog boxes:

```
PMProbeStructure = celintensityread(' ', ' ');
```

See Also
affygcrma | affyinvarsetnorm | affyprobeseqread | affyread | affyrma | affysnpintensitysplit | agferead | gcrma | gcrmabackadj | gprread | ilmnbsread | probelibraryinfo | probesetlink | probesetlookup | probesetplot | probesetvalues | rmabackadj | rmasummary | sptread

cghcbs

Perform circular binary segmentation (CBS) on array-based comparative genomic hybridization (aCGH) data

Syntax

```
SegmentStruct = cghcbs(CGHData)
SegmentStruct = cghcbs(CGHData, ...'Alpha', AlphaValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Permutations',
PermutationsValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Method', MethodValue, ...)
SegmentStruct = cghcbs(CGHData, ...'StoppingRule',
StoppingRuleValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Smooth', SmoothValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Prune', PruneValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Errsum', ErrsumValue, ...)
SegmentStruct = cghcbs(CGHData, ...'WindowSize',
WindowSizeValue, ...)
SegmentStruct = cghcbs(CGHData, ...'SampleIndex',
SampleIndexValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Chromosome',
ChromosomeValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Showplot', ShowplotValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Verbose', VerboseValue, ...)
```

Input Arguments

CGHData	Array-based comparative genomic hybridization (aCGH) data in either of the following forms:
	
	• Structure with the following fields:
	
	    • Sample — Cell array of strings containing the sample names (optional).
	
	    • Chromosome — Vector containing the chromosome numbers on which the clones are located.

	• GenomicPosition — Vector containing the genomic positions (in any unit) to which the clones are mapped. • Log2Ratio — Matrix containing $\log_2$ ratio of test to reference signal intensity for each clone. Each row corresponds to a clone, and each column corresponds to a sample. • Matrix in which each row corresponds to a clone. The first column contains the chromosome number, the second column contains the genomic position, and the remaining columns each contain the $\log_2$ ratio of test to reference signal intensity for a sample.
AlphaValue	Scalar that specifies the significance level for the statistical tests to accept change points. Default is `0.01`.
PermutationsValue	Scalar that specifies the number of permutations used for p-value estimation. Default is `10,000`.
MethodValue	String that specifies the method to estimate the p-values. Choices are `'Perm'` or `'Hybrid'` (default). `'Perm'` does a full permutation, while `'Hybrid'` uses a faster, tail probability-based permutation. When using the `'Hybrid'` method, the `'Perm'` method is applied automatically when segment data length becomes less than 200.
StoppingRuleValue	Controls the use of a heuristic stopping rule, based on the method described by Venkatraman and Olshen (2007), to declare a change without performing the full number of permutations for the p-value estimation, whenever it becomes very likely that a change has been detected. Choices are `true` or `false` (default). **Tip** Set this property to `true` to increase processing speed. Set this property to `false` to maximize accuracy.
SmoothValue	Controls the smoothing of outliers before segmenting using the procedure explained by Olshen et al. (2004). Choices are `true` (default) or `false`.

PruneValue	Controls the elimination of change points identified due to local trends in the data that are not indicative of real copy number change, using the procedure explained by Olshen et al. (2004). Choices are `true` or `false` (default).
ErrsumValue	Scalar that specifies the allowed proportional increase in the error sum of squares when eliminating change points using the `'Prune'` property. Commonly used values are `0.05` and `0.1`. Default is `0.05`.
WindowSizeValue	Scalar that specifies the size of the window (in data points) used to divide the data when using the `'Perm'` method on large data sets. Default is `200`.
SampleIndexValue	A single sample index or a vector of sample indices that specify the sample(s) to analyze. Default is all sample indices.
ChromosomeValue	A single chromosome number or a vector of chromosome numbers that specify the data to analyze. Default is all chromosome numbers.
ShowplotValue	Controls the display of plots of the segment means over the original data. Choices are either: • `true` — All chromosomes in all samples are plotted. If there are multiple samples in *CGHData*, then each sample is plotted in a separate Figure window. • `false` — No plot. • `W` — The layout displays all chromosomes in the whole genome in one plot in the Figure window. • `S` — The layout displays each chromosome in a subplot in the Figure window. • *I* — An integer specifying only one of the chromosomes in *CGHData* to be plotted. Default is: • `false` — When return values are specified. • `true` and `W` — When return values are not specified.
VerboseValue	Controls the display of a progress report of the analysis. Choices are `true` (default) or `false`.

Output Arguments

| *SegmentStruct*	Structure containing segmentation information in the following fields:
	• Sample — Sample name from *CGHData* input argument. If the input argument does not include sample names, then sample names are assigned as Sample1, Sample2, and so forth.
	• SegmentData — Structure array containing segment data for the sample in the following fields:
	• Chromosome — Chromosome number on which the segment is located.
	• Start — Genomic position at the start of the segment (in the same units as used for the *CGHData* input).
	• End — Genomic position at the end of the segment (in the same units as used for the *CGHData* input).
	• Mean — Mean value of the $\log_2$ ratio of the test to reference signal intensity for the segment.

Description

SegmentStruct = cghcbs(*CGHData*) performs circular binary segmentation (CBS) on array-based comparative genomic hybridization (aCGH) data to determine the copy number alteration segments (neighboring regions of DNA that exhibit a statistical difference in copy number) and change points.

Note: The CBS algorithm recursively splits chromosomes into segments based on a maximum t statistic estimated by permutation. This computation can be time consuming. If n = number of data points, then computation time ~ $O(n^2)$.

SegmentStruct = cghcbs(*CGHData*, ...'*PropertyName*', *PropertyValue*, ...) calls cghcbs with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each

PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

SegmentStruct = cghcbs(*CGHData*, ...'Alpha', *AlphaValue*, ...) specifies the significance level for the statistical tests to accept change points. Default is `0.01`.

SegmentStruct = cghcbs(*CGHData*, ...'Permutations', *PermutationsValue*, ...) specifies the number of permutations used for p-value estimation. Default is `10,000`.

SegmentStruct = cghcbs(*CGHData*, ...'Method', *MethodValue*, ...) specifies the method to estimate the p-values. Choices are `'Perm'` or `'Hybrid'` (default). `'Perm'` does a full permutation, while `'Hybrid'` uses a faster, tail probability-based permutation. When using the `'Hybrid'` method, the `'Perm'` method is applied automatically when segment data length becomes less than 200.

SegmentStruct = cghcbs(*CGHData*, ...'StoppingRule', *StoppingRuleValue*, ...) controls the use of a heuristic stopping rule, based on the method described by Venkatraman and Olshen (2007), to declare a change without performing the full number of permutations for the p-value estimation, whenever it becomes very likely that a change has been detected. Choices are `true` or `false` (default).

SegmentStruct = cghcbs(*CGHData*, ...'Smooth', *SmoothValue*, ...) controls the smoothing of outliers before segmenting, using the procedure explained by Olshen et al. (2004). Choices are `true` (default) or `false`.

SegmentStruct = cghcbs(*CGHData*, ...'Prune', *PruneValue*, ...) controls the elimination of change points identified due to local trends in the data that are not indicative of real copy number change, using the procedure explained by Olshen et al. (2004). Choices are `true` or `false` (default).

SegmentStruct = cghcbs(*CGHData*, ...'Errsum', *ErrsumValue*, ...) specifies the allowed proportional increase in the error sum of squares when eliminating change points using the `'Prune'` property. Commonly used values are `0.05` and `0.1`. Default is `0.05`.

SegmentStruct = cghcbs(*CGHData*, ...'WindowSize', *WindowSizeValue*, ...) specifies the size of the window (in data points) used to divide the data when using the `'Perm'` method on large data sets. Default is `200`.

SegmentStruct = cghcbs(*CGHData*, ...'SampleIndex',
SampleIndexValue, ...) analyzes only the sample(s) specified by
SampleIndexValue, which can be a single sample index or a vector of sample indices.
Default is all sample indices.

SegmentStruct = cghcbs(*CGHData*, ...'Chromosome',
ChromosomeValue, ...) analyzes only the data on the chromosomes specified
by *ChromosomeValue*, which can be a single chromosome number or a vector of
chromosome numbers. Default is all chromosome numbers.

SegmentStruct = cghcbs(*CGHData*, ...'Showplot', *ShowplotValue*, ...)
controls the display of plots of the segment means over the original data. Choices are
true, false, W, S, or *I*, an integer specifying one of the chromosomes in *CGHData*.
When *ShowplotValue* is true, all chromosomes in all samples are plotted. If there
are multiple samples in *CGHData*, then each sample is plotted in a separate Figure
window. When *ShowplotValue* is W, the layout displays all chromosomes in one plot in
the Figure window. When *ShowplotValue* is S, the layout displays each chromosome
in a subplot in the Figure window. When *ShowplotValue* is *I*, only the specified
chromosome is plotted. Default is either:

- false — When return values are specified.
- true and W — When return values are not specified.

SegmentStruct = cghcbs(*CGHData*, ...'Verbose', *VerboseValue*, ...)
controls the display of a progress report of the analysis. Choices are true (default) or
false.

Examples

Analyzing Data from the Coriell Cell Line Study

1   Load a MAT-file, included with the Bioinformatics Toolbox software, which contains
    coriell_data, a structure of array-based CGH data.

    load coriell_baccgh

2   Analyze all chromosomes of sample 3 (GM05296) of the aCGH data and return
    segmentation data in a structure, S. Plot the segment means over the original data
    for all chromosomes of this sample.

    S = cghcbs(coriell_data,'sampleindex',3,'showplot',true);
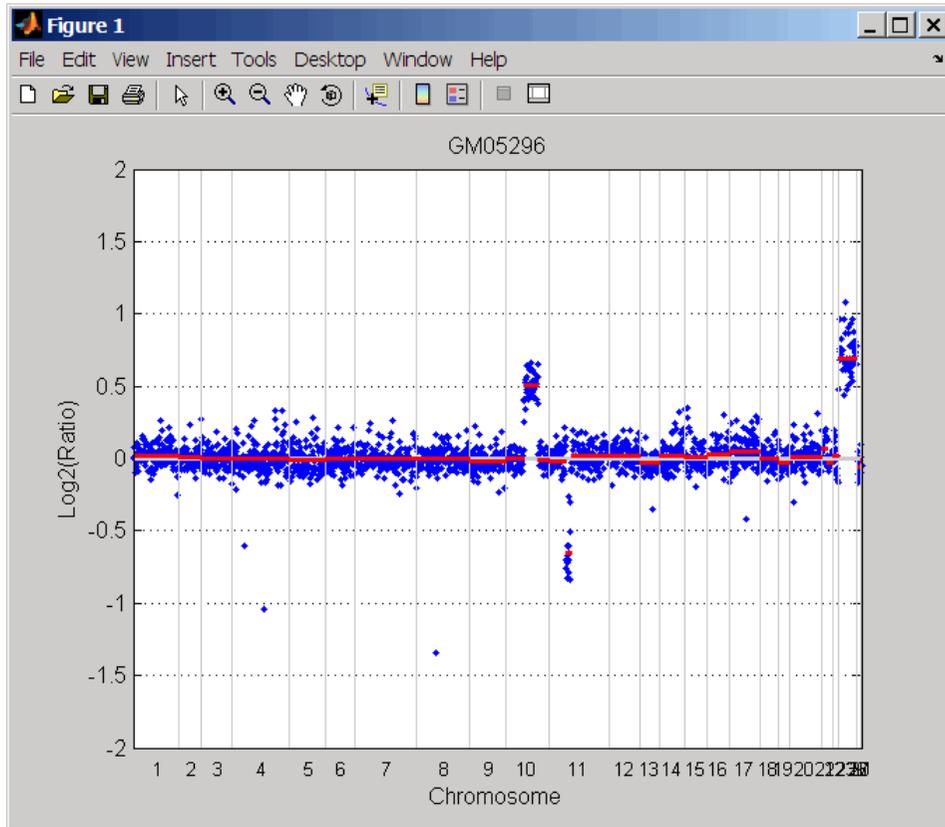
Chromosome 10 shows a gain, while chromosome 11 shows a loss.

The `coriell_baccgh.mat` file used in this example contains data from Snijders et al., 2001.

Analyzing Data from a Pancreatic Cancer Study

1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains `pancrea_data`, a structure of array-based CGH data from a pancreatic cancer study.

```
load pancrea_oligocgh
```

2 Analyze only chromosome 9 in sample 32 of the CGH data and return the segmentation data in a structure, `PS`. Plot the segment means over the original data for chromosome 9 in this sample.

```
PS = cghcbs(pancrea_data,'sampleindex',32,'chromosome',9,...
            'showplot',9);
```
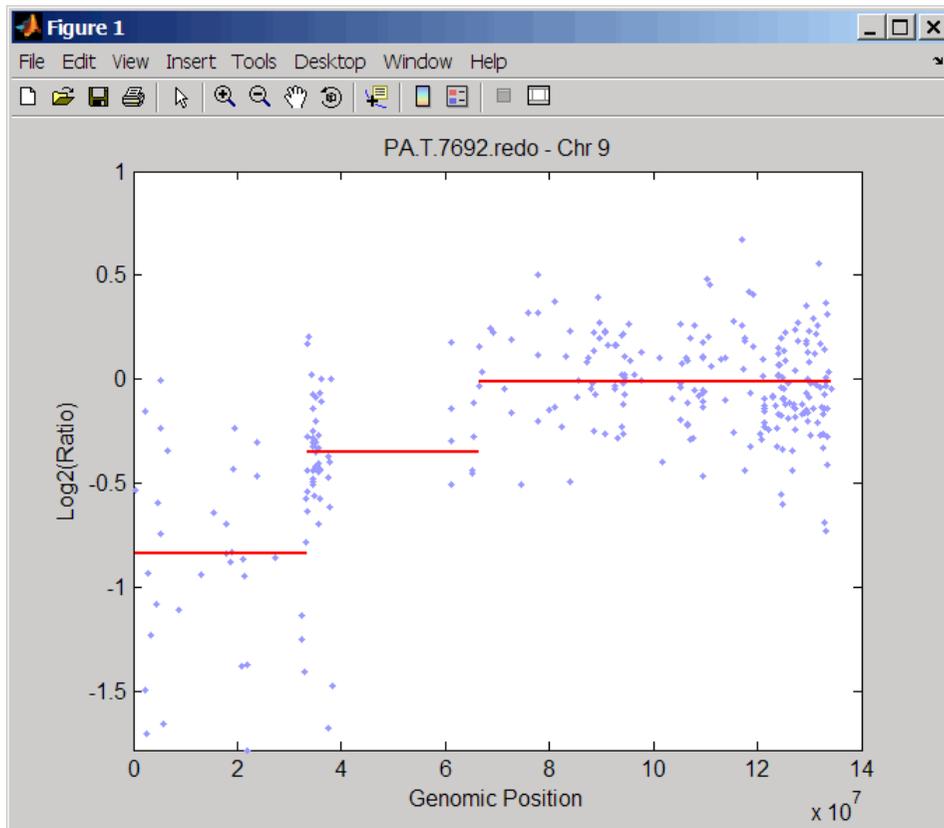

Chromosome 9 contains two segments that indicate losses. For more detailed information on interpreting the data, see Aguirre et al. (2004).

3 Use the `chromosomeplot` function with the `'addtoplot'` property to add the ideogram of chromosome 9 for *Homo sapiens* to the plot of the segmentation data.

```
chromosomeplot('hs_cytoBand.txt', 9, 'addtoplot', gca)
```
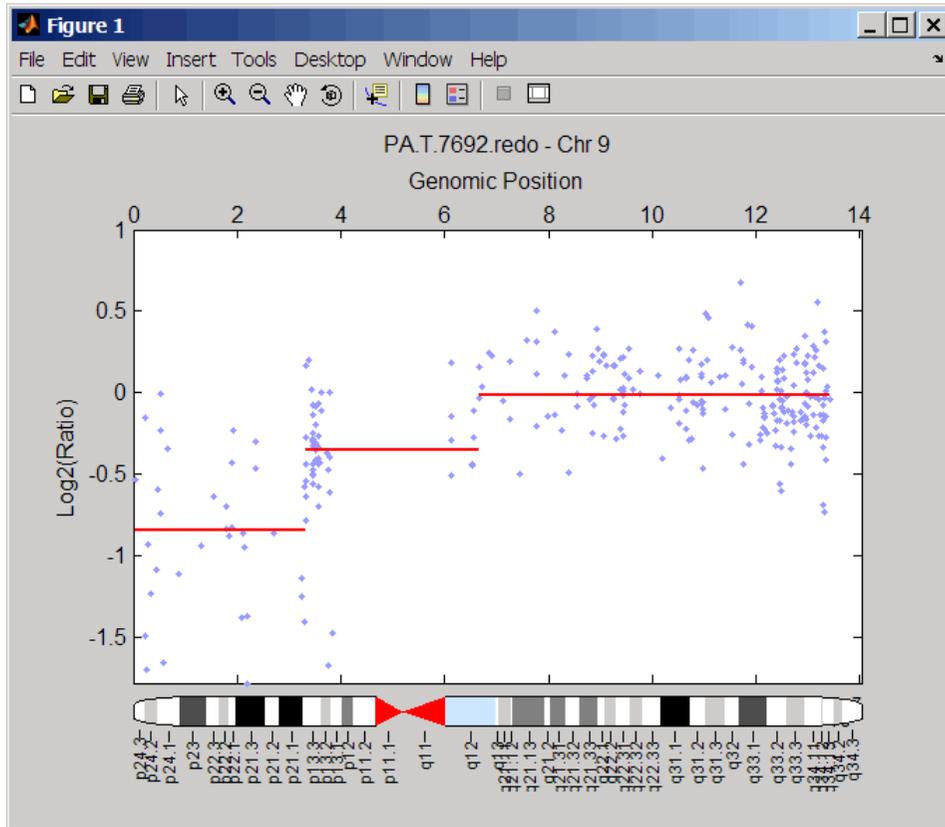
The `pancrea_oligocgh.mat` file used in this example contains data from Aguirre et al., 2004.

Displaying Copy Number Alteration Regions Aligned to a Chromosome Ideogram

1   Create a structure containing segment gain and loss information for chromosomes 10 and 11 from sample 3 from the Coriell cell line study, making sure the segment data is in bp units. (You can determine copy number variance (CNV) information by exploring S, the structure of segments returned by the `cghcbs` function in Analyzing Data from the Coriell Cell Line Study.) For the `'CNVType'` field, use 1 to indicate a loss and 2 to indicate a gain.

```
cnvStruct = struct('Chromosome', [10 11],...
```

```
'CNVType', [2 1],...
'Start', [S.SegmentData(10).Start(2),...
 S.SegmentData(11).Start(2)]*1000,...
'End',   [S.SegmentData(10).End(2),...
 S.SegmentData(11).End(2)]*1000)

cnvStruct =

    Chromosome: [10 11]
    CNVType: [2 1]
    Start: [66905000 35416000]
    End: [110412000 43357000]
```

2 Pass the structure to the chromosomeplot function using the 'CNV' property to display the copy number gains (green) and losses (red) aligned to the human chromosome ideogram. Specify kb units for the display of segment information in the data tip.

```
chromosomeplot('hs_cytoBand.txt', 'cnv', cnvStruct, 'unit', 2)
```
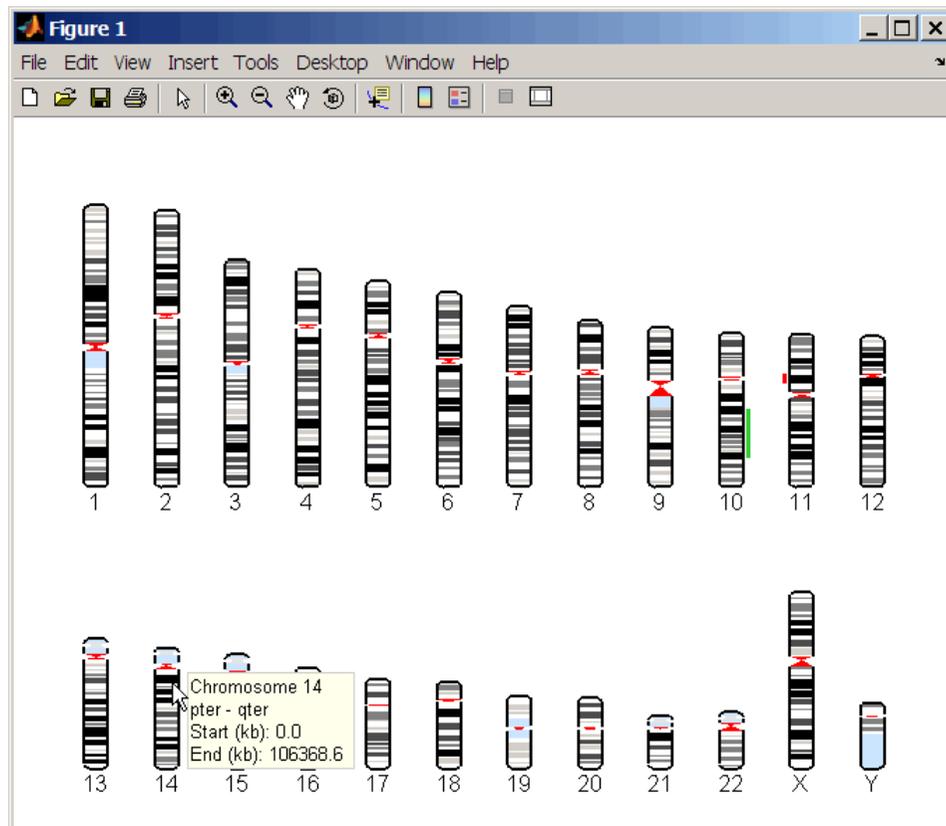
The `coriell_baccgh.mat` file used in this example contains data from Snijders et al., 2001.

References

[1] Olshen, A.B., Venkatraman, E.S., Lucito, R., and Wigler, M. (2004). Circular binary segmentation for the analysis of array-based DNA copy number data. Biostatistics *5, 4*, 557–572.

[2] Venkatraman, E.S., and Olshen, A.B. (2007). A Faster Circular Binary Segmentation Algorithm for the Analysis of Array CGH Data. Bioinformatics *23(6)*, 657–663.

[3] Venkatraman, E.S., and Olshen, A.B. (2006). DNAcopy: A Package for Analyzing DNA Copy Data. http://www.bioconductor.org/packages/2.1/bioc/html/DNAcopy.html

[4] Snijders, A.M., Nowak, N., Segraves, R., Blackwood, S., Brown, N., Conroy, J., Hamilton, G., Hindle, A.K., Huey, B., Kimura, K., Law, S., Myambo, K., Palmer, J., Ylstra, B., Yue, J.P., Gray, J.W., Jain, A.N., Pinkel, D., and Albertson, D.G. (2001). Assembly of microarrays for genome-wide measurement of DNA copy number. Nature Genetics *29*, 263–264.

[5] Aguirre, A.J., Brennan, C., Bailey, G., Sinha, R., Feng, B., Leo, C., Zhang, Y., Zhang, J., Gans, J.D., Bardeesy, N., Cauwels, C., Cordon-Cardo, C., Redston, M.S., DePinho, R.A., and Chin, L. (2004). High-resolution characterization of the pancreatic adenocarcinoma genome. PNAS *101, 24*, 9067–9072.

See Also
chromosomeplot | cytobandread

cghfreqplot

Display frequency of DNA copy number alterations across multiple samples

Syntax

FreqStruct = cghfreqplot(*CGHData*)
FreqStruct = cghfreqplot(*CGHData*, ...'Threshold',
ThresholdValue, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'Group', *GroupValue*, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'Subgrp', *SubgrpValue*, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'Subplot', *SubplotValue*, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'Cutoff', *CutoffValue*, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'Chromosome',
ChromosomeValue, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'IncludeX', *IncludeXValue*, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'IncludeY', *IncludeYValue*, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'Chrominfo',
ChrominfoValue, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'ShowCentr',
ShowCentrValue, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'Color', *ColorValue*, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'YLim', *YLimValue*, ...)
FreqStruct = cghfreqplot(*CGHData*, ...'Titles', *TitlesValue*, ...)

Input Arguments

CGHData	Array-based comparative genomic hybridization (aCGH) data in either of the following forms:
	• Structure with the following fields:
	• Sample — Cell array of strings containing the sample names (optional).
	• Chromosome — Vector containing the chromosome numbers on which the clones are located.

	• GenomicPosition — Vector containing the genomic positions (in bp, kb, or mb units) to which the clones are mapped. • Log2Ratio — Matrix containing $\log_2$ ratio of test to reference signal intensity for each clone. Each row corresponds to a clone, and each column corresponds to a sample. • Matrix in which each row corresponds to a clone. The first column contains the chromosome number, the second column contains the genomic position, and the remaining columns each contain the $\log_2$ ratio of test to reference signal intensity for a sample.
ThresholdValue	Positive scalar or vector that specifies the gain/loss threshold. A clone is considered to be a gain if its $\log_2$ ratio is above *ThresholdValue*, and a loss if its $\log_2$ ratio is below negative *ThresholdValue*. The *ThresholdValue* is applied as follows: • If a positive scalar, it is the gain and loss threshold for all the samples. • If a two-element vector, the first element is the gain threshold for all samples, and the second element is the loss threshold for all samples. • If a vector of the same length as the number of samples, each element in the vector is considered as a unique gain and loss threshold for each sample. Default is 0.25.

GroupValue	Specifies the sample groups to calculate the frequency from. Choices are: • A vector of sample column indices (for data with only one group). The samples specified in the vector are considered a group. • A cell array of vectors of sample column indices (for data divided into multiple groups). Each element in the cell array is considered a group. Default is a single group of all the samples in *CGHData*.
SubgrpValue	Controls the analysis of samples by subgroups. Choices are `true` (default) or `false`.
SubplotValue	Controls the display of all plots in one Figure window when more than one subgroup is analyzed. Choices are `true` (default) or `false` (displays plots in separate windows).
CutoffValue	Scalar or two-element numeric vector that specifies a cutoff, which controls the plotting of only the clones with frequency gains or losses greater than or equal to *CutoffValue*. If a two-element vector, the first element is the cutoff for gains, and the second element is for losses. Default is `0`.
ChromosomeValue	Single chromosome number or a vector of chromosome numbers that specify the chromosomes for which to display frequency plots. Default is all chromosomes in *CGHData*.
IncludeXValue	Controls the inclusion of the X chromosome in the analysis. Choices are `true` (default) or `false`.
IncludeYValue	Controls the inclusion of the Y chromosome in the analysis. Choices are `true` or `false` (default) .

ChrominfoValue	Cytogenetic banding information specified by either of the following: • Structure returned by the `cytobandread` function • String specifying the file name of an NCBI ideogram text file or a UCSC Genome Browser cytoband text file Default is *Homo sapiens* cytogenetic banding information from the UCSC Genome Browser, NCBI Build 36.1 (http://genome.UCSC.edu).
ShowCentrValue	Controls the display of the centromere positions as vertical dashed lines in the frequency plot. Choices are `true` (default) or `false`. **Tip** The centromere positions are obtained from *ChrominfoValue*.
ColorValue	Color scheme for the vertical lines in the plot, indicating the frequency of the gains and losses, specified by either of the following: • Name of or handle to a function that returns a colormap • M-by-3 matrix containing RGB values. If M equals 1, then that single color is used for all gains and losses. If M equals 2 or more, then the first row is used for gains, the second row is used for losses, and remaining rows are ignored. For example, `[0 1 0;1 0 0]` specifies green for gain and red for loss. The default color scheme is a range of colors from pure green (gain = 1) through yellow (0) to pure red (loss = −1).
YLimValue	Two-element vector specifying the minimum and maximum values on the vertical axis. Default is `[1, -1]`.
TitlesValue	Single string or a cell array of strings that specifies titles for the group(s), which are added to the tops of the plot(s).

Output Arguments

FreqStruct	Structure containing frequency data in the following fields:
	
	• Group — Structure array, with each structure representing a group of samples. Each structure contains the following fields:
	
	    • Sample — Cell array containing names of samples within the group.
	    • GainFrequency — Column vector containing the average gain for each clone for a group of samples.
	    • LossFrequency — Column vector containing the average loss for each clone for a group of samples.
	
	• Chromosome — Column vector containing the chromosome numbers on which the clones are located.
	
	• GenomicPosition — Column vector containing the genomic positions of the clones.
	
	Tip You can use this output structure as input to the cghfreqplot function.

Description

FreqStruct = cghfreqplot(*CGHData*) displays the frequency of copy number gain or loss across multiple samples for each clone on an array against their genomic position along the chromosomes.

FreqStruct = cghfreqplot(*CGHData*, ...'*PropertyName*', *PropertyValue*, ...) calls cghfreqplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

FreqStruct = cghfreqplot(*CGHData*, ...'Threshold', *ThresholdValue*, ...) specifies the gain/loss threshold. A clone is considered to be

a gain if its $\log_2$ ratio is above *ThresholdValue*, and a loss if its $\log_2$ ratio is below negative *ThresholdValue*.

The *ThresholdValue* is applied as follows:

- If a positive scalar, it is the gain and loss threshold for all the samples.
- If a two-element vector, the first element is the gain threshold for all samples, and the second element is the loss threshold for all samples.
- If a vector of the same length as the number of samples, each element in the vector is considered as a unique gain and loss threshold for each sample.

Default is `0.25`.

FreqStruct = cghfreqplot(*CGHData*, ...'Group', *GroupValue*, ...) specifies the sample groups to calculate the frequency from. Choices are:

- A vector of sample column indices (for data with only one group). The samples specified in the vector are considered a group.
- A cell array of vectors of sample column indices (for data divided into multiple groups). Each element in the cell array is considered a group.

Default is a single group of all the samples in *CGHData*.

FreqStruct = cghfreqplot(*CGHData*, ...'Subgrp', *SubgrpValue*, ...) controls the analysis of samples by subgroups. Choices are `true` (default) or `false`.

FreqStruct = cghfreqplot(*CGHData*, ...'Subplot', *SubplotValue*, ...) controls the display of all plots in one Figure window when more than one subgroup is analyzed. Choices are `true` (default) or `false` (displays plots in separate windows).

FreqStruct = cghfreqplot(*CGHData*, ...'Cutoff', *CutoffValue*, ...) specifies a cutoff value, which controls the plotting of only the clones with frequency gains or losses greater than or equal to *CutoffValue*. *CutoffValue* is a scalar or two-element numeric vector. If a two-element numeric vector, the first element is the cutoff for gains, and the second element is for losses. Default is `0`.

FreqStruct = cghfreqplot(*CGHData*, ...'Chromosome', *ChromosomeValue*, ...) displays the frequency plots only of chromosome(s) specified by *ChromosomeValue*, which can be a single chromosome number or a vector of chromosome numbers. Default is all chromosomes in *CGHData*.

FreqStruct = cghfreqplot(*CGHData*, ...'IncludeX', *IncludeXValue*, ...) controls the inclusion of the X chromosome in the analysis. Choices are true (default) or false.

FreqStruct = cghfreqplot(*CGHData*, ...'IncludeY', *IncludeYValue*, ...) controls the inclusion of the Y chromosome in the analysis. Choices are true or false (default).

FreqStruct = cghfreqplot(*CGHData*, ...'Chrominfo', *ChrominfoValue*, ...) specifies the cytogenetic banding information for the chromosomes. *ChrominfoValue* can be either of the following

- Structure returned by the cytobandread function
- String specifying the file name of an NCBI ideogram text file or a UCSC Genome Browser cytoband text file

Default is *Homo sapiens* cytogenetic banding information from the UCSC Genome Browser, NCBI Build 36.1 (http://genome.UCSC.edu).

Tip You can download files containing cytogenetic G-banding data from the NCBI or UCSC Genome Browser ftp site. For example, you can download the cytogenetic banding data for *Homo sapiens* from:

ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/mapview/ideogram.gz
or

ftp://hgdownload.cse.ucsc.edu/goldenPath/hg18/database/cytoBandIdeo.txt.gz

---

FreqStruct = cghfreqplot(*CGHData*, ...'ShowCentr', *ShowCentrValue*, ...) controls the display of the centromere positions as vertical dashed lines in the frequency plot. Choices are true (default) or false.

Tip The centromere positions are obtained from *ChrominfoValue*.

FreqStruct = cghfreqplot(*CGHData*, ...'Color', *ColorValue*, ...) specifies a color scheme for the vertical lines in the plot, indicating the frequency of the gains and losses. Choices are:

- Name of or handle to a function that returns a colormap.

- M-by-3 matrix containing RGB values. If M equals 1, then that single color is used for all gains and losses. If M equals 2 or more, then the first row is used for gains, the second row is used for losses, and remaining rows are ignored. For example, [0 1 0;1 0 0] specifies green for gain and red for loss.

The default color scheme is a range of colors from pure green (gain = 1) through yellow (0) to pure red (loss = –1).

FreqStruct = cghfreqplot(*CGHData*, ...'YLim', *YLimValue*, ...) specifies the y vertical limits for the frequency plot. *YLimValue* is a two-element vector specifying the minimum and maximum values on the vertical axis. Default is [1, -1].

FreqStruct = cghfreqplot(*CGHData*, ...'Titles', *TitlesValue*, ...) specifies titles for the group(s), which are added to the tops of the plot(s). *TitlesValue* can be a single string or a cell array of strings.

Examples

Plotting Data from the Coriell Cell Line Study

1   Load a MAT-file, included with the Bioinformatics Toolbox software, which contains coriell_data, a structure of array-based CGH data.

    ```
    load coriell_baccgh
    ```

2   Display a frequency plot of the copy number alterations across all samples in the Coriell aCGH data.

    ```
    Struct = cghfreqplot(coriell_data);
    ```
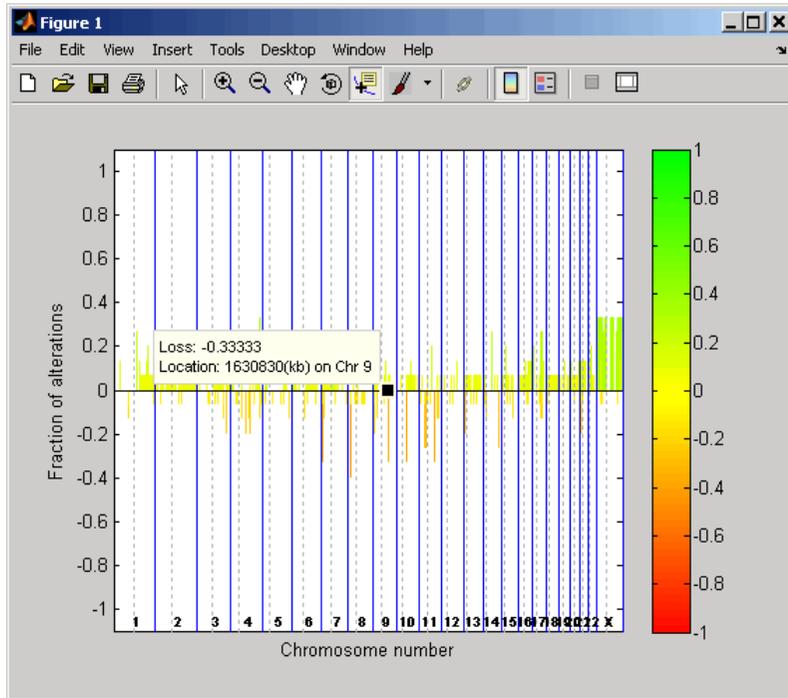
3. View data tips for the data, chromosomes, and centromeres by clicking the Data

   Cursor ![icon] button on the toolbar, then clicking data, a blue chromosome boundary line, or a dotted centromere line in the plot. To delete this data tip, right-click it, then select **Delete Current Datatip**.

4. Display a color bar indicating the degree of gain or loss by clicking the Insert

   Colorbar ![icon] button on the toolbar.

The `coriell_baccgh.mat` file used in this example contains data from Snijders et al., 2001.

Plotting Pancreatic Cancer Study Data Using a Green and Red Color Scheme

1. Load a MAT-file, included with the Bioinformatics Toolbox software, which contains `pancrea_data`, a structure of array-based CGH data from a pancreatic cancer study.

```
load pancrea_oligocgh
```

2 Display a frequency plot of the copy number alterations across all samples in the pancreatic cancer data, using a green and red color scheme.

```
cghfreqplot(pancrea_data, 'Color', [0 1 0; 1 0 0])
```
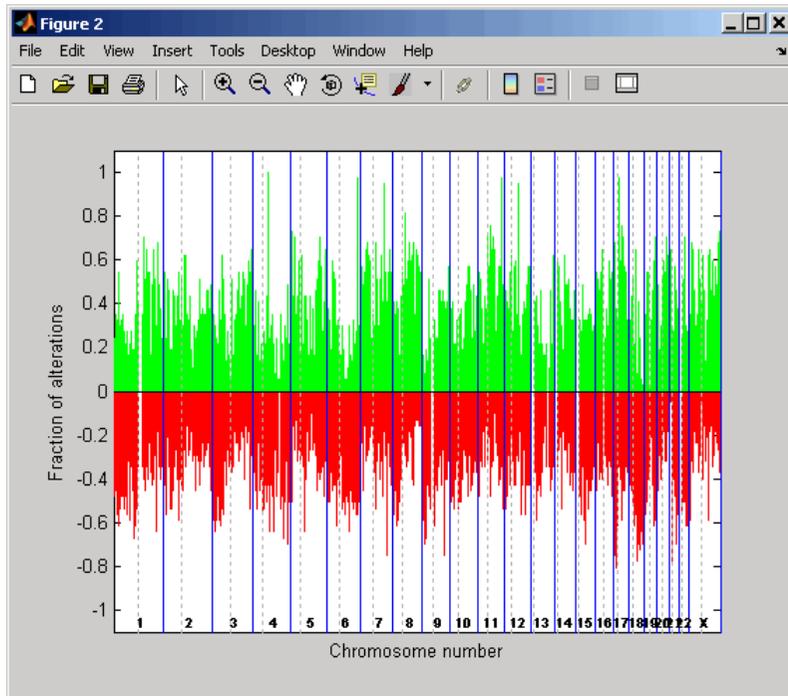

The `pancrea_oligocgh.mat` file used in this example contains data from Aguirre et al., 2004.

Plotting Groups of aCGH Data, Specifying a Frequency Value Cutoff, and Adding a Chromosome Ideogram

1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains `pancrea_data`, a structure of array-based CGH data from a pancreatic cancer study.

```
load pancrea_oligocgh
```

2   Define two groups of data.

```
grp1 = strncmp('PA.C', pancrea_data.Sample,4);
grp1_ind = find(grp1);
grp2 = strncmp('PA.T', pancrea_data.Sample,4);
grp2_ind = find(grp2);
```

3   Display a frequency plot of the copy number alterations across all samples in the two groups and limit the plotting to only the clones with frequency gains or losses greater than or equal to 0.25.

```
SP = cghfreqplot(pancrea_data, 'Group', {grp1_ind, grp2_ind},...
                'Title', {'CL', 'PT'}, 'Cutoff', 0.25);
```
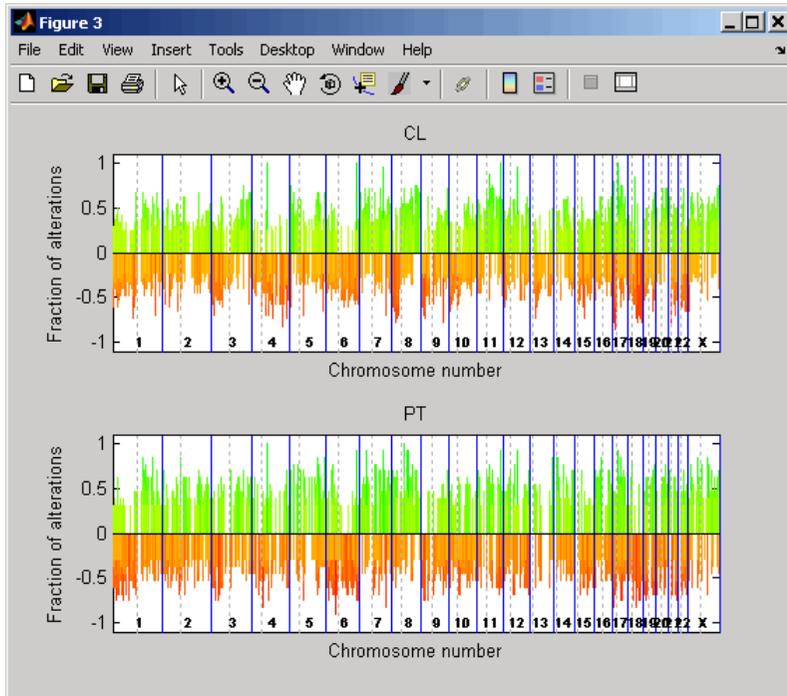

4   Display a frequency plot of the copy number alterations across all samples in the first group and limit the plot to chromosome 4 only.

```
SP = cghfreqplot(pancrea_data, 'Group', grp1_ind, ...
                'Title', 'CL Group on Chr 4', 'Chromosome', 4);
```
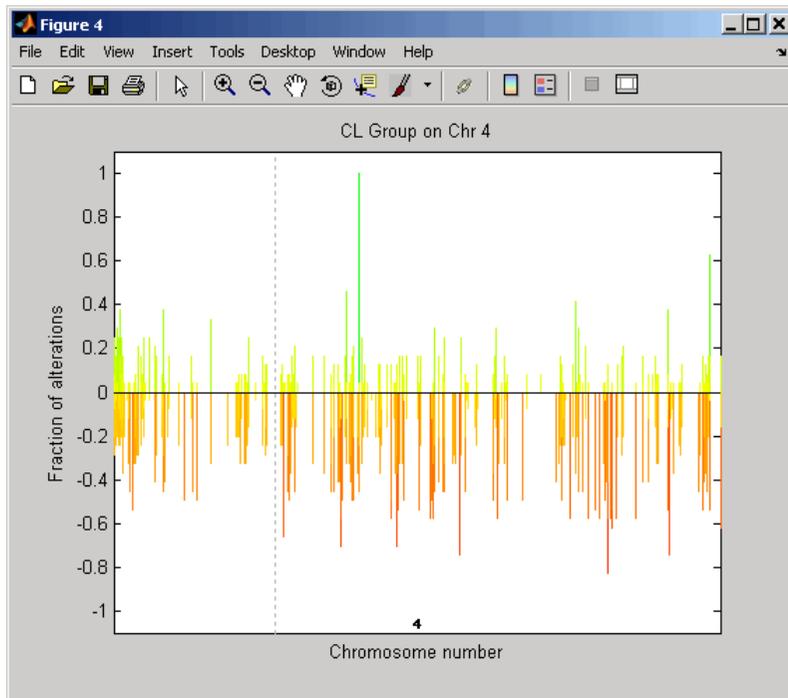
5 Use the `chromosomeplot` function with the `'addtoplot'` property to add the ideogram of chromosome 4 for *Homo sapiens* to this frequency plot. Because the plot of the frequency data from the pancreatic cancer study is in kb units, use the `'Unit'` property to convert the ideogram data to kb units.

```
chromosomeplot('hs_cytoBand.txt', 4, 'addtoplot', gca, 'unit', 2);
```
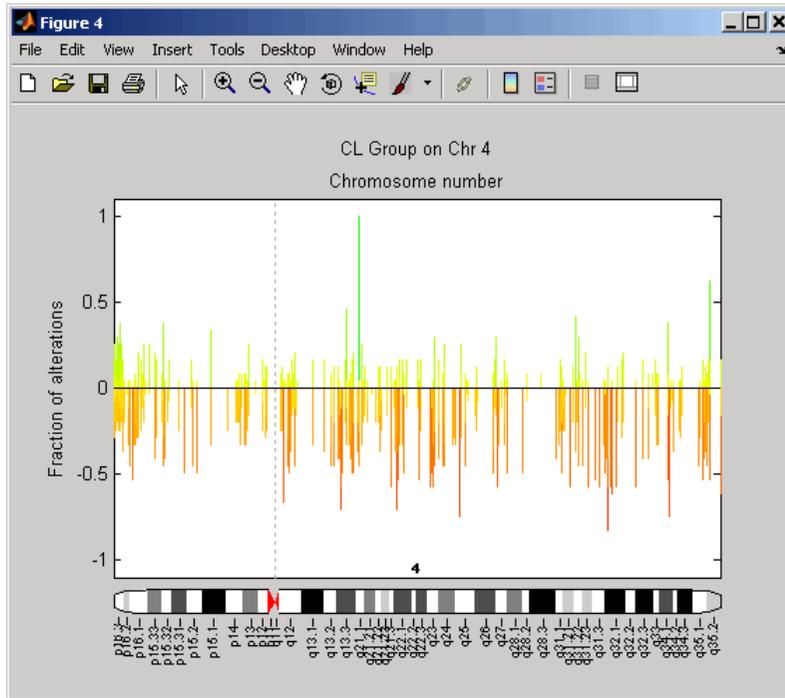
The `pancrea_oligocgh.mat` file used in this example contains data from Aguirre et al., 2004.

References

[1] Snijders, A.M., Nowak, N., Segraves, R., Blackwood, S., Brown, N., Conroy, J., Hamilton, G., Hindle, A.K., Huey, B., Kimura, K., Law, S., Myambo, K., Palmer, J., Ylstra, B., Yue, J.P., Gray, J.W., Jain, A.N., Pinkel, D., and Albertson, D.G. (2001). Assembly of microarrays for genome-wide measurement of DNA copy number. Nature Genetics *29*, 263–264.

[2] Aguirre, A.J., Brennan, C., Bailey, G., Sinha, R., Feng, B., Leo, C., Zhang, Y., Zhang, J., Gans, J.D., Bardeesy, N., Cauwels, C., Cordon-Cardo, C., Redston, M.S., DePinho, R.A., and Chin, L. (2004). High-resolution characterization of the pancreatic adenocarcinoma genome. PNAS *101, 24*, 9067–9072.

See Also

cghcbs | chromosomeplot | cytobandread

chromosomeplot

Plot chromosome ideogram with G-banding pattern

Syntax

```
chromosomeplot(CytoData)
chromosomeplot(CytoData, ChromNum)
chromosomeplot(CytoData, ChromNum, ...,'Orientation',
OrientationValue, ...)
chromosomeplot(CytoData, ChromNum, ...,'ShowBandLabel',
ShowBandLabelValue, ...)
chromosomeplot(CytoData, ChromNum, ...,'AddToPlot',
AddToPlotValue, ...)
chromosomeplot(..., 'Unit', UnitValue, ...)
chromosomeplot(..., 'CNV', CNVValue, ...)
```

Arguments

CytoData	Either of the following:
	
	• String specifying a file containing cytogenetic G-banding data (in bp units), such as an NCBI ideogram text file or a UCSC Genome Browser cytoband text file.
	
	• Structure containing cytogenetic G-banding data (in bp units) in the following fields:
	
	   • ChromLabels
	   • BandStartBPs
	   • BandEndBPs
	   • BandLabels
	   • GieStains
	
	Tip Use the cytobandread function to create the structure to use for *CytoData*.

ChromNum	Scalar or string specifying a single chromosome to plot. Valid entries are integers, `'X'`, and `'Y'`.
	Note: Setting *ChromNum* to `0` will plot ideograms for all chromosomes.
OrientationValue	String or number that specifies the orientation of the ideogram of a single chromosome specified by *ChromNum*. Choices are `'Vertical'` or `1` (default) and `'Horizontal'` or `2`.
ShowBandLabelValue	Controls the display of band labels (such as `q25.3`) when plotting a single chromosome ideogram, specified by *ChromNum*. Choices are `true` (default) or `false`.
AddToPlotValue	Variable name of a figure axis to which to add the single chromosome ideogram, specified by *ChromNum*.
	Note: If you use this property to add the ideogram to a plot of genomic data that is in units other than bp, use the `'Unit'` property to convert the ideogram data to the appropriate units.
	Tip Before printing a figure containing an added chromosome ideogram, change the background to white by issuing the following command: `set(gcf,'color','w')`
UnitValue	Integer that specifies the units (base pairs, kilo base pairs, or mega base pairs) for the starting and ending genomic positions. This unit is used in the data tip displayed when you hover the cursor over chromosomes in the ideogram. This unit can also be used when using the `'AddToPlot'` property to add the ideogram to a plot that is in units other than bp. Choices are `1` (bp), `2` (kb), or `3` (mb). Default is `1` (bp).

CNVValue	Controls the display of copy number variance (CNV) data, provided by *CNVValue*, aligned to the chromosome ideogram. Gains are shown in green to the right or above the ideogram, while losses are shown in red to the left or below the ideogram. *CNVValue* is a structure array containing the four fields described in the table below.

Description

chromosomeplot(*CytoData*) plots the ideogram of all chromosomes, using information from *CytoData*, a structure containing cytogenetic G-banding data (in bp units), or a string specifying a file containing cytogenetic G-banding data (in bp units), such as an NCBI ideogram text file or a UCSC Genome Browser cytoband text file. The G bands distinguish different areas of the chromosome. For example, for the *Homo sapiens* ideogram, possible G bands are:

- gneg — white
- gpos25 — light gray
- gpos50 — medium gray
- gpos75 — dark gray
- gpos100 — black
- acen — red (centromere)
- stalk — light blue (regions with repeats)
- gvar — indented region

Darker bands are AT-rich, while lighter bands are GC-rich.

chromosomeplot(*CytoData*, *ChromNum*) plots the ideogram of a single chromosome specified by *ChromNum*.

chromosomeplot(..., '*PropertyName*', *PropertyValue*, ...) calls chromosomeplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

chromosomeplot(*CytoData*, *ChromNum*, ...,'Orientation', *OrientationValue*, ...) specifies the orientation of the ideogram of a single

chromosome specified by *ChromNum*. Choices are `'Vertical'` or `1` (default) and `'Horizontal'` or `2`.

Note: When plotting the ideogram of all chromosomes, the orientation is always vertical.

---

chromosomeplot(*CytoData*, *ChromNum*, ...,`'ShowBandLabel'`, *ShowBandLabelValue*, ...) displays band labels (such as q25.3) when plotting a single chromosome ideogram, specified by *ChromNum*. Choices are `true` (default) or `false`.

chromosomeplot(*CytoData*, *ChromNum*, ...,`'AddToPlot'`, *AddToPlotValue*, ...) adds the single chromosome ideogram, specified by *ChromNum*, to a figure axis specified by *AddToPlotValue*.

Note: If you use this property to add the ideogram to a plot of genomic data that is in units other than bp, use the `'Unit'` property to convert the ideogram data to the appropriate units.

Tip Before printing a figure containing an added chromosome ideogram, change the background to white by issuing the following command:

```
set(gcf,'color','w')
```

---

chromosomeplot(..., `'Unit'`, *UnitValue*, ...) specifies the units (base pairs, kilo base pairs, or mega base pairs) for the starting and ending genomic positions. This unit is used in the data tip displayed when you hover the cursor over chromosomes in the ideogram. This unit can also be used when using the `'AddToPlot'` property to add the ideogram to a plot that is in units other than bp. Choices are `1` (bp), `2` (kb), or `3` (mb). Default is `1` (bp).

chromosomeplot(..., `'CNV'`, *CNVValue*, ...) displays copy number variance (CNV) data, provided by *CNVValue*, aligned to the chromosome ideogram. Gains are shown in green to the right or above the ideogram, while losses are shown in red to the left or below the ideogram. *CNVValue* is a structure array containing the following fields. Each field must contain the same number of elements.

Field	Description

Chromosome	Either of the following:
	
	 • Numeric vector containing the chromosome number on which each CNV is located.
	
	 Note: For the sex chromosome, X, use *N*, where *N* = number of autosomes + 1. For the sex chromosome, Y, use *M*, where *M* = number of autosomes + 2. For example, for *Homo sapiens* use 23 for X and 24 for Y, and for *Mus musculus* (lab mouse), use 20 for X and 21 for Y.
	
	 • Character array containing the chromosome number on which each CNV is located.
	
	 Note: Using a character array lets you use the characters X and Y (instead of numbers) for sex chromosomes. However, all elements in the array must be the same width, which may require you to add spaces to the strings. For example:
	
	 `[' 1'; ' 2'; '10'; ' X']`
	
	 Or you can use the `char` function with a cell array to create a character array of the chromosome numbers and letters. For example:
	
	 `char({'1', '2', '10', 'X'})`

CNVType	Numeric vector containing the type of each CNV, either 1 (loss) or 2 (gain).
Start	Numeric vector containing the starting genomic position of each CNV. Units must be in base pairs.
End	Numeric vector containing the ending genomic position of each CNV. Units must be in base pairs.

Examples

Plotting Chromosome Ideograms

1 Read the cytogenetic banding information for *Homo sapiens* into a structure.

```
hs_cytobands = cytobandread('hs_cytoBand.txt')

hs_cytobands =

     ChromLabels: {862x1 cell}
    BandStartBPs: [862x1 int32]
      BandEndBPs: [862x1 int32]
      BandLabels: {862x1 cell}
       GieStains: {862x1 cell}
```

2 Plot the entire chromosome ideogram for *Homo sapiens*.

```
chromosomeplot(hs_cytobands);
title('Human Karyogram')
```
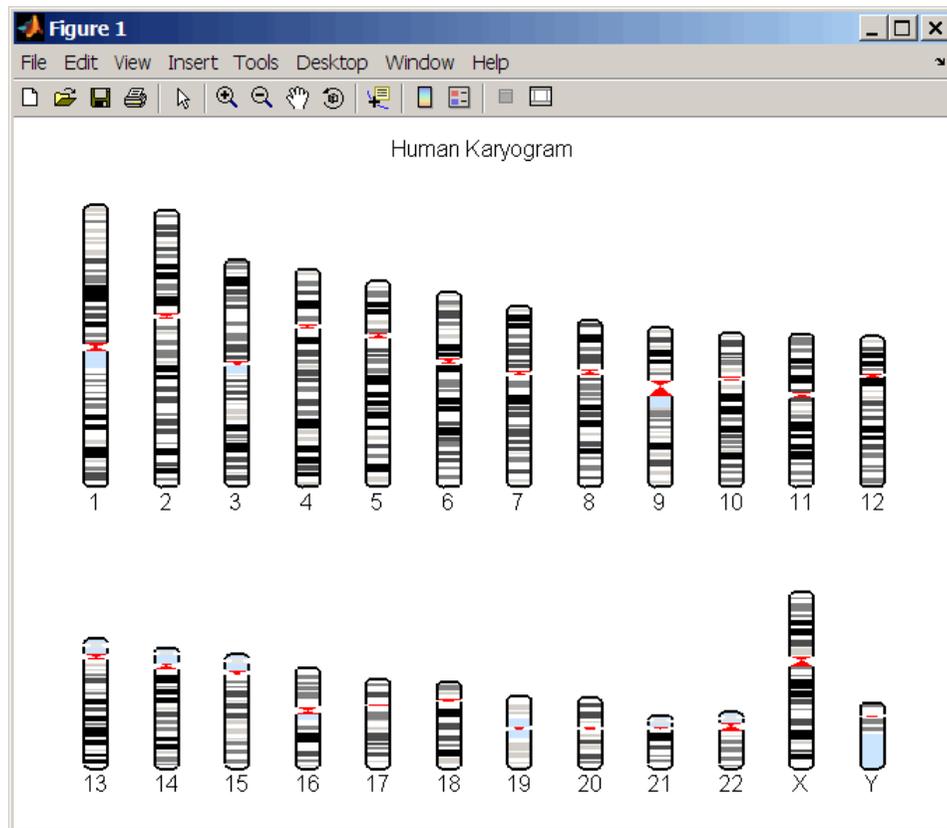
3 Display the ideogram of only chromosome 7 for *Homo sapiens* by right-clicking chromosome 7 in the plot, then selecting **Display in New Figure > Vertical**.
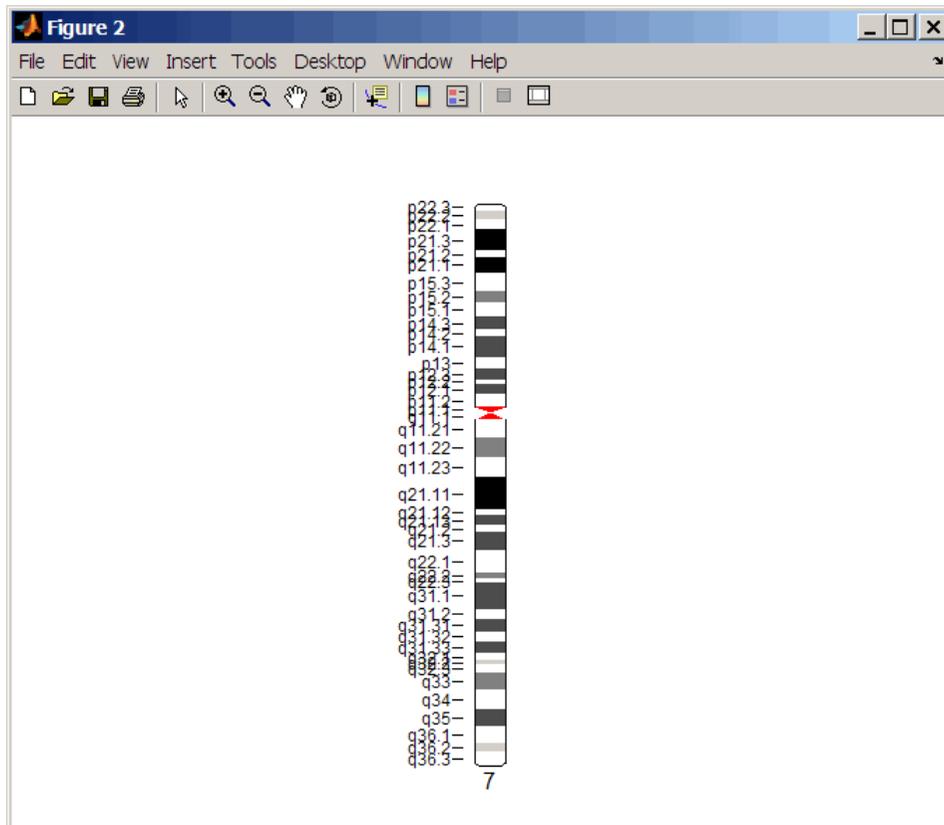
4 Plot the ideogram of only chromosome 15 for *Homo sapiens* in a horizontal orientation. Set the units used in the data tip to kilo base pairs.

```
chromosomeplot(hs_cytobands, 15, 'Orientation', 2, 'Unit', 2);
```
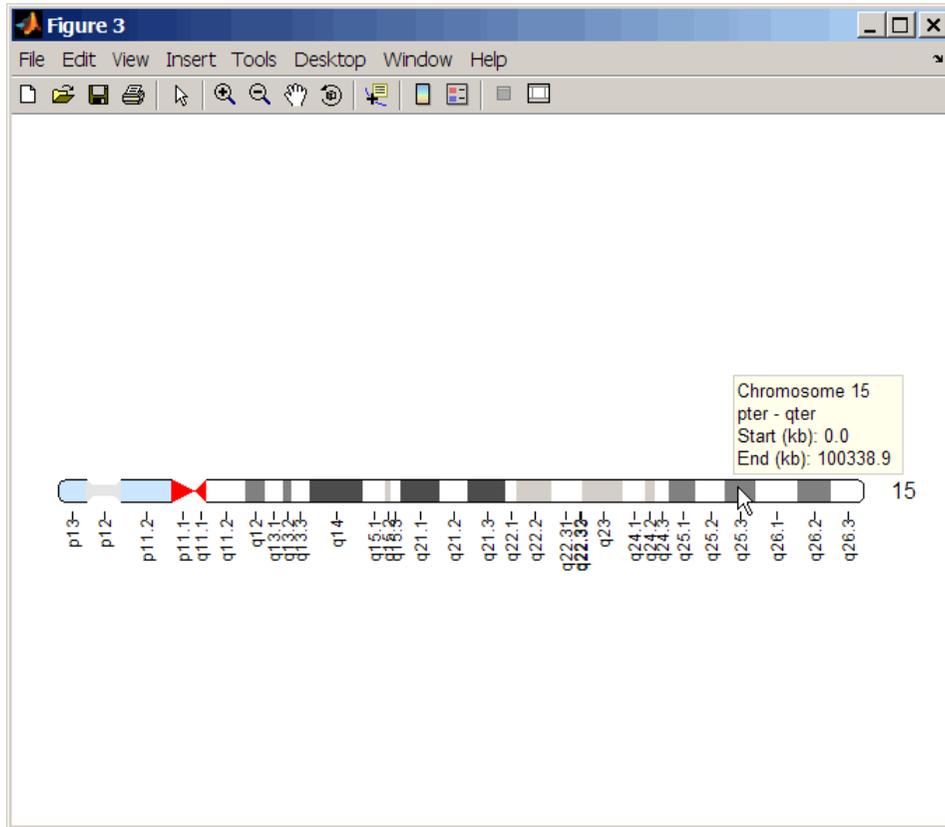
5 View a data tip with information about the chromosome by hovering the cursor over the chromosome. View a data tip with detailed information about a specific band by clicking the Data Cursor ⬚ button on the toolbar, then clicking the band in the plot. To delete this data tip, right-click it, then select **Delete Current Datatip**.

Tip You can change the orientation of a single chromosome ideogram by right-clicking, selecting **Display > Vertical** or **Horizontal**. You can show or hide the band labels of a single chromosome ideogram by right-clicking, then selecting **Show G-band Labels** or **Hide G-band Labels**.

Adding a Chromosome Ideogram to a Plot

1   Load a MAT-file, included with the Bioinformatics Toolbox software, which contains `coriell_data`, a structure of CGH data.

```
load coriell_baccgh
```

2   Use the `cghcbs` function to analyze chromosome 10 of sample 3 (GM05296) of the CGH data and return copy number variance (CNV) data in a structure, `S`. Plot the segment means over the original data for only chromosome 10 of sample 3.

```
S = cghcbs(coriell_data,'sampleindex',3,'chromosome',10,...
           'showplot',10);
```
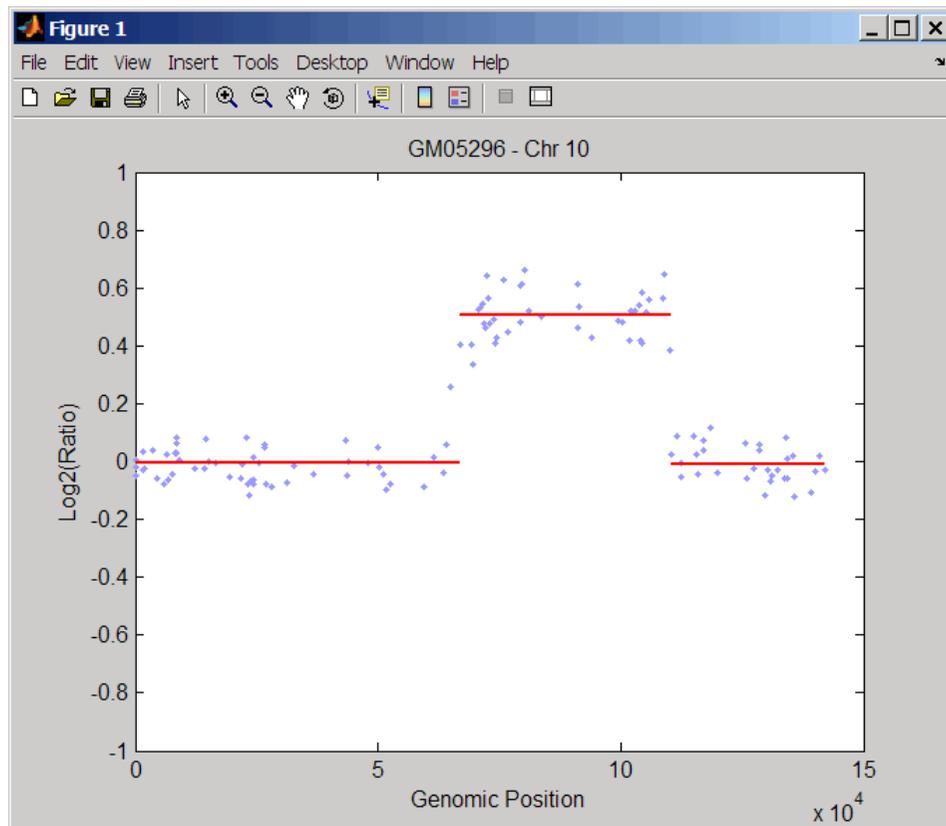
3 Use the `chromosomeplot` function with the `'addtoplot'` property to add the ideogram of chromosome 10 for *Homo sapiens* to the plot. Because the plot of the CNV data from the Coriell cell line study is in kb units, use the `'Unit'` property to convert the ideogram data to kb units.

```
chromosomeplot('hs_cytoBand.txt', 10, 'addtoplot', gca,...
               'Unit', 2)
```
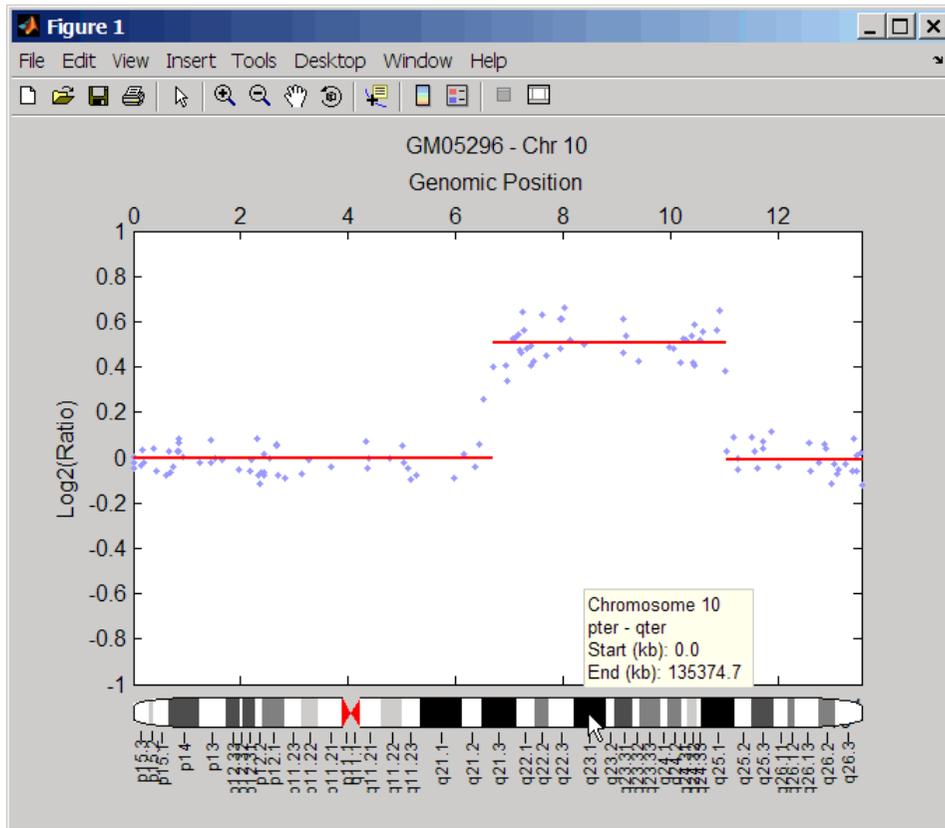

Tip Before printing the above figure containing an added chromosome ideogram, change the background to white by issuing the following command:

```
set(gcf,'color','w')
```

Displaying Copy Number Alteration Regions Aligned to a Chromosome Ideogram

1 Create a structure containing segment gain and loss information for chromosomes 10, 11, and X from sample 3 from the Coriell cell line study, making sure the segment data is in bp units. (You can determine copy number variance (CNV) information by exploring S, the structure of segments returned by the cghcbs function in step 2 in Adding a Chromosome Ideogram to a Plot.) For the 'CNVType' field, use 1 to indicate a loss and 2 to indicate a gain.

```
cnvStruct = struct('Chromosome', char({'10', '11', 'X'}),...
                   'CNVType', [2 1 2],...
                   'Start', [66905000 25416000 1],...
                   'End',   [110412000 39389000 154913755]);
```

2 Pass the structure to the chromosomeplot function using the 'CNV' property to display the copy number gains (green) and losses (red) aligned to the human chromosome ideogram.

```
chromosomeplot('hs_cytoBand.txt', 'cnv', cnvStruct);
```
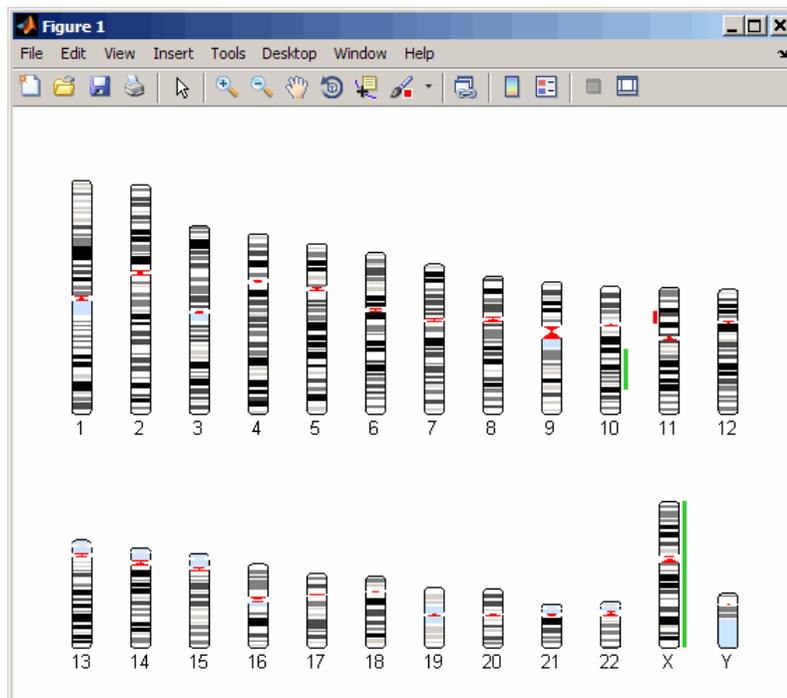
The `coriell_baccgh.mat` file used in this example contains data from Snijders et al., 2001.

References

[1] Snijders, A.M., Nowak, N., Segraves, R., Blackwood, S., Brown, N., Conroy, J., Hamilton, G., Hindle, A.K., Huey, B., Kimura, K., Law, S., Myambo, K., Palmer, J., Ylstra, B., Yue, J.P., Gray, J.W., Jain, A.N., Pinkel, D., and Albertson, D.G. (2001). Assembly of microarrays for genome-wide measurement of DNA copy number. Nature Genetics *29*, 263–264.

See Also
cghcbs | cytobandread

cigar2align

Convert unaligned sequences to aligned sequences using Compact Idiosyncratic Gapped Alignment Report (CIGAR) format strings

Syntax

```
Alignment = cigar2align(Seqs,Cigars)
[GapSeq, Indices] = cigar2align(Seqs,Cigars)
... = cigar2align(Seqs,Cigars,Name,Value)
```

Description

`Alignment = cigar2align(Seqs,Cigars)` converts unaligned sequences in Seqs, a cell array of strings, into Alignment, a matrix of aligned sequences, using the information stored in Cigars, a cell array of CIGAR strings.

`[GapSeq, Indices] = cigar2align(Seqs,Cigars)` converts unaligned sequences in Seqs, a cell array of strings, into GapSeq, a cell array of strings of aligned sequences, and also returns Indices, a vector of numeric indices, using the information stored in Cigars, a cell array of CIGAR strings. When an alignment has many columns, this syntax uses less memory and is faster.

`... = cigar2align(Seqs,Cigars,Name,Value)` converts unaligned sequences in Seqs, a cell array of strings, into Alignment, a matrix of aligned sequences, using the information stored in Cigars, a cell array of CIGAR strings, with additional options specified by one or more Name,Value pair arguments.

Input Arguments

Seqs

Cell array of strings containing unaligned sequences. Seqs must contain the same number of elements as Cigars.

Default:

`Cigars`

Cell array of valid CIGAR strings. Cigars must contain the same number of elements as Seqs.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`'Start'`

Vector of positive integers specifying the reference sequence position at which each aligned sequence starts. By default, each aligned sequence starts at position 1 of the reference sequence.

Default:

`'GapsInRef'`

Logical specifying whether to display positions in the aligned sequences that correspond to gaps in the reference sequence. Choices are `true` (1) or `false` (0). If your reference sequence has gaps and you set GapsInRef to `false` (0), and then later use Alignment as input to `align2cigar`, the returned CIGAR strings will not match the original CIGAR strings.

Default: `false` (0)

`'SoftClipping'`

Logical specifying whether to include characters in the aligned read sequences corresponding to soft clipping ends. Choices are `true` (1) or `false` (0).

Default: `false` (0)

`'OffsetPad'`

Logical specifying whether to add padding blanks to the left of each aligned read sequence to represent the offset of the start position from the first position of the

reference sequence. Choices are `true` (1) or `false` (0). When `false`, the matrix of aligned sequences starts at the start position of the leftmost aligned read sequence.

Default: `false` (0)

Output Arguments

`Alignment`

Matrix of aligned sequences, in which the number of rows equals the number of strings in Seqs.

`GapSeq`

Cell array of strings of aligned sequences, in which the number strings equals the number of strings in Seqs.

`Indices`

Vector of numeric indices indicating the starting column for each aligned sequence in Alignment. These indices are not necessarily the same as the start positions in the reference sequence for each aligned sequence. This is because either of the following:

- The reference sequence can be extended to account for insertions.
- An aligned sequence can have leading soft clippings, padding, or insertion characters.

Examples

Create a cell array of strings containing unaligned sequences, create a cell array of strings containing corresponding CIGAR strings associated with a reference sequence of `ACGTATGC`, and then reconstruct the alignment:

```
r = {'ACGACTGC', 'ACGTTGC', 'AGGTATC'}; % unaligned sequences
c = {'3M1D1M1I3M', '4M1D1P3M', '5M1P1M1D1M'}; % cigar strings
aln1 = cigar2align(r, c)

aln1 =

ACG-ATGC
ACGT-TGC
```

```
AGGTAT-C
```

Reconstruct the same alignment to display positions in the aligned sequences that correspond to gaps in the reference sequence:

```
aln2 = cigar2align(r, c,'GapsInRef',true)

aln2 =

ACG-ACTGC
ACGT--TGC
AGGTA-T-C
```

Reconstruct the alignment adding an offset padding of 5:

```
aln3 = cigar2align(r, c, 'start', [5 5 5], 'OffsetPad', true)

aln3 =

    ACG-ATGC
    ACGT-TGC
    AGGTAT-C
```

Alternatives

If your CIGAR information is captured in the `Signature` property of a `BioMap` object, you can use the `getAlignment` method to construct the alignment.

More About

Algorithms

When `cigar2align` reconstructs the alignment, it does not display hard clipped positions (H) or soft clipped positions (S). Also, it does not consider soft clipped positions as start positions for aligned sequences.

- "Manage Short-Read Sequence Data in Objects"
- Sequence Read Archive
- SAM format specification

References

[1] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Goncalo, A., and Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics *25, 16*, 2078–2079.

See Also

getAlignment | align2cigar | seqalignviewer | getBaseCoverage | getCompactAlignment | BioMap

classperf

Evaluate performance of classifier

Syntax

```
classperf
CP = classperf(truelabels)
CP = classperf(truelabels, classout)
CP = classperf(..., 'Positive', PositiveValue, 'Negative',
NegativeValue)
classperf(CP, classout)
classperf(CP, classout, testidx)
```

Input Arguments

truelabels	True class labels for each observation, specified by one of the following: • Numeric vector • Cell array of strings **Note:** When used in a cross-validation design experiment, *truelabels* should have the same size as the total number of observations.
classout	Classifier output, specified by one of the following: • Numeric vector • Cell array of strings **Note:** *classout* must contain the same number of elements as *truelabels*.
PositiveValue	Numeric vector or cell array of strings that specifies the positive labels to identify the target class(es). Default is the first class returned by grp2idx(*truelabels*).

NegativeValue	Numeric vector or cell array of strings that specifies the negative labels to identify the control class(es). Default is all classes other than the first class returned by grp2idx(*truelabels*).
testidx	Vector that indicates the observations that were used in the current validation. Choices are: • Index vector • Logical index vector of the same size as *truelabels* used to construct the classifier performance object

Output Arguments

CP	Classifier performance object with performance properties listed in the following table.

Description

classperf provides an interface to keep track of the performance during the validation of classifiers. classperf creates and, optionally, updates a classifier performance object, *CP*, which accumulates the results of the classifier. The performance properties of a classifier performance object are listed in the following table.

classperf, without input arguments, displays all the performance properties of a classifier performance object.

CP = classperf(*truelabels*) creates and initializes an empty classifier performance object. *CP* is the handle to the object. *truelabels* is a vector or cell array of strings containing the true class labels for every observation. When used in a cross-validation design experiment, *truelabels* must have the same size as the total number of observations.

CP = classperf(*truelabels*, *classout*) creates *CP* using *truelabels*, then updates *CP* using the classifier output, *classout*.

Tip This syntax is useful when you want to know the performance of a single validation.

CP = classperf(..., 'Positive', *PositiveValue*, 'Negative', *NegativeValue*) specifies the positive and negative labels to identify the target and the control classes, respectively. These labels are used to compute clinical diagnostic test performance.

If *truelabels* is a numeric vector, *PositiveValue* and *NegativeValue* must be numeric vectors whose entries are subsets of grp2idx(*truelabels*). If *truelabels* is a cell array of strings, *PositiveValue* and *NegativeValue* can be cell arrays of strings or numeric vectors whose entries are subsets of grp2idx(*truelabels*). *PositiveValue* defaults to the first class returned by grp2idx(*truelabels*), while *NegativeValue* defaults to all other classes.

PositiveValue and *NegativeValue* must consist of disjoint sets of the labels used in *truelabels*. For example, if

```
truelabels = [1 2 2 1 3 4 4 1 3 3 3 2]
```

you could set

```
p = [1 2];
n = [3 4];
```

For example, if you have a data set with data from six samples: five different types of cancer (ovarian, lung, prostate, skin, brain) and no cancer, then ClassLabels = {'Ovarian', 'Lung', 'Prostate', 'Skin', 'Brain', 'Healthy'}.

You could test a detector for lung cancer by using a *PositiveValue* of 2, and a *NegativeValue* = [1 3 4 5 6].

Or you can test for any type of cancer by using *PositiveValue* = [1 2 3 4 5] and a *NegativeValue* of 6.

In clinical tests, inconclusive values such as '' or NaN are counted as false negatives for the computation of the specificity, and as false positives for the computation of the sensitivity. That is, inconclusive results may decrease the diagnostic value of the test. Tested observations for which *truelabels* is not within the union of *PositiveValue* and *NegativeValue* are not considered. However, tested observations that result in a class not covered by the vector *truelabels* are counted as inconclusive.

classperf(*CP*, *classout*) updates *CP*, the classifier performance object, with the classifier output *classout*. *classout* must be the same size as *truelabels*, the vector

or cell array used to construct the classifier performance object. When *classout* is a cell array of strings, an empty string, `''`, represents an inconclusive result of the classifier. For numeric arrays, NaN represents an inconclusive result.

classperf(*CP*, *classout*, *testidx*) updates *CP*, the classifier performance object, with the classifier output *classout*. *classout* has a smaller size than *truelabels*. *testidx* is an index vector or a logical index vector of the same size as *truelabels*, the vector or cell array used to construct the classifier performance object. *testidx* indicates the observations that were used in the current validation.

Note: In the two previous syntaxes, you do not need to create a separate output variable to update the classifier performance object, *CP*.

Properties of a Classifier Performance Object

You can access classifier performance object properties by using the get function

```
get(CP, 'ControlClasses')
```

or using dot notation

```
CP.ControlClasses
```

You cannot directly modify the classifier performance object properties by using the set function, with the exception of the Label and Description properties.

Tip To modify properties, use either of the following syntaxes:

```
classperf(CP, classout)
classperf(CP, classout, testidx)
```

Property	Description
Label	String to label the classifier performance object. Default is `''`.
Description	String to describe the classifier performance object. Default is `''`.

Property	Description
`ClassLabels`	Numeric vector or cell array of strings specifying a unique set of class labels from `unique(`*`truelabels`*`)`.
`GroundTruth`	Numeric vector or cell array of strings that specifies the true class labels for each observation. The number of elements = `NumberOfObservations`.
`NumberOfObservations`	Positive integer specifying the number of observations in the study.
`ControlClasses`	Indices to the `ClassLabels` vector or cell array, indicating which classes to be considered as the control or negative classes in a diagnostic test. **Tip** You set the `ControlClasses` property with the `'Negative'` property name/value pair. If you do not specify the `'Negative'` property, `ControlClasses` defaults to all classes other than the first class returned by `grp2idx(`*`truelabels`*`)`.
`TargetClasses`	Indices to the `ClassLabels` vector or cell array, indicating which classes to be considered as the target or positive classes in a diagnostic test. **Tip** You set the `TargetClasses` property with the `'Positive'` property name/value pair. If you do not specify the `'Positive'` property, `TargetClasses` defaults to the first class returned by `grp2idx(`*`truelabels`*`)`.
`ValidationCounter`	Positive integer specifying the number of validations performed.

Property	Description
SampleDistribution	Numeric vector indicating how many times each sample was considered in the validation. For example, if you use resubstitution, SampleDistribution is a vector of ones and ValidationCounter = 1. If you have a ten-fold cross-validation, SampleDistribution is also a vector of ones, but ValidationCounter = 10. **Tip** SampleDistribution is more useful when doing Monte Carlo partitions of the test sets, as this will help determine if all the samples are being equally tested.
ErrorDistribution	Numeric vector indicating how many times each sample was misclassified.
SampleDistributionByClass	Numeric vector indicating the frequency of the true classes in the validation.
ErrorDistributionByClass	Numeric vector indicating the frequency of errors for each class in the validation.
CountingMatrix	The classification confusion matrix. The order of rows and columns is the same as grp2idx(*truelabels*). Columns represent the true classes, and rows represent the classifier prediction. The last row in CountingMatrix is reserved to count inconclusive results. There are some families of classifiers that can reserve the right to make a hard class assignment; this can be based on metrics, such as the posterior probabilities, or on how close a sample is to the class boundaries.
CorrectRate	Correctly Classified Samples / Classified Samples **Note:** Inconclusive results are not counted.

Property	Description
ErrorRate	Incorrectly Classified Samples / Classified Samples **Note:** Inconclusive results are not counted.
LastCorrectRate	The following equation applies only to samples considered the last time the classifier performance object was updated: Correctly Classified Samples / Classified Samples
LastErrorRate	The following equation applies only to samples considered the last time the classifier performance object was updated: Incorrectly Classified Samples / Classified Samples
InconclusiveRate	Nonclassified Samples / Total Number of Samples
ClassifiedRate	Classified Samples / Total Number of Samples
Sensitivity	Correctly Classified Positive Samples / True Positive Samples **Note:** Inconclusive results that are true positives are counted as errors for computing Sensitivity (following a conservative approach). This is the same as being incorrectly classified as negatives.
Specificity	Correctly Classified Negative Samples / True Negative Samples **Note:** Inconclusive results that are true negatives are counted as errors for computing Specificity (following a conservative approach). This is the same as being incorrectly classified as positives.

Property	Description
PositivePredictiveValue	Correctly Classified Positive Samples / Positive Classified Samples **Note:** Inconclusive results are classified as negatives when computing `PositivePredictiveValue`.
NegativePredictiveValue	Correctly Classified Negative Samples / Negative Classified Samples **Note:** Inconclusive results are classified as positives when computing `NegativePredictiveValue`.
PositiveLikelihood	$\text{Sensitivity} / (1 - \text{Specificity})$
NegativeLikelihood	$(1 - \text{Sensitivity}) / \text{Specificity}$
Prevalence	True Positive Samples / Total Number of Samples
DiagnosticTable	A 2-by-2 numeric array with diagnostic counts. The first row indicates the number of samples that were classified as positive, with the number of true positives in the first column, and the number of false positives in the second column. The second row indicates the number of samples that were classified as negative, with the number of false negatives in the first column, and the number of true negatives in the second column. Correct classifications appear in the diagonal elements, and errors appear in the off-diagonal elements. Inconclusive results are considered errors and counted in the off-diagonal elements. For an illustration of a diagnostic table, see below.

Example Diagnostic Table

In a cancer study of ten patients, suppose we get the following results:

Patient	Classifier Output	Has Cancer
1	Positive	Yes
2	Positive	Yes
3	Positive	Yes
4	Positive	No
5	Negative	Yes
6	Negative	No
7	Negative	No
8	Negative	No
9	Negative	No
10	Inconclusive	Yes

The diagnostic table would look as follows:

```
                  True  State

                   1      0
                 ┌──────┬──────┐
Classifier    1  │  3   │  1   │
Output           ├──────┼──────┤
              0  │ 1+1  │  4   │
                 └──────┴──────┘
```

Examples

```
% Classify the fisheriris data with a K-Nearest Neighbor classifier
load fisheriris
c = knnclassify(meas,meas,species,4,'euclidean','Consensus');
cp = classperf(species,c)
get(cp)

% 10-fold cross-validation on the fisheriris data using linear
% discriminant analysis and the third column as only feature for
```

```
% classification
load fisheriris
indices = crossvalind('Kfold',species,10);
cp = classperf(species); % initializes the CP object
for i = 1:10
    test = (indices == i); train = ~test;
    class = classify(meas(test,3),meas(train,3),species(train));
    % updates the CP object with the current classification results
    classperf(cp,class,test)
end
cp.CorrectRate % queries for the correct classification rate


cp =

 biolearning.classperformance

                         Label: ''
                   Description: ''
                   ClassLabels: {3x1 cell}
                    truelabels: [150x1 double]
           NumberOfObservations: 150
                 ControlClasses: [2x1 double]
                  TargetClasses: 1
              ValidationCounter: 1
             SampleDistribution: [150x1 double]
              ErrorDistribution: [150x1 double]
      SampleDistributionByClass: [3x1 double]
       ErrorDistributionByClass: [3x1 double]
                 CountingMatrix: [4x3 double]
                    CorrectRate: 1
                      ErrorRate: 0
                InconclusiveRate: 0.0733
                  ClassifiedRate: 0.9267
                    Sensitivity: 1
                    Specificity: 0.8900
         PositivePredictiveValue: 0.8197
         NegativePredictiveValue: 1
              PositiveLikelihood: 9.0909
              NegativeLikelihood: 0
                      Prevalence: 0.3333
                  DiagnosticTable: [2x2 double]


ans =
    0.9467
```

See Also
crossvalind | classify | grp2idx | knnclassify | svmclassify

cleave

Cleave amino acid sequence with enzyme

Syntax

Fragments = cleave(*SeqAA*, *Enzyme*)
Fragments = cleave(*SeqAA*, *PeptidePattern*, *Position*)
[*Fragments*, *CuttingSites*] = cleave(...)
[*Fragments*, *CuttingSites*, *Lengths*] = cleave(...)
[*Fragments*, *CuttingSites*, *Lengths*, *Missed*] = cleave(...)
cleave(..., 'PartialDigest', *PartialDigestValue*, ...)
cleave(..., 'MissedSites', *MissedSitesValue*, ...)
cleave(..., 'Exception', *ExceptionValue*, ...)

Input Arguments

SeqAA	One of the following: • String of single-letter codes specifying an amino acid sequence. • Row vector of integers specifying an amino acid sequence. • MATLAB structure containing a `Sequence` field that contains an amino acid sequence, such as returned by `fastaread`, `getgenpept`, `genpeptread`, `getpdb`, or `pdbread`. Examples: `'ARN'` or `[1 2 3]`.
Enzyme	String specifying a name or abbreviation code for an enzyme or compound for which the literature specifies a cleavage rule. **Tip** Use the `cleavelookup` function to display the names of enzymes and compounds in the cleavage rule library.

PeptidePattern	Short amino acid sequence to search for in *SeqAA*, a larger sequence. *PeptidePattern* can be any of the following: • Character string • Vector of integers • Regular expression
Position	Integer from 0 to the length of the *PeptidePattern*, that specifies a position in the *PeptidePattern* to cleave. **Note:** Position 0 corresponds to the N terminal end of *PeptidePattern*.
PartialDigestValue	Value from 0 to 1 (default) specifying the probability that a cleavage site will be cleaved.
MissedSitesValue	Nonnegative integer specifying the maximum number of missed cleavage sites. The output includes all possible peptide fragments that can result from missing *MissedSitesValue* or less cleavage sites. Default is 0, which is equivalent to an ideal digestion.
ExceptionValue	Regular expression specifying an exception rule to the cleavage rule associated with *Enzyme*. By default, cleave applies no exception rule.

Output Arguments

Fragments	Cell array of strings representing the fragments from the cleavage.
CuttingSites	Numeric vector containing indices representing the cleavage sites. **Note:** The cleave function adds a 0 to the list, so numel(*CuttingSites*)==numel(*Fragments*). Use *CuttingSites* + 1 to point to the first amino acid of every fragment respective to the original sequence.

Lengths	Numeric vector containing the length of each fragment.
Missed	Numeric vector containing the number of missed cleavage sites for every peptide fragment.

Description

Fragments = cleave(*SeqAA, Enzyme*) cuts *SeqAA*, an amino acid sequence, into parts at the cleavage sites specific for *Enzyme*, a string specifying a name or abbreviation code for an enzyme or compound for which the literature specifies a cleavage rule. It returns *Fragments*, a cell array of strings representing the fragments from the cleavage.

Tip Use the cleavelookup function to display the names of enzymes and compounds in the cleavage rule library.

Fragments = cleave(*SeqAA, PeptidePattern, Position*) cuts *SeqAA*, an amino acid sequence, into parts at the cleavage sites specified by a peptide pattern and position.

[*Fragments, CuttingSites*] = cleave(...) returns a numeric vector containing indices representing the cleavage sites.

Note: The cleave function adds a 0 to the list, so numel(*CuttingSites*)==numel(*Fragments*). Use *CuttingSites* + 1 to point to the first amino acid of every fragment respective to the original sequence.

[*Fragments, CuttingSites, Lengths*] = cleave(...) returns a numeric vector containing the length of each fragment.

[*Fragments, CuttingSites, Lengths, Missed*] = cleave(...) returns a numeric vector containing the number of missed cleavage sites for every fragment.

cleave(..., '*PropertyName*', *PropertyValue*, ...) calls cleave with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

cleave(..., 'PartialDigest', *PartialDigestValue*, ...) simulates a partial digestion where *PartialDigestValue* is the probability of a cleavage site being cut. *PartialDigestValue* is a value from 0 to 1 (default).

This table lists some common proteases and their cleavage sites.

Protease	Peptide Pattern	Position
Aspartic acid N	D	1
Chymotrypsin	[WYF](?!P)	1
Glutamine C	[ED](?!P)	1
Lysine C	[K](?!P)	1
Trypsin	[KR](?!P)	1

cleave(..., 'MissedSites', *MissedSitesValue*, ...) returns all possible peptide fragments that can result from missing *MissedSitesValue* or less cleavage sites. *MissedSitesValue* is a nonnegative integer. Default is 0, which is equivalent to an ideal digestion.

cleave(..., 'Exception', *ExceptionValue*, ...) specifies an exception rule to the cleavage rule associated with *Enzyme*. *ExceptionValue* is a regular expression. By default, cleave applies no exception rule.

Examples

1 Retrieve a protein sequence from the GenPept database.

    S = getgenpept('AAA59174');

2 Cleave the sequence using proteinase K.

    [partsPK, sitesPK, lengthsPK] = cleave(S.Sequence, ...
     'proteinase K');

3 Display the indices of the cleavage sites, lengths, and sequences of the first ten fragments.

    for i=1:10
         fprintf('%5d%5d %s\n',sitesPK(i),lengthsPK(i),partsPK{i})
       end

     0   3   MGT

```
 3     6     GGRRGA
 9     1     A
10     1     A
11     1     A
12     2     PL
14     1     L
15     1     V
16     1     A
17     1     V
```

4 Cleave the same sequence using one of trypsin's cleavage rules (cleave after K or R when the next residue is not P).

```
[partsT, sitesT, lengthsT] = cleave(S.Sequence,'[KR](?!P)',1);
```

5 Display the indices of the cleavage sites, lengths, and sequences of the first ten fragments.

```
for i=1:10
      fprintf('%5d%5d   %s\n',sitesT(i),lengthsT(i),partsT{i})
   end
```

```
  0     6     MGTGGR
  6     1     R
  7    34     GAAAAPLLVAVAALLLGAAGHLYPGEVCPGMDIR
 41     5     NNLTR
 46    21     LHELENCSVIEGHLQILLMFK
 67     7     TRPEDFR
 74     6     DLSFPK
 80    12     LIMITDYLLLFR
 92     8     VYGLESLK
100    10     DLFPNLTVIR
```

6 Cleave the same sequence using trypsin's cleavage rule, but allow for one missed cleavage site.

```
[partsT2, sitesT2, lengthsT2, missedT2] = cleave(S.Sequence, ...
                                    'trypsin','missedsites',1);
```

7 Cleave the same sequence using trypsin's cleavage rule, except do not to cleave after K when K is following by a D.

```
partsT3 = cleave(S.Sequence, 'trypsin', 'exception', 'KD');
```

See Also
cleavelookup | rebasecuts | restrict | seqshowwords | regexp

cleavelookup

Find cleavage rule for enzyme or compound

Syntax

```
cleavelookup
cleavelookup('Code', CodeValue)
cleavelookup('Name', NameValue)
```

Arguments

CodeValue	String specifying a code representing an abbreviation code for an enzyme or compound. For valid codes, see the table Cleave Lookup.
NameValue	String specifying an enzyme or compound name. For valid names, see the table Cleave Lookup.

Description

`cleavelookup` displays a table of abbreviation codes, cleavage positions, cleavage patterns, and full names of enzymes and compounds for which cleavage rules are specified by the cleavage rule library. For more information, see the ExPASy PeptideCutter tool.

Cleave Lookup

Code	Position	Pattern	Full Name
ARG-C	1	R	ARG-C proteinase
ASP-N	2	D	ASP-N endopeptidase
BNPS	1	W	BNPS-Skatole
CASP1	1	(?<=[FWYL]\w[HAT])D(?=[^PEDQKR])	Caspase 1

Code	Position	Pattern	Full Name
CASP2	1	(?<=DVA)D(?=[^PEDQKR])	Caspase 2
CASP3	1	(?<=DMQ)D(?=[^PEDQKR])	Caspase 3
CASP4	1	(?<=LEV)D(?=[^PEDQKR])	Caspase 4
CASP5	1	(?<=[LW]EH)D	Caspase 5
CASP6	1	(?<=VE[HI])D(?=[^PEDQKR])	Caspase 6
CASP7	1	(?<=DEV)D(?=[^PEDQKR])	Caspase 7
CASP8	1	(?<=[IL]ET)D(?=[^PEDQKR])	Caspase 8
CASP9	1	(?<=LEH)D	Caspase 9
CASP10	1	(?<=IEA)D	Caspase 10
CH-HI	1	([FY](?=[^P]))|(W(?=[^MP]))	Chymotrypsin-high specificity
CH-LO	1	([FLY](?=[^P]))|(W(?=[^MP]))| (M(?=[^PY]))|(H(?=[^DMPW]))	Chymotrypsin-low specificity
CLOST	1	R	Clostripain
CNBR	1	M	CNBR
ENTKIN	1	(?<=[DN][DN][DN])K	Enterokinase
FACTXA	1	(?<=[AFGILTVM][DE]G)R	Factor XA
FORMIC	1	D	Formic acid
GLUEND	1	E	Glutamyl endopeptidase
GRANB	1	(?<=IEP)D	Granzyme B
HYDROX	1	N(?=G)	Hydroxylamine
IODOB	1	W	Iodosobenzoic acid
LYSC	1	K	Lysc
NTCB	1	C	NTCB
PEPS	1	((?<=[^HKR][^P])[^R](?=[FLWY][^P]))| ((?<=[^HKR][^P])[FLWY](?=\w[^P]))	Pepsin PH = 1.3

Code	Position	Pattern	Full Name
PEPS2	1	`((?<=[^HKR][^P])[^R](?=[FL][^P]))|((?<=[^HKR][^P])[FL](?=\w[^P]))`	Pepsin PH > 2
PROEND	1	`(?<=[HKR])P(?=[^P])`	Proline endopeptidase
PROTK	1	`[AEFILTVWY]`	Proteinase K
STAPHP	1	`(?<=[^E])E`	Staphylococcal peptidase I
THERMO	1	`[^DE](?=[AFILMV])`	Thermolysin
THROMB	1	`((?<=\w\wG)R(?=G))| ((?<=[AFGILTVM][AFGILTVWA]P)R(?=[^DE][^DE]))`	Thrombin
TRYPS	1	`((?<=\w)[KR](?=[^P]))| ((?<=W)K(?=P))|((?<=M)R(?=P))`	Trypsin

cleavelookup('Code', *CodeValue*) displays the cleavage position, cleavage pattern, and full name of the enzyme or compound specified by *CodeValue*, a string specifying an abbreviation code.

cleavelookup('Name', *NameValue*) displays the cleavage position, cleavage pattern, and abbreviation code of the enzyme or compound specified by *NameValue*, a string specifying an enzyme or compound name.

Examples

Using cleavelookup with an Enzyme Name

Display the cleavage position, cleavage pattern, and abbreviation code of the enzyme Caspase 1.

```
cleavelookup('name', 'CASPASE 1')

ans =

1 (?<=[FWYL]\w[HAT])D(?=[^PEDQKR]) CASP1
```

Using cleavelookup with an Abbreviation Code

Display the cleavage position, cleavage pattern, and full name of the enzyme with a abbreviation code of CASP1.

```
cleavelookup('code', 'CASP1')

ans =

1 (?<=[FWYL]\w[HAT])D(?=[^PEDQKR]) CASPASE 1
```

See Also
```
cleave | rebasecuts | restrict
```

cluster (phytree)

Validate clusters in phylogenetic tree

Syntax

```
LeafClusters = cluster(Tree, Threshold)
[LeafClusters, NodeClusters] = cluster(Tree, Threshold)
[LeafClusters, NodeClusters, Branches] = cluster(Tree, Threshold)
cluster(..., 'Criterion', CriterionValue, ...)
cluster(..., 'MaxClust', MaxClustValue, ...)
cluster(..., 'Distances', DistancesValue, ...)
```

Input Arguments

Tree	Phylogenetic tree object created, such as created with the `phytree` constructor function.
Threshold	Scalar specifying a threshold value.
CriterionValue	String specifying the criterion to determine the number of clusters as a function of the species pairwise distances. Choices are: • `'maximum'` (default) — Maximum within cluster pairwise distance (W_{max}). Cluster splitting stops when $W_{max} \leq$ *Threshold*. • `'median'` — Median within cluster pairwise distance (W_{med}). Cluster splitting stops when $W_{med} \leq$ *Threshold*. • `'average'` — Average within cluster pairwise distance (W_{avg}). Cluster splitting stops when $W_{avg} \leq$ *Threshold*. • `'ratio'` — Between/within cluster pairwise distance ratio, defined as $BW_{rat} = (\text{trace}(B)/(k - \mathbf{1})) / (\text{trace}(W)/(n - k))$ where *B* and *W* are the between- and within-scatter matrices, respectively. *k* is the number of clusters, and *n* is the number

	of species in the tree. Cluster splitting stops when $BW_{rat} \geq$ *Threshold*.
	• 'gain' — Within cluster pairwise distance gain, defined as $W_{gain} = (\text{trace}(W_{old})/ (\text{trace}(W)$ - 1) * $(n$ - k - 1))$ where W and W_{old} are the within-scatter matrices for k and k - 1, respectively. k is the number of clusters, and n is the number of species in the tree. Cluster splitting stops when $W_{gain} \leq$ *Threshold*.
	• 'silhouette' — Average silhouette width (SW_{avg}). SW_{avg} ranges from -1 to +1. Cluster splitting stops when $SW_{avg} \geq$ *Threshold*. For more information, see silhouette.
MaxClustValue	Positive integer specifying the maximum number of possible clusters for the tested partitions. Default is the number of leaves in the tree.
	Tip When using the 'maximum', 'median', or 'average' criteria, set *Threshold* to [] (empty) to force cluster to return *MaxClustValue* clusters. It does so because such metrics monotonically decrease as k increases.
	Tip When using the 'ratio', 'gain', or 'silhouette' criteria, you may find it hard to estimate an appropriate *Threshold* in advance. Set *Threshold* to [] (empty) to find the optimal number of clusters below *MaxClustValue*. Also, set *MaxClustValue* to a small value to avoid expensive computation due to testing all possible number of clusters.
DistancesValue	Matrix of pairwise distances, such as returned by the seqpdist function, containing biological distances between each pair of sequences. cluster substitutes this matrix for the patristic distances in *Tree*. For example, this matrix can contain the real sample pairwise distances.

Output Arguments

LeafClusters	Column vector containing a cluster index for each species (leaf) in *Tree*, a phylogenetic tree object.
NodeClusters	Column vector containing the cluster index for each leaf node and branch node in *Tree*.
	Tip Use the *LeafClusters* or *NodeClusters* output vectors with the handle returned by the plot method to modify graphic elements of the phylogenetic tree object. For more information, see "Examples" on page 1-424.
Branches	Two-column matrix containing, for each step in the algorithm, the index of the branch being considered and the value of the criterion. Each row corresponds to a step in the algorithm. The first column contains branch indices, and the second column contains criterion values.
	Tip To obtain the whole curve of the criterion versus the number of clusters in *Branches*, set *Threshold* to [] (empty) and do not specify a *MaxClustValue*. Be aware that computation of some criteria can be computationally intensive.

Description

LeafClusters = cluster(*Tree, Threshold*) returns a column vector containing a cluster index for each species (leaf) in a phylogenetic tree object. It determines the optimal number of clusters as follows:

- Starting with two clusters (k = 2), selects the partition that optimizes the criterion specified by the 'Criterion' property
- Increments k by 1 and again selects the optimal partition
- Continues incrementing k and selecting the optimal partition until a criterion value = *Threshold* or k = the maximum number of clusters (that is, number of leaves)
- From all possible k values, selects the k value whose partition optimizes the criterion

[*LeafClusters*, *NodeClusters*] = cluster(*Tree*, *Threshold*) returns a column vector containing the cluster index for each leaf node and branch node in *Tree*.

[*LeafClusters*, *NodeClusters*, *Branches*] = cluster(*Tree*, *Threshold*) returns a two-column matrix containing, for each step in the algorithm, the index of the branch being considered and the value of the criterion. Each row corresponds to a step in the algorithm. The first column contains branch indices, and the second column contains criterion values.

cluster(..., '*PropertyName*', *PropertyValue*, ...) calls cluster with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:.

cluster(..., 'Criterion', *CriterionValue*, ...) specifies the criterion to determine the number of clusters as a function of the species pairwise distances.

cluster(..., 'MaxClust', *MaxClustValue*, ...) specifies the maximum number of possible clusters for the tested partitions. Default is the number of leaves in the tree.

cluster(..., 'Distances', *DistancesValue*, ...) substitutes the patristic distances in *Tree* with a user-provided pairwise distance matrix.

Examples

Validate the clusters in a phylogenetic tree:

```
% Read sequences from a multiple alignment file into a MATLAB
% structure
gagaa = multialignread('aagag.aln');

% Build a phylogenetic tree from the sequences
gag_tree = seqneighjoin(seqpdist(gagaa),'equivar',gagaa);

% Validate the clusters in the tree and find the best partition
% using the 'gain' criterion
[i,j] = cluster(gag_tree,[],'criterion','gain','maxclust',10);

% Use the returned vector of indices to color the branches of each
% cluster in a plot of the tree
```

```
h = plot(gag_tree);
set(h.BranchLines(j==2),'Color','b')
set(h.BranchLines(j==1),'Color','r')
```
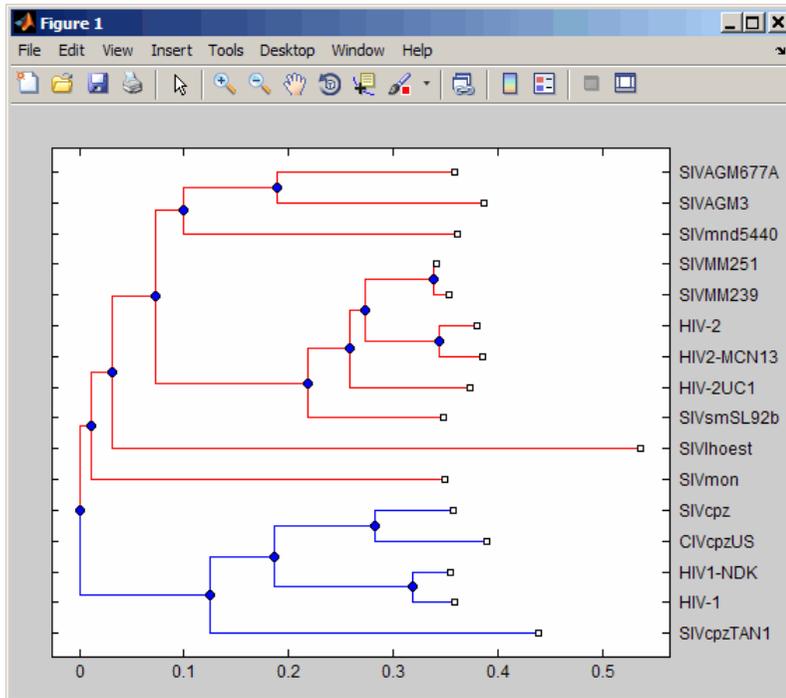

More About

- phytree object

References

[1] Dudoit, S. and Fridlyan, J. (2002). A prediction-based resampling method for estimating the number of clusters in a dataset. Genome Biology *3(7)*, research 0036.1–0036.21.

[2] Theodoridis, S. and Koutroumbas, K. (1999). Pattern Recognition (Academic Press), pp. 434–435.

[3] Kaufman, L. and Rousseeuw, P.J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis* (New York, Wiley).

[4] Calinski, R. and Harabasz, J. (1974). A dendrite method for cluster analysis. Commun Statistics *3*, 1–27.

[5] Hartigan, J.A. (1985). Statistical theory in clustering. J Classification *2*, 63–76.

See Also

```
phytree | plot | view | cluster | phytreeread | phytreeviewer | seqlinkage
| seqneighjoin | seqpdist | silhouette
```

clustergram object

Object containing hierarchical clustering analysis data

Description

A clustergram object contains hierarchical clustering analysis data that you can view in a heat map and dendrograms.

Create a clustergram object using the object constructor function `clustergram`. View a graphical representation of the clustergram object in a heat map and dendrograms using the `view` method.

The clustergram class is a subclass of the HeatMap class.

Method Summary

Following are methods of a clustergram object:

addTitle (clustergram)	Add title to clustergram
addXLabel (clustergram)	Label *x*-axis of clustergram
addYLabel (clustergram)	Label *y*-axis of clustergram
clusterGroup (clustergram)	Select cluster group
get (clustergram)	Retrieve information about clustergram object
plot (clustergram)	Render clustergram and dendrograms for clustergram object
set (clustergram)	Set property of clustergram object

view (clustergram)

View clustergram and dendrograms of clustergram object

Property Summary

Properties for Clustering Analysis and Clustergram Creation

Property Name	Description
Standardize	String or number specifying the dimension for standardizing the data values. This property transforms the standardized values so that the mean is 0 and the standard deviation is 1 in the specified dimension. Choices are: • `'column'` or `1` — Standardize along the columns of data. • `'row'` or `2` — Standardize along the rows of data. • `'none'` or `3` (default) — Do not standardize.
Cluster	String or number specifying the dimension for clustering the values in the data. Choices are: • `'column'` or `1` — Cluster along the columns of data only, which results in clustered rows. • `'row'` or `2` — Cluster along the rows of data only, which results in clustered columns. • `'all'` or `3` (default) — Cluster along the columns of data, then cluster along the rows of row-clustered data.
RowPDist	String specifying the distance metric to pass to the `pdist` function (Statistics and Machine Learning Toolbox™ software) to calculate the pairwise distances between rows. For information on choices, see the `pdist` function. Default is `'euclidean'`. --- **Note:** If the distance metric requires extra arguments, then *RowPDistValue* is a cell array. For example, to use the Minkowski distance with exponent P, you would use `{'minkowski', P}`.

Property Name	Description
ColumnPDist	String specifying the distance metric to pass to the pdist function (Statistics and Machine Learning Toolbox software) to calculate the pairwise distances between columns. For information on choices, see the pdist function. Default is 'euclidean'.
	Note: If the distance metric requires extra arguments, then *ColumnPDistValue* is a cell array. For example, to use the Minkowski distance with exponent P, you would use {'minkowski', P}.
Linkage	String or two-element cell array of strings specifying the linkage method to pass to the linkage function (Statistics and Machine Learning Toolbox software) to create the hierarchical cluster tree for rows and columns. If a two-element cell array of strings, this property uses the first element for linkage between rows, and the second element for linkage between columns. For information on choices, see the linkage function. Default is 'average'.
Dendrogram	Scalar or two-element numeric vector or cell array of strings specifying the 'colorthreshold' property to pass to the dendrogram function (Statistics and Machine Learning Toolbox software) to create the dendrogram plot. If a two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns. For more information, see the dendrogram function.
OptimalLeafOrder	Enables or disables the optimal leaf ordering calculation, which determines the leaf order that maximizes the similarity between neighboring leaves. Choices are true (enable) or false (disable). Default depends on the size of *Data*, the matrix of data used to create the clustergram object. If the number of rows or columns in *Data* exceeds 1500, default is false; otherwise, default is true.

Property Name	Description
	Tip Disabling the optimal leaf ordering calculation can be useful when working with large data sets, because this calculation consumes a lot of memory and time.
Colormap	Either of the following: • *M*-by-3 matrix of RGB values • Name or function handle of a function that returns a colormap, such as `redgreencmap` or `redbluecmap` Default is `redgreencmap`.
DisplayRange	Positive scalar specifying the display range of standardized values. Default is **3**, which means there is a color variation for values between –3 and 3, but values >3 are the same color as 3, and values < –3 are the same color as –3. For example, if you specify `redgreencmap` for the `'Colormap'` property, pure red represents values ≥ *DisplayRangeValue*, and pure green represents values ≤ –*DisplayRangeValue*.
Symmetric	Forces the color scale of the heat map to be symmetric around zero. Choices are `true` (default) or `false`.
LogTrans	Controls the $\log_2$ transform of the data from natural scale. Choices are `true` or `false` (default).
DisplayRatio	Either of the following: • Scalar • Two-element vector This property specifies the ratio of space that the row and column dendrograms occupy relative to the heat map. If `DisplayRatio` is a scalar, it is used as the ratio for both dendrograms. If `DisplayRatio` is a two-element vector, the first element is used for the ratio of the row dendrogram width to the heat map width, and the second element is used for the ratio of the column dendrogram height to the

Property Name	Description
	heat map height. The second element is ignored for one-dimensional clustergrams. Default is 1/5.
ImputeFun	One of the following: • Name of a function that imputes missing data. • Handle to a function that imputes missing data. • Cell array where the first element is the name of or handle to a function that imputes missing data. The remaining elements are property name/property value pairs used as inputs to the function.
ShowDendrogram	Shows and hides the dendrogram tree diagrams with the clustergram. Choices are 'on' (default) or 'off'. **Tip** After displaying a clustergram in a Clustergram window, click the Show Dendrogram ▦ button on the toolbar to show and hide the dendrograms.

Properties for Group Labels

Property Name	Description
RowGroupMarker	Structure or structure array containing information for annotating the groups (clusters) of rows determined by the clustergram function. The structure or structures contain the following fields. If a single structure, then the fields contain a cell array of elements. If a structure array, then the fields contain one element: • GroupNumber — Scalar specifying the row group number to annotate. • Annotation — String specifying text to annotate the row group. • Color — String or three-element vector of RGB values specifying a color to label the row group. For more information on specifying colors, see ColorSpec. If this field is empty, default is 'blue'.

Property Name	Description
ColumnGroupMarker	Structure or structure array containing information for annotating the groups (clusters) of columns determined by the clustergram function. The structure or structures contain the following fields. If a single structure, then the fields contain a cell array of elements. If a structure array, then the fields contain one element: • GroupNumber — Scalar specifying the column group number to annotate. • Annotation — String specifying text to annotate the column group. • Color — String or three-element vector of RGB values specifying a color to label the column group. For more information on specifying colors, see ColorSpec. If this field is empty, default is 'blue'.

Properties for Row and Column Labels

Property Name	Description
RowLabels	Vector of numbers or cell array of text strings to label the rows in the dendrogram and heat map. Default is a vector of values 1 through *M*, where *M* is the number of rows in *Data*, the matrix of data used by the clustergram function to create the clustergram object.
ColumnLabels	Vector of numbers or cell array of text strings to label the columns in the dendrogram and heat map. Default is a vector of values 1 through *M*, where *M* is the number of columns in *Data*, the matrix of data used by the clustergram function to create the clustergram object.
ColumnLabelsLocation	Read-only string specifying the location of the column labels. For clustergram objects, it is always 'bottom' (default).
RowLabelsLocation	Read-only string specifying the location of the row labels. For clustergram objects, it is always 'right' (default).
RowLabelsColor	Structure or structure array containing color information for labeling the rows (*y*-axis) of the clustergram. The structure or structures contain the following fields. If a single structure,

Property Name	Description
	then the fields contain a cell array of elements. If a structure array, then the fields contain one element: • Labels — String specifying a row label listed in the RowLabels vector. • Colors — String or three-element vector of RGB values specifying a color for the row label specified in the Labels field. For more information on specifying colors, see ColorSpec. If this field is empty, default colors are assigned to the row label.
ColumnLabelsColor	Structure or structure array containing color information for labeling the columns (*x*-axis) of the clustergram. The structure or structures contain the following fields. If a single structure, then the fields contain a cell array of elements. If a structure array, then the fields contain one element: • Labels — String specifying a column label listed in the ColumnLabels vector. • Colors — String or three-element vector of RGB values specifying a color for the column label specified in the Labels field. For more information on specifying colors, see ColorSpec. If this field is empty, default colors are assigned to the column label.
LabelsWithMarkers	Controls the display of colored markers instead of colored text for the row labels and column labels. Choices are true or false (default).
RowLabelsRotate	Numeric value in degrees rotation specifying the orientation of row (*y*-axis) labels. Default is 0 degrees, which is horizontal. Positive values cause counterclockwise rotation.
ColumnLabelsRotate	Numeric value in degrees rotation specifying the orientation of column (*x*-axis) labels. Default is 90 degrees, which is vertical. Values greater than 90 degrees cause counterclockwise rotation.

Properties for Annotating Data

Property Name	Description
Annotate	Controls the display of intensity values on each area of the heat map. Choices are `true` or `false` (default).
	Tip After displaying a clustergram in a Clustergram window, click the Annotate ![button] button on the toolbar to show and hide the intensity values.
AnnotPrecision	Positive integer specifying the precision of the intensity values when displayed on the heat map. Default is `2`.
AnnotColor	String or three-element vector of RGB values specifying a color, which is used for the text of the intensity values when displayed on the heat map. Default is `'white'`. For more information on specifying colors, see `ColorSpec`.

Examples

Note: The following examples use the `get` and `set` methods with property names and values of a clustergram object. When supplying a *PropertyName*, be aware that it is case sensitive.

Determining Properties and Property Values of a Clustergram Object

Display all properties and their current values of a clustergram object, *CGobj*:

```
get(CGobj)
```

Return all properties and their current values of *CGobj*, a clustergram object, to *CGstruct*, a scalar structure, in which each field name is a property of a clustergram object, and each field contains the value of that property:

```
CGstruct = get(CGobj)
```

Return the value of a specific property of a clustergram object, *CGobj*, using either:

```
PropertyValue = get(CGobj, 'PropertyName')
```

PropertyValue = CGobj.PropertyName

Return the value of specific properties of a clustergram object, *CGobj*:

[*Property1Value*, *Property2Value*, ...] = get(*CGobj*, ...
'*Property1Name*', '*Property2Name*', ...)

Determining Possible Values of Clustergram Object Properties

Display possible values for all properties that have a fixed set of property values in a clustergram object, *CGobj*:

set(*CGobj*)

Display possible values for a specific property that has a fixed set of property values in a clustergram object, *CGobj*:

set(*CGobj*, '*PropertyName*')

Specifying Properties of a Clustergram Object

Set a specific property of a clustergram object, *CGobj*, using either:

set(*CGobj*, '*PropertyName*', *PropertyValue*)

CGobj.PropertyName = PropertyValue

Set multiple properties of a clustergram object, *CGobj*:

set(*CGobj*, '*Property1Name*', *Property1Value*, ...
    '*Property2Name*', *Property2Value*, ...)

More About

• HeatMap object

See Also

clustergram | addXLabel | clusterGroup | plot | set | addTitle | addYLabel
| get | view | display

clustergram

Compute hierarchical clustering, display dendrogram and heat map, and create clustergram object

Syntax

```
CGobj = clustergram(Data)
CGobj = clustergram(Data, ...'RowLabels', RowLabelsValue, ...)
CGobj = clustergram(Data, ...'ColumnLabels', ColumnLabelsValue, ...)
CGobj = clustergram(Data, ...'Standardize', StandardizeValue, ...)
CGobj = clustergram(Data, ...'Cluster', ClusterValue, ...)
CGobj = clustergram(Data, ...'RowPDist', RowPDistValue, ...)
CGobj = clustergram(Data, ...'ColumnPDist', ColumnPDistValue, ...)
CGobj = clustergram(Data, ...'Linkage', LinkageValue, ...)
CGobj = clustergram(Data, ...'Dendrogram', DendrogramValue, ...)
CGobj = clustergram(Data, ...'OptimalLeafOrder',
OptimalLeafOrderValue, ...)
CGobj = clustergram(Data, ...'Colormap', ColormapValue, ...)
CGobj = clustergram(Data, ...'DisplayRange', DisplayRangeValue, ...)
CGobj = clustergram(Data, ...'Symmetric', SymmetricValue, ...)
CGobj = clustergram(Data, ...'LogTrans', LogTransValue, ...)
CGobj = clustergram(Data, ...'DisplayRatio', DisplayRatioValue, ...)
CGobj = clustergram(Data, ...'ImputeFun', ImputeFunValue, ...)
CGobj = clustergram(Data, ...'RowGroupMarker',
RowGroupMarkerValue, ...)
CGobj = clustergram(Data, ...'ColumnGroupMarker',
ColumnGroupMarkerValue, ...)
```

Arguments

Data	DataMatrix object or numeric matrix of data. If the matrix contains gene expression data, typically each row corresponds to a gene and each column corresponds to a sample.

RowLabelsValue	Vector of numbers or cell array of text strings to label the rows in the dendrogram and heat map. Default is a vector of values 1 through *M*, where *M* is the number of rows in *Data*.
	Note: If the number of row labels is 200 or more, the labels do not appear in the clustergram plot unless you zoom in on the plot.
ColumnLabelsValue	Vector of numbers or cell array of text strings to label the columns in the dendrogram and heat map. Default is a vector of values 1 through *N*, where *N* is the number of columns in *Data*.
	Note: If the number of column labels is 200 or more, the labels do not appear in the clustergram plot unless you zoom in on the plot.
StandardizeValue	String or number specifying the dimension for standardizing the values in *Data*. The clustergram function transforms the standardized values so that the mean is 0 and the standard deviation is 1 in the specified dimension. Choices are: • `'column'` or 1 — Standardize along the columns of data. • `'row'` or 2 — Standardize along the rows of data. • `'none'` or 3 (default) — Do not standardize.

ClusterValue	String or number specifying the dimension for clustering the values in *Data*. Choices are: • `'column'` or `1` — Cluster along the columns of data only, which results in clustered rows. • `'row'` or `2` — Cluster along the rows of data only, which results in clustered columns. • `'all'` or `3` (default) — Cluster along the columns of data, then cluster along the rows of row-clustered data.
RowPDistValue	String, function handle, or cell array specifying the distance metric to pass to the `pdist` function (Statistics and Machine Learning Toolbox software) to calculate the pairwise distances between rows. For information on choices, see the `pdist` function. Default is `'euclidean'`. **Note:** If the distance metric requires extra arguments, then *RowistValue* is a cell array. For example, to use the Minkowski distance with exponent P, you would use `{'minkowski', P}`.
ColumnPDistValue	String, function handle, or cell array specifying the distance metric to pass to the `pdist` function (Statistics and Machine Learning Toolbox software) to use to calculate the pairwise distances between columns. For information on choices, see the `pdist` function. Default is `'euclidean'`. **Note:** If the distance metric requires extra arguments, then *ColumnPDistValue* is a cell array. For example, to use the Minkowski distance with exponent P, you would use `{'minkowski', P}`.

LinkageValue	String or two-element cell array of strings specifying the linkage method to pass to the `linkage` function (Statistics and Machine Learning Toolbox software) to create the hierarchical cluster tree for rows and columns. If a two-element cell array of strings, the `clustergram` function uses the first element for linkage between rows, and the second element for linkage between columns. For information on choices, see the `linkage` function. Default is `'average'`.
	Tip To specify the linkage method for only one dimension, set the other dimension to `''`.
DendrogramValue	Scalar or two-element numeric vector or cell array of strings specifying the `'colorthreshold'` property to pass to the `dendrogram` function (Statistics and Machine Learning Toolbox software) to create the dendrogram plot. If a two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns. For more information, see the `dendrogram` function.
	Tip To specify the `'colorthreshold'` property for only one dimension, set the other dimension to `''`.
OptimalLeafOrderValue	Enables or disables the optimal leaf ordering calculation, which determines the leaf order that maximizes the similarity between neighboring leaves. Choices are `true` (enable) or `false` (disable). Default depends on the size of *Data*. If the number of rows or columns in *Data* exceeds 1500, default is `false`; otherwise, default is `true`.
	Note: Disabling the optimal leaf ordering calculation can be useful when working with large data sets, because this calculation consumes a lot of memory and time.

ColormapValue	Either of the following:
	- *M*-by-3 matrix of RGB values
	- Name of or handle to a function that returns a colormap, such as redgreencmap or redbluecmap
	Default is redgreencmap, in which red represents values above the mean, black represents the mean, and green represents values below the mean of a row (gene) across all columns (samples).
DisplayRangeValue	Positive scalar that specifies the display range of standardized values. Default is 3, which means there is a color variation for values between −3 and 3, but values >3 are the same color as 3, and values < −3 are the same color as −3.
	For example, if you specify redgreencmap for the 'Colormap' property, pure red represents values ≥ *DisplayRangeValue*, and pure green represents values ≤ −*DisplayRangeValue*.
SymmetricValue	Forces the color scale of the heat map to be symmetric around zero. Choices are true (default) or false.
LogTransValue	Controls the $\log_2$ transform of *Data* from natural scale. Choices are true or false (default).

DisplayRatioValue	Either of the following: • Scalar • Two-element vector This property specifies the ratio of space that the row and column dendrograms occupy relative to the heat map. If *DisplayRatioValue* is a scalar, the `clustergram` function uses it as the ratio for both dendrograms. If *DisplayRatioValue* is a two-element vector, the `clustergram` function uses the first element for the ratio of the row dendrogram width to the heat map width, and the second element for the ratio of the column dendrogram height to the heat map height. The `clustergram` function ignores the second element for one-dimensional clustergrams. Default is `1/5`.
ImputeFunValue	One of the following: • Name of a function that imputes missing data. • Handle to a function that imputes missing data. • Cell array where the first element is the name of or handle to a function that imputes missing data. The remaining elements are property name/property value pairs used as inputs to the function. **Caution** If data points are missing, use the `'ImputeFun'` property. Otherwise, the `clustergram` function errors.

RowGroupMarkerValue	Structure or structure array containing information for annotating the groups (clusters) of rows determined by the `clustergram` function. The structure or structures contain the following fields. If a single structure, then the fields contain a cell array of elements. If a structure array, then the fields contain a single element. • `GroupNumber` — Scalar specifying the row group number to annotate. • `Annotation` — String specifying text to annotate the row group. • `Color` — String or three-element vector of RGB values specifying a color, which the `clustergram` function uses to label the row group. For more information on specifying colors, see `ColorSpec`. If this field is empty, default is `'blue'`.
ColumnGroupMarkerValue	Structure or structure array containing information for annotating the groups (clusters) of columns determined by the `clustergram` function. The structure or structures contain the following fields. If a single structure, then the fields contain a cell array of elements. If a structure array, then the fields contain a single element. • `GroupNumber` — Scalar specifying the column group number to annotate. • `Annotation` — String specifying text to annotate the column group. • `Color` — String or three-element vector of RGB values specifying a color, which the `clustergram` function uses to label the column group. For more information on specifying colors, see `ColorSpec`. If this field is empty, default is `'blue'`.

Description

CGobj = clustergram(*Data*) performs hierarchical clustering analysis on the values in *Data*, a DataMatrix object or numeric matrix. It creates *CGobj*, an object containing the analysis data, and displays a dendrogram and heat map. It uses hierarchical clustering with Euclidean distance metric and average linkage to generate the hierarchical tree. It clusters first along the columns (producing row-clustered data), and then along the rows in the matrix *Data*. If *Data* contains gene expression data, typically the rows correspond to genes and the columns correspond to samples.

CGobj = clustergram(*Data*, ...'*PropertyName*', *PropertyValue*, ...) calls clustergram with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

CGobj = clustergram(*Data*, ...'RowLabels', *RowLabelsValue*, ...) uses the contents of *RowLabelsValue*, a vector of numbers or cell array of text strings, as labels for the rows in the dendrogram and heat map. Default is a vector of values 1 through *M*, where *M* is the number of rows in *Data*.

CGobj = clustergram(*Data*, ...'ColumnLabels', *ColumnLabelsValue*, ...) uses the contents of *ColumnLabelsValue*, a vector of numbers or cell array of text strings, as labels for the columns in the dendrogram and heat map. Default is a vector of values 1 through *M*, where *M* is the number of columns in *Data*.

CGobj = clustergram(*Data*, ...'Standardize', *StandardizeValue*, ...) specifies the dimension for standardizing the values in *Data*. The clustergram function transforms the standardized values so that the mean is 0 and the standard deviation is 1 in the specified dimension. *StandardizeValue* can be:

- 'column' or 1 — Standardize along the columns of data.
- 'row' or 2 (default) — Standardize along the rows of data.
- 'none' or 3 — Do not standardize.

CGobj = clustergram(*Data*, ...'Cluster', *ClusterValue*, ...) specifies the dimension for clustering the values in *Data*. *ClusterValue* can be:

- 'column' or 1 — Cluster along the columns of data only, which results in clustered rows.
- 'row' or 2 — Cluster along the rows of data only, which results in clustered columns.

- `'all'` or `3` (default) — Cluster along the columns of data, then cluster along the rows of row-clustered data.

CGobj = clustergram(*Data*, ...'RowPDist', *RowPDistValue*, ...) specifies the distance metric to pass to the `pdist` function (Statistics and Machine Learning Toolbox software) to use to calculate the pairwise distances between rows. *RowPDistValue* is a string, function handle, or cell array. For information on choices, see the `pdist` function. Default is `'euclidean'`.

CGobj = clustergram(*Data*, ...'ColumnPDist', *ColumnPDistValue*, ...) specifies the distance metric to pass to the `pdist` function (Statistics and Machine Learning Toolbox software) to use to calculate the pairwise distances between columns. *ColumnPDistValue* is a string, function handle, or cell array. For information on choices, see the `pdist` function. Default is `'euclidean'`.

Note: If the distance metric requires extra arguments, then *RowPDistValue* or *ColumnPDistValue* is a cell array. For example, to use the Minkowski distance with exponent P, you would use `{'minkowski', P}`.

CGobj = clustergram(*Data*, ...'Linkage', *LinkageValue*, ...) specifies the linkage method to pass to the `linkage` function (Statistics and Machine Learning Toolbox software) to use to create the hierarchical cluster tree for rows and columns. *LinkageValue* is a string or two-element cell array of strings. If a two-element cell array of strings, the `clustergram` function uses first element for linkage between rows, and the second element for linkage between columns. For information on choices, see the `linkage` function. Default is `'average'`.

Tip To specify the linkage method for only one dimension, set the other dimension to `' '`.

CGobj = clustergram(*Data*, ...'Dendrogram', *DendrogramValue*, ...) specifies the `'colorthreshold'` property to pass to the `dendrogram` function (Statistics and Machine Learning Toolbox software) to create the dendrogram plot. *DendrogramValue* is a scalar or two-element numeric vector or cell array of strings that specifies the `'colorthreshold'` property. If a two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns. For more information, see the `dendrogram` function.

Tip To specify the `'colorthreshold'` property for only one dimension, set the other dimension to `''`.

CGobj = clustergram(*Data*, ...'OptimalLeafOrder', *OptimalLeafOrderValue*, ...) enables or disables the optimal leaf ordering calculation, which determines the leaf order that maximizes the similarity between neighboring leaves. Choices are `true` (enable) or `false` (disable). Default depends on the size of *Data*. If the number of rows or columns in *Data* exceeds 1500, default is `false`; otherwise, default is `true`.

Tip Disabling the optimal leaf ordering calculation can be useful when working with large data sets, because this calculation consumes a lot of memory and time.

CGobj = clustergram(*Data*, ...'Colormap', *ColormapValue*, ...) specifies the colormap to use to create the clustergram. The colormap controls the colors used to display the heat map. *ColormapValue* is either an *M*-by-3 matrix of RGB values or the name of or handle to a function that returns a colormap, such as `redgreencmap` or `redbluecmap`. Default is `redgreencmap`.

Note: In `redgreencmap`, red represents values above the mean, black represents the mean, and green represents values below the mean of a row (gene) across all columns (samples). In `redbluecmap`, red represents values above the mean, white represents the mean, and blue represents values below the mean of a row (gene) across all columns (samples).

CGobj = clustergram(*Data*, ...'DisplayRange', *DisplayRangeValue*, ...) specifies the display range of standardized values. *DisplayRangeValue* must be a positive scalar. Default is 3, which means there is a color variation for values between −3 and 3, but values >3 are the same color as 3, and values < −3 are the same color as −3.

For example, if you specify `redgreencmap` for the `'Colormap'` property, pure red represents values ≥ *DisplayRangeValue*, and pure green represents values ≤ −*DisplayRangeValue*.

CGobj = clustergram(*Data*, ...'Symmetric', *SymmetricValue*, ...) controls whether the color scale of the heat map is symmetric around zero. *SymmetricValue* can be `true` (default) or `false`.

CGobj = clustergram(*Data*, ...'LogTrans', *LogTransValue*, ...) controls the $\log_2$ transform of *Data* from natural scale. Choices are `true` or `false` (default).

CGobj = clustergram(*Data*, ...'DisplayRatio', *DisplayRatioValue*, ...) specifies the ratio of space that the row and column dendrograms occupy relative to the heat map. If *DisplayRatioValue* is a scalar, the `clustergram` function uses it as the ratio for both dendrograms. If *DisplayRatioValue* is a two-element vector, the `clustergram` function uses the first element for the ratio of the row dendrogram width to the heat map width, and the second element for the ratio of the column dendrogram height to the heat map height. The `clustergram` function ignores the second element for one-dimensional clustergrams. Default is `1/5`.

CGobj = clustergram(*Data*, ...'ImputeFun', *ImputeFunValue*, ...) specifies a function and optional inputs that impute missing data. *ImputeFunValue* can be any of the following:

- Name of a function that imputes missing data.
- Handle to a function that imputes missing data.
- Cell array where the first element is the name of or handle to a function that imputes missing data. The remaining elements are property name/property value pairs used as inputs to the function.

Tip If data points are missing, use the `'ImputeFun'` property. Otherwise, the `clustergram` function errors.

CGobj = clustergram(*Data*, ...'RowGroupMarker', *RowGroupMarkerValue*, ...) specifies a structure or structure array containing information for annotating the groups (clusters) of rows determined by the `clustergram` function.

CGobj = clustergram(*Data*, ...'ColumnGroupMarker', *ColumnGroupMarkerValue*, ...) specifies a structure or structure array containing information for annotating the groups of columns determined by the `clustergram` function.

Tip If necessary, view row labels (right) and column labels (bottom) by clicking the Zoom

In ![zoom icon] button on the toolbar to zoom the clustergram.

Examples

The following example uses data from an experiment (DeRisi et al., 1997) that used DNA microarrays to study temporal gene expression of almost all genes in *Saccharomyces cerevisiae* (yeast) during the metabolic shift from fermentation to respiration. Expression levels were measured at seven time points during the diauxic shift.

1 Load the MAT-file, provided with Bioinformatics Toolbox, that contains filtered yeast data.

```
load filteredyeastdata
```

This MAT-file includes three variables, which are added to the MATLABWorkspace:

- yeastvalues — A matrix of gene expression data from *Saccharomyces cerevisiae* (yeast) during the metabolic shift from fermentation to respiration
- genes — A cell array of GenBank accession numbers for labeling the rows in yeastvalues
- times — A vector of time values for labeling the columns in yeastvalues

2 Create a clustergram object and display the heat map from the gene expression data in the first 30 rows of the yeastvalues matrix and standardize along the rows of data.

```
cgo = clustergram(yeastvalues(1:30,:),'Standardize','Row')
Clustergram object with 30 rows of nodes and 7 columns of nodes.
```
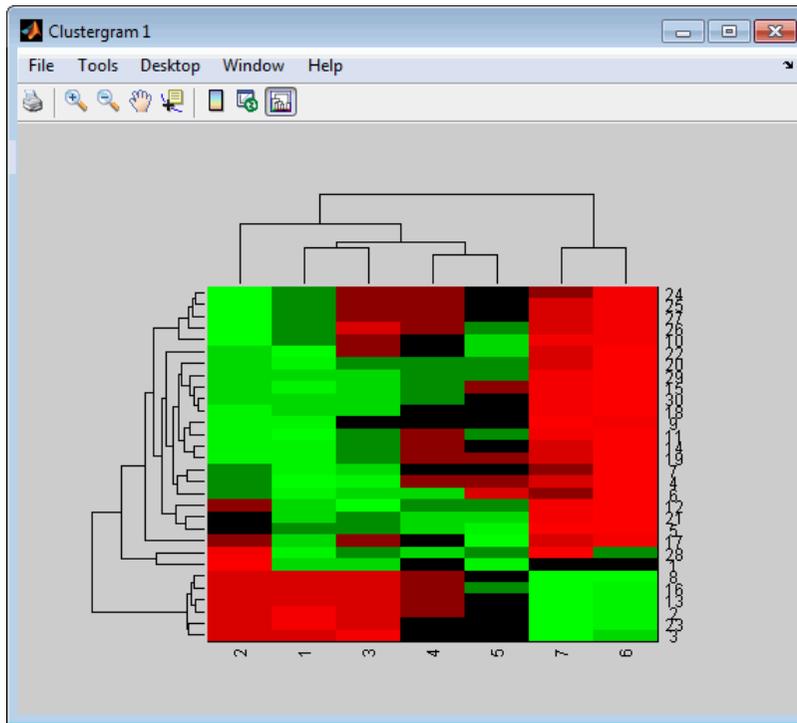
3 Use the `set` method and the `genes` and `times` vectors to add meaningful row and column labels to the clustergram.

```
set(cgo,'RowLabels',genes(1:30),'ColumnLabels',times)
```
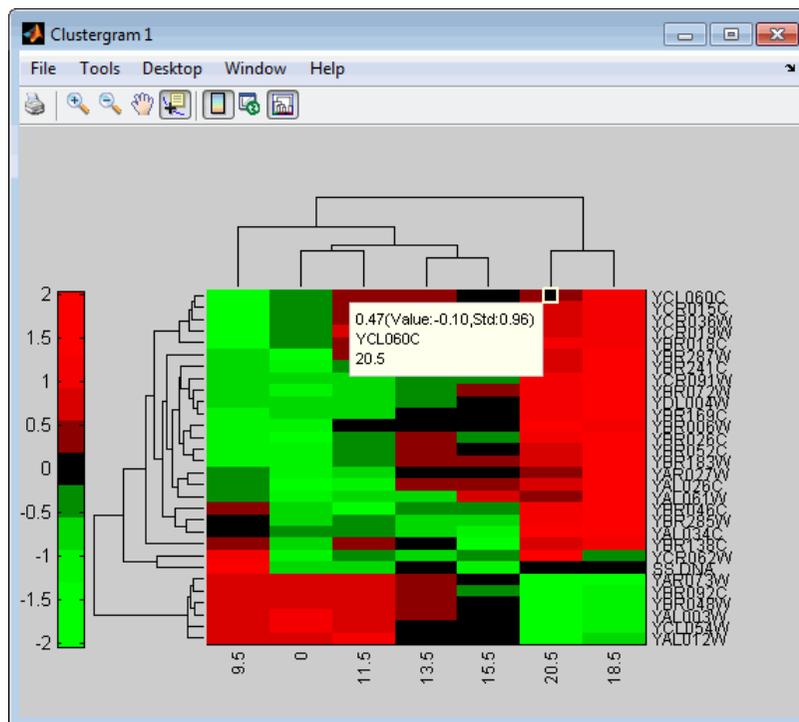
4
Add a color bar to the clustergram by clicking the Insert Colorbar ▢ button on the toolbar.

5 View a data tip containing the intensity value, row label, and column label for

a specific area of the heat map by clicking the Data Cursor ▣ button on the toolbar, then clicking an area in the heat map. To delete this data tip, right-click it, then select **Delete Current Datatip**.

6 Display intensity values for each area of the heat map by clicking the Annotate

▣ button on the toolbar. Click the Annotate button again to remove the intensity values.

Tip If the amount of data is large enough, the cells within the clustergram are too small to display the intensity annotations. Zoom the clustergram to see the intensity annotations.

---

7 Remove the dendrogram tree diagrams from the figure by clicking the Show

Dendrogram ![icon] button on the toolbar. Click the Show Dendrogram button again to display the dendrograms.

8 Use the `get` method to display the properties of the clustergram object, `cgo`:

```
get(cgo)
```

```
              Cluster: 'ALL'
              RowPDist: {'Euclidean'}
           ColumnPDist: {'Euclidean'}
               Linkage: {'Average'}
            Dendrogram: {}
       OptimalLeafOrder: 1
              LogTrans: O
           DisplayRatio: [0.2000 0.2000]
         RowGroupMarker: []
      ColumnGroupMarker: []
         ShowDendrogram: 'on'
           ColumnLabels: {' 9.5' '  0' '11.5' '13.5' '15.5' '20.5' '18.5'}
              RowLabels: {30x1 cell}
     ColumnLabelsRotate: 90
        RowLabelsRotate: O
   ColumnLabelsLocation: 'bottom'
      RowLabelsLocation: 'right'
            Standardize: 'ROW'
              Symmetric: 1
           DisplayRange: 3
               Colormap: [11x3 double]
              ImputeFun: []
               Annotate: 'off'
         AnnotPrecision: 2
             AnnotColor: 'w'
      ColumnLabelsColor: []
         RowLabelsColor: []
      LabelsWithMarkers: O
```

9 Change the clustering parameters by changing the linkage method and changing the color of the groups of nodes in the dendrogram whose linkage is less than a threshold of 3.

```
set(cgo,'Linkage','complete','Dendrogram',3)
```
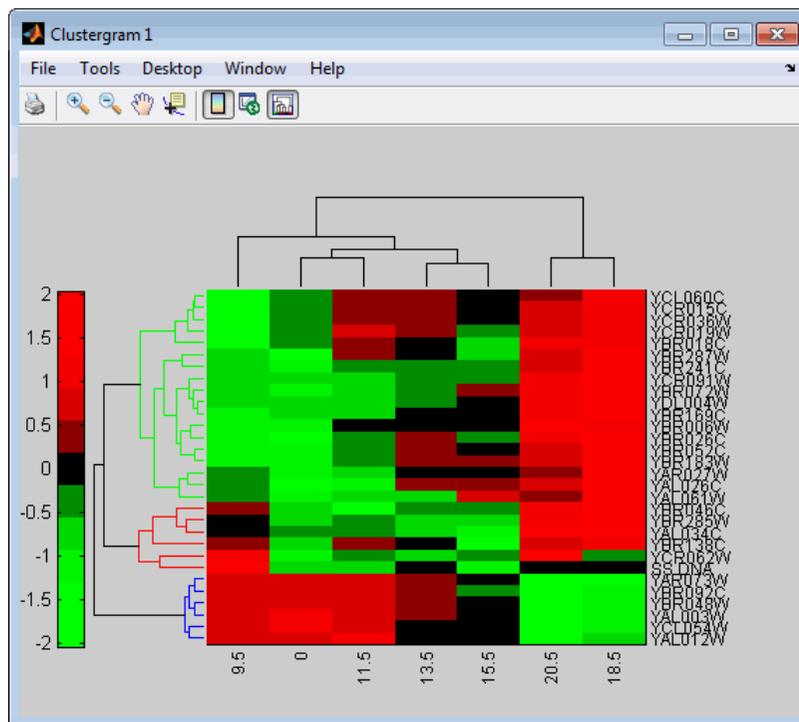
10 Place the cursor on a branch node in the dendrogram to highlight (in blue) the group associated with it. Press and hold the mouse button to display a data tip listing the group number and the nodes (genes or samples) in the group.

11 Right-click a branch node in the dendrogram to display a menu of options.

The following options are available:

- **Set Group Color** — Change the cluster group color.
- **Print Group to Figure** — Print the group to a Figure window.
- **Copy Group to New Clustergram** — Copy the group to a new Clustergram window.
- **Export Group to Workspace** — Create a clustergram object of the group in the MATLAB Workspace.
- **Export Group Info to Workspace** — Create a structure containing information about the group in the MATLAB Workspace. The structure contains these fields:

  - GroupNames — Cell array of text strings containing the names of the row or column groups.

- `RowNodeNames` — Cell array of text strings containing the names of the row nodes.

- `ColumnNodeNames` — Cell array of text strings containing the names of the column nodes.

- `ExprValues` — An *M*-by-*N* matrix of intensity values, where *M* and *N* are the number of row nodes and of column nodes respectively. If the matrix contains gene expression data, typically each row corresponds to a gene and each column corresponds to sample.

12 Create a clustergram object in the MATLAB Workspace of Group 18 by right-clicking it, then selecting **Export Group to Workspace**. In the Export to Workspace dialog box, type **Group18**, then click **OK**.

13 Use the `get` method to display the properties of the clustergram object, `Group18`.

```
get(Group18)

              Cluster: 'ALL'
              RowPDist: {'Euclidean'}
           ColumnPDist: {'Euclidean'}
               Linkage: 'complete'
            Dendrogram: 3
        OptimalLeafOrder: 1
              LogTrans: 0
          DisplayRatio: [0.2000 0.2000]
         RowGroupMarker: []
      ColumnGroupMarker: []
         ShowDendrogram: 'on'
           ColumnLabels: {' 9.5'  '  0'  '11.5'  '13.5'  '15.5'  '20.5'  '18.5'}
              RowLabels: {3x1 cell}
      ColumnLabelsRotate: 90
         RowLabelsRotate: 0
    ColumnLabelsLocation: 'bottom'
       RowLabelsLocation: 'right'
             Standardize: 'ROW'
               Symmetric: 1
            DisplayRange: 3
                 Colormap: [11x3 double]
                ImputeFun: []
                 Annotate: 'off'
           AnnotPrecision: 2
               AnnotColor: 'w'
       ColumnLabelsColor: []
          RowLabelsColor: []
       LabelsWithMarkers: 0
```

14 Use the `view` method to view the clustergram (dendrograms and heat map) of the clustergram object, `Group18`.

```
view(Group18)
```
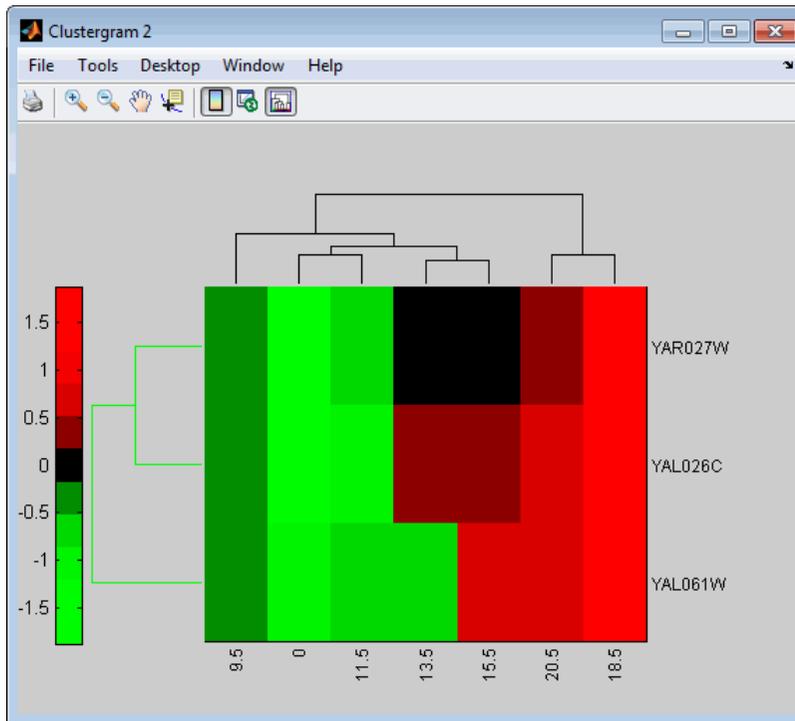
15 View all the gene expression data using a diverging red and blue colormap and standardize along the rows of data.

```
cgo_all = clustergram(yeastvalues,'Colormap',redbluecmap,'Standardize','Row')
Clustergram object with 614 rows of nodes and 7 columns of nodes.
```
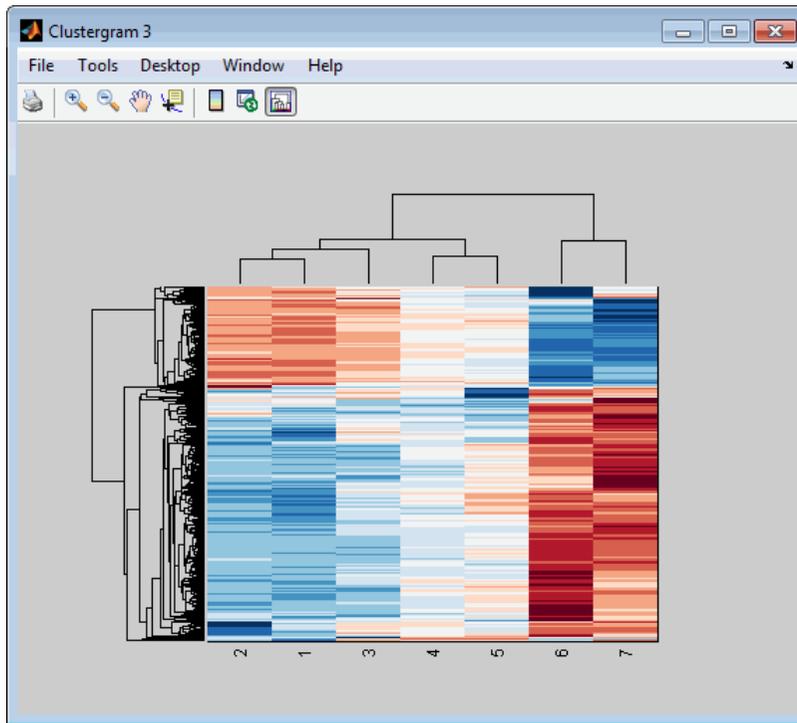
16 Create structure arrays to specify marker colors and annotations for two groups of rows (510 and 593) and two groups of columns (4 and 5).

```
rm = struct('GroupNumber',{510,593},'Annotation',{'A','B'},...
    'Color',{'b','m'});
cm = struct('GroupNumber',{4,5},'Annotation',{'Time1','Time2'},...
    'Color',{[1 1 0],[0.6 0.6 1]});
```

17 Use the `'RowGroupMarker'` and `'ColumnGroupMarker'` properties to add the color markers and annotations to the clustergram.

```
set(cgo_all,'RowGroupMarker',rm,'ColumnGroupMarker',cm)
```
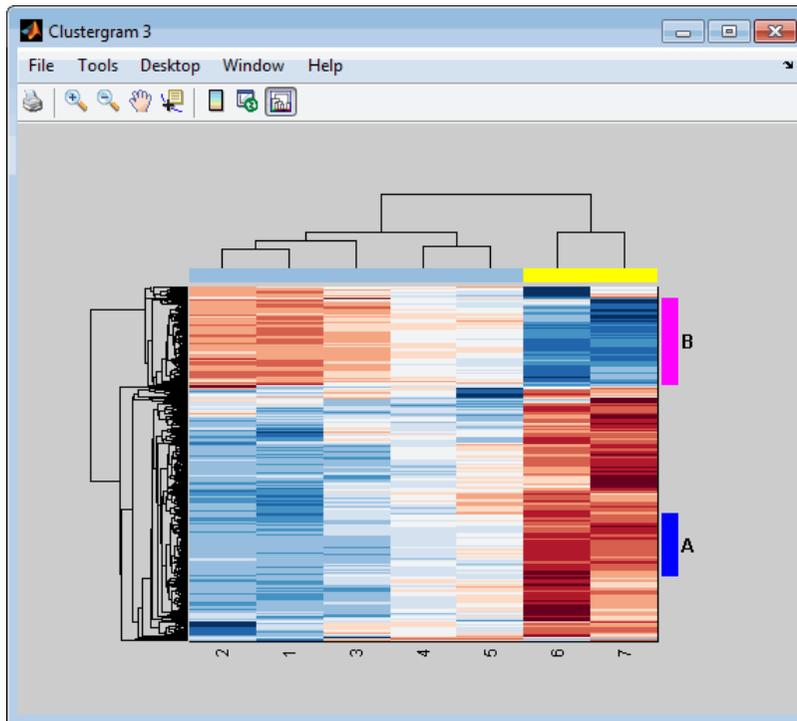
18 Click the color column markers to display the annotations.

More About

·   clustergram object

References

[1] Bar-Joseph, Z., Gifford, D.K., and Jaakkola, T.S. (2001). Fast optimal leaf ordering for hierarchical clustering. Bioinformatics *17*, Suppl 1:S22 – 9. PMID: 11472989.

[2] Eisen, M.B., Spellman, P.T., Brown, P.O., and Botstein, D. (1998). Cluster analysis and display of genome-wide expression patterns. Proc Natl Acad Sci USA *95*, 14863–8.

[3] DeRisi, J.L., Iyer, V.R., and Brown, P.O. (1997). Exploring the metabolic and genetic control of gene expression on a genomic scale. Science *278*, 680–686s.

[4] Golub, T.R., Slonim, D.K., and Tamayo, P., et al. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science *286 (15)*, 531–537.

See Also

redbluecmap | addXLabel | clustergroup | plot | set | redgreencmap | addTitle | addYLabel | get | view | cluster | dendrogram | linkage | pdist

clusterGroup (clustergram)

Select cluster group

Syntax

```
clusterGroup(CGobj1, GroupIndex, Dim)
CGobj2 = clusterGroup(CGobj1, GroupIndex, Dim)
CGStruct = clusterGroup(CGobj1, GroupIndex, Dim, 'InfoOnly',
InfoOnlyValue)
CGStruct = clusterGroup(CGobj1, GroupIndex, Dim, 'Color',
ColorValue)
```

Input Arguments

CGobj1	Clustergram object created with the function `clustergram`.
GroupIndex	Positive integer specifying a group index for a cluster in *CGobj1*.
Dim	String specifying the dimension of the cluster group. Choices are `'column'` or `'row'`.
InfoOnlyValue	Controls the return of a structure (instead of a clustergram object) containing information about the cluster group. Choices are `true` or `false` (default).
ColorValue	Color to highlight the dendrogram of the selected cluster group. Specify the color with one of the following: • Three-element numeric vector of RGB values • String containing a predefined single-letter color code • String containing a predefined color name For example, to use cyan, enter `[0 1 1]`, `'c'`, or `'cyan'`. For more information on specifying colors, see `ColorSpec`.

Output Arguments

CGobj2	Clustergram object created from the selected cluster group in *CGobj1*.
CGStruct	Structure containing information about the cluster group in the following fields: • `GroupNames` — Cell array of text strings containing the names of the row or column groups in the selected cluster group. • `RowNodeNames` — Cell array of text strings containing the names of the row nodes in the selected cluster group. • `ColumnNodeNames` — Cell array of text strings containing the names of the column nodes in the selected cluster group. • `ExprValues` — An *M*-by-*N* matrix of intensity values, where *M* and *N* are the number of row nodes and of column nodes respectively in the selected cluster group. If the matrix contains gene expression data, typically each row corresponds to a gene and each column corresponds to a sample.

Description

clusterGroup(*CGobj1*, *GroupIndex*, *Dim*) selects and highlights a cluster group in the Clustergram window, specified by a clustergram object, group index, and dimension.

CGobj2 = clusterGroup(*CGobj1*, *GroupIndex*, *Dim*) creates a clustergram object from the specified cluster group. This syntax is equivalent to selecting the **Export Group to Workspace** command from the context menu after right-clicking a group in the Clustergram window.

CGStruct = clusterGroup(*CGobj1*, *GroupIndex*, *Dim*, 'InfoOnly', *InfoOnlyValue*) controls the return of a structure (instead of a clustergram object) containing information about the cluster group. Choices are true or false (default). Setting this property to true is equivalent to selecting the **Export Group Info to Workspace** command from the context menu after right-clicking a group in the Clustergram window.

CGStruct = clusterGroup(*CGobj1*, *GroupIndex*, *Dim*, 'Color', *ColorValue*) specifies a color for the dendrogram of the selected cluster group.

Examples

Select and highlight column cluster Group 4 in the Clustergram window, from the clustergram object created in the first two steps of the "Examples" on page 1-447 section of the `clustergram` function reference page.

```
clusterGroup(cgo,4,'column')
```
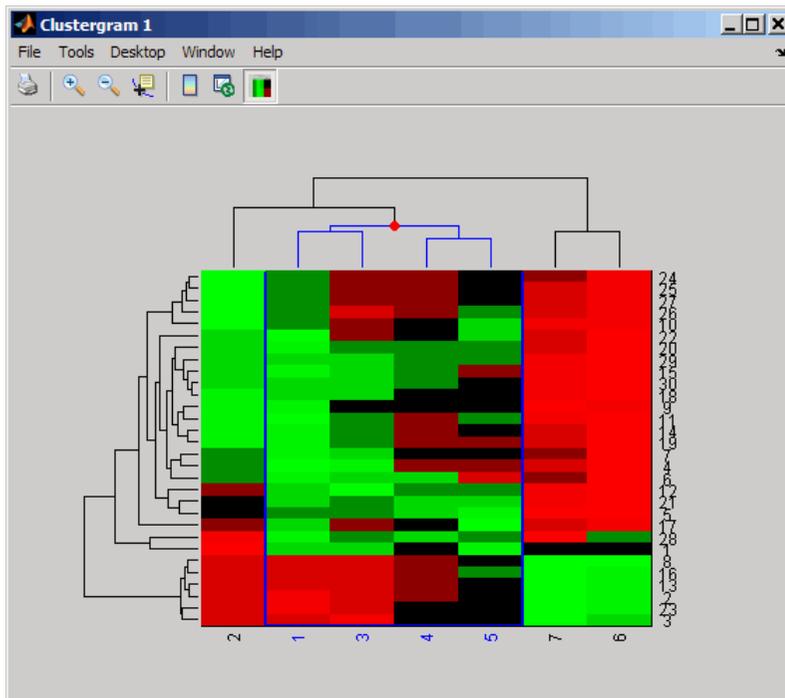

More About

- clustergram object

See Also

`clustergram` | `set` | `view` | `get`

codonbias

Calculate codon frequency for each amino acid coded for in nucleotide sequence

Syntax

CodonFreq = codonbias(*SeqNT*)
CodonFreq = codonbias(*SeqNT*, ...'GeneticCode',
GeneticCodeValue, ...)
CodonFreq = codonbias(*SeqNT*, ...'Frame', *FrameValue*, ...)
CodonFreq = codonbias(*SeqNT*, ...'Reverse', *ReverseValue*, ...)
CodonFreq = codonbias(*SeqNT*, ...'Ambiguous', *AmbiguousValue*, ...)
CodonFreq = codonbias(*SeqNT*, ...'Pie', *PieValue*, ...)

Input Arguments

SeqNT	One of the following: • String of codes specifying a nucleotide sequence • Row vector of integers specifying a nucleotide sequence • MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, or `getgenbank` Valid characters include `A`, `C`, `G`, `T`, and `U`. `codonbias` does not count ambiguous nucleotides or gaps.
GeneticCodeValue	Integer or string specifying a genetic code number or code name from the table Genetic Code. Default is `1` or `'Standard'`. **Tip** If you use a code name, you can truncate the name to the first two letters of the name.

FrameValue	Integer specifying a reading frame in the nucleotide sequence. Choices are 1 (default), 2, or 3.
ReverseValue	Controls the return of the codon frequency for the reverse complement sequence of the nucleotide sequence specified by *SeqNT*. Choices are true or false (default).
AmbiguousValue	String specifying how to treat codons containing ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N). Choices are: • 'ignore' (default) — Skips codons containing ambiguous characters • 'prorate' — Counts codons containing ambiguous characters and distributes them proportionately in the appropriate codon fields. For example, the counts for the codon ART are distributed evenly between the AAT and AGT fields. • 'warn' — Skips codons containing ambiguous characters and displays a warning.
PieValue	Controls the creation of a figure of 20 pie charts, one for each amino acid. Choices are true or false (default).

Output Arguments

CodonFreq	MATLAB structure containing a field for each amino acid, each of which contains the associated codon frequencies as percentages.

Description

Many amino acids are coded by two or more nucleic acid codons. However, the probability that a specific codon (from all possible codons for an amino acid) is used to code an amino acid varies between sequences. Knowing the frequency of each codon in a protein coding sequence for each amino acid is a useful statistic.

CodonFreq = codonbias(*SeqNT*) calculates the codon frequency in percent for each amino acid coded for in *SeqNT*, a nucleotide sequence, and returns the results in *CodonFreq*, a MATLAB structure containing a field for each amino acid.

CodonFreq = codonbias(*SeqNT*, ...'*PropertyName*', *PropertyValue*, ...) calls codonbias with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

CodonFreq = codonbias(*SeqNT*, ...'GeneticCode', *GeneticCodeValue*, ...) specifies a genetic code. Choices for *GenetidCodeValue* are an integer or string specifying a code number or code name from the table Genetic Code. If you use a code name, you can truncate the name to the first two characters of the name. Default is 1 or 'Standard'.

Tip If you use a code name, you can truncate the name to the first two letters of the name.

CodonFreq = codonbias(*SeqNT*, ...'Frame', *FrameValue*, ...) calculates the codon frequency in the reading frame specified by *FrameValue*, which can be 1 (default), 2, or 3.

CodonFreq = codonbias(*SeqNT*, ...'Reverse', *ReverseValue*, ...) controls the return of the codon frequency for the reverse complement of the nucleotide sequence specified by *SeqNT*. Choices are true or false (default).

CodonFreq = codonbias(*SeqNT*, ...'Ambiguous', *AmbiguousValue*, ...) specifies how to treat codons containing ambiguous nucleotide characters. Choices are 'ignore' (default), 'prorate', and 'warn'.

CodonFreq = codonbias(*SeqNT*, ...'Pie', *PieValue*, ...) controls the creation of a figure of 20 pie charts, one for each amino acid. Choices are true or false (default).

Genetic Code

Code Number	Code Name
1	Standard

Code Number	Code Name
2	`Vertebrate Mitochondrial`
3	`Yeast Mitochondrial`
4	`Mold`, `Protozoan`, `Coelenterate Mitochondrial`, and `Mycoplasma/Spiroplasma`
5	`Invertebrate Mitochondrial`
6	`Ciliate`, `Dasycladacean`, and `Hexamita Nuclear`
9	`Echinoderm Mitochondrial`
10	`Euplotid Nuclear`
11	`Bacterial` and `Plant Plastid`
12	`Alternative Yeast Nuclear`
13	`Ascidian Mitochondrial`
14	`Flatworm Mitochondrial`
15	`Blepharisma Nuclear`
16	`Chlorophycean Mitochondrial`
21	`Trematode Mitochondrial`
22	`Scenedesmus Obliquus Mitochondrial`
23	`Thraustochytrium Mitochondrial`

Examples

Calculate Codon Frequency for Each Amino Acid

Import a nucleotide sequence from the GenBank database into the MATLAB software. For example, retrieve the DNA sequence that codes for a human insulin receptor.

```
S = getgenbank('M10051');
```

Calculate the codon frequency for each amino acid coded for by the DNA sequence, and then plot the results.

```
cb = codonbias(S.Sequence,'PIE',true)
```
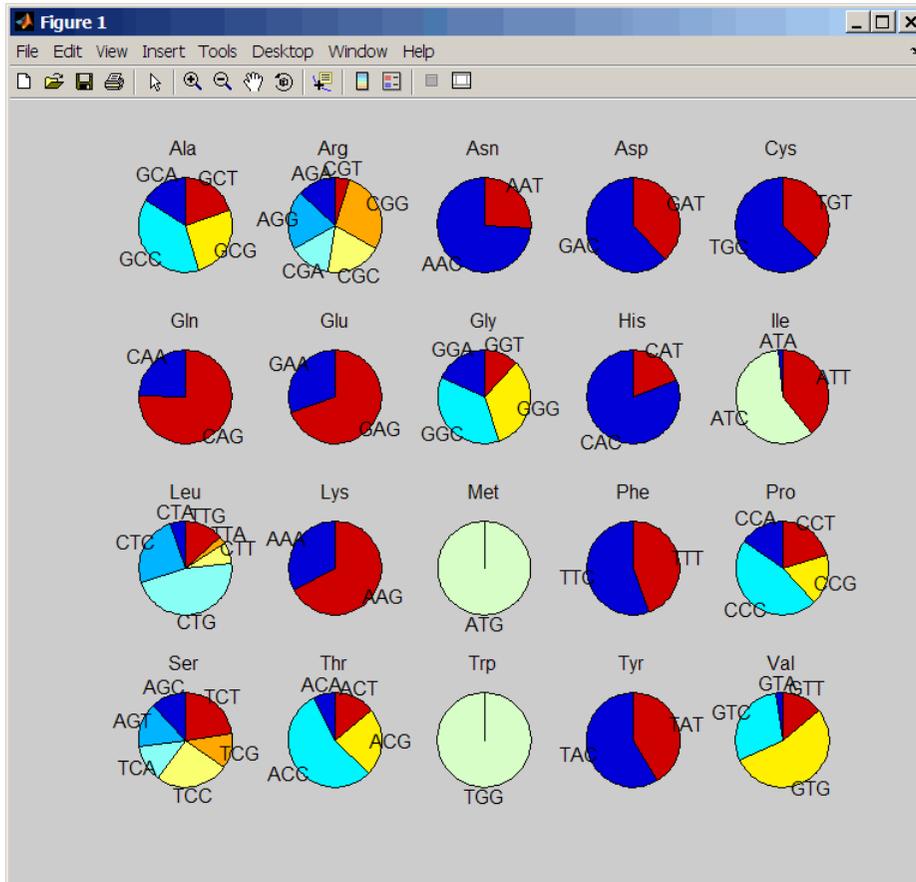
Get the codon frequency for the alanine (A) amino acid.

```
cb.Ala

ans =

    Codon: {'GCA' "GCC' "GCG' 'GCT'}
     Freq: [0.1600 0.3867 0.2533 02000]
```

See Also
aminolookup | codoncount | geneticcode | nt2aa

codoncount

Count codons in nucleotide sequence

Syntax

```
Codons = codoncount(SeqNT)
[Codons, CodonArray] = codoncount(SeqNT)
... = codoncount(SeqNT, ...'Frame', FrameValue, ...)
... = codoncount(SeqNT, ...'Reverse', ReverseValue, ...)
... = codoncount(SeqNT, ...'Ambiguous', AmbiguousValue, ...)
... = codoncount(SeqNT, ...'Figure', FigureValue, ...)
... = codoncount(SeqNT, ...'GeneticCode', GeneticCodeValue, ...)
```

Input Arguments

SeqNT	One of the following: • String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers • Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes • MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, or `getgenbank`. Examples: `'ACGT'` or `[1 2 3 4]`
FrameValue	Integer specifying a reading frame in the nucleotide sequence. Choices are `1` (default), `2`, or `3`.
ReverseValue	Controls the return of the codon count for the reverse complement sequence of the nucleotide sequence specified by *SeqNT*. Choices are `true` or `false` (default).

AmbiguousValue	String specifying how to treat codons containing ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N). Choices are: • `'ignore'` (default) — Skips codons containing ambiguous characters • `'bundle'` — Counts codons containing ambiguous characters and reports the total count in the Ambiguous field of the *Codons* output structure. • `'prorate'` — Counts codons containing ambiguous characters and distributes them proportionately in the appropriate codon fields containing standard nucleotide characters. For example, the counts for the codon ART are distributed evenly between the AAT and AGT fields. • `'warn'` — Skips codons containing ambiguous characters and displays a warning.
FigureValue	Controls the display of a heat map of the codon counts. Choices are true or false (default).
GeneticCodeValue	Integer or string specifying a genetic code number or code name from the table Genetic Code. Default is 1 or `'Standard'`. You can also specify `'None'`. --- **Tip** If you use a code name, you can truncate the name to the first two letters of the name.

Output Arguments

Codons	MATLAB structure containing fields for the 64 possible codons (AAA, AAC, AAG, ..., TTG, TTT), which contain the codon counts in *SeqNT*.
CodonArray	A 4-by-4-by-4 array containing the raw count data for each codon. The three dimensions correspond to the three positions in the codon, and the indices to each element are represented by 1 = A, 2 = C, 3 = G, and 4 = T. For example, the element (2,3,4) in the array contains the number of CGT codons.

Description

Codons = codoncount(*SeqNT*) counts the codons in *SeqNT*, a nucleotide sequence, and returns the codon counts in *Codons*, a MATLAB structure containing fields for the 64 possible codons (AAA, AAC, AAG, ..., TTG, TTT).

- For sequences that have codons containing the character U, these codons are added to the corresponding codons containing a T.
- If the sequence contains gaps indicated by a hyphen (-), then codons containing gaps are ignored.
- If the sequence contains unrecognized characters, then codons containing these characters are ignored, and the following warning message appears:

  `Warning: Unknown symbols appear in the sequence. These will be ignored.`

[*Codons, CodonArray*] = codoncount(*SeqNT*) returns *CodonArray*, a 4-by-4-by-4 array containing the raw count data for each codon. The three dimensions correspond to the three positions in the codon, and the indices to each element are represented by 1 = A, 2 = C, 3 = G, and 4 = T. For example, the element (2,3,4) in the array contains the number of CGT codons.

... = codoncount(*SeqNT*, ...'*PropertyName*', *PropertyValue*, ...) calls codoncount with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = codoncount(*SeqNT*, ...'Frame', *FrameValue*, ...) counts the codons in the reading frame specified by *FrameValue*, which can be 1 (default), 2, or 3.

... = codoncount(*SeqNT*, ...'Reverse', *ReverseValue*, ...) controls the return of the codon count for the reverse complement sequence of *SeqNT*. Choices are true or false (default).

... = codoncount(*SeqNT*, ...'Ambiguous', *AmbiguousValue*, ...) specifies how to treat codons containing ambiguous nucleotide characters. Choices are:

- 'ignore' (default)
- 'bundle'
- 'prorate'
- 'warn'

`... = codoncount(`*`SeqNT`*`, ...`'Figure'`, `*`FigureValue`*`, ...)` controls the display of a heat map of the codon counts. Choices are `true` or `false` (default).

`... = codoncount(`*`SeqNT`*`, ...`'GeneticCode'`, `*`GeneticCodeValue`*`, ...)` controls the overlay of a grid on the heat map figure. The grid groups the synonymous codons according to *`GeneticCodeValue`*.

Examples

• Count the codons in a nucleotide sequence.

```
codons = codoncount('AAACGTTA')

codons =

        AAA: 1  ATC: 0  CGG: 0  GCT: 0  TCA: 0
        AAC: 0  ATG: 0  CGT: 1  GGA: 0  TCC: 0
        AAG: 0  ATT: 0  CTA: 0  GGC: 0  TCG: 0
        AAT: 0  CAA: 0  CTC: 0  GGG: 0  TCT: 0
        ACA: 0  CAC: 0  CTG: 0  GGT: 0  TGA: 0
        ACC: 0  CAG: 0  CTT: 0  GTA: 0  TGC: 0
        ACG: 0  CAT: 0  GAA: 0  GTC: 0  TGG: 0
        ACT: 0  CCA: 0  GAC: 0  GTG: 0  TGT: 0
        AGA: 0  CCC: 0  GAG: 0  GTT: 0  TTA: 0
        AGC: 0  CCG: 0  GAT: 0  TAA: 0  TTC: 0
        AGG: 0  CCT: 0  GCA: 0  TAC: 0  TTG: 0
        AGT: 0  CGA: 0  GCC: 0  TAG: 0  TTT: 0
        ATA: 0  CGC: 0  GCG: 0  TAT: 0
```

• Count the codons in the second frame for the reverse complement of a sequence.

```
r2codons = codoncount('AAACGTTA','Frame',2,'Reverse',true)

r2codons =

        AAA: 0  ATC: 0  CGG: 0  GCT: 0  TCA: 0
        AAC: 1  ATG: 0  CGT: 0  GGA: 0  TCC: 0
        AAG: 0  ATT: 0  CTA: 0  GGC: 0  TCG: 0
        AAT: 0  CAA: 0  CTC: 0  GGG: 0  TCT: 0
        ACA: 0  CAC: 0  CTG: 0  GGT: 0  TGA: 0
        ACC: 0  CAG: 0  CTT: 0  GTA: 0  TGC: 0
        ACG: 0  CAT: 0  GAA: 0  GTC: 0  TGG: 0
        ACT: 0  CCA: 0  GAC: 0  GTG: 0  TGT: 0
```

```
AGA: 0   CCC: 0   GAG: 0   GTT: 1   TTA: 0
AGC: 0   CCG: 0   GAT: 0   TAA: 0   TTC: 0
AGG: 0   CCT: 0   GCA: 0   TAC: 0   TTG: 0
AGT: 0   CGA: 0   GCC: 0   TAG: 0   TTT: 0
ATA: 0   CGC: 0   GCG: 0   TAT: 0
```

- Create a heat map of the codons in a random nucleotide sequence and overlay a grid that groups the synonymous codons according to the Standard genetic code.

```
a = randseq(1000);
codoncount(a,'Figure', true);
```
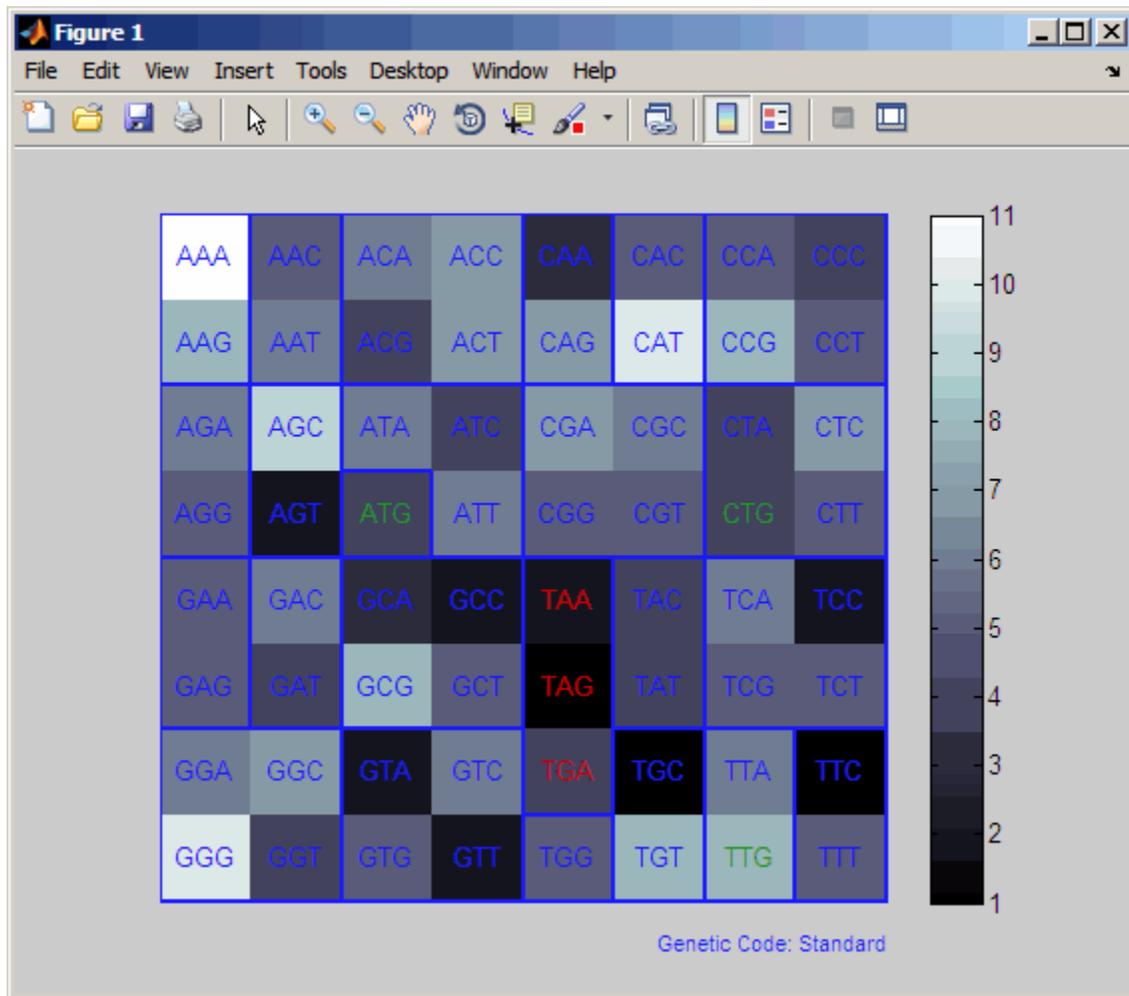
See Also

aacount | basecount | baselookup | codonbias | dimercount | nmercount | ntdensity | seqcomplement | seqrcomplement | seqreverse | seqwordcount

colnames (DataMatrix)

Retrieve or set column names of DataMatrix object

Syntax

```
ReturnColNames = colnames(DMObj)
ReturnColNames = colnames(DMObj, ColIndices)
DMObjNew = colnames(DMObj, ColIndices, ColNames)
```

Input Arguments

DMObj	DataMatrix object, such as created by DataMatrix (object constructor).
ColIndices	One or more columns in *DMObj*, specified by any of the following: • Positive integer • Vector of positive integers • String specifying a column name • Cell array of strings • Logical vector
ColNames	Column names specified by any of the following: • Numeric vector • Cell array of strings • Character array • Single string, which is used as a prefix for column names, with column numbers appended to the prefix • Logical true or false (default). If true, unique column names are assigned using the format col1, col2, col3, etc. If false, no column names are assigned.

	Note: The number of elements in *ColNames* must equal the number of elements in *ColIndices*.

Output Arguments

ReturnColNames	String or cell array of strings containing column names in *DMObj*.
DMObjNew	DataMatrix object created with names specified by *ColIndices* and *ColNames*.

Description

ReturnColNames = colnames(*DMObj*) returns *ReturnColNames*, a cell array of strings specifying the column names in *DMObj*, a DataMatrix object.

ReturnColNames = colnames(*DMObj*, *ColIndices*) returns the column names specified by *ColIndices*. *ColIndices* can be a positive integer, vector of positive integers, string specifying a column name, cell array of strings, or a logical vector.

DMObjNew = colnames(*DMObj*, *ColIndices*, *ColNames*) returns *DMObjNew*, a DataMatrix object with columns specified by *ColIndices* set to the names specified by *ColNames*. The number of elements in *ColIndices* must equal the number of elements in *ColNames*.

More About

- DataMatrix object

See Also
DataMatrix | rownames

combine

Class: bioma.data.ExptData
Package: bioma.data

Combine two ExptData objects

Syntax

NewEDObj = combine(*EDObj1*, *EDObj2*)

Description

NewEDObj = combine(*EDObj1*, *EDObj2*) combines data from two ExptData objects and returns a new ExptData object. The number and names of features (rows) in both ExptData objects must match. The number and names of samples (columns) in both ExptData objects must match.

Input Arguments

EDObj#

Object of the bioma.data.ExptData class.

Default:

See Also
bioma.data.ExptData

How To

·   "Representing Expression Data Values in ExptData Objects"

combine

Class: bioma.data.MetaData
Package: bioma.data

Combine two MetaData objects

Syntax

NewMDObj = combine(*MDObj1, MDObj2*)

Description

NewMDObj = combine(*MDObj1, MDObj2*) combines data from two MetaData objects
and returns a new MetaData object. The sample or feature names in the two MetaData
objects being combined must be unique. The variable names in the two MetaData objects
can be unique or the same. If a variable name is common to the two MetaData objects,
then the variable occupies one column in the new MetaData object. Variable names
unique to either of the two MetaData objects occupy their own column and contain values
only for the samples or features where the variable is present.

Input Arguments

MDObj#

Object of the bioma.data.MetaData class.

Default:

See Also
bioma.data.MetaData

How To
· "Representing Sample and Feature Metadata in MetaData Objects"

combine

Class: bioma.data.MIAME
Package: bioma.data

Combine two MIAME objects

Syntax

NewMIAMEObj = combine(*MIAMEObj1*, *MIAMEObj2*)

Description

NewMIAMEObj = combine(*MIAMEObj1*, *MIAMEObj2*) combines data from two MIAME objects and returns a new MIAME object. The combine method concatenates the properties of the two objects together.

Input Arguments

MIAMEObj#

Object of the bioma.data.MIAME class.

Default:

Examples

Construct two MIAME objects, and then combine them:

```
% Create a MATLAB structure containing GEO Series data
geoStruct1 = getgeodata('GSE4616');
% Create a second MATLAB structure containing GEO Series data
geoStruct2 = getgeodata('GSE11287');
% Import bioma.data package to make constructor function
% available
import bioma.data.*
```

```
% Construct MIAME object from the first structure
MIAMEObj1 = MIAME(geoStruct1);
% Construct MIAME object from the second structure
MIAMEObj2 = MIAME(geoStruct2);
% Combine the two MIAME objects
newMIAMEObj = combine(MIAMEObj1, MIAMEObj2)
```

See Also
```
bioma.data.MIAME
```

How To
- "Representing Experiment Information in a MIAME Object"

combine

Class: BioRead

Combine two objects

Syntax

NewObj = combine(*BioObj1*, *BioObj2*)
NewObj = combine(*BioObj1*, *BioObj2*, Name,Value)

Description

NewObj = combine(*BioObj1*, *BioObj2*) combines data from two objects of the same class and returns a new object. The combine method concatenates the properties of the two objects.

NewObj = combine(*BioObj1*, *BioObj2*, Name,Value) combines data from two objects of the same class with additional options specified by one or more Name,Value pair arguments.

Input Arguments

BioObj#

Object of the BioRead or BioMap class.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'Name'

String describing *NewObj*. This string populates the Name property of *NewObj*.

Default:

Output Arguments

NewObj

Object of the `BioRead` or `BioMap` class.

Examples

Construct two `BioRead` objects, and then combine them:

```
% Create two structures with data from a FASTQ file
struct1 = fastqread('SRR005164_1_50.fastq', 'blockread', [1 10],...
                    'trimheaders', true);
struct2 = fastqread('SRR005164_1_50.fastq', 'blockread', [11 20],...
                    'trimheaders', true);
% Construct two BioRead objects from the two structures
BRObj1 = BioRead(struct1);
BRObj2 = BioRead(struct2);
% Combine the two BioRead objects and set the Name property
% of the new object
NewBRObj = combine(BRObj1, BRObj2, 'Name', 'BRObj1 + BRObj2')

NewBRObj =

  BioRead with properties:

     Quality: {20x1 cell}
    Sequence: {20x1 cell}
      Header: {20x1 cell}
       NSeqs: 20
        Name: 'BRObj1 + BRObj2'
```

See Also
BioRead | BioMap

How To

· "Manage Short-Read Sequence Data in Objects"

Related Links

· Sequence Read Archive
· SAM format specification

conncomp (biograph)

Find strongly or weakly connected components in biograph object

Syntax

```
[S, C] = conncomp(BGObj)
[S, C] = conncomp(BGObj, ...'Directed', DirectedValue, ...)
[S, C] = conncomp(BGObj, ...'Weak', WeakValue, ...)
```

Arguments

BGObj	Biograph object created by biograph (object constructor).
DirectedValue	Property that indicates whether the graph is directed or undirected. Enter false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true. A DFS-based algorithm computes the connected components. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.
WeakValue	Property that indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set *WeakValue* to true to find weakly connected components. Default is false, which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.

Description

[*S*, *C*] = conncomp(*BGObj*) finds the strongly connected components of an N-by-N adjacency matrix extracted from a biograph object, *BGObj* using Tarjan's algorithm. A strongly connected component is a maximal group of nodes that are mutually reachable without violating the edge directions. The N-by-N sparse matrix represents a directed graph; all nonzero entries in the matrix indicate the presence of an edge.

The number of components found is returned in *S*, and *C* is a vector indicating to which component each node belongs.

Tarjan's algorithm has a time complexity of O(N+E), where N and E are the number of nodes and edges respectively.

[*S*, *C*] = conncomp(*BGObj*, ...'*PropertyName*', *PropertyValue*, ...) calls conncomp with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[*S*, *C*] = conncomp(*BGObj*, ...'Directed', *DirectedValue*, ...) indicates whether the graph is directed or undirected. Set *DirectedValue* to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true. A DFS-based algorithm computes the connected components. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.

[*S*, *C*] = conncomp(*BGObj*, ...'Weak', *WeakValue*, ...) indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set *WeakValue* to true to find weakly connected components. Default is false, which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.

Note: By definition, a single node can be a strongly connected component.

Note: A directed acyclic graph (DAG) cannot have any strongly connected components larger than one.

More About

- biograph object

References

[1] Tarjan, R.E., (1972). Depth first search and linear graph algorithms. SIAM Journal on Computing *1(2)*, 146–160.

[2] Sedgewick, R., (2002). Algorithms in C++, Part 5 Graph Algorithms (Addison-Wesley).

[3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also
```
biograph | allshortestpaths | isomorphism | maxflow | shortestpath |
traverse | graphconncomp | isdag | isspantree | minspantree | topoorder
```

cpgisland

Locate CpG islands in DNA sequence

Syntax

```
cpgStruct = cpgisland(SeqDNA)
cpgStruct = cpgisland(SeqDNA, ...'Window', WindowValue, ...)
cpgStruct = cpgisland(SeqDNA, ...'MinIsland', MinIslandValue, ...)
cpgStruct = cpgisland(SeqDNA, ...'GCmin', GCminValue, ...)
cpgStruct = cpgisland(SeqDNA, ...'CpGoe', CpGoeValue, ...)
cpgStruct = cpgisland(SeqDNA, ...'Plot', PlotValue, ...)
```

Input Arguments

SeqDNA	One of the following: • String of codes specifying a DNA nucleotide sequence • Row vector of integers specifying a DNA nucleotide sequence • MATLAB structure containing a `Sequence` field that contains a DNA nucleotide sequence, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, or `getgenbank` Valid characters include `A`, `C`, `G`, and `T`. `cpgisland` does not count ambiguous nucleotides or gaps.
WindowValue	Integer specifying the window size for calculating GC content and CpGobserved/CpGexpected ratios. Default is `100` bases. A smaller window size increases the noise in a plot.
MinIslandValue	Integer specifying the minimum number of consecutive marked bases to report as a CpG island. Default is `200` bases.
GCminValue	Value specifying the minimum GC percent in a window needed to mark a base. Choices are a value between `0` and `1`. Default is `0.5`.

CpGoeValue	Value specifying the minimum CpGobserved/CpGexpected ratio in each window needed to mark a base. Choices are a value between 0 and 1. Default is 0.6. This ratio is defined as: `CPGobs/CpGexp = (NumCpGs*Length)/(NumGs*NumCs)`
PlotValue	Controls the plotting of GC content, CpGoe content, CpG islands greater than the minimum island size, and all potential CpG islands for the specified criteria. Choices are true or false (default).

Output Arguments

cpgStruct	MATLAB structure containing the starting and ending bases of the CpG islands greater than the minimum island size.

Description

cpgStruct = cpgisland(*SeqDNA*) searches *SeqDNA*, a DNA nucleotide sequence, for CpG islands with a GC content greater than 50% and a CpGobserved/CpGexpected ratio greater than 60%. It marks bases meeting this criteria within a moving window of 100 DNA bases and then returns the results in *cpgStruct*, a MATLAB structure containing the starting and ending bases of the CpG islands greater than the minimum island size of 200 bases.

cpgStruct = cpgisland(*SeqDNA*, ...'*PropertyName*', *PropertyValue*, ...) calls cpgisland with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

cpgStruct = cpgisland(*SeqDNA*, ...'Window', *WindowValue*, ...) specifies the window size for calculating GC content and CpGobserved/CpGexpected ratios. Default is 100 bases. A smaller window size increases the noise in a plot.

cpgStruct = cpgisland(*SeqDNA*, ...'MinIsland', *MinIslandValue*, ...) specifies the minimum number of consecutive marked bases to report as a CpG island. Default is 200 bases.

cpgStruct = cpgisland(*SeqDNA*, ...'GCmin', *GCminValue*, ...) specifies the minimum GC percent in a window needed to mark a base. Choices are a value between 0 and 1. Default is 0.5.

cpgStruct = cpgisland(*SeqDNA*, ...'CpGoe', *CpGoeValue*, ...) specifies the minimum CpGobserved/CpGexpected ratio in each window needed to mark a base. Choices are a value between 0 and 1. Default is 0.6. This ratio is defined as:

```
CPGobs/CpGexp = (NumCpGs*Length)/(NumGs*NumCs)
```

cpgStruct = cpgisland(*SeqDNA*, ...'Plot', *PlotValue*, ...) controls the plotting of GC content, CpGoe content, CpG islands greater than the minimum island size, and all potential CpG islands for the specified criteria. Choices are true or false (default).

Examples

1   Import a nucleotide sequence from the GenBank database. For example, retrieve a sequence from *Homo sapiens* chromosome 12.

```
S = getgenbank('AC156455');
```

2   Calculate the CpG islands in the sequence and plot the results.

```
cpgisland(S.Sequence,'PLOT',true)

ans =

    Starts: [4510 29359]
     Stops: [5468 29604]
```

The CpG islands greater than 200 bases in length are listed and a plot displays.
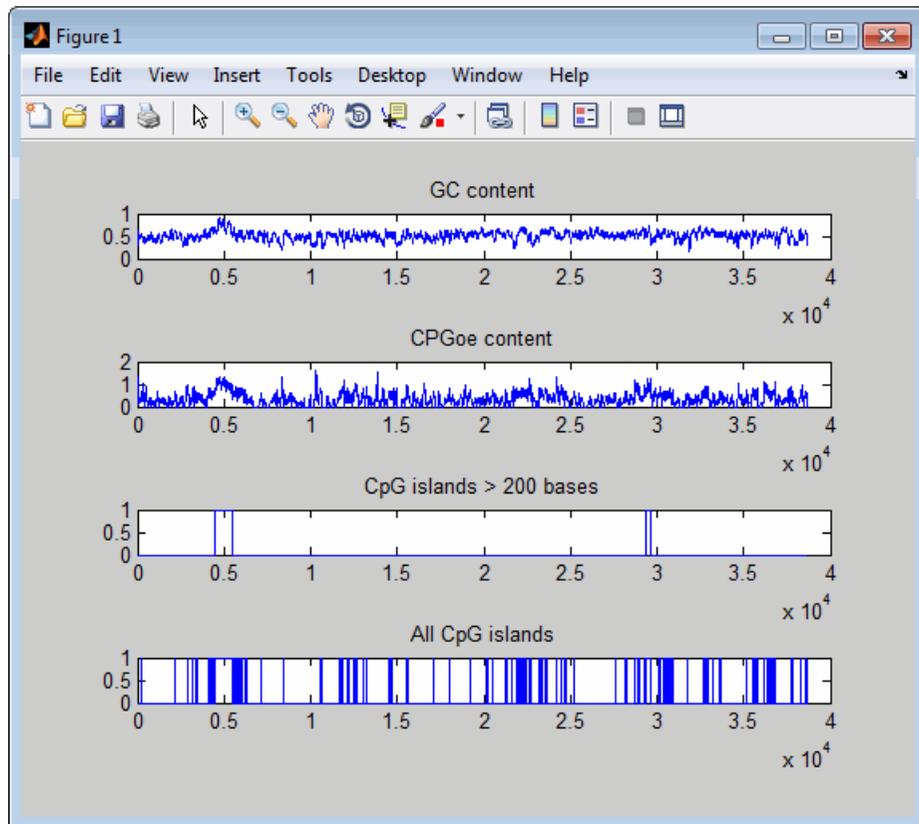
See Also

basecount | ntdensity | seqshoworfs

crossvalind

Generate cross-validation indices

Syntax

```
Indices = crossvalind('Kfold', N, K)
[Train, Test] = crossvalind('HoldOut', N, P)
[Train, Test] = crossvalind('LeaveMOut', N, M)
[Train, Test] = crossvalind('Resubstitution', N, [P,Q])
[...] = crossvalind(Method, Group, ...)
[...] = crossvalind(Method, Group, ..., 'Classes', C)
[...] = crossvalind(Method, Group, ..., 'Min', MinValue)
```

Description

Indices = `crossvalind('Kfold', N, K)` returns randomly generated indices for a K-fold cross-validation of `N` observations. `Indices` contains equal (or approximately equal) proportions of the integers `1` through `K` that define a partition of the `N` observations into `K` disjoint subsets. Repeated calls return different randomly generated partitions. `K` defaults to `5` when omitted. In K-fold cross-validation, `K-1` folds are used for training and the last fold is used for evaluation. This process is repeated `K` times, leaving one different fold for evaluation each time.

`[Train, Test] = crossvalind('HoldOut', N, P)` returns logical index vectors for cross-validation of N observations by randomly selecting `P*N` (approximately) observations to hold out for the evaluation set. `P` must be a scalar between `0` and `1`. `P` defaults to `0.5` when omitted, corresponding to holding `50%` out. Using holdout cross-validation within a loop is similar to K-fold cross-validation one time outside the loop, except that non-disjointed subsets are assigned to each evaluation.

`[Train, Test] = crossvalind('LeaveMOut', N, M)`, where M is an integer, returns logical index vectors for cross-validation of N observations by randomly selecting M of the observations to hold out for the evaluation set. M defaults to `1` when omitted. Using `'LeaveMOut'` cross-validation within a loop does not guarantee disjointed evaluation sets. To guarantee disjointed evaluation sets, use `'Kfold'` instead.

[Train, Test] = crossvalind('Resubstitution', N, [P,Q]) returns logical index vectors of indices for cross-validation of N observations by randomly selecting P*N observations for the evaluation set and Q*N observations for training. Sets are selected in order to minimize the number of observations that are used in both sets. P and Q are scalars between 0 and 1. Q=1-P corresponds to holding out (100*P)%, while P=Q=1 corresponds to full resubstitution. [P,Q] defaults to [1,1] when omitted.

[...] = crossvalind(Method, Group, ...) takes the group structure of the data into account. Group is a grouping vector that defines the class for each observation. Group can be a numeric vector, a string array, or a cell array of strings. The partition of the groups depends on the type of cross-validation: For K-fold, each group is divided into K subsets, approximately equal in size. For all others, approximately equal numbers of observations from each group are selected for the evaluation set. In both cases the training set contains at least one observation from each group.

[...] = crossvalind(Method, Group, ..., 'Classes', C) restricts the observations to only those values specified in C. C can be a numeric vector, a string array, or a cell array of strings, but it is of the same form as Group. If one output argument is specified, it contains the value 0 for observations belonging to excluded classes. If two output arguments are specified, both will contain the logical value false for observations belonging to excluded classes.

[...] = crossvalind(Method, Group, ..., 'Min', MinValue) sets the minimum number of observations that each group has in the training set. Min defaults to 1. Setting a large value for Min can help to balance the training groups, but adds partial resubstitution when there are not enough observations. You cannot set Min when using K-fold cross-validation.

Examples

Note: The crossvalind function creates random partitions, which depend on the state of the default random stream. Therefore, your results from the following examples will vary from those shown.

Create a 10-fold cross-validation to compute classification error.

```
load fisheriris
indices = crossvalind('Kfold',species,10);
```

```
cp = classperf(species);
for i = 1:10
    test = (indices == i); train = ~test;
    class = classify(meas(test,:),meas(train,:),species(train,:));
    classperf(cp,class,test)
end
cp.ErrorRate

ans =

    0.0200
```

Approximate a leave-one-out prediction error estimate.

```
load carbig
x = Displacement; y = Acceleration;
N = length(x);
sse = 0;
for i = 1:100
    [train,test] = crossvalind('LeaveMOut',N,1);
    yhat = polyval(polyfit(x(train),y(train),2),x(test));
    sse = sse + sum((yhat - y(test)).^2);
end
CVerr = sse / 100

CVerr =

    4.9750
```

Divide cancer data 60/40 without using the `'Benign'` observations. Assume groups are the true labels of the observations.

```
labels = {'Cancer','Benign','Control'};
groups = labels(ceil(rand(100,1)*3));
[train,test] = crossvalind('holdout',groups,0.6,'classes',...
                           {'Control','Cancer'});
sum(test) % Total groups allocated for testing

ans =

    35

sum(train) % Total groups allocated for training

ans =
```

26

More About

- knnclassify

See Also
```
classperf | classify | grp2idx | svmclassify
```

cytobandread

Read cytogenetic banding information

Syntax

CytoStruct = cytobandread(*File*)

Input Arguments

File	String specifying a file containing cytogenetic G-banding data, such as an NCBI ideogram text file or a UCSC Genome Browser cytoband text file.

Output Arguments

CytoStruct	Structure containing cytogenetic G-banding data in the following fields: • ChromLabels • BandStartBPs • BandEndBPs • BandLabels • GieStains

Description

CytoStruct = cytobandread(*File*) reads *File*, which is a string specifying a file containing cytogenetic G-banding data, and returns *CytoStruct*, which is a structure containing the following fields.

Field	Description
ChromLabels	Cell array containing the chromosome label (number or letter) on which each band is located.

Field	Description
BandStartBPs	Column vector containing the number of the base pair at the start of each band.
BandEndBPs	Column vector containing the number of the base pair at the end of each band.
BandLabels	Cell array containing the FISH label of each band, for example, p32.3.
GieStains	Cell array containing the Giemsa staining result for each band. Possible stain results depend on the species. For example, for *Homo sapiens*, the possibilities are: • gneg • gpos25 • gpos50 • gpos75 • gpos100 • acen • stalk • gvar

Tip You can download files containing cytogenetic G-banding data from the NCBI or UCSC Genome Browser ftp site. For example, you can download the cytogenetic banding data for *Homo sapiens* from:

ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/mapview/ideogram.gz
or

ftp://hgdownload.cse.ucsc.edu/goldenPath/hg18/database/cytoBandIdeo.txt.gz

Examples

Read the cytogenetic banding information for *Homo sapiens* into a structure.

```
hs_cytobands = cytobandread('hs_cytoBand.txt')
```

```
hs_cytobands =

    ChromLabels: {862x1 cell}
   BandStartBPs: [862x1 int32]
     BandEndBPs: [862x1 int32]
     BandLabels: {862x1 cell}
      GieStains: {862x1 cell}
```

See Also
chromosomeplot

DataMatrix object

Data structure encapsulating data and metadata from microarray experiment so that it can be indexed by gene or probe identifiers and by sample identifiers

Description

A DataMatrix object is a data structure encapsulating measurement data and feature metadata from a microarray experiment so that it can be indexed by gene or probe identifiers and by sample identifiers. A DataMatrix object stores experimental data in a matrix, with rows typically corresponding to gene names or probe identifiers, and columns typically corresponding to sample identifiers. A DataMatrix object also stores metadata, such as the gene names or probe identifiers and sample identifiers, in row names and column names.

You create a DataMatrix object using the object constructor function `DataMatrix`.

Property Summary

Properties of a DataMatrix Object

Property	Description
Name	String that describes the DataMatrix object. Default is `''`.
RowNames	Empty array or cell array of strings that specifies the names for the rows, typically gene names or probe identifiers. The number of elements in the cell array must equal the number of rows in the matrix. Default is an empty array.
ColNames	Empty array or cell array of strings that specifies the names for the columns, typically sample identifiers. The number of elements in the cell array must equal the number of columns in the matrix.
NRows	Read-only. Positive number that specifies the number of rows in the matrix. **Note:** You cannot modify this property directly. You can access it using the `get` method.

Property	Description
NCols	Read-only. Positive number that specifies the number of columns in the matrix. **Note:** You cannot modify this property directly. You can access it using the `get` method.
NDims	Read-only. Positive number that specifies the number of dimensions in the matrix. **Note:** You cannot modify this property directly. You can access it using the `get` method.
ElementClass	Read-only. String that specifies the class type of the elements in the DataMatrix object, such as `single` or `double`. **Note:** You cannot modify this property directly. You can access it using the `get` method.

Method Summary

General Methods of a DataMatrix Object

Method	Description
colnames	Retrieve or set column names of DataMatrix object.
disp	Display DataMatrix object.
display	Display DataMatrix object, printing DataMatrix object name. To invoke this method, enter the name of a DataMatrix object at the command prompt.
dmwrite	Write DataMatrix object to text file.
double	Convert DataMatrix object to double-precision array.
get	Retrieve information about DataMatrix object.
isempty	Determine if DataMatrix object is empty.
isfinite	Determine if DataMatrix object elements are finite.

Method	Description
isinf	Determine if DataMatrix object elements are infinite.
isnan	Determine if DataMatrix object elements are NaN.
isscalar	Determine if DataMatrix object is scalar.
isequal	Test DataMatrix objects for equality.
isequaln	Test DataMatrix objects for equality, treating NaNs as equal.
isvector	Determine if DataMatrix object is vector.
length	Return length of DataMatrix object.
ndims	Return number of dimensions in DataMatrix object.
numel	Return number of elements in DataMatrix object.
plot	Draw 2-D line plot of DataMatrix object.
rownames	Retrieve or set row names of DataMatrix object.
set	Set property of DataMatrix object.
single	Convert DataMatrix object to single-precision array.
size	Return size of DataMatrix object.

Methods for Manipulating the Data in a DataMatrix Object

Method	Description
cat	Concatenate DataMatrix objects. The horzcat and vertcat methods implement special cases.
horzcat	Concatenate DataMatrix objects horizontally.
sortcols	Sort columns of DataMatrix object in ascending or descending order.
sortrows	Sort rows of DataMatrix object in ascending or descending order.
subsasgn	Subscripted assignment for DataMatrix object. To invoke this method, use parentheses or dot indexing described in "Accessing Data in DataMatrix Objects".

Method	Description
subsref	Subscripted reference for DataMatrix object. To invoke this method, use parentheses or dot indexing described in "Accessing Data in DataMatrix Objects".
transpose	Transpose DataMatrix object.
vertcat	Concatenate DataMatrix objects vertically.

Descriptive Statistics and Statistical Learning Methods

Method	Description
kmeans	K-means clustering.
max	Return maximum values in DataMatrix object.
mean	Return average or mean values in DataMatrix object.
median	Return median values in DataMatrix object.
min	Return minimum values in DataMatrix object.
nanmax	Return maximum values in DataMatrix object ignoring NaN values.
nanmean	Return average or mean values in DataMatrix object ignoring NaN values.
nanmedian	Return median values in DataMatrix object ignoring NaN values.
nanmin	Return minimum values in DataMatrix object ignoring NaN values.
nanstd	Return standard deviation values in DataMatrix object ignoring NaN values.
nansum	Return sum of elements in DataMatrix object ignoring NaN values.
nanvar	Return variance values in DataMatrix object ignoring NaN values.
pca	Principal component analysis on data.
pdist	Pairwise distance.
std	Return standard deviation values in DataMatrix object.
sum	Return sum of elements in DataMatrix object.

Method	Description
var	Return variance values in DataMatrix object.

Unary Methods — Exponential

Method	Description
exp	Exponential.
log	Natural logarithm.
log10	Common (base 10) logarithm.
log2	Base 2 logarithm and dissect floating-point numbers into exponent and mantissa.
pow2	Base 2 power and scale floating-point numbers.
sqrt	Square root.

Unary Methods — Integer

Method	Description
ceil	Round DataMatrix object toward infinity.
fix	Round DataMatrix object toward zero.
floor	Round DataMatrix object toward minus infinity.
round	Round DataMatrix object to nearest integer.

Unary Methods — Custom

Method	Description
dmarrayfun	Apply function to each element in DataMatrix object.

Binary Methods — Arithmetic Operator

Operator	Method	Description
+	plus	Add DataMatrix objects
-	minus	Subtract DataMatrix objects.
.*	times	Multiply DataMatrix objects.
./	rdivide	Right array divide DataMatrix objects.
.\	ldivide	Left array divide DataMatrix objects.

Operator	Method	Description
.^	power	Array power DataMatrix objects.

Binary Methods — Relational Operator

Operator	Method	Description
<	lt	Test DataMatrix objects for less than.
<=	le	Test DataMatrix objects for less than or equal to.
>	gt	Test DataMatrix objects for greater than.
>=	ge	Test DataMatrix objects for greater than or equal to.
==	eq	Test DataMatrix objects for equality.
~=	ne	Test DataMatrix objects for inequality.

Binary Methods — Custom

Method	Description
dmbsxfun	Apply element-by-element binary operation to two DataMatrix objects with singleton expansion enabled.

Examples

Determining Properties and Property Values of a DataMatrix Object

You can display all properties and their current values of a DataMatrix object, *DMobj*, by using the following syntax:

get(*DMobj*)

You can return all properties and their current values of *DMobj*, a DataMatrix object, to *DMstruct*, a scalar structure in which each field name is a property of a DataMatrix object, and each field contains the value of that property, by using the following syntax:

DMstruct = get(*DMobj*)

You can return the value of a specific property of a DataMatrix object, *DMobj*, by using either of the following syntaxes:

PropertyValue = get(*DMObj*, '*PropertyName*')

```
PropertyValue = DMObj.PropertyName
```

You can return the value of specific properties of a DataMatrix object, *DMobj*, by using the following syntax:

```
[Property1Value, Property2Value, ...] = get(DMobj, ...
'Property1Name', 'Property2Name', ...)
```

Determining Possible Values of DataMatrix Object Properties

You can display possible values for all properties that have a fixed set of property values in a DataMatrix object, *DMobj*, by using the following syntax:

```
set(DMobj)
```

You can display possible values for a specific property that has a fixed set of property values in a DataMatrix object, *DMobj*, by using the following syntax:

```
set(DMObj, 'PropertyName')
```

Specifying Properties of a DataMatrix Object

You can set a specific property of a DataMatrix object, *DMObj*, by using either of the following syntaxes:

```
DMObj = set(DMObj, 'PropertyName', PropertyValue)
```

```
DMObj.PropertyName = PropertyValue
```

You can set multiple properties of a DataMatrix object, *DMobj*, by using the following syntax:

```
set(DMobj, 'PropertyName1', PropertyValue1, ...
    'PropertyName2', PropertyValue2, ...)
```

Note: For more examples of creating and using DataMatrix objects, see "Representing Expression Data Values in DataMatrix Objects".

See Also

```
DataMatrix | disp | dmbsxfun | double | ge | get | horzcat | isequaln | le |
lt | mean | min | ndims | ne | plot | plus | rdivide | set | sortcols | std |
sum | var | colnames | dmarrayfun | dmwrite | eq | gt | isequal | ldivide |
```

max | median | minus | numel | power | rownames | single | sortrows | times |
vertcat

DataMatrix

Create DataMatrix object

Syntax

```
DMobj = DataMatrix(Matrix)
DMobj = DataMatrix(Matrix, RowNames, ColumnNames)
DMobj = DataMatrix('File', FileName)
DMobj = DataMatrix(..., 'RowNames', RowNamesValue, ...)
DMobj = DataMatrix(..., 'ColNames', ColNamesValue, ...)
DMobj = DataMatrix(..., 'Name', NameValue, ...)
DMobj = DataMatrix('File', FileName, ...'Delimiter',
DelimiterValue, ...)
DMobj = DataMatrix('File', FileName, ...'HLine', HLineValue, ...)
DMobj = DataMatrix('File', FileName, ...'Rows', RowsValue, ...)
DMobj = DataMatrix('File', FileName, ...'Columns',
ColumnsValue, ...)
```

Arguments

Matrix	Two-dimensional numeric or logical array.
RowNames	Row names for the DataMatrix object, specified by a numeric vector, character array, or cell array of strings, whose elements are equal in number to the number of rows in *Matrix*. *RowNames* are typically gene names or probe identifiers from a microarray experiment.
	Note: The row names do not need to be unique.
ColumnNames	Column names for the DataMatrix object, specified by a numeric vector, character array, or cell array of strings, whose elements are equal in number to the number of columns in *Matrix*. *ColumnNames* are typically sample identifiers from a microarray experiment.

	Note: The column names do not need to be unique.
FileName	String specifying a file name or a path and file name of a tab-delimited TXT or XLS file that contains table-oriented data and metadata.
	Note: Typically, the first row of the table contains column names, the first column contains row names, and the numeric data starts at the 2,2 position. The DataMatrix function will detect if the first column does not contain row names, and read data from the first column. However, if the first row does not contain header text (column names), set the HLine property to 0.
RowNamesValue, *ColNamesValue*	Row names or column names for the DataMatrix object. Choices are: • Numeric vector, character array, or a cell array of strings, whose elements are equal in number to the number of rows or number of columns of numeric data in the input matrix. • A single string, which is used as a prefix for row or column names. Numbers will be appended to the prefix. • true — Unique row or column names will be assigned using the formats row1, row2, row3, etc., or col1, col2, col3, etc. • false — Default. No row or column names are assigned.
	Note: The row or column names do not need to be unique.
NameValue	String specifying a name for the DataMatrix object. Default is ' '.

DelimiterValue	String specifying a delimiter symbol to use for the input file. Typical choices are: • `' '` • `'\t'` (default) • `','` • `';'` • `'|'`
HLineValue	Positive integer that specifies which row of the input file contains the column header text (column names). Default is `1`. When creating the DataMatrix object *DMobj*, the `DataMatrix` function loads data from (*HLineValue* + `1`) to the end of the file. **Tip** If the input file does not contain column header text (column names), set *HLineValue* to `0`.
RowsValue, *ColumnsValue*	A subset of rows or columns in `File`, for the `DataMatrix` function to use to create the DataMatrix object. Choices are: • Cell array of strings • Character array • Numeric or logical vector

Description

A DataMatrix object encapsulates measurement data and feature metadata from a microarray experiment so that it can be indexed by gene names or probe identifiers and by sample identifiers. For examples of creating and using DataMatrix objects, see "Representing Expression Data Values in DataMatrix Objects".

Note: The DataMatrix constructor function is part of the microarray object package. To make it available, type the following in the MATLAB command line:

```
import bioma.data.*
```

Otherwise, use `bioma.data.DataMatrix` instead of `DataMatrix`, in the following syntaxes.

DMobj = `DataMatrix`(*Matrix*) creates a DataMatrix object, *DMobj*, from *Matrix*, a two-dimensional numeric or logical array. *Matrix* can also be a DataMatrix object.

DMobj = `DataMatrix`(*Matrix*, *RowNames*, *ColumnNames*) creates a DataMatrix object, *DMobj*, from *Matrix*, a two-dimensional numeric or logical array, with row names and column names specified by *RowNames* and *ColumnNames*. *RowNames* and *ColumnNames* can be a numeric vector, character array, or cell array of strings, whose elements are equal in number to the number of rows and number of columns, respectively, in *Matrix*. *RowNames* are typically gene names or probe identifiers, while *ColumnNames* are typically sample identifiers.

Note: The row or column names do not need to be unique.

DMobj = `DataMatrix`('File', *FileName*) creates a DataMatrix object, *DMobj*, from *FileName*, a string specifying a file name or a path and file name of a tab-delimited TXT or XLS file that contains table-oriented data and metadata.

Note: Typically, the first row of the table contains column names, the first column contains row names, and the numeric data starts at the 2,2 position. The `DataMatrix` function will detect if the first column does not contain row names, and read data from the first column. However, if the first row does not contain header text (column names), set the `HLine` property to 0.

DMobj = `DataMatrix`(..., '*PropertyName*', *PropertyValue*, ...) calls `DataMatrix` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

DMobj = `DataMatrix`(..., 'RowNames', *RowNamesValue*, ...) specifies row names for *DMobj*. *RowNamesValue* can be any of the following:

• Numeric vector, character array, or a cell array of strings, whose elements are equal in number to the number of rows of numeric data in the input matrix.

- A single string, which is used as a prefix for row names. Row numbers will be appended to the prefix.
- `true` — Unique row names will be assigned using the format `row1`, `row2`, `row3`, etc.
- `false` — Default. No row names are assigned.

Note: The row names do not need to be unique.

DMobj = DataMatrix(..., 'ColNames', *ColNamesValue*, ...) specifies column names for *DMobj*. *ColNamesValue* can be any of the following:

- Numeric vector, character array, or a cell array of strings, whose elements are equal in number to the number of columns of numeric data in the input matrix.
- A single string, which is used as a prefix for column names. Column numbers will be appended to the prefix.
- `true` — Unique column names will be assigned using the format `col1`, `col2`, `col3`, etc.
- `false` — Default. No column names are assigned.

Note: The column names do not need to be unique.

DMobj = DataMatrix(..., 'Name', *NameValue*, ...) specifies a name for *DMobj*. Default is ''.

DMobj = DataMatrix('File', *FileName*, ...'Delimiter', *DelimiterValue*, ...) specifies a delimiter symbol to use for the input file. Typical choices are:

- `' '`
- `'\t'` (default)
- `','`
- `';'`
- `'|'`

DMobj = DataMatrix('File', *FileName*, ...'HLine', *HLineValue*, ...) specifies which row of the input file contains the column header text (column names).

HLineValue is a positive integer. Default is 1. When creating the DataMatrix object *DMobj*, the DataMatrix function loads data from (*HLineValue* + 1) to the end of the file.

Tip If the input file does not contain column header text (column names), set *HLineValue* to 0.

DMobj = DataMatrix('File', *FileName*, ...'Rows', *RowsValue*, ...) specifies a subset of row names in File for the DataMatrix function to use to create *DMobj*. *RowsValue* can be a cell array of strings, a character array, or a numeric or logical vector.

DMobj = DataMatrix('File', *FileName*, ...'Columns', *ColumnsValue*, ...) specifies a subset of column names in File for the DataMatrix function to use to create *DMobj*. *ColumnsValue* can be a cell array of strings, a character array, or a numeric or logical vector.

Examples

For examples of creating and using DataMatrix objects, see "Representing Expression Data Values in DataMatrix Objects".

More About

- DataMatrix object

See Also

disp | dmbsxfun | double | ge | get | horzcat | isequaln | le | lt | mean | min | ndims | ne | plot | plus | rdivide | set | sortcols | std | sum | var | colnames | dmarrayfun | dmwrite | eq | gt | isequal | ldivide | max | median | minus | numel | power | rownames | single | sortrows | times | vertcat

dayhoff

Return Dayhoff scoring matrix

Syntax

ScoringMatrix = dayhoff

Description

ScoringMatrix = dayhoff returns a PAM250 type scoring matrix. The order of amino acids in the matrix is A R N D C Q E G H I L K M F P S T W Y V B Z X *.

See Also

blosum | gonnet | localalign | nuc44 | nwalign | pam | swalign

dimercount

Count dimers in nucleotide sequence

Syntax

```
Dimers = dimercount(SeqNT)
[Dimers, Percent] = dimercount(SeqNT)
... = dimercount(SeqNT, 'Ambiguous', AmbiguousValue)
... = dimercount(SeqNT, 'Chart', ChartValue)
```

Input Arguments

SeqNT	One of the following: • String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers. • Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes. • MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, or `getgenbank`. Examples: `'ACGT'` or `[1 2 3 4]`
AmbiguousValue	String specifying how to treat dimers containing ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N). Choices are: • `'ignore'` (default) — Skips dimers containing ambiguous characters • `'bundle'` — Counts dimers containing ambiguous characters and reports the total count in the `Ambiguous` field of the *Dimers* output structure.

	• `'prorate'` — Counts dimers containing ambiguous characters and distributes them proportionately in the appropriate dimer fields containing standard nucleotide characters. For example, the counts for the dimer AR are distributed evenly between the AA and AG fields.
	• `'warn'` — Skips dimers containing ambiguous characters and displays a warning.
ChartValue	String specifying a chart type. Choices are `'pie'` or `'bar'`.

Output Arguments

Dimers	MATLAB structure containing the fields AA, AC, AG, AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, and TT, which contain the dimer counts in *SeqNT*.
Percent	A 4-by-4 matrix with the relative proportions of the dimers in *SeqNT*. The rows correspond to A, C, G, and T in the first element of the dimer, and the columns correspond to A, C, G, and T in the second element of the dimer.

Description

Dimers = dimercount(*SeqNT*) counts the nucleotide dimers in *SeqNT*, a nucleotide sequence, and returns the dimer counts in *Dimers*, a MATLAB structure containing the fields AA, AC, AG, AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, and TT.

- For sequences that have dimers with the character U, these dimers are added to the corresponding dimers containing a T.

- If the sequence contains gaps indicated by a hyphen (-), the gaps are ignored, and the two characters on either side of the gap are counted as a dimer.

- If the sequence contains unrecognized characters, then dimers containing these characters are ignored, and the following warning message appears:

  ```
  Warning: Unknown symbols appear in the sequence. These will be ignored.
  ```

[*Dimers*, *Percent*] = dimercount(*SeqNT*) returns *Percent*, a 4-by-4 matrix with the relative proportions of the dimers in *SeqNT*. The rows correspond to A, C, G, and T in

the first element of the dimer, and the columns correspond to A, C, G, and T in the second element of the dimer.

`... = dimercount(`*`SeqNT`*`, 'Ambiguous',` *`AmbiguousValue`*`)` specifies how to treat dimers containing ambiguous nucleotide characters. Choices are:

- `'ignore'` (default)
- `'bundle'`
- `'prorate'`
- `'warn'`

`... = dimercount(`*`SeqNT`*`, 'Chart',` *`ChartValue`*`)` creates a chart showing the relative proportions of the dimers. *ChartValue* can be `'pie'` or `'bar'`.

Examples

Count the dimers in a nucleotide sequence and display a matrix of the percentage of each dimer.

```
[Dimers, Percent] = dimercount('TAGCTGGCCAAGCGAGCTTG')

Dimers =

    AA: 1
    AC: 0
    AG: 3
    AT: 0
    CA: 1
    CC: 1
    CG: 1
    CT: 2
    GA: 1
    GC: 4
    GG: 1
    GT: 0
    TA: 1
    TC: 0
    TG: 2
    TT: 1
```

```
Percent =

    0.0526         0    0.1579         0
    0.0526    0.0526    0.0526    0.1053
    0.0526    0.2105    0.0526         0
    0.0526         0    0.1053    0.0526
```

See Also

aacount | basecount | baselookup | codoncount | nmercount | ntdensity

disp (DataMatrix)

Display DataMatrix object

Syntax

disp(*DMObj*)

Arguments

DMObj	DataMatrix object, such as created by DataMatrix (object constructor).

Description

disp(*DMObj*) displays the DataMatrix object *DMObj*, including row names and column names, without printing the DataMatrix object name.

More About

·   DataMatrix object

See Also
DataMatrix

dmarrayfun (DataMatrix)

Apply function to each element in DataMatrix object

Syntax

```
DMObjNew1 = dmarrayfun(Func, DMObj1)
DMObjNew1 = dmarrayfun(Func, DMObj1, DMObj2, ...)
[DMObjNew1, DMObjNew2, ...] = dmarrayfun(Func, DMObj1, ...)
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'UniformOutput',
UniformOutputValue, ...)
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'DataMatrixOutput',
DataMatrixOutputValue, ...)
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'Rows',
RowsValue, ...)
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'Columns',
ColumnsValue, ...)
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'ErrorHandler',
ErrorHandlerValue, ...)
```

Input Arguments

Func	Function handle for a function that returns one or more scalars, and returns values of the same class each time it is called.
DMObj1	DataMatrix object, such as created by DataMatrix (object constructor).
DMObj2	Either of the following: • DataMatrix object, such as created by DataMatrix (object constructor) • MATLAB numeric array

	Note: *DMObj2* and subsequent input objects or arrays must be the same size (number of rows and columns) as *DMObj1*.
UniformOutputValue	Specifies whether *Func* must return output values without encapsulation in a cell array. Choices are true (default) or false. If true, dmarrayfun must return scalar values that can be concatenated into an array. These values can also be a cell array. If false, dmarrayfun returns a cell array (or multiple cell arrays), where the I,Jth cell contains the value equal to *Func*(*DMObj1*(I,J),...).
DataMatrixOutputValue	Specifies whether return values must be DataMatrix objects. Choices are true (default) or false. If you set the 'UniformOutput' property to false, this property is ignored.
RowsValue, *ColumnsValue*	Specifies the rows or columns to which to apply the function. Choices are: • Positive integer • Vector of positive integers • String specifying a row or column name • Cell array of strings • Logical vector
ErrorHandlerValue	Specifies a function handle to a function that dmarrayfun calls if the call to *Func* fails.

Output Arguments

DMObjNew1, *DMObjNew2*	DataMatrix objects created from applying the function to each element in one or more DataMatrix objects. The size (number of rows and columns), row names, and column names will be the same as *DMObj1*.

Description

DMObjNew1 = dmarrayfun(*Func, DMObj1*) applies the function specified by *Func* to each element in *DMObj1*, a DataMatrix object, and returns the results in *DMObjNew1*, a new DataMatrix object. *DMObjNew1* has the same size (number of rows and columns), row names, and column names as *DMObj1*. The I,Jth element of *DMObjNew1* is equal to *Func*(*DMObj1*(I,J)), where *Func* is a function handle for a function that takes one input argument, returns one scalar value, and returns values of the same class each time it is called.

DMObjNew1 = dmarrayfun(*Func, DMObj1, DMObj2, ...*) evaluates the function specified by *Func* using elements in *DMObj1, DMObj2*, etc. as input arguments. The I,Jth element of *DMObjNew1* is equal to *Func*(*DMObj1*(I,J), *DMObj2*(I,J), ...), where *Func* is a function handle for a function that takes multiple input arguments, returns one scalar, and returns values of the same class each time it is called.

[*DMObjNew1, DMObjNew2, ...*] = dmarrayfun(*Func, DMObj1, ...*) evaluates the function specified by *Func* using elements in *DMObj1*, and possibly other input arguments. *Func* is a function handle for a function that takes one or more input arguments, returns multiple scalars, and returns values of the same class each time it is called. It returns DataMatrix objects *DMObjNew1*, *DMObjNew2*, etc. with each one corresponding to one of the outputs of *Func*. The outputs of *Func* may be of different classes, however, but each output must be the same each time it is called.

[*DMObjNew1, ...*] = dmarrayfun(*Func, DMObj1, ...*'*PropertyName*', *PropertyValue, ...*) calls dmarrayfun with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

[*DMObjNew1, ...*] = dmarrayfun(*Func, DMObj1, ...*'UniformOutput', *UniformOutputValue, ...*) specifies whether *Func* must return output values without encapsulation in a cell array. Choices are true (default) or false. If true, dmarrayfun must return scalar values that can be concatenated into an array. These values can also be a cell array. If false, dmarrayfun returns a cell array (or multiple cell arrays), where the I,Jth cell contains the value equal to *Func*(*DMObj1*(I,J),...).

[*DMObjNew1, ...*] = dmarrayfun(*Func, DMObj1, ...*'DataMatrixOutput', *DataMatrixOutputValue, ...*) specifies whether return values must be DataMatrix objects. Choices are true (default) or false. If you set the 'UniformOutput' property to false, this property is ignored.

[*DMObjNew1*, ...] = dmarrayfun(*Func*, *DMObj1*, ...'Rows', *RowsValue*, ...) applies the function only to the rows in the DataMatrix object specified by *RowsValue*, which can be a positive integer, vector of positive integers, string specifying a row name, cell array of strings, or a logical vector.

[*DMObjNew1*, ...] = dmarrayfun(*Func*, *DMObj1*, ...'Columns', *ColumnsValue*, ...) applies the function only to the columns in the DataMatrix object specified by *ColsValue*, which can be a positive integer, vector of positive integers, string specifying a column name, cell array of strings, or a logical vector.

[*DMObjNew1*, ...] = dmarrayfun(*Func*, *DMObj1*, ...'ErrorHandler', *ErrorHandlerValue*, ...) specifies a function handle to a function that dmarrayfun calls if the call to *Func* fails. The error handling function will be called with these input arguments:

- Structure with the following fields:

  - identifier — Identifier of the error
  - message — Error message text
  - index — Linear index into the input array(s) at which the error occurred
- Set of input arguments at which the call to the function failed

If you do not specify *ErrorHandlerValue*, dmarrayfun rethrows the error from the call to *Func*.

More About

- DataMatrix object

See Also

DataMatrix | dmbsxfun | arrayfun

dmbsxfun (DataMatrix)

Apply element-by-element binary operation to two DataMatrix objects with singleton expansion enabled

Syntax

DMObjNew = dmbsxfun(*Func*, *DMObj1*, *DMObj2*)

Input Arguments

Func	Function handle for a function or a built-in function. For more information on built-in functions, see bsxfun.
DMObj1, *DMObj2*	Either of the following: • DataMatrix object, such as created by DataMatrix (object constructor) • MATLAB numeric array At least one of these input arguments must be a DataMatrix object.

Output Arguments

DMObjNew	DataMatrix object or MATLAB numeric array created from element-by-element binary operation of two DataMatrix objects with singleton expansion enabled.

Description

DMObjNew = dmbsxfun(*Func*, *DMObj1*, *DMObj2*) applies an element-by-element binary operation to the DataMatrix objects *DMObj1* and *DMObj2*, with singleton expansion enabled. *Func* is a function handle, and can be for a function or a built-in function. For more information on built-in functions, see bsxfun.

DMObj1 and *DMObj2* can be DataMatrix objects or MATLAB numeric arrays; however, at least one of these input arguments must be a DataMatrix object. *DMObj1* and *DMObj2* must have the same number of rows or the same number or columns. If they don't have the same number of rows, then one must be a row vector and its rows are expanded down to be equal to the larger matrix. If they don't have the same number of columns, then one must be a column vector and its columns are expanded across to be equal to the larger matrix.

DMObjNew is a DataMatrix object, unless the larger input argument is a MATLAB numeric array; then *DMObjNew* is also a numeric array. The size (number of rows and columns) of *DMObjNew* is equal to the larger of the two input arguments. The row names and column names of *DMObjNew* come from the larger input argument, or, if both inputs are the same size, from the first input argument.

Examples

1. Use the `DataMatrix` constructor function to create a DataMatrix object.

```
A = bioma.data.DataMatrix(magic(3), 'RowNames', true, ...
                          'ColNames',true)
```

2. Use the built-in function `@minus` to subtract the column means from this DataMatrix object.

```
A = dmbsxfun(@minus, A, mean(A))
```

More About

· DataMatrix object

See Also

`DataMatrix | bsxfun`

dmNames

Class: bioma.data.ExptData
Package: bioma.data

Retrieve or set Name properties of DataMatrix objects in ExptData object

Syntax

DMNames = dmNames(*EDObj*)
DMNames = dmNames(*EDObj*, *Subset*)
NewEDObj = dmNames(*EDObj*, *Subset*, *NewDMNames*)

Description

DMNames = dmNames(*EDObj*) returns a cell array of strings specifying the Name properties of all the DataMatrix objects in an ExptData object.

DMNames = dmNames(*EDObj*, *Subset*) returns a cell array of strings specifying the Name properties of a subset of the DataMatrix objects in an ExptData object.

NewEDObj = dmNames(*EDObj*, *Subset*, *NewDMNames*) replaces the Name properties of DataMatrix objects specified by *Subset* in *EDObj*, an ExptData object, with *NewDMNames*, and returns *NewEDObj*, a new ExptData object.

Input Arguments

EDObj

Object of the bioma.data.ExptData class.

Default:

Subset

One of the following to specify the names of a subset of the DataMatrix objects in an ExptData object:

- String specifying a name

- Cell array of strings specifying names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewDMNames

New names for specific DataMatrix objects within an ExptData object, specified by one of the following:

- Numeric vector
- String or cell array of strings
- String, which `dmNames` uses as a prefix for the DataMatrix object names, with numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `dmNames` assigns unique names using the format `DM1`, `DM2`, etc.

The number of elements in *NewDMNames* must equal the number of DataMatrix objects specified by *Subset*.

Default:

Output Arguments

DMNames

Cell array of strings specifying the names of all or some of the DataMatrix objects in an ExptData object.

NewEDObj

Object of the `bioma.data.ExptData` class, returned after replacing names of specific DataMatrix objects.

Examples

Construct an ExptData object, and then retrieve the names of DataMatrix objects from it:

```
% Import bioma.data package to make constructor functions
% available
import bioma.data.*
% Create DataMatrix object from .txt file containing
% expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Retrieve DataMatrix object names
DMNames = dmNames(EDObj);
```

See Also

bioma.data.ExptData | DataMatrix | elementNames | featureNames |
sampleNames

How To

- "Representing Expression Data Values in ExptData Objects"

dmwrite (DataMatrix)

Write DataMatrix object to text file

Syntax

```
dmwrite(DMObj, File)
dmwrite(..., 'Delimiter', DelimiterValue, ...)
dmwrite(..., 'Precision', PrecisionValue, ...)
dmwrite(..., 'Header', HeaderValue, ...)
dmwrite(..., 'Annotated', AnnotatedValue, ...)
dmwrite(..., 'Append', AppendValue, ...)
```

Arguments

DMObj	DataMatrix object, such as created by `DataMatrix` (object constructor).
File	String specifying either a file name or a path and file name for saving the text file.
DelimiterValue	String specifying a delimiter symbol to use as a matrix column separator. Typical choices are: • `' '` • `'\t'` (default) • `','` • `';'` • `'|'`
PrecisionValue	Precision for writing the data to the text file, specified by either: • Positive integer specifying the number of significant digits • C-style format string starting with `%`, such as `'%6.5f'` Default is `5`.
HeaderValue	String specifying the first line of the text file. Default is the `Name` property for the DataMatrix object.

AnnotatedValue	Controls the writing of row and column names to the text file. Choices are `true` (default) or `false`.
AppendValue	Controls the appending of *DMObj* to *File* when it is an existing file. Choices are `true` or `false` (default). If `false`, `dmwrite` overwrites *File*.

Description

dmwrite(*DMObj*, *File*) writes a DataMatrix object to a text file using the delimiter \t to separate DataMatrix columns. dmwrite writes the data starting at the first column of the first row in the destination file.

dmwrite(..., '*PropertyName*', *PropertyValue*, ...) calls dmwrite with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

dmwrite(..., 'Delimiter', *DelimiterValue*, ...) specifies a delimiter symbol to use as a column separator for separating matrix columns. Default is '\t'.

dmwrite(..., 'Precision', *PrecisionValue*, ...) specifies the precision for writing the data to the text file. Default is 5.

dmwrite(..., 'Header', *HeaderValue*, ...) specifies the first line of the text file. Default is the Name property for the DataMatrix object.

dmwrite(..., 'Annotated', *AnnotatedValue*, ...) controls the writing of row and column names to the text file. Choices are `true` (default) or `false`.

dmwrite(..., 'Append', *AppendValue*, ...) controls the appending of *DMObj* to *File* when it is an existing file. Choices are `true` or `false` (default). If `false`, dmwrite overwrites *File*.

Examples

Create a DataMatrix object and write the contents to a text file:

```
% Create a DataMatrix object
dmobj = bioma.data.DataMatrix(rand(2,3), {'Row1', 'Row2'}, ...
                                    {'Col1', 'Col2', 'Col3'})
% Write the DataMatrix object to a text file
dmwrite(dmobj,'testdm.txt')
```

More About

- DataMatrix object

See Also

DataMatrix

dna2rna

Convert DNA sequence to RNA sequence

Syntax

SeqRNA = dna2rna(*SeqDNA*)

Arguments

SeqDNA	DNA sequence specified by any of the following: • Character string with the characters A, C, G, T, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N, • Row vector of integers from the table Mapping Nucleotide Integers to Letter Codes. • MATLAB structure containing a Sequence field that contains a DNA sequence, such as returned by fastaread, fastqread, emblread, getembl, genbankread, or getgenbank.

Description

SeqRNA = dna2rna(*SeqDNA*) converts a DNA sequence to an RNA sequence by converting any thymine nucleotides (T) in the DNA sequence to uracil nucleotides (U). The RNA sequence is returned in the same format as the DNA sequence. For example, if *SeqDNA* is a vector of integers, then so is *SeqRNA*.

Examples

Convert a DNA sequence to an RNA sequence.

```
rna = dna2rna('ACGATGAGTCATGCTT')

rna =
```

ACGAUGAGUCAUGCUU

See Also
rna2dna | regexp | strrep

dnds

Estimate synonymous and nonsynonymous substitution rates

Syntax

```
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2)
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ...'GeneticCode',
GeneticCodeValue, ...)
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ...'Method',
MethodValue, ...)
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ...'Window',
WindowValue, ...)
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ...'AdjustStops',
AdjustStopsValue, ...)
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ...'Verbose',
VerboseValue, ...)
```

Input Arguments

SeqNT1, *SeqNT2*	Nucleotide sequences. Enter either a string or a structure with the field `Sequence`.
GeneticCodeValue	Property to specify a genetic code. Enter a Code Number or a string with a Code Name from the table Genetic Code. If you use a Code Name, you can truncate it to the first two characters. Default is `1` or `Standard`.
MethodValue	String specifying the method for calculating substitution rates. Choices are: • `NG` (default) — Nei-Gojobori method (1986) uses the number of synonymous and nonsynonymous substitutions and the number of potentially synonymous and nonsynonymous sites. Based on the Jukes-Cantor model. • `LWL` — Li-Wu-Luo method (1985) uses the number of transitional and transversional substitutions at

	three different levels of degeneracy of the genetic code. Based on Kimura's two-parameter model. • PBL — Pamilo-Bianchi-Li method (1993) is similar to the Li-Wu-Luo method, but with bias correction. Use this method when the number of transitions is much larger than the number of transversions.
WindowValue	Integer specifying the sliding window size, in codons, for calculating substitution rates and variances.
AdjustStopsValue	Controls whether stop codons are excluded from calculations. Choices are `true` (default) or `false`.
VerboseValue	Property to control the display of the codons considered in the computations and their amino acid translations. Choices are `true` or `false` (default). --- **Tip** Specify `true` to use this display to manually verify the codon alignment of the two input sequences. The presence of stop codons (`*`) in the amino acid translation can indicate that *SeqNT1* and *SeqNT2* are not codon-aligned.

Output Arguments

Dn	Nonsynonymous substitution rate(s).
Ds	Synonymous substitution rate(s).
Vardn	Variance for the nonsynonymous substitution rate(s).
Vards	Variance for the synonymous substitutions rate(s).

Description

[*Dn, Ds, Vardn, Vards*] = dnds(*SeqNT1, SeqNT2*) estimates the synonymous and nonsynonymous substitution rates per site between the two homologous nucleotide sequences, *SeqNT1* and *SeqNT2*, by comparing codons using the Nei-Gojobori method.

dnds returns:

- *Dn* — Nonsynonymous substitution rate(s).
- *Ds* — Synonymous substitution rate(s).
- *Vardn* — Variance for the nonsynonymous substitution rate(s).
- *Vards* — Variance for the synonymous substitutions rate(s).

This analysis:

- Assumes that the nucleotide sequences, *SeqNT1* and *SeqNT2*, are codon-aligned, that is, do not have frame shifts

---

Tip If your sequences are not codon-aligned, use the `nt2aa` function to convert them to amino acid sequences, use the `nwalign` function to globally align them, then use the `seqinsertgaps` function to recover the corresponding codon-aligned nucleotide sequences. For an example, see "Estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences" on page 1-534.

- Excludes codons that include ambiguous nucleotide characters or gaps
- Considers the number of codons in the shorter of the two nucleotide sequences

---

Caution If *SeqNT1* and *SeqNT2* are too short or too divergent, saturation can be reached, and `dnds` returns NaNs and a warning message.

[*Dn, Ds, Vardn, Vards*] = dnds(*SeqNT1, SeqNT2, ...'PropertyName', PropertyValue, ...*) calls dnds with optional properties that use property name/ property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

[*Dn, Ds, Vardn, Vards*] = dnds(*SeqNT1, SeqNT2, ...'GeneticCode', GeneticCodeValue, ...*) calculates synonymous and nonsynonymous substitution rates using the specified genetic code. Enter a Code Number or a string with a Code Name from the table Genetic Code. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard.

[*Dn, Ds, Vardn, Vards*] = dnds(*SeqNT1, SeqNT2, ...'Method', MethodValue, ...*) allows you to calculate synonymous and nonsynonymous substitution rates using the following algorithms:

- NG (default) — Nei-Gojobori method (1986) uses the number of synonymous and nonsynonymous substitutions and the number of potentially synonymous and nonsynonymous sites. Based on the Jukes-Cantor model.

- LWL — Li-Wu-Luo method (1985) uses the number of transitional and transversional substitutions at three different levels of degeneracy of the genetic code. Based on Kimura's two-parameter model.

- PBL — Pamilo-Bianchi-Li method (1993) is similar to the Li-Wu-Luo method, but with bias correction. Use this method when the number of transitions is much larger than the number of transversions.

[*Dn, Ds, Vardn, Vards*] = dnds(*SeqNT1*, *SeqNT2*, ...'Window', *WindowValue*, ...) performs the calculations over a sliding window, specified in codons. Each output is an array containing a rate or variance for each window.

[*Dn, Ds, Vardn, Vards*] = dnds(*SeqNT1*, *SeqNT2*, ...'AdjustStops', *AdjustStopsValue*, ...) controls whether stop codons are excluded from calculations. Choices are true (default) or false.

Tip When the 'AdjustStops' property is set to true, the following are true:

- Stop codons are excluded from frequency tables.

- Paths containing stop codons are not counted in the Nei-Gojobori method.

[*Dn, Ds, Vardn, Vards*] = dnds(*SeqNT1*, *SeqNT2*, ...'Verbose', *VerboseValue*, ...) controls the display of the codons considered in the computations and their amino acid translations. Choices are true or false (default).

Tip Specify true to use this display to manually verify the codon alignment of the two input sequences, *SeqNT1* and *SeqNT2*. The presence of stop codons (*) in the amino acid translation can indicate that *SeqNT1* and *SeqNT2* are not codon-aligned.

Examples

Estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences

This example shows how to estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences that are not codon-aligned.

This example uses two nucleotide sequences representing the human HEXA gene (accession number: NM_000520) and mouse HEXA gene (accession number: AK080777).

If you have live internet connection, you can use getgenbank function to retrieve the sequence information from the NCBI data repository and load the data into MATLAB®.

```
humanHEXA = getgenbank('NM_000520');
mouseHEXA = getgenbank('AK080777');
```

For your convenience, MATLAB provides these two sequences in the following mat file. Note that data in public databases are frequently updated and curated, and the results in this example may slightly differ if you use the latest data.

```
load hexosaminidase.mat
```

Extract the coding regions from the two nucleotide sequences.

```
humanHEXA_cds = featuresparse(humanHEXA,'feature','CDS','Sequence',true);
mouseHEXA_cds = featuresparse(mouseHEXA,'feature','CDS','Sequence',true);
```

Align the amino acid sequences converted from the nucleotide sequences.

```
[sc,al] = nwalign(nt2aa(humanHEXA_cds),nt2aa(mouseHEXA_cds),'extendgap',1);
```

Use the seqinsertgaps function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

```
humanHEXA_aligned = seqinsertgaps(humanHEXA_cds,al(1,:))
mouseHEXA_aligned = seqinsertgaps(mouseHEXA_cds,al(3,:))
```

```
humanHEXA_aligned =

atgacaagctccaggctttggttttcgctgctgctggcggcagcgttcgcaggacgggcgacggccctctggccctggcctcagaa
```

```
mouseHEXA_aligned =
```

```
atggccggctgcaggctctgggtttcgctgctgctggcggcggcgttggcttgcttggccacggcactgtggccgtggccccagta
```

Estimate the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences and also display the codons considered in the computations and their amino acid translations.

```
[nonsynSubRate,synSubRate] = dnds(humanHEXA_aligned,mouseHEXA_aligned,'verbose',true)

DNDS:
Codons considered in the computations:
ATGACAAGCTCCAGGCTTTGGTTTTCGCTGCTGCTGGCGGCAGCGTTCGCAGGACGGGCGACGGCCCTCTGGCCCTGGCCTCAGAA
ATGGCCGGCTGCAGGCTCTGGGTTTCGCTGCTGCTGGCGGCGGCGTTGGCTTGCTTGGCCACGGCACTGTGGCCGTGGCCCCAGTA
Translations:
M  T  S  S  R  L  W  F  S  L  L  L  A  A  A  F  A  G  R  A  T  A  L  W  P  W  P  Q  N
M  A  G  C  R  L  W  V  S  L  L  L  A  A  A  L  A  C  L  A  T  A  L  W  P  W  P  Q  Y

nonsynSubRate =

    0.0933


synSubRate =

    0.5181
```

More About

· dndsml

· geneticcode

· nt2aa

· seqpdist

References

[1] Li, W., Wu, C., and Luo, C. (1985). A new method for estimating synonymous and nonsynonymous rates of nucleotide substitution considering the relative

likelihood of nucleotide and codon changes. Molecular Biology and Evolution *2(2)*, 150–174.

[2] Nei, M., and Gojobori, T. (1986). Simple methods for estimating the numbers of synonymous and nonsynonymous nucleotide substitutions. Molecular Biology and Evolution *3(5)*, 418–426.

[3] Nei, M., and Jin, L. (1989). Variances of the average numbers of nucleotide substitutions within and between populations. Molecular Biology and Evolution *6(3)*, 290–300.

[4] Nei, M., and Kumar, S. (2000). Synonymous and nonsynonymous nucleotide substitutions" in Molecular Evolution and Phylogenetics (Oxford University Press).

[5] Pamilo, P., and Bianchi, N. (1993). Evolution of the Zfx And Zfy genes: rates and interdependence between the genes. Molecular Biology and Evolution *10(2)*, 271–281.

See Also
`featuresparse` | `nwalign` | `seqinsertgaps`

dndsml

Estimate synonymous and nonsynonymous substitution rates using maximum likelihood method

Syntax

```
[Dn, Ds, Like] = dndsml(SeqNT1, SeqNT2)
[Dn, Ds, Like] = dndsml(SeqNT1, SeqNT2, ...'GeneticCode',
GeneticCodeValue, ...)
[Dn, Ds, Like] = dndsml(SeqNT1, SeqNT2, ...'Verbose',
VerboseValue, ...)
```

Input Arguments

SeqNT1, *SeqNT2*	Nucleotide sequences. Enter either a string or a structure with the field `Sequence`.
GeneticCodeValue	Property to specify a genetic code. Enter a Code Number or a string with a Code Name from the table Genetic Code. If you use a Code Name, you can truncate it to the first two characters. Default is `1` or `Standard`.
VerboseValue	Property to control the display of the codons considered in the computations and their amino acid translations. Choices are `true` or `false` (default). **Tip** Specify `true` to use this display to manually verify the codon alignment of the two input sequences. The presence of stop codons (*) in the amino acid translation can indicate that *SeqNT1* and *SeqNT2* are not codon-aligned.

Output Arguments

Dn	Nonsynonymous substitution rate(s).
Ds	Synonymous substitution rate(s).

Like	Likelihood of estimate of substitution rates.

Description

[*Dn, Ds, Like*] = dndsml(*SeqNT1, SeqNT2*) estimates the synonymous and nonsynonymous substitution rates between the two homologous sequences, *SeqNT1* and *SeqNT2*, using the Goldman-Yang method (1994). This maximum likelihood method estimates an explicit model for codon substitution that accounts for transition/transversion rate bias and base/codon frequency bias. Then it uses the model to correct synonymous and nonsynonymous counts to account for multiple substitutions at the same site. The maximum likelihood method is best suited when the sample size is significant (larger than 100 bases) and when the sequences being compared can have transition/transversion rate biases and base/codon frequency biases.

dndsml returns:

- *Dn* — Nonsynonymous substitution rate(s).
- *Ds* — Synonymous substitution rate(s).
- *Like* — Likelihood of this estimate.

This analysis:

- Assumes that the nucleotide sequences, *SeqNT1* and *SeqNT2*, are codon-aligned, that is, do not have frame shifts.

  > **Tip** If your sequences are not codon-aligned, use the nt2aa function to convert them to amino acid sequences, use the nwalign function to globally align them, then use the seqinsertgaps function to recover the corresponding codon-aligned nucleotide sequences. For an example, see "Estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences using maximum likelihood method" on page 1-539.

- Excludes any ambiguous nucleotide characters or codons that include gaps.
- Considers the number of codons in the shorter of the two nucleotide sequences.

> **Caution** If *SeqNT1* and *SeqNT2* are too short or too divergent, saturation can be reached, and dndsml returns NaNs and a warning message.

[*Dn, Ds, Like*] = dndsml(*SeqNT1*, *SeqNT2*, ...'*PropertyName*', *PropertyValue*, ...) calls dnds with optional properties that use property name/ property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

[*Dn, Ds, Like*] = dndsml(*SeqNT1*, *SeqNT2*, ...'GeneticCode', *GeneticCodeValue*, ...) calculates synonymous and nonsynonymous substitution rates using the specified genetic code. Enter a Code Number or a string with a Code Name from the table Genetic Code. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard.

[*Dn, Ds, Like*] = dndsml(*SeqNT1*, *SeqNT2*, ...'Verbose', *VerboseValue*, ...) controls the display of the codons considered in the computations and their amino acid translations. Choices are true or false (default).

Tip Specify true to use this display to manually verify the codon alignment of the two input sequences, *SeqNT1* and *SeqNT2*. The presence of stop codons (*) in the amino acid translation can indicate that *SeqNT1* and *SeqNT2* are not codon-aligned.

Examples

Estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences using maximum likelihood method

This example shows how to estimate synonymous and nonsynonymous substitution rates between two nucleotide sequences that are not codon-aligned using maximum likelihood method.

This example uses two nucleotide sequences representing the human HEXA gene (accession number: NM_000520) and mouse HEXA gene (accession number: AK080777).

If you have live internet connection, you can use getgenbank function to retrieve the sequence information from the NCBI data repository and load the data into MATLAB®.

```
humanHEXA = getgenbank('NM_000520');
mouseHEXA = getgenbank('AK080777');
```

For your convenience, MATLAB provides these two sequences in the following mat file. Note that data in public databases are frequently updated and curated, and the results in this example may slightly differ if you use the latest data.

```
load hexosaminidase.mat
```

Extract the coding regions from the two nucleotide sequences.

```
humanHEXA_cds = featuresparse(humanHEXA,'feature','CDS','Sequence',true);
mouseHEXA_cds = featuresparse(mouseHEXA,'feature','CDS','Sequence',true);
```

Align the amino acid sequences converted from the nucleotide sequences.

```
[sc,al] = nwalign(nt2aa(humanHEXA_cds),nt2aa(mouseHEXA_cds),'extendgap',1);
```

Use the `seqinsertgaps` function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

```
humanHEXA_aligned = seqinsertgaps(humanHEXA_cds,al(1,:))
mouseHEXA_aligned = seqinsertgaps(mouseHEXA_cds,al(3,:))
```

```
humanHEXA_aligned =

atgacaagctccaggctttggttttcgctgctgctggcggcagcgttcgcaggacgggcgacggccctctggccctggcctcagaac
```

```
mouseHEXA_aligned =

atggccggctgcaggctctgggtttcgctgctgctggcggcggcgttggcttgcttggccacggcactgtggccgtggccccagtac
```

Estimate the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences and also display the codons considered in the computations and their amino acid translations.

```
[nonsynSubRate,synSubRate] = dndsml(humanHEXA_aligned,mouseHEXA_aligned,'verbose',true)
```

```
DNDSML:
Codons considered in the computations:
ATGACAAGCTCCAGGCTTTGGTTTTCGCTGCTGCTGGCGGCAGCGTTCGCAGGACGGGCGACGGCCCTCTGGCCCTGGCCTCAGAAC
ATGGCCGGCTGCAGGCTCTGGGTTTCGCTGCTGCTGGCGGCGGCGTTGGCTTGCTTGGCCACGGCACTGTGGCCGTGGCCCCAGTAC
Translations:
M  T  S  S  R  L  W  F  S  L  L  L  A  A  A  F  A  G  R  A  T  A  L  W  P  W  P  Q  N
M  A  G  C  R  L  W  V  S  L  L  L  A  A  A  L  A  C  L  A  T  A  L  W  P  W  P  Q  Y
```

```
Initial estimates: Kappa=3.301203, dn=0.093274, ds=0.518095, t=0.353716
ML estimates: Kappa=2.498253, omega(dn/ds)=0.185577, t=0.602465

nonsynSubRate =

    0.0943


synSubRate =

    0.5080
```

More About

- dnds
- geneticcode
- nt2aa
- seqpdist

References

[1] Tamura, K., and Mei, M. (1993). Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. Molecular Biology and Evolution *10*, 512–526.

[2] Yang, Z., and Nielsen, R. (2000). Estimating synonymous and nonsynonymous substitution rates under realistic evolutionary models. Molecular Biology and Evolution *17*, 32–43.

[3] Goldman, N., and Yang, Z. (1994). A Codon-based Model of Nucleotide Substitution for Protein-coding DNA Sequences. Mol. Biol. Evol. *11(5)*, 725–736.

See Also
featuresparse | nwalign | seqinsertgaps

dolayout (biograph)

Calculate node positions and edge trajectories

Syntax

```
dolayout(BGobj)
dolayout(BGobj, 'Paths', PathsOnlyValue)
```

Arguments

BGobj	Biograph object created by the biograph function (object constructor).
PathsOnlyValue	Controls the calculation of only the edge paths, leaving the nodes at their current positions. Choices are true or false (default).

Description

dolayout(*BGobj*) calls the layout engine to calculate the optimal position for each node so that its 2-D rendering is clean and uncluttered, and then calculates the best curves to represent the edges. The layout engine uses the following properties of the biograph object:

- LayoutType — Specifies the layout engine as 'hierarchical', 'equilibrium', or 'radial'.

- LayoutScale — Rescales the sizes of the node before calling the layout engine. This gives more space to the layout and reduces the overlapping of nodes.

- NodeAutoSize — Controls precalculating the node size before calling the layout engine. When NodeAutoSize is set to 'on', the layout engine uses the node properties FontSize and Shape, and the biograph object property LayoutScale to precalculate the actual size of each node. When NodeAutoSize is set to 'off', the layout engine uses the node property Size.

For more information on the above properties, see Properties of a Biograph Object. For an example of accessing and specifying the above properties of a biograph object, see "Create a Biograph object and specify its properties" on page 1-231.

dolayout(*BGobj*, 'Paths', *PathsOnlyValue*) controls the calculation of only the edge paths, leaving the nodes at their current positions. Choices are `true` or `false` (default).

Examples

Create a Biograph Object and Calculate Node Positions and Edge Trajectories

This example shows how to create a biograph object and calculate node positions and edge trajectories.

Create a biograph object.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg = biograph(cm)

Biograph object with 5 nodes and 9 edges.
```

Nodes do not have positions yet.

```
bg.nodes(1).Position

ans =

    []
```

Call the layout engine and render the graph.

```
dolayout(bg);
bg.nodes(1).Position

ans =

   100    206
```

```
view(bg)
```
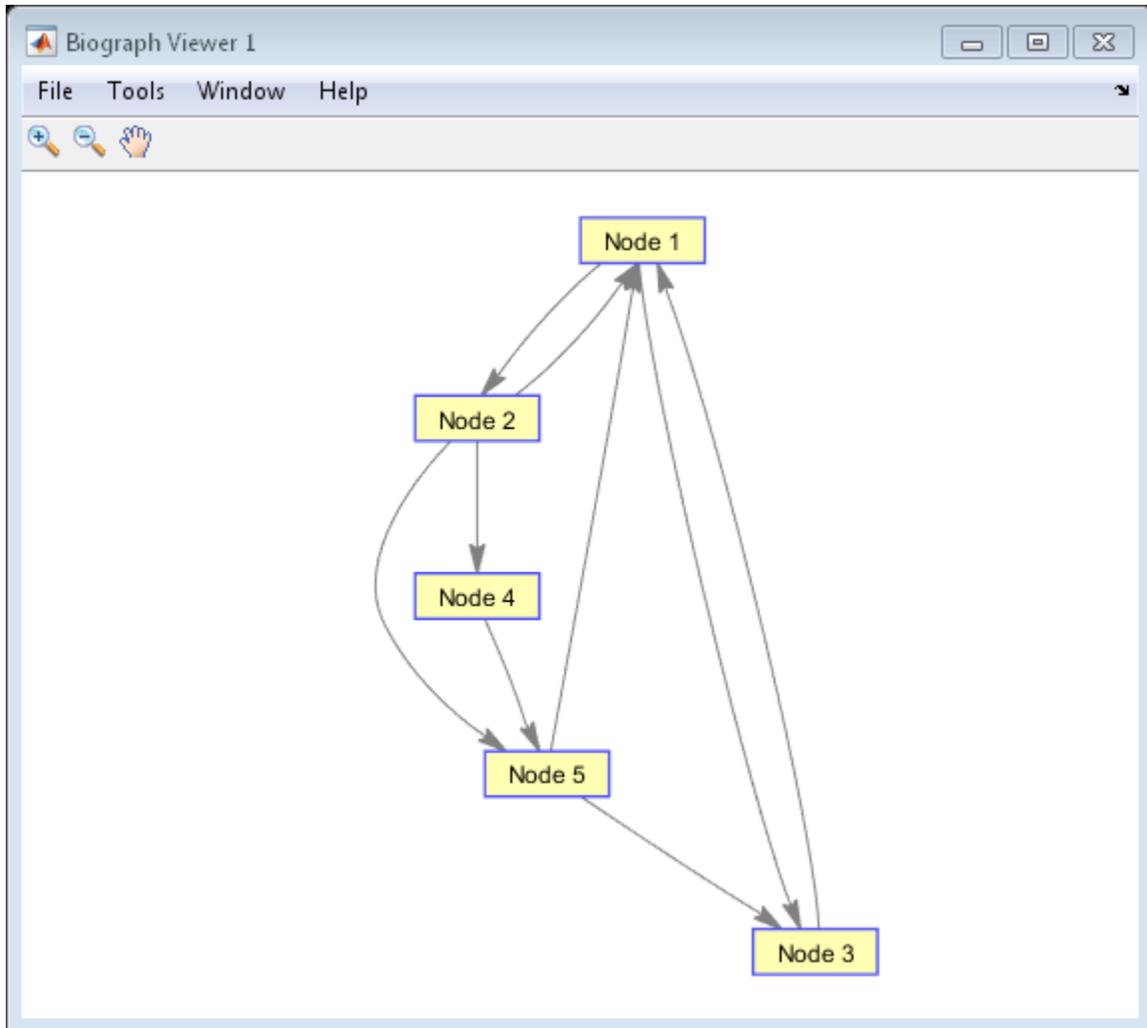

Manually modify a node position and recalculate the paths only.

```
bg.nodes(1).Position = [150 150];
```

```
dolayout(bg, 'Pathsonly', true);
view(bg)
```
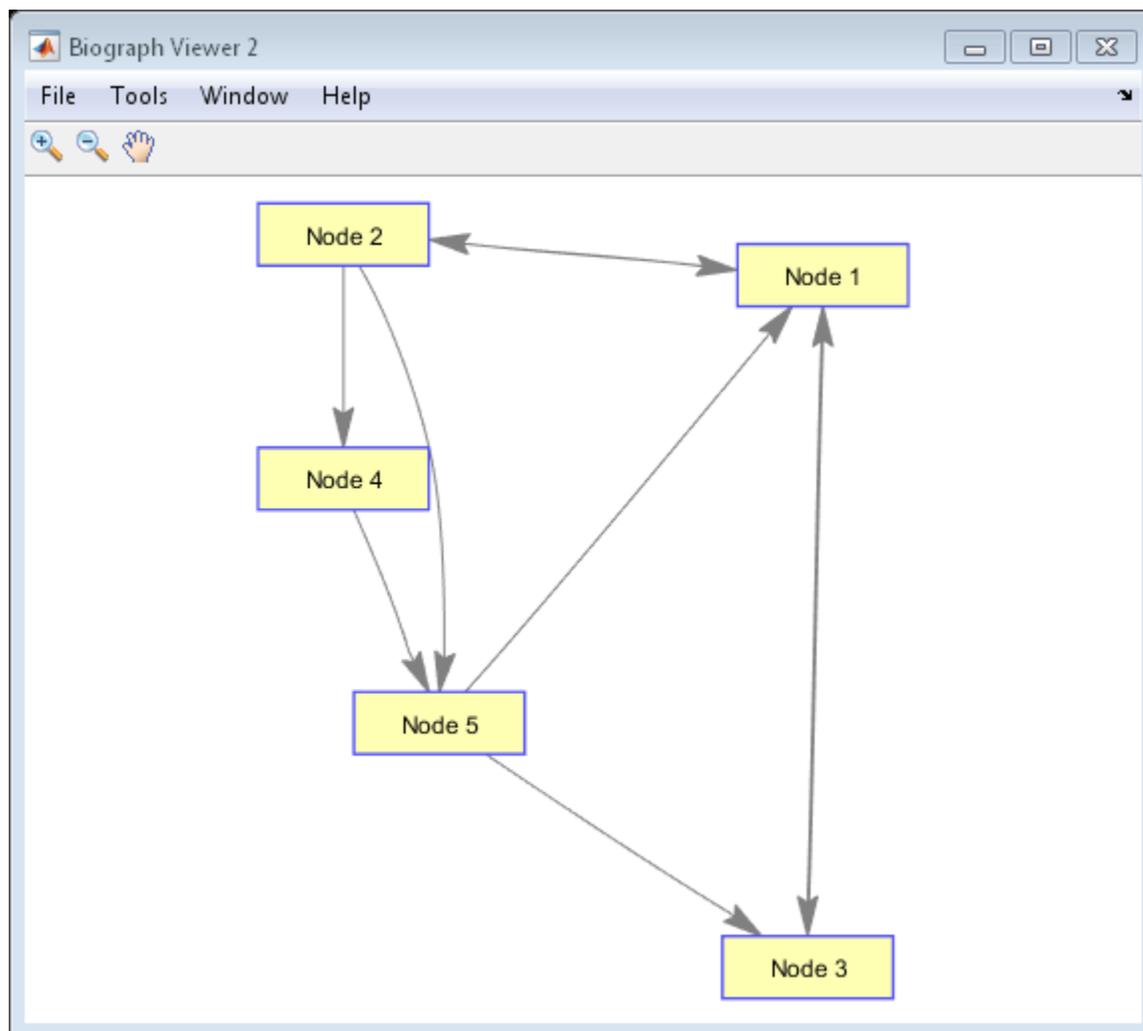

More About

- biograph object

See Also

biograph | get | getdescendants | getnodesbyid | set | view | dolayout |
getancestors | getedgesbynodeid | getrelatives

double (DataMatrix)

Convert DataMatrix object to double-precision array

Syntax

```
B = double(DMObj)
B = double(DMObj, Rows)
B = double(DMObj, Rows, Cols)
```

Input Arguments

DMObj	DataMatrix object, such as created by DataMatrix (object constructor).
Rows, *Cols*	Row(s) or column(s) in *DMObj*, specified by one of the following: • Scalar • Vector of positive integers • String specifying a row or column name • Cell array of row or column names • Logical vector

Output Arguments

B	MATLAB numeric array.

Description

B = double(*DMObj*) converts *DMObj*, a DataMatrix object, to a double-precision array, which it returns in *B*.

B = double(*DMObj*, *Rows*) converts a subset of *DMObj*, a DataMatrix object, specified by *Rows*, to a double-precision array, which it returns in *B*. *Rows* can be a positive integer,

vector of positive integers, string specifying a row name, cell array of row names, or a logical vector.

B = double(*DMObj, Rows, Cols*) converts a subset of *DMObj*, a DataMatrix object, specified by *Rows* and *Cols*, to a double-precision array, which it returns in*B*. *Cols* can be a positive integer, vector of positive integers, string specifying a column name, cell array of column names, or a logical vector.

More About

• DataMatrix object

See Also
DataMatrix | single

elementData

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set data element (DataMatrix object) in ExpressionSet object

Syntax

DMObj = elementData(*ESObj, Element*)
NewESObj = elementData(*ESObj, Element, NewDMObj*)

Description

DMObj = elementData(*ESObj, Element*) returns the DataMatrix object from an ExpressionSet object, specified by *Element*, a positive integer or a string specifying an element name.

NewESObj = elementData(*ESObj, Element, NewDMObj*) replaces the DataMatrix object specified by *Element* in *ESObj*, an ExpressionSet object, with *NewDMObj*, a new DataMatrix object, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

Element

Element (DataMatrix object) in an ExpressionSet object, specified by either of the following:

- Positive integer
- String specifying the element name

Default:

NewDMObj

Object of the DataMatrix class. The sample names and feature names in *NewDMObj* must match the sample names and feature names in the DataMatrix object specified by *Element*.

Default:

Output Arguments

DMObj

Object of the DataMatrix class, returned from the ExptData object of an ExpressionSet object.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing a specified data element (DataMatrix object).

Examples

Construct an ExpressionSet object, ESObj, as described in the "Examples" on page 1-   section of the `bioma.ExpressionSet` class reference page. Extract a DataMatrix object from it:

```
% Extract first DataMatrix object
ExtractedDMObj = elementData(ESObj, 1);
```

See Also

bioma.ExpressionSet | bioma.data.ExptData | DataMatrix

How To

• "Managing Gene Expression Data in Objects"

elementData

Class: bioma.data.ExptData
Package: bioma.data

Retrieve or set data element (DataMatrix object) in ExptData object

Syntax

DMObj = elementData(*EDObj, Element*)
NewEDObj = elementData(*EDObj, Element, NewDMObj*)

Description

DMObj = elementData(*EDObj, Element*) returns the DataMatrix object from an
ExptData object, specified by *Element*, a positive integer or string specifying an element
name.

NewEDObj = elementData(*EDObj, Element, NewDMObj*) replaces the element
(DataMatrix object) specified by *Element* in *EDObj*, an ExptData object, with *NewDMObj*,
a new DataMatrix object, and returns *NewEDObj*, a new ExptData object.

Input Arguments

EDObj

Object of the `bioma.data.ExptData` class.

Default:

Element

Element (DataMatrix object) in an ExptData object, specified by either of the following:

- Positive integer
- String specifying the element name

Default:

NewDMObj

Object of the DataMatrix class. The sample names and feature names in *NewDMObj* must match the sample names and feature names of *EDObj*.

Default:

Output Arguments

DMObj

Object of the DataMatrix class, returned from an ExptData object.

NewEDObj

Object of the `bioma.data.ExptData` class, returned after replacing a data element (DataMatrix object).

Examples

Construct an ExptData object, and then extract a DataMatrix object from it:

```
% Import bioma.data package to make constructor functions
% available
import bioma.data.*
% Create DataMatrix object from .txt file containing
% expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Extract first DataMatrix object
ExtractedDMObj = elementData(EDObj, 1);
```

See Also
`bioma.data.ExptData` | `DataMatrix`

How To
· "Representing Expression Data Values in ExptData Objects"

elementNames

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set element names of DataMatrix objects in ExpressionSet object

Syntax

```
ElmtNames = elementNames(ESObj)
ElmtNames = elementNames(ESObj, Subset)
NewESObj = elementNames(ESObj, Subset, NewElmtNames)
```

Description

ElmtNames = elementNames(*ESObj*) returns a cell array of strings specifying the element names of all the data elements (DataMatrix objects) stored in the ExptData object in an ExpressionSet object.

ElmtNames = elementNames(*ESObj*, *Subset*) returns a cell array of strings specifying the element names of a subset of the data elements (DataMatrix objects) in the ExptData object in an ExpressionSet object.

NewESObj = elementNames(*ESObj*, *Subset*, *NewElmtNames*) replaces the element names of the data elements (DataMatrix objects) specified by *Subset* in *ESObj*, an ExpressionSet object, with *NewElmtNames*, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

Subset

One of the following to specify the element names of a subset of the data elements (DataMatrix objects) in the ExptData object of an ExpressionSet object:

- String specifying an element name
- Cell array of strings specifying element names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewElmtNames

New element names for specific data elements (DataMatrix objects) within an ExpressionSet object, specified by one of the following:

- Numeric vector
- String or cell array of strings
- String, which `elementNames` uses as a prefix for the element names, with element numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `elementNames` assigns unique element names using the format `Elmt1`, `Elmt2`, etc.

The number of elements in *NewElmtNames* must equal the number of elements specified by *Subset*.

Default:

Output Arguments

ElmtNames

Cell array of strings specifying the element names of all or some of the data elements (DataMatrix objects) in the ExptData object of an ExpressionSet object.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing element names of specific data elements (DataMatrix objects).

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1- section of the `bioma.ExpressionSet` class reference page. Retrieve the element names of the DataMatrix objects in it:

```
% Retrieve element names of DataMatrix objects
ENames = elementNames(ESObj);
```

See Also

`bioma.ExpressionSet` | `bioma.data.ExptData` | `DataMatrix` | `exptData`

How To

- "Managing Gene Expression Data in Objects"

elementNames

Class: bioma.data.ExptData
Package: bioma.data

Retrieve or set element names of DataMatrix objects in ExptData object

Syntax

```
ElmtNames = elementNames(EDObj)
ElmtNames = elementNames(EDObj, Subset)
NewEDObj = elementNames(EDObj, Subset, NewElmtNames)
```

Description

ElmtNames = elementNames(*EDObj*) returns a cell array of strings specifying the element names of all the data elements (DataMatrix objects) stored in an ExptData object.

ElmtNames = elementNames(*EDObj*, *Subset*) returns a cell array of strings specifying the element names of a subset of the data elements (DataMatrix objects) stored in an ExptData object.

NewEDObj = elementNames(*EDObj*, *Subset*, *NewElmtNames*) replaces the element names of the data elements (DataMatrix objects) specified by *Subset* in *EDObj*, an ExptData object, with *NewElmtNames*, and returns *NewEDObj*, a new ExptData object.

Input Arguments

EDObj

Object of the `bioma.data.ExptData` class.

Default:

Subset

One of the following to specify the element names of a subset of the data elements (DataMatrix objects) in an ExptData object:

- String specifying an element name
- Cell array of strings specifying element names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewElmtNames

New element names for specific data elements (DataMatrix objects) within an ExptData object, specified by one of the following:

- Numeric vector
- String or cell array of strings
- String, which `elementNames` uses as a prefix for the element names, with element numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `elementNames` assigns unique element names using the format `Elmt1`, `Elmt2`, etc.

The number of elements in *NewElmtNames* must equal the number of elements specified by *Subset*.

Default:

Output Arguments

ElmtNames

Cell array of strings specifying the element names of all or some of the data elements (DataMatrix objects) in an ExptData object.

NewEDObj

Object of the `bioma.data.ExptData` class, returned after replacing element names of specific data elements (DataMatrix objects).

Examples

Construct an ExptData object, and then retrieve the element names of DataMatrix objects from it:

```
% Import bioma.data package to make constructor functions
% available
import bioma.data.*
% Create DataMatrix object from .txt file containing
% expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Retrieve element names of DataMatrix objects
ENames = elementNames(EDObj);
```

See Also
bioma.data.ExptData | dmNames | DataMatrix | featureNames | sampleNames

How To
- "Representing Expression Data Values in ExptData Objects"

emblread

Read data from EMBL file

Syntax

EMBLData = emblread(*File*)
EMBLSeq = emblread (*File*, 'SequenceOnly', *SequenceOnlyValue*)

Input Arguments

File	Either of the following: • String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is an EMBL-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. • MATLAB character array that contains the text of an EMBL-formatted file **Tip** You can use the getembl function with the 'ToFile' property to retrieve data from the European Molecular Biology Laboratory (EMBL) database and create an EMBL-formatted file.
SequenceOnlyValue	Controls the reading of only the sequence without the metadata. Choices are true or false (default).

Output Arguments

EMBLData	MATLAB structure with fields corresponding to EMBL data.
EMBLSeq	MATLAB character string representing the sequence.

Description

EMBLData = emblread(*File*) reads data from *File*, an EMBL-formatted file, and creates *EMBLData*, a MATLAB structure containing fields corresponding to the EMBL two-character line type code, based on release 107 of the EMBL-Bank flat file format. Each line type code is stored as a separate element in the structure. For a list of the EMBL two-character line type codes, see http://www.ebi.ac.uk/embl/Documentation/User_manual/usrman.html.

Note: Topology information was not included in EMBL flat files before release 87 of the database. When reading a file created before release 87, EMBLREAD returns an empty `Identification.Topology` field.

Note: The entry name is no longer displayed in the ID line of EMBL flat files in release 87. When reading a file created in release 87, EMBLREAD returns the accession number in the `Identification.EntryName` field.

EMBLSeq = emblread (*File*, 'SequenceOnly', *SequenceOnlyValue*) controls the reading of only the sequence without the metadata. Choices are `true` or `false` (default).

Examples

Retrieve sequence information from the Web, save to a file, and then read back into the MATLAB software.

1 Use the `getembl` function and `ToFile` property to retrieve sequence information from the Web and save to an EMBL-formatted file.

```
getembl('XOO558','ToFile','rat_protein.txt');
```

2 Read data from the EMBL-formatted file and create a MATLAB structure.

```
EMBLData = emblread('rat_protein.txt')

EMBLData =

            Identification: [1x1 struct]
```

```
                 Accession: 'X00558'
           SequenceVersion: 'X00558.1'
               DateCreated: '13-JUN-1985 (Rel. O6, Created)'
               DateUpdated: [1x46 char]
               Description: [1x75 char]
                   Keyword: [1x75 char]
            OrganismSpecies: [1x75 char]
     OrganismClassification: [3x75 char]
                  Organelle: ''
                  Reference: {[1x1 struct]}
     DatabaseCrossReference: ''
                   Comments: ''
                   Assembly: ''
                    Feature: [23x75 char]
                  BaseCount: [1x1 struct]
                   Sequence: [1x877 char]
```

See Also
fastaread | genbankread | genpeptread | getembl | pdbread | seqviewer

eq (DataMatrix)

Test DataMatrix objects for equality

Syntax

T = eq(*DMObj1*, *DMObj2*)
T = *DMObj1* == *DMObj2*
T = eq(*DMObj1*, *B*)
T = *DMObj1* == *B*
T = eq(*B*, *DMObj1*)
T = *B* == *DMObj1*

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

T	Logical matrix of the same size as *DMObj1* and *DMObj2* or *DMObj1* and *B*. It contains logical 1 (true) where elements in the first input are equal to the corresponding element in the second input, and logical 0 (false) when they are not equal.

Description

T = eq(*DMObj1*, *DMObj2*) or the equivalent *T* = *DMObj1* == *DMObj2* compares each element in DataMatrix object *DMObj1* to the corresponding element in DataMatrix object *DMObj2*, and returns *T*, a logical matrix of the same size as *DMObj1* and *DMObj2*, containing logical 1 (true) where elements in *DMObj1* are equal to the corresponding element in *DMObj2*, and logical 0 (false) when they are not equal. *DMObj1* and *DMObj2*

must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). *DMObj1* and *DMObj2* can have different `Name` properties.

T = `eq`(*DMObj1*, *B*) or the equivalent *T* = *DMObj1* `==` *B* compares each element in DataMatrix object *DMObj1* to the corresponding element in *B*, a numeric or logical array, and returns *T*, a logical matrix of the same size as *DMObj1* and *B*, containing logical 1 (true) where elements in *DMObj1* are equal to the corresponding element in *B*, and logical 0 (false) when they are not equal. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

T = `eq`(*B*, *DMObj1*) or the equivalent *T* = *B* `==` *DMObj1* compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are equal to the corresponding element in *DMObj1*, and logical 0 (false) when they are not equal. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls *T* = `eq`(*X*, *Y*) for the syntax *T* = *X* `==` *Y* when *X* or *Y* is a DataMatrix object.

More About

· DataMatrix object

See Also

`DataMatrix` | `ne`

evalrasmolscript

Send RasMol script commands to Molecule Viewer window

Syntax

evalrasmolscript(*FigureHandle*, *Command*)

Arguments

FigureHandle	Figure handle to a molecule viewer returned by the `molviewer` function.
Command	Any of the following: • String specifying one or more RasMol script commands. Use a `;` to separate commands. • Character array or cell array containing strings specifying RasMol script commands. **Note:** For a complete list of RasMol script commands, see `http://www.stolaf.edu/academics/chemapps/jmol/docs/` • String specifying a file name or a path and file name of a text file containing Jmol script commands. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.

Description

evalrasmolscript(*FigureHandle*, *Command*) sends the RasMol script commands specified by *Command* to *FigureHandle*, the figure handle of a Molecule Viewer window created using the `molviewer` function.

Examples

1   Use the `molviewer` function to create a figure handle to a Molecule Viewer window.

```
FH = molviewer('2DHB')
```

2   Use the `evalrasmolscript` function to send script commands to the molecule viewer that change the background to black and spin the molecule.

```
evalrasmolscript(FH, 'background white; spin')
```

See Also

`getpdb` | `molviewer` | `pdbread` | `pdbwrite`

expressions

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set `Expressions` DataMatrix object from ExpressionSet object

Syntax

ExpressionsDMObj = expressions(*ESObj*)
NewESObj = expressions(*ESObj, NewDMObj*)

Description

ExpressionsDMObj = expressions(*ESObj*) returns the `Expressions` element
(DataMatrix object), which contains expression values, from an ExpressionSet object.

NewESObj = expressions(*ESObj, NewDMObj*) replaces the `Expressions`
element (DataMatrix object) in *ESObj*, an ExpressionSet object, with *NewDMObj*, a new
DataMatrix object, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

NewDMObj

Object of the DataMatrix class.

Default:

Output Arguments

ExpressionsDMObj

DataMatrix object containing the expression values from the `Expressions` DataMatrix object within an ExpressionSet object.

NewESObj

ExpressionSet object returned after replacing the `Expressions` DataMatrix object.

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1-   section of the `bioma.ExpressionSet` class reference page. Extract the `Expressions` DataMatrix object from it:

```
% Extract expression values from Expressions DataMatrix object
ExpressionsDMObj = expressions(ESObj);
```

See Also
bioma.ExpressionSet | bioma.data.ExptData | DataMatrix

How To
·   "Managing Gene Expression Data in Objects"

exprprofrange

Calculate range of gene expression profiles

Syntax

Range = exprprofrange(*Data*)
[*Range, LogRange*] = exprprofrange(*Data*)
... = exprprofrange(*Data*, 'ShowHist', *ShowHistValue*)

Arguments

Data	DataMatrix object or numeric matrix of expression values, where each row corresponds to a gene.
ShowHistValue	Controls the display of a histogram with range data. Default is: • false — When output values are specified. • true — When output values are not specified.

Description

Range = exprprofrange(*Data*) calculates the range of each expression profile in *Data*, a DataMatrix object or numeric matrix of expression values, where each row corresponds to a gene.

[*Range, LogRange*] = exprprofrange(*Data*) returns the log range, that is, log(max(prof))- log(min(prof)), of each expression profile. If you do not specify output arguments, exprprofrange displays a histogram bar plot of the range.

... = exprprofrange(*Data*, 'ShowHist', *ShowHistValue*) controls the display of a histogram with range data. Choices for *ShowHistValue* are true or false.

Examples

1   Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: yeastvalues, a matrix of gene

expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`

```
load yeastdata
```

2 Calculate the range of expression profiles for yeast data as gene expression changes during the metabolic shift from fermentation to respiration. Display a histogram of the data.

```
range = exprprofrange(yeastvalues,'ShowHist',true);
```

See Also
`exprprofvar` | `generangefilter`

exprprofvar

Calculate variance of gene expression profiles

Syntax

```
Variance = exprprofvar(Data)
exprprofvar(..., 'PropertyName', PropertyValue,...)
exprprofvar(..., 'ShowHist', ShowHistValue)
```

Arguments

Data	DataMatrix object or numeric matrix of expression values, where each row corresponds to a gene.
ShowHistValue	Controls the display of a histogram with variance data. Default is: • `false` — When output values are specified. • `true` — When output values are not specified.

Description

Variance = exprprofvar(*Data*) calculates the variance of each expression profile in *Data*, a DataMatrix object or numeric matrix of expression values, where each row corresponds to a gene. If you do not specify output arguments, this function displays a histogram bar plot of the range.

exprprofvar(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

exprprofvar(..., 'ShowHist', *ShowHistValue*) controls the display of a histogram with range data. Choices for *ShowHistValue* are `true` or `false`.

Examples

1   Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: `yeastvalues`, a matrix of gene expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`

```
load yeastdata
```

2   Calculate the variance of expression profiles for yeast data as gene expression changes during the metabolic shift from fermentation to respiration. Display a histogram of the data.

```
datavar = exprprofvar(yeastvalues,'ShowHist',true);
```

See Also
exprprofrange | generangefilter | genevarfilter

exprWrite

Class: bioma.ExpressionSet
Package: bioma

Write expression values in ExpressionSet object to text file

Syntax

```
exprWrite(ESObj, File)
exprWrite(..., 'Delimiter', DelimiterValue, ...)
exprWrite(..., 'Precision', PrecisionValue, ...)
exprWrite(..., 'Header', HeaderValue, ...)
exprWrite(..., 'Annotated', AnnotatedValue, ...)
exprWrite(..., 'Append', AppendValue, ...)
```

Description

exprWrite(*ESObj*, *File*) writes the expression values in the `Expressions` element (DataMatrix object) from an ExpressionSet object to a text file, using the delimiter `\t` to separate columns. `exprWrite` writes the data starting at the first column of the first row in the destination file.

exprWrite(..., '*PropertyName*', *PropertyValue*, ...) calls `exprWrite` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

exprWrite(..., 'Delimiter', *DelimiterValue*, ...) specifies a delimiter symbol to use as a column separator. Default is '`\t`'.

exprWrite(..., 'Precision', *PrecisionValue*, ...) specifies the precision for writing the data to the text file. Default is `5`.

exprWrite(..., 'Header', *HeaderValue*, ...) specifies the first line of the text file. Default is the `Name` property for the DataMatrix object.

exprWrite(..., 'Annotated', *AnnotatedValue*, ...) controls the writing of row and column names to the text file. Choices are `true` (default) or `false`.

exprWrite(..., 'Append', *AppendValue*, ...) controls the appending of the expression values to *File* when it is an existing file. Choices are `true` or `false` (default). If `false`, exprWrite overwrites *File*.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

File

String specifying either a file name or a path and file name for saving the expression values. If you specify only a file name, `exprWrite` saves the file to the MATLAB Current Folder.

Default:

DelimiterValue

String specifying a delimiter symbol to use as a matrix column separator. Typical choices are:

- `' '`
- `'\t'` (default)
- `','`
- `';'`
- `'|'`

Default:

PrecisionValue

Precision for writing the data to the text file, specified by either:

- Positive integer specifying the number of significant digits
- C-style format string starting with %, such as `'%6.5f'`

Default: 5

HeaderValue

String specifying the first line of the text file. Default is the `Name` property for the DataMatrix object.

Default:

AnnotatedValue

Controls the writing of row and column names to the text file. Choices are `true` (default) or `false`.

Default:

AppendValue

Controls the appending of the expression values to *File* when it is an existing file. Choices are `true` or `false` (default). If `false`, `exprWrite` overwrites *File*.

Default:

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1-   section of the `bioma.ExpressionSet` class reference page. Write the expression values in the ExpressionSet object to a text file:

```
% Write expression values to text file
exprWrite(ESObj, 'myexpressiondata.txt')
```

See Also
bioma.ExpressionSet | bioma.data.ExptData | dmwrite | DataMatrix

How To
- "Managing Gene Expression Data in Objects"

exptData

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set experiment data in ExpressionSet object

Syntax

```
ExptDataObj = exptData(ESObj)
NewESObj = exptData(ESObj, NewExptDataObj)
```

Description

ExptDataObj = exptData(*ESObj*) returns the ExptData object stored in an ExpressionSet object.

NewESObj = exptData(*ESObj, NewExptDataObj*) replaces the ExptData object in *ESObj*, an ExpressionSet object, with *NewExptDataObj*, a new ExptData object, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

NewExptDataObj

Object of the `bioma.data.ExptData` class.

Default:

Output Arguments

ExptDataObj

Object of the `bioma.data.ExptData` class.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing the ExptData object.

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1-   section of the `bioma.ExpressionSet` class reference page. Retrieve the ExptData object stored in the ExpressionSet object:

```
% Retrieve the ExptData object
NewEDObj = exptData(ESObj);
```

See Also
`bioma.ExpressionSet` | `bioma.data.ExptData` | `DataMatrix` | `featureData` | `sampleData`

How To
•   "Managing Gene Expression Data in Objects"

exptInfo

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set experiment information in ExpressionSet object

Syntax

```
MIAMEObj = exptInfo(ESObj)
NewESObj = exptInfo(ESObj, NewMIAMEObj)
```

Description

MIAMEObj = exptInfo(*ESObj*) returns a MIAME object containing experiment information from an ExpressionSet object.

NewESObj = exptInfo(*ESObj*, *NewMIAMEObj*) replaces the MIAME object in *ESObj*, an ExpressionSet object, with *NewMIAMEObj*, a new MIAME object, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

NewMIAMEObj

Object of the `bioma.data.MIAME` class.

Default:

Output Arguments

MIAMEObj

Object of the `bioma.data.MIAME` class.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing the MIAME object.

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1-   section of the `bioma.ExpressionSet` class reference page. Retrieve the MIAME object stored in the ExpressionSet object:

```
% Retrieve the MIAME object
NewMIAMEObj = exptInfo(ESObj);
```

See Also
bioma.ExpressionSet | bioma.data.MIAME

How To
• "Managing Gene Expression Data in Objects"

Related Links

•

• http://www.mged.org/Workgroups/MIAME/miame.html

fastainfo

Return information about FASTA file

Syntax

InfoStruct = fastainfo(*File*)

Description

InfoStruct = fastainfo(*File*) returns a MATLAB structure containing summary information about a FASTA-formatted file.

Input Arguments

File

FASTA-formatted file specified by one of the following:

- String specifying a file name or path and file name of a FASTA-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.
- URL pointing to a FASTA-formatted file.
- MATLAB character array containing the text of a FASTA-formatted file.

Default:

Output Arguments

InfoStruct

MATLAB structure containing summary information about a FASTA-formatted file. The structure contains the following fields.

Field	Description
`Filename`	Name of the file.
`FilePath`	Path to the file
`FileModDate`	Modification date of the file.
`FileSize`	Size of the file in bytes.
`NumberOfEntries`	Number of sequence entries in the file.
`Header`	If *File* contains only one sequence, then this is a string containing the header information from the FASTA-formatted file. Otherwise, this field is empty.
`Length`	If *File* contains only one sequence, then this is a scalar specifying the length of the sequence. Otherwise, this field is empty.

Examples

Return a summary of the contents of a FASTA file:

```
info = fastainfo('p53nt.txt')

info =

            Filename: 'p53nt.txt'
            FilePath: 'D:\2010_08_24_h11m43s32_job6027_pass\matlab\toolbox\bioinfo\biode
         FileModDate: '31-Mar-2003 11:44:27'
            FileSize: 2764
     NumberOfEntries: 1
              Header: [1x94 char]
              Length: 2629
```

See Also
`BioIndexedFile` | `fastaread` | `fastawrite` | `fastqinfo` | `fastqread` | `fastqwrite` | `sffinfo` | `sffread` | `saminfo` | `samread`

fastaread

Read data from FASTA file

Syntax

```
FASTAData = fastaread(File)
[Header, Sequence] = fastaread(File)
... = fastaread(File, ...'IgnoreGaps', IgnoreGapsValue, ...)
... = fastaread(File, ...'Blockread', BlockreadValue, ...)
... = fastaread(File, ...'TrimHeaders', TrimHeadersValue, ...)
```

Input Arguments

File	Either of the following: • String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a FASTA-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. • MATLAB character array that contains the text of a FASTA-formatted file.
IgnoreGapsValue	Controls the removal of gap symbols. Choices are `true` or `false` (default).
BlockreadValue	Scalar or vector that controls the reading of a single sequence entry or block of sequence entries from a FASTA-formatted file containing multiple sequences. Enter a scalar *N* to read the *N*th entry in the file. Enter a 1-by-2 vector [*M1*, *M2*] to read the block of entries starting at the *M1* entry and ending at the *M2* entry. To read all remaining entries in the file starting at the *M1* entry, enter a positive value for *M1* and enter `Inf` for *M2*.
TrimHeadersValue	Specifies whether to trim the header after the first white space character. White space characters include a space

(char(32)) and a tab (char(9)). Choices are `true` or `false` (default).

Output Arguments

FASTAData	MATLAB structure with the fields `Header` and `Sequence`.

Description

`fastaread` reads data from a FASTA-formatted file into a MATLAB structure with the following fields.

Field	Description
`Header`	Header information.
`Sequence`	Single letter-code representation of a nucleotide sequence.

A FASTA-formatted file begins with a right angle bracket (>) and a single line description. Following this description is the sequence as a series of lines with fewer than 80 characters. Sequences must use the standard IUB/IUPAC amino acid and nucleotide letter codes.

For a list of codes, see `aminolookup` and `baselookup`.

FASTAData = `fastaread`(*File*) reads a FASTA-formatted file and returns the data in a structure. *FASTAData*.`Header` is the header information, while *FASTAData*.`Sequence` is the sequence stored as a string of letters.

[*Header*, *Sequence*] = `fastaread`(*File*) reads data from a file into separate variables. If the file contains multiple sequences, then `Header` and `Sequence` are cell arrays of header and sequence information.

... = `fastaread`(*File*, ...'*PropertyName*', *PropertyValue*, ...) calls `fastaread` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. The property name/value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). These property name/property value pairs are as follows:

... = fastaread(*File*, ...'IgnoreGaps', *IgnoreGapsValue*, ...), when *IgnoreGapsValue* is true, removes any gap symbol ('-' or '.') from the sequences. Default is false.

... = fastaread(*File*, ...'Blockread', *BlockreadValue*, ...) lets you read in a single sequence entry or block of sequence entries from a file containing multiple sequences. If *BlockreadValue* is a scalar *N*, then fastaread reads the *N*th entry in the file. If *BlockreadValue* is a 1-by-2 vector [*M1, M2*], then fastaread reads the block of entries starting at the *M1* entry and ending at the *M2* entry. To read all remaining entries in the file starting at the *M1* entry, enter a positive value for *M1* and enter Inf for *M2*.

... = fastaread(*File*, ...'TrimHeaders', *TrimHeadersValue*, ...) specifies whether to trim the header to the first white space.

Examples

Read the sequence for the human p53 tumor gene:

```
p53nt = fastaread('p53nt.txt')
```

Read the sequence for the human p53 tumor protein:

```
p53aa = fastaread('p53aa.txt')
```

Read a block of entries from a FASTA file:

```
% Read the contents of reads 5 through 10 into an array of
% structures
pf2_5_10 = fastaread('pf00002.fa', 'blockread', [5 10], ...
                     'ignoregaps',true)
pf2_5_10 =

6x1 struct array with fields:
    Header
    Sequence
```

See Also

aminolookup | baselookup | BioIndexedFile | emblread | fastainfo | fastawrite | fastqinfo | fastqread | fastqwrite | genbankread | genpeptread | multialignread | saminfo | samread | seqprofile | seqviewer | sffinfo | sffread

fastawrite

Write to file using FASTA format

Syntax

```
fastawrite(File, Data)
fastawrite(File, Header, Sequence)
```

Arguments

File	String specifying either a file name or a path and file name for saving the FASTA-formatted data. If you specify only a file name, `fastawrite` saves the file to the MATLAB Current Folder. If you specify an existing file, `fastawrite` appends the data to the file, instead of overwriting the file.
Data	Any of the following: • String containing a sequence • MATLAB structure containing the fields `Header` and `Sequence` • MATLAB structure containing sequence information from the GenBank or GenPept database, such as returned by `genbankread`, `getgenbank`, `genpeptread`, or `getgenpept`.
Header	String or name of variable containing information about the sequence. This text appears in the header of the FASTA-formatted file, *File*.
Sequence	String or name of variable containing an amino acid or nucleotide sequence using the standard IUB/IUPAC letter or integer codes. For a list of valid characters, see Amino Acid Lookup or Nucleotide Lookup.

Description

`fastawrite(File, Data)` writes the contents of *Data* to *File*, a FASTA-formatted file. If you specify an existing FASTA-formatted file, `fastawrite` appends the data to the file, instead of overwriting the file.

fastawrite(*File*, *Header*, *Sequence*) writes the specified header and sequence information to *File*, a FASTA-formatted file.

Tip To append FASTA-formatted data to an existing file, simply specify that file name. fastawrite adds the data to the end of the file.

If you are using fastawrite in a script, you can disable the append warning message by entering the following command lines before the fastawrite command:

```
warnState = warning %Save the current warning state
warning('off','Bioinfo:fastawrite:AppendToFile');
```
Then enter the following command line after the fastawrite command:

```
warning(warnState) %Reset warning state to previous settings
```

Examples

Writing a Coding Region to a FASTA-Formatted File

1 Retrieve the sequence for the human p53 gene from the GenBank database.

```
seq = getgenbank('NM_000546');
```

2 Read the coordinates of the coding region in the CDS line.

```
start = seq.CDS.indices(1)

start =

    198

stop = seq.CDS.indices(2)

stop =

    1379
```

3 Extract the coding region.

```
codingSeq = seq.Sequence(start:stop);
```

4 Write the coding region to a FASTA-formatted file, specifying `Coding region for p53` for the Header in the file, and `p53coding.txt` for the file name.

```
fastawrite('p53coding.txt','Coding region for p53',codingSeq);
```

Saving Multiple Sequences to a FASTA-Formatted File

1   Write two nucleotide sequences to a MATLAB structure containing the fields
    Header and Sequence.

```
data(1).Sequence = 'ACACAGGAAA';
data(1).Header = 'First sequence';
data(2).Sequence = 'ACGTCAGGTC';
data(2).Header = 'Second sequence';
```

2   Write the sequences to a FASTA-formatted file, specifying my_sequences.txt for
    the file name.

```
fastawrite('my_sequences.txt', data)
```

3   Display the FASTA-formatted file, my_sequences.txt.

```
type('my_sequences.txt')

>First sequence
ACACAGGAAA

>Second sequence
ACGTCAGGTC
```

Appending Sequences to a FASTA-Formatted File

1   If you haven't already done so, create the FASTA-formatted file,
    my_sequences.txt, described in Saving Multiple Sequences to a FASTA-
    Formatted File.

2   Append a third sequence to the file.

```
fastawrite('my_sequences.txt','Third sequence','TACTGACTTC')
```

3   Display the FASTA-formatted file, my_sequences.txt.

```
type('my_sequences.txt')

>First sequence
ACACAGGAAA

>Second sequence
ACGTCAGGTC
```

```
>Third sequence
TACTGACTTC
```

See Also

fastainfo | fastaread | fastqinfo | fastqread | fastqwrite | genbankread
| genpeptread | getgenbank | getgenpept | multialignwrite | saminfo |
samread | seqviewer | sffinfo | sffread

fastqinfo

Return information about FASTQ file

Syntax

InfoStruct = fastqinfo(*File*)

Description

InfoStruct = fastqinfo(*File*) returns a MATLAB structure containing summary information about a FASTQ-formatted file.

Input Arguments

File

String specifying a file name or path and file name of a FASTQ-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

Default:

Output Arguments

InfoStruct

MATLAB structure containing summary information about a FASTQ-formatted file. The structure contains the following fields.

Field	Description
Filename	Name of the file.
FilePath	Path to the file
FileModDate	Modification date of the file.

Field	Description
FileSize	Size of the file in bytes.
NumberOfEntries	Number of sequence reads in the file.

Examples

Return a summary of the contents of a FASTQ file:

```
info = fastqinfo('SRR005164_1_50.fastq')

info =

          Filename: 'SRR005164_1_50.fastq'
          FilePath: 'D:\2010_08_24_h11m43s32_job6027_pass\matlab\toolbox\bioinfo\biode
       FileModDate: '03-Mar-2009 14:21:51'
          FileSize: 16702
   NumberOfEntries: 50
```

More About

•   http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?
    cmd=show&f=main&m=main&s=main

See Also

BioIndexedFile | BioRead | fastqread | fastqwrite | fastainfo | fastaread
| fastawrite | sffinfo | sffread | saminfo | samread

fastqread

Read data from FASTQ file

Syntax

```
FASTQStruct = fastqread(File)
[Header, Sequence] = fastqread(File)
[Header, Sequence, Qual] = fastqread(File)
fastqread(..., 'Blockread', BlockreadValue, ...)
fastqread(..., 'HeaderOnly', HeaderOnlyValue, ...)
fastqread(..., 'TrimHeaders', TrimHeadersValue, ...)
```

Description

FASTQStruct = `fastqread`(*File*) reads a FASTQ-formatted file and returns the data in a MATLAB array of structures.

[*Header*, *Sequence*] = `fastqread`(*File*) returns only the header and sequence data in two separate variables.

[*Header*, *Sequence*, *Qual*] = `fastqread`(*File*) returns the data in three separate variables.

`fastqread`(..., '*PropertyName*', *PropertyValue*, ...) calls `fastqread` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

`fastqread`(..., 'Blockread', *BlockreadValue*, ...) reads a single sequence entry or block of sequence entries from a FASTQ-formatted file containing multiple sequences.

`fastqread`(..., 'HeaderOnly', *HeaderOnlyValue*, ...) specifies whether to return only the header information.

fastqread(..., 'TrimHeaders', *TrimHeadersValue*, ...) specifies whether to trim the header to the first white space.

Input Arguments

File

Either of the following:

- String specifying a file name or path and file name of a FASTQ-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.
- MATLAB character array that contains the text of a FASTQ-formatted file.

Default:

BlockreadValue

Scalar or vector that controls the reading of a single sequence entry or block of sequence entries from a FASTQ-formatted file containing multiple sequences. Enter a scalar *N* to read the *N*th entry in the file. Enter a 1-by-2 vector [*M1, M2*] to read a block of entries starting at the *M1* entry and ending at the *M2* entry. To read all remaining entries in the file starting at the *M1* entry, enter a positive value for *M1* and enter Inf for *M2*.

Default:

HeaderOnlyValue

Specifies whether to return only the header information. Choices are true or false (default).

Default:

TrimHeadersValue

Specifies whether to trim the header after the first white space character. White space characters include a space (char(32)) and a tab (char(9)). Choices are true or false (default).

Default:

Output Arguments

`FASTQStruct`

Array of structures containing information from a FASTQ-formatted file. There is one structure for each sequence read or entry in the file. Each structure contains the following fields.

Field	Description
Header	Header information.
Sequence	Single letter-code representation of a nucleotide sequence.
Quality	ASCII representation of per-base quality scores for a nucleotide sequence.

`Header`

Variable containing header information or, if the FASTQ-formatted file contains multiple sequences, a cell array containing header information.

`Sequence`

Variable containing sequence information or, if the FASTQ-formatted file contains multiple sequences, a cell array containing sequence information.

`Qual`

Variable containing quality information or, if the FASTQ-formatted file contains multiple sequences, a cell array containing quality information.

Definitions

A FASTQ-formatted file contains nucleotide sequence and quality information on four lines:

- **Line 1** — Header information prefixed with an @ symbol
- **Line 2** — Nucleotide sequence
- **Line 3** — Header information prefixed with a + symbol
- **Line 4** — ASCII representation of per-base quality scores for the nucleotide sequence using Phred or Solexa encoding

Examples

Read a FASTQ file into an array of structures:

```
% Read the contents of a FASTQ-formatted file into
% an array of structures
reads = fastqread('SRR005164_1_50.fastq')

reads =

1x50 struct array with fields:
    Header
    Sequence
    Quality
```

Read a FASTQ file into three separate variables:

```
% Read the contents of a FASTQ-formatted file into
% three separate variables
[headers,seqs,quals] = fastqread('SRR005164_1_50.fastq');
```

Read a block of entries from a FASTQ file:

```
% Read the contents of reads 5 through 10 into
% an array of structures
reads_5_10 = fastqread('SRR005164_1_50.fastq', 'blockread', [5 10])

1x6 struct array with fields:
    Header
    Sequence
    Quality
```

More About

- http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?
  cmd=show&f=main&m=main&s=main

See Also

BioIndexedFile | BioRead | fastqwrite | fastaread | fastawrite | fastainfo
| fastqinfo | soapread | sffinfo | sffread | saminfo | samread | bamread |
baminfo | bamindexread

fastqwrite

Write to file using FASTQ format

Syntax

```
fastqwrite(File, FASTQStruct)
fastqwrite(File, Header, Sequence, Qual)
```

Description

fastqwrite(*File*, *FASTQStruct*) writes the contents of a MATLAB structure or array of structures to a FASTQ-formatted file. If you specify an existing FASTQ-formatted file, fastqwrite appends the data to the file, instead of overwriting the file.

fastqwrite(*File*, *Header*, *Sequence*, *Qual*) writes header, sequence, and quality information to a FASTQ-formatted file.

Tip To append FASTQ-formatted data to an existing file, simply specify that file name. fastqwrite adds the data to the end of the file.

If you are using fastqwrite in a script, you can disable the append warning message by entering the following command lines before the fastqwrite command:

```
warnState = warning %Save the current warning state
warning('off','Bioinfo:fastqwrite:AppendToFile');
```
Then enter the following command line after the fastqwrite command:

```
warning(warnState) %Reset warning state to previous settings
```

Input Arguments

File

String specifying either a file name or a path and file name for saving the FASTQ-formatted data. If you specify only a file name, fastqwrite saves the file to the

MATLAB Current Folder. If you specify an existing file, `fastqwrite` appends the data to the file, instead of overwriting the file.

Default:

FASTQStruct

MATLAB structure or array of structures containing the fields `Header`, `Sequence`, and `Quality`, such as returned by `fastqread`.

Default:

Header

String or name of a variable containing information about the nucleotide sequence. This text appears in the header of the FASTQ-formatted file, *File*.

Default:

Sequence

String or name of a variable containing a nucleotide sequence using the standard IUB/IUPAC letter or integer codes. For a list of valid characters, see Amino Acid Lookup or Nucleotide Lookup.

Default:

Qual

String or name of a variable containing ASCII representation of per-base quality scores for a nucleotide sequence.

Default:

Definitions

A FASTQ-formatted file contains nucleotide sequence and quality information on four lines:

- **Line 1** — Header information prefixed with an @ symbol
- **Line 2** — Nucleotide sequence

- **Line 3** — Header information prefixed with a + symbol
- **Line 4** — ASCII representation of per-base quality scores for the nucleotide sequence using Phred or Solexa encoding

Examples

Write multiple sequences to a FASTQ file from an array of structures:

```
% Read the contents of a FASTQ-formatted file into
% an array of structures
reads = fastqread('SRR005164_1_50.fastq');
% Create another array of structures for the first five reads
reads5 = reads(1:5);
% Write the first five reads to a separate FASTQ-formatted file
fastqwrite('fiveReads.fastq', reads5)
```

Write a single sequence to a FASTQ file from separate variables:

```
% Create separate variables for the header, sequence, and
% quality information of a nucleotide sequence
h = 'MYSEQ-OOO_1_1_1_953_493';
s = 'GTTACCATGATGTTATTTCTTCATTTGGAGGTAAAA';
q = ']]]]]]]]]]]]]]]]]]]]]]]]T]]]]RJRZTQLOA';
% Write the information to a FASTQ-formatted file
fastqwrite('oneRead.fastq', h, s, q)
```

More About

- Amino Acid Lookup
- Nucleotide Lookup
- http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?
  cmd=show&f=main&m=main&s=main

See Also

```
fastqread | fastqinfo | fastaread | fastawrite | fastainfo | sffinfo |
sffread | saminfo | samread | BioRead
```

featureData

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set feature metadata in ExpressionSet object

Syntax

```
MetaDataObj = featureData(ESObj)
NewESObj = featureData(ESObj, NewMetaDataObj)
```

Description

MetaDataObj = featureData(*ESObj*) returns a MetaData object containing the feature metadata from an ExpressionSet object.

NewESObj = featureData(*ESObj*, *NewMetaDataObj*) replaces the feature metadata in *ESObj*, an ExpressionSet object, with *NewMetaDataObj*, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the bioma.ExpressionSet class.

Default:

NewMetaDataObj

Object of the bioma.data.MetaData class, containing feature metadata, stored in two dataset arrays. The feature names and variable names in *NewMetaDataObj* must match the feature names and variable names in the *MetaDataObj* being replaced in the ExpressionSet object, *ESObj*.

Default:

Output Arguments

`MetaDataObj`

Object of the `bioma.data.MetaData` class, containing the feature metadata, stored in two dataset arrays.

`NewESObj`

Object of the `bioma.ExpressionSet` class, returned after replacing the MetaData object containing the feature metadata.

See Also
`bioma.ExpressionSet` | `bioma.data.MetaData` | `featureNames` | `sampleData`

How To

· "Managing Gene Expression Data in Objects"

featureNames

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set feature names in ExpressionSet object

Syntax

FeatNames = featureNames(*ESObj*)
FeatNames = featureNames(*ESObj*, *Subset*)
NewESObj = featureNames(*ESObj*, *Subset*, *NewFeatNames*)

Description

FeatNames = featureNames(*ESObj*) returns a cell array of strings specifying all feature names in an ExpressionSet object.

FeatNames = featureNames(*ESObj*, *Subset*) returns a cell array of strings specifying a subset the feature names in an ExpressionSet object.

NewESObj = featureNames(*ESObj*, *Subset*, *NewFeatNames*) replaces the feature names specified by *Subset* in *ESObj*, an ExpressionSet object, with *NewFeatNames*, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the bioma.ExpressionSet class.

Default:

Subset

One of the following to specify a subset of the feature names in an ExpressionSet object:

- String specifying a feature name
- Cell array of strings specifying feature names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewFeatNames

New feature names for specific feature names within an ExpressionSet object, specified by one of the following:

- Numeric vector
- String or cell array of strings
- String, which `featureNames` uses as a prefix for the feature names, with feature numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `featureNames` assigns unique feature names using the format `Feature1`, `Feature2`, etc.

The number of feature names in *NewFeatNames* must equal the number of features specified by *Subset*.

Default:

Output Arguments

FeatNames

Cell array of strings specifying all or some of the feature names in an ExpressionSet object. The feature names are the row names in the DataMatrix objects in the ExpressionSet object. The feature names are also the row names of the *VarValues* dataset array in the MetaData object in the ExpressionSet object.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing specific feature names.

See Also

`bioma.ExpressionSet` | `bioma.data.ExptData` | `bioma.data.MetaData` | `sampleNames` | `DataMatrix`

How To

·   "Managing Gene Expression Data in Objects"

featureNames

Class: bioma.data.ExptData
Package: bioma.data

Retrieve or set feature names in ExptData object

Syntax

```
FeatNames = featureNames(EDObj)
FeatNames = featureNames(EDObj, Subset)
NewESObj = featureNames(EDObj, Subset, NewFeatNames)
```

Description

FeatNames = `featureNames(`*EDObj*`)` returns a cell array of strings specifying all feature names in an ExptData object.

FeatNames = `featureNames(`*EDObj*`, `*Subset*`)` returns a cell array of strings specifying a subset the feature names in an ExptData object.

NewESObj = `featureNames(`*EDObj*`, `*Subset*`, `*NewFeatNames*`)` replaces the feature names specified by *Subset* in *EDObj*, an ExptData object, with *NewFeatNames*, and returns *NewEDObj*, a new ExptData object.

Input Arguments

EDObj

Object of the `bioma.data.ExptData` class.

Default:

Subset

One of the following to specify a subset of the feature names in an ExptData object:

- String specifying a feature name

- Cell array of strings specifying feature names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewFeatNames

New feature names for specific feature names within an ExptData object, specified by one of the following:

- Numeric vector
- String or cell array of strings
- String, which `featureNames` uses as a prefix for the feature names, with feature numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `featureNames` assigns unique feature names using the format `Feature1`, `Feature2`, etc.

The number of feature names in *NewFeatNames* must equal the number of features specified by *Subset*.

Default:

Output Arguments

FeatNames

Cell array of strings specifying all or some of the feature names in an ExptData object. The feature names are the row names in the DataMatrix objects in the ExptData object.

NewEDObj

Object of the `bioma.data.ExptData` class, returned after replacing specific feature names.

Examples

Construct an ExptData object, and then retrieve the feature names from it:

```
% Import bioma.data package to make constructor functions
% available
import bioma.data.*
% Create DataMatrix object from .txt file containing
% expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Retrieve feature names
FNames = featureNames(EDObj);
```

See Also

bioma.data.ExptData | dmNames | DataMatrix | elementNames | sampleNames

How To

· "Representing Expression Data Values in ExptData Objects"

featuresmap

Draw linear or circular map of features from GenBank structure

Syntax

```
featuresmap(GBStructure)
featuresmap(GBStructure, FeatList)
featuresmap(GBStructure, FeatList, Levels)
featuresmap(GBStructure, Levels)
[Handles, OutFeatList] = featuresmap(...)
featuresmap(..., 'FontSize', FontSizeValue, ...)
featuresmap(..., 'ColorMap', ColorMapValue, ...)
featuresmap(..., 'Qualifiers', QualifiersValue, ...)
featuresmap(..., 'ShowPositions', ShowPositionsValue, ...)
```

Arguments

GBStructure	GenBank structure, typically created using the getgenbank or the genbankread function.
FeatList	Cell array of features (from the list of all features in the GenBank structure) to include in or exclude from the map. • If FeatList is a cell array of features, these features are mapped. Any features in FeatList not found in the GenBank structure are ignored. • If FeatList includes '-' as the first string in the cell array, then the remaining strings (features) are not mapped. By default, FeatList is the a list of all features in the GenBank structure.
Levels	Vector of N integers, where N is the number of features. Each integer represents the level in the map for the corresponding feature. For example, if Levels = [1, 1, 2, 3, 3], the first two features would appear on level

	1, the third feature on level 2, and the fourth and fifth features on level 3. By default, *Levels* = [1:N].
FontSizeValue	Scalar that sets the font size (points) for the annotations of the features. Default is 9.
ColorMapValue	Three-column matrix, to specify a list of colors to use for each feature. This matrix replaces the default matrix, which specifies the following colors and order: blue, green, red, cyan, magenta, yellow, brown, light green, orange, purple, gold, and silver. In the matrix, each row corresponds to a color, and each column specifies red, green, and blue intensity respectively. Valid values for the RGB intensities are 0.0 to 1.0.
QualifiersValue	Cell array of strings to specify an ordered list of qualifiers to search for in the structure and use as annotations. For each feature, the first matching qualifier found from the list is used for its annotation. If a feature does not include any of the qualifiers, no annotation displays for that feature. By default, *QualifiersValue* = {'gene', 'product', 'locus_tag', 'note', 'db_xref', 'protein_id'}. Provide your own *QualifiersValue* to limit or expand the list of qualifiers or change the search order.
	Tip Set *QualifiersValue* = {} to create a map with no annotations.
	Tip To determine all qualifiers available for a given feature, do either of the following:
	• Create the map, and then click a feature or its annotation to list all qualifiers for that feature.
	• Use the featuresparse command to parse all the features into a new structure, and then use the fieldnames command to list the qualifiers for a specific feature. See Determining Qualifiers for a Specific Feature.

ShowPositionsValue	Property to add the sequence position to the annotation label for each feature. Enter `true` to add the sequence position. Default is `false`.

Description

featuresmap(*GBStructure*) creates a linear or circular map of all features from a GenBank structure, typically created using the `getgenbank` or the `genbankread` function.

featuresmap(*GBStructure*, *FeatList*) creates a linear or circular map of a subset of features from a GenBank structure. *FeatList* lets you specify features (from the list of all features in the GenBank structure) to include in or exclude from the map.

- If *FeatList* is a cell array of features, these features are mapped. Any features in *FeatList* not found in the GenBank structure are ignored.
- If *FeatList* includes '-' as the first string in the cell array, then the remaining strings (features) are not mapped.

By default, *FeatList* is a list of all features in the GenBank structure.

featuresmap(*GBStructure*, *FeatList*, *Levels*) or featuresmap(*GBStructure*, *Levels*) indicates which level on the map each feature is drawn. Level 1 is the left-most (linear map) or inner-most (circular map) level, and level N is the right-most (linear map) or outer-most (circular map) level, where N is the number of features.

Levels is a vector of N integers, where N is the number of features. Each integer represents the level in the map for the corresponding feature. For example, if *Levels* = [1, 1, 2, 3, 3], the first two features would appear on level 1, the third feature on level 2, and the fourth and fifth features on level 3. By default, *Levels* = [1:N].

[*Handles*, *OutFeatList*] = featuresmap(...) returns a list of handles for each feature in *OutFeatList*. It also returns *OutFeatList*, which is a cell array of the mapped features.

Tip Use *Handles* and *OutFeatList* with the `legend` command to create a legend of features.

`featuresmap(..., 'PropertyName', PropertyValue, ...)` defines optional properties that use property name/value pairs in any order. These property name/value pairs are as follows:

`featuresmap(..., 'FontSize', FontSizeValue, ...)` sets the font size (points) for the annotations of the features. Default *FontSizeValue* is `9`.

`featuresmap(..., 'ColorMap', ColorMapValue, ...)` specifies a list of colors to use for each feature. This matrix replaces the default matrix, which specifies the following colors and order: blue, green, red, cyan, magenta, yellow, brown, light green, orange, purple, gold, and silver. *ColorMapValue* is a three-column matrix, where each row corresponds to a color, and each column specifies red, green, and blue intensity respectively. Valid values for the RGB intensities are `0.0` to `1.0`.

`featuresmap(..., 'Qualifiers', QualifiersValue, ...)` lets you specify an ordered list of qualifiers to search for and use as annotations. For each feature, the first matching qualifier found from the list is used for its annotation. If a feature does not include any of the qualifiers, no annotation displays for that feature. *QualifiersValue* is a cell array of strings. By default, *QualifiersValue* = {`'gene', 'product', 'locus_tag', 'note', 'db_xref', 'protein_id'`}. Provide your own *QualifiersValue* to limit or expand the list of qualifiers or change the search order.

Tip Set *QualifiersValue* = `{}` to create a map with no annotations.

Tip To determine all qualifiers available for a given feature, do either of the following:

- Create the map, and then click a feature or its annotation to list all qualifiers for that feature.

- Use the `featuresparse` command to parse all the features into a new structure, and then use the `fieldnames` command to list the qualifiers for a specific feature. See Determining Qualifiers for a Specific Feature.

`featuresmap(..., 'ShowPositions', ShowPositionsValue, ...)` lets you add the sequence position to the annotation label. If *ShowPositionsValue* is `true`, sequence positions are added to the annotation labels. Default is `false`.
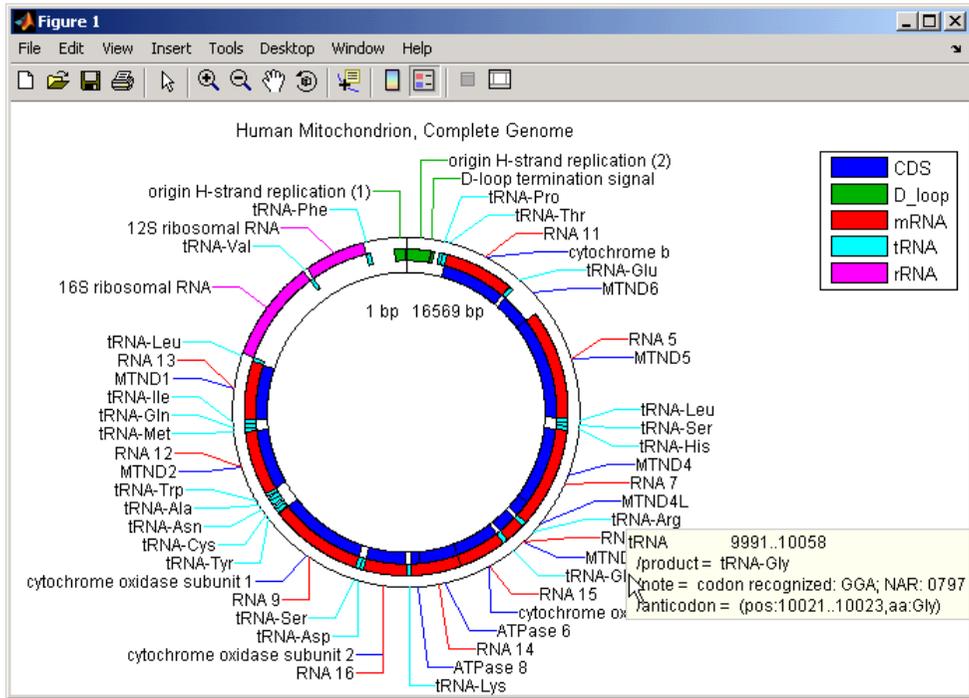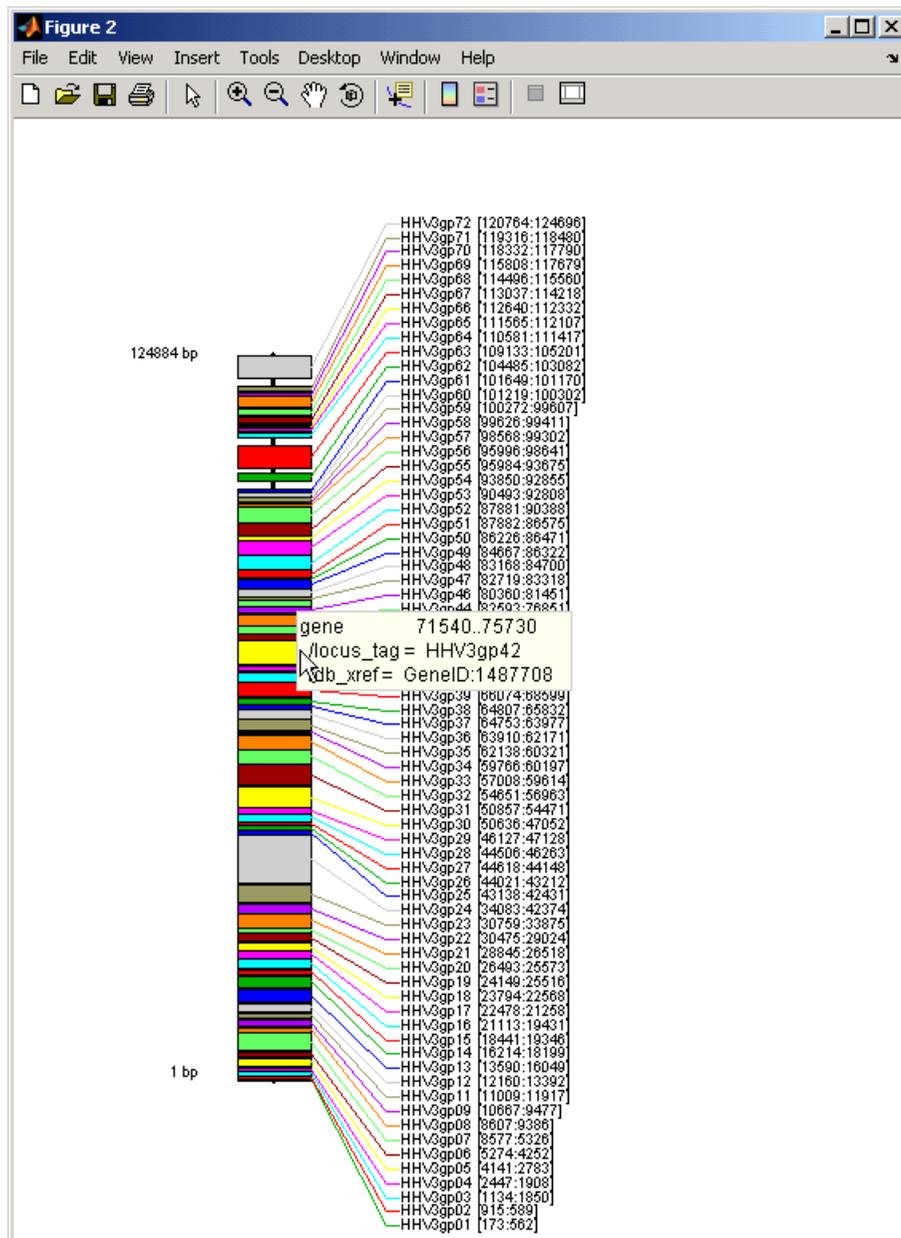
After creating a map:

- Click a feature or annotation to display a list of all qualifiers for that feature.
- Zoom the plot by clicking the following buttons:

 or

Examples

Creating a Circular Map with a Legend

The following example creates a circular map of five different features mapped on three levels. It also uses outputs from the `featuresmap` function as inputs to the `legend` function to add a legend to the map.

```
GBStructure = getgenbank('J01415');
[Handles, OutFeatList] = featuresmap(GBStructure, ...
    {'CDS','D_loop','mRNA','tRNA','rRNA'}, [1 2 2 2 3])
legend(Handles, OutFeatList, 'interpreter', 'none', ...
    'location','bestoutside')
title('Human Mitochondrion, Complete Genome')
```

Creating a Linear Map with Sequence Position Labels and Changed Font Size

The following example creates a linear map showing only the gene feature. It changes the font of the labels to seven points and includes the sequence position in the labels.

```
herpes = getgenbank('NC_001348');
featuresmap(herpes,{'gene'},'fontsize',7,'showpositions',true)
title('Genes in Human herpesvirus 3 (strain Dumas)')
```

Determining Qualifiers for a Specific Feature

The following example uses the `getgenbank` function to create a GenBank structure, `GBStructure`. It then uses the `featuresparse` function to parse the features in the GenBank structure into a new structure, `features`. It then uses the `fieldnames` function to return all qualifiers for one of the features, `D_loop`.

```
GenBankStructure = getgenbank('J01415');
features = featuresparse (GenBankStructure)
features =
```

```
          source: [1x1 struct]
          D_loop: [1x2 struct]
       rep_origin: [1x3 struct]
      repeat_unit: [1x4 struct]
      misc_signal: [1x1 struct]
         misc_RNA: [1x1 struct]
        variation: [1x17 struct]
             tRNA: [1x22 struct]
             rRNA: [1x2 struct]
             mRNA: [1x10 struct]
              CDS: [1x13 struct]
          conflict: [1x1 struct]

fieldnames(features.D_loop)

ans =

    'Location'
    'Indices'
    'note'
    'citation'
```

See Also
featuresparse | genbankread | getgenbank | seqviewer

featuresparse

Parse features from GenBank, GenPept, or EMBL data

Syntax

```
FeatStruct = featuresparse(Features)
FeatStruct = featuresparse(Features, ...'Feature',
FeatureValue, ...)
FeatStruct = featuresparse(Features, ...'Sequence',
SequenceValue, ...)
```

Input Arguments

Features	Any of the following: • String containing GenBank, GenPept, or EMBL features • MATLAB character array including text describing GenBank, GenPept, or EMBL features • MATLAB structure with fields corresponding to GenBank, GenPept, or EMBL data, such as those returned by `genbankread`, `genpeptread`, `emblread`, `getgenbank`, `getgenpept`, or `getembl`
FeatureValue	Name of a feature contained in *Features*. When specified, `featuresparse` returns only the substructure that corresponds to this feature. If there are multiple features with the same *FeatureValue*, then *FeatStruct* is an array of structures.
SequenceValue	Property to control the extraction, when possible, of the sequences respective to each feature, joining and complementing pieces of the source sequence and storing them in the `Sequence` field of the returned structure, *FeatStruct*. When extracting the sequence from an incomplete CDS feature, `featuresparse` uses the `codon_start` qualifier to adjust the frame of the sequence. Choices are `true` or `false` (default).

Output Arguments

FeatStruct	Output structure containing a field for every database feature. Each field name in *FeatStruct* matches the corresponding feature name in the GenBank, GenPept, or EMBL database, with the exceptions listed in the table below. Fields in *FeatStruct* contain substructures with feature qualifiers as fields. In the GenBank, GenPept, and EMBL databases, for each feature, the only mandatory qualifier is its location, which `featuresparse` translates to the field `Location`. When possible, `featuresparse` also translates this location to numeric indices, creating an `Indices` field.
	Note: If you use the `Indices` field to extract sequence information, you may need to complement the sequences.

Description

FeatStruct = featuresparse(*Features*) parses the features from *Features*, which contains GenBank, GenPept, or EMBL features. *Features* can be a:

- String containing GenBank, GenPept, or EMBL features
- MATLAB character array including text describing GenBank, GenPept, or EMBL features
- MATLAB structure with fields corresponding to GenBank, GenPept, or EMBL data, such as those returned by genbankread, genpeptread, emblread, getgenbank, getgenpept, or getembl

FeatStruct is the output structure containing a field for every database feature. Each field name in *FeatStruct* matches the corresponding feature name in the GenBank, GenPept, or EMBL database, with the following exceptions.

Feature Name in GenBank, GenPept, or EMBL Database	Field Name in MATLAB Structure
-10_signal	minus_10_signal

Feature Name in GenBank, GenPept, or EMBL Database	Field Name in MATLAB Structure
`-35_signal`	`minus_35_signal`
`3'UTR`	`three_prime_UTR`
`3'clip`	`three_prime_clip`
`5'UTR`	`five_prime_UTR`
`5'clip`	`five_prime_clip`
`D-loop`	`D_loop`

Fields in *FeatStruct* contain substructures with feature qualifiers as fields. In the GenBank, GenPept, and EMBL databases, for each feature, the only mandatory qualifier is its location, which `featuresparse` translates to the field `Location`. When possible, `featuresparse` also translates this location to numeric indices, creating an `Indices` field.

Note: If you use the `Indices` field to extract sequence information, you may need to complement the sequences.

FeatStruct = featuresparse (*Features*, ...'*PropertyName*', *PropertyValue*, ...) calls `featuresparse` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

FeatStruct = featuresparse(*Features*, ...'Feature', *FeatureValue*, ...) returns only the substructure that corresponds to *FeatureValue*, the name of a feature contained in *Features*. If there are multiple features with the same *FeatureValue*, then *FeatStruct* is an array of structures.

FeatStruct = featuresparse(*Features*, ...'Sequence', *SequenceValue*, ...) controls the extraction, when possible, of the sequences respective to each feature, joining and complementing pieces of the source sequence and storing them in the field `Sequence`. When extracting the sequence from an incomplete CDS feature, `featuresparse` uses the `codon_start` qualifier to adjust the frame of the sequence. Choices are `true` or `false` (default).

Examples

Obtaining All Features from a GenBank File

The following example obtains all the features stored in the GenBank file `nm175642.txt`:

```
gbkStruct = genbankread('nm175642.txt');
features = featuresparse(gbkStruct)

features =

    source: [1x1 struct]
      gene: [1x1 struct]
       CDS: [1x1 struct]
```

Obtaining a Subset of Features from a GenBank Record

The following example obtains only the coding sequences (CDS) feature of the *Caenorhabditis elegans* cosmid record (accession number Z92777) from the GenBank database:

```
worm = getgenbank('Z92777');
CDS = featuresparse(worm,'feature','cds')

CDS =

1x12 struct array with fields:
    Location
    Indices
    locus_tag
    standard_name
    note
    codon_start
    product
    protein_id
    db_xref
    translation
```

Extracting Sequences for Each Feature

1 Retrieve two nucleotide sequences from the GenBank database for the neuraminidase (NA) protein of two strains of the Influenza A virus (H5N1).

```
hk01 = getgenbank('AF509094');
vt04 = getgenbank('DQ094287');
```

2 Extract the sequence of the coding region for the neuraminidase (NA) protein from the two nucleotide sequences. The sequences of the coding regions are stored in the Sequence fields of the returned structures, hk01_cds and vt04_cds.

```
hk01_cds = featuresparse(hk01,'feature','CDS','Sequence',true);
vt04_cds = featuresparse(vt04,'feature','CDS','Sequence',true);
```

3 Once you have extracted the nucleotide sequences, you can use the nt2aa and nwalign functions to align the amino acids sequences converted from the nucleotide sequences.

```
[sc,al]=nwalign(nt2aa(hk01_cds),nt2aa(vt04_cds),'extendgap',1);
```

4 Then you can use the seqinsertgaps function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

```
hk01_aligned = seqinsertgaps(hk01_cds,al(1,:))
vt04_aligned = seqinsertgaps(vt04_cds,al(3,:))
```

5 Once you have code aligned the two sequences, you can use them as input to other functions such as dnds, which calculates the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences. By setting Verbose to true, you can also display the codons considered in the computations and their amino acid translations.

```
[dn,ds] = dnds(hk01_aligned,vt04_aligned,'verbose',true)
```

See Also

emblread | genbankread | genpeptread | getgenbank | getgenpept

featureVarDesc

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set feature variable descriptions in ExpressionSet object

Syntax

DSVarDescriptions = featureVarDesc(*ESObj*)
NewESObj = featureVarDesc(*ESObj*, *NewDSVarDescriptions*)

Description

DSVarDescriptions = featureVarDesc(*ESObj*) returns a dataset array
containing the feature variable names and descriptions from the MetaData object in an
ExpressionSet object.

NewESObj = featureVarDesc(*ESObj*, *NewDSVarDescriptions*) replaces
the feature variable descriptions in *ESObj*, an ExpressionSet object, with
NewDSVarDescriptions, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the bioma.ExpressionSet class.

Default:

NewDSVarDescriptions

Descriptions of the feature variable names, specified by either of the following:

- A new dataset array containing the feature variable names and descriptions. In
  this dataset array, each row corresponds to a variable. The first column contains
  the variable name, and the second column (VariableDescription) contains a

description of the variable. The row names (variable names) must match the row names (variable names) in *DSVarDescriptions*, the dataset array being replaced in the MetaData object in the ExpressionSet object, *ESObj*.

- Cell array of strings containing descriptions of the feature variables. The number of elements in *VarDesc* must equal the number of row names (variable names) in *DSVarDescriptions*, the dataset array being replaced in the MetaData object in the ExpressionSet object, *ESObj*.

Default:

Output Arguments

DSVarDescriptions

A dataset array containing the feature variable names and descriptions from the MetaData object of an ExpressionSet object. In this dataset array, each row corresponds to a variable. The first column contains the variable name, and the second column (VariableDescription) contains a description of the variable.

NewESObj

Object of the bioma.ExpressionSet class, returned after replacing the dataset array containing the feature variable descriptions.

See Also
bioma.ExpressionSet | bioma.data.MetaData | variableDesc

How To

- "Managing Gene Expression Data in Objects"

featureVarNames

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set feature variable names in ExpressionSet object

Syntax

FeatVarNames = featureVarNames(*ESObj*)
FeatVarNames = featureVarNames(*ESObj*, *Subset*)
NewESObj = featureVarNames(*ESObj*, *Subset*, *NewFeatVarNames*)

Description

FeatVarNames = featureVarNames(*ESObj*) returns a cell array of strings specifying all feature variable names in an ExpressionSet object.

FeatVarNames = featureVarNames(*ESObj*, *Subset*) returns a cell array of strings specifying a subset the feature variable names in an ExpressionSet object.

NewESObj = featureVarNames(*ESObj*, *Subset*, *NewFeatVarNames*) replaces the feature variable names specified by *Subset* in *ESObj*, an ExpressionSet object, with *NewFeatVarNames*, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the bioma.ExpressionSet class.

Default:

Subset

One of the following to specify a subset of the feature variable names in an ExpressionSet object:

· String specifying a feature variable name

- Cell array of strings specifying feature variable names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewFeatVarNames

New feature variable names for specific feature variable names within an ExpressionSet object, specified by one of the following:

- Numeric vector
- Cell array of strings
- Character array
- String, which `featureVarNames` uses as a prefix for the feature variable names, with feature variable numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `featureVarNames` assigns unique feature variable names using the format `Var1`, `Var2`, etc.

The number of feature variable names in *NewFeatVarNames* must equal the number of feature variable names specified by *Subset*.

Default:

Output Arguments

FeatVarNames

Cell array of strings specifying all or some of the feature variable names in an ExpressionSet object. The feature variable names are the column names of the *VarValues* dataset array. The feature variable names are also the row names of the *VarDescriptions* dataset array. Both dataset arrays are in the MetaData object in the ExpressionSet object.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing specific feature names.

See Also
`bioma.ExpressionSet` | `bioma.data.MetaData` | `sampleNames` | `featureNames` | `sampleVarNames`

How To
- "Managing Gene Expression Data in Objects"

featureVarValues

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set feature variable data values in ExpressionSet object

Syntax

```
DSVarValues = featureVarValues(ESObj)
NewESObj = featureVarValues(ESObj, NewDSVarValues)
```

Description

DSVarValues = featureVarValues(*ESObj*) returns a dataset array containing the measured value of each variable per feature from the MetaData object of an ExpressionSet object.

NewESObj = featureVarValues(*ESObj*, *NewDSVarValues*) replaces the feature variable values in *ESObj*, an ExpressionSet object, with *NewDSVarValues*, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

NewDSVarValues

A dataset array containing a value for each variable per feature. In this dataset array, the columns correspond to variables and rows correspond to feature. The row names (feature names) must match the row names (feature names) in *DSVarValues*, the dataset array being replaced in the MetaData object in the ExpressionSet object, *ESObj*.

Default:

Output Arguments

DSVarValues

A dataset array containing the measured value of each variable per feature from the MetaData object of an ExpressionSet object. In this dataset array, the columns correspond to variables and rows correspond to features.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing the dataset array containing the feature variable values.

See Also
`bioma.ExpressionSet` | `bioma.data.MetaData` | `variableValues`

How To
•   "Managing Gene Expression Data in Objects"

galread

Read microarray data from GenePix array list file

Syntax

GALData = galread('*File*')

Arguments

File	GenePix® array list formatted file (GAL). Enter a file name, or enter a path and file name.

Description

galread reads data from a GenePix formatted file into a MATLAB structure.

GALData = galread('*File*') reads in a GenePix array list formatted file (*File*) and creates a structure (GALData) containing the following fields.

Field
Header
BlockData
IDs
Names

The field BlockData is an N-by-3 array. The columns of this array are the block data, the column data, and the row data respectively. For more information on the GAL format, see

http://support.moleculardevices.com/pages/software/gn_genepix_file_formats.html#gal

For a list of supported file format versions, see

http://support.moleculardevices.com/pages/software/gn_genepix_file_formats.html

See Also
affyread | geoseriesread | geosoftread | gprread | ilmnbsread |
imageneread | sptread

gcrma

Perform GC Robust Multi-array Average (GCRMA) background adjustment, quantile normalization, and median-polish summarization on Affymetrix microarray probe-level data

Syntax

ExpressionMatrix = gcrma(*PMMatrix*, *MMMatrix*, *ProbeIndices*, *AffinPM*, *AffinMM*)
ExpressionMatrix = gcrma(*PMMatrix*, *MMMatrix*, *ProbeIndices*, *SequenceMatrix*)
ExpressionMatrix = gcrma(..., 'ChipIndex', *ChipIndexValue*, ...)
ExpressionMatrix = gcrma(..., 'OpticalCorr', *OpticalCorrValue*, ...)
ExpressionMatrix = gcrma(..., 'CorrConst', *CorrConstValue*, ...)
ExpressionMatrix = gcrma(..., 'Method', *MethodValue*, ...)
ExpressionMatrix = gcrma(..., 'TuningParam', *TuningParamValue*, ...)
ExpressionMatrix = gcrma(..., 'GSBCorr', *GSBCorrValue*, ...)
ExpressionMatrix = gcrma(..., 'Normalize', *NormalizeValue*, ...)
ExpressionMatrix = gcrma(..., 'Verbose', *VerboseValue*, ...)

Input Arguments

PMMatrix	Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.) **Tip** You can use the `PMIntensities` matrix returned by the `celintensityread` function.
MMMatrix	Matrix of intensity values where each row corresponds to a mismatch (MM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)

	Tip You can use the `MMIntensities` matrix returned by the `celintensityread` function.
ProbeIndices	Column vector containing probe indices. Probes within a probe set are numbered 0 through $N - 1$, where N is the number of probes in the probe set.
	Tip You can use the `affyprobeseqread` function to generate this column vector.
AffinPM	Column vector of PM probe affinities.
	Tip You can use the `affyprobeaffinities` function to generate this column vector.
AffinMM	Column vector of MM probe affinities.
	Tip You can use the `affyprobeaffinities` function to generate this column vector.
SequenceMatrix	An N-by-25 matrix of sequence information for the perfect match (PM) probes on the Affymetrix GeneChip array, where N is the number of probes on the array. Each row corresponds to a probe, and each column corresponds to one of the 25 sequence positions. Nucleotides in the sequences are represented by one of the following integers: • 0 — None • 1 — A • 2 — C • 3 — G • 4 — T
	Tip You can use the `affyprobeseqread` function to generate this matrix. If you have this sequence information in letter representation, you can convert it to integer representation using the `nt2int` function.

ChipIndexValue	Positive integer specifying a column index in *MMMatrix*, which specifies a chip. This chip intensity data is used to compute probe affinities. Default is 1.
OpticalCorrValue	Controls the use of optical background correction on the PM and MM intensity values in *PMMatrix* and *MMMatrix*. Choices are true (default) or false.
CorrConstValue	Value that specifies the correlation constant, rho, for background intensity for each PM/MM probe pair. Choices are any value ≥ 0 and ≤ 1. Default is 0.7.
MethodValue	String that specifies the method to estimate the signal. Choices are MLE, a faster, ad hoc Maximum Likelihood Estimate method, or EB, a slower, more formal, empirical Bayes method. Default is MLE.
TuningParamValue	Value that specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB). **Tip** For information on determining a setting for this parameter, see Wu et al., 2004.
GSBCorrValue	Specifies whether to perform gene-specific binding (GSB) correction using probe affinity data. Choices are true (default) or false. If there is no probe affinity information, this property is ignored.
NormalizeValue	Controls whether quantile normalization is performed on background adjusted data. Choices are true (default) or false.
VerboseValue	Controls the display of a progress report showing the number of each chip as it is completed. Choices are true (default) or false.

Output Arguments

ExpressionMatrix	Matrix of log$_2$ expression values where each row corresponds to a gene (probe set) and each column corresponds to an Affymetrix CEL file, which represents a single chip.

Description

ExpressionMatrix = gcrma(*PMMatrix, MMMatrix, ProbeIndices, AffinPM, AffinMM*) performs GCRMA background adjustment, quantile normalization, and median-polish summarization on Affymetrix microarray probe-level data using probe affinity data. *ExpressionMatrix* is a matrix of log$_2$ expression values where each row corresponds to a gene (probe set) and each column corresponds to an Affymetrix CEL file, which represents a single chip.

Note: There is no column in *ExpressionMatrix* that contains probe set or gene information.

ExpressionMatrix = gcrma(*PMMatrix, MMMatrix, ProbeIndices, SequenceMatrix*) performs GCRMA background adjustment, quantile normalization, and Robust Multi-array Average (RMA) summarization on Affymetrix microarray probe-level data using probe sequence data to compute probe affinity data. *ExpressionMatrix* is a matrix of log$_2$ expression values where each row corresponds to a gene (probe set) and each column corresponds to an Affymetrix CEL file, which represents a single chip.

Note: If *AffinPM* and *AffinMM* affinity data and *SequenceMatrix* sequence data are not available, you can still use the gcrma function by entering an empty matrix for these inputs in the syntax.

ExpressionMatrix = gcrma(...'*PropertyName*', *PropertyValue*, ...) calls gcrma with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

ExpressionMatrix = gcrma(..., 'ChipIndex', *ChipIndexValue*, ...) computes probe affinities from MM probe intensity data from the chip with the specified column index in *MMMatrix*. Default *ChipIndexValue* is 1. If *AffinPM* and *AffinMM* affinity data are provided, this property is ignored.

ExpressionMatrix = gcrma(..., 'OpticalCorr', *OpticalCorrValue*, ...) controls the use of optical background correction on the PM and MM intensity values in *PMMatrix* and *MMMatrix*. Choices are true (default) or false.

ExpressionMatrix = gcrma(..., 'CorrConst', *CorrConstValue*, ...) specifies the correlation constant, rho, for background intensity for each PM/MM probe pair. Choices are any value ≥ 0 and ≤ 1. Default is 0.7.

ExpressionMatrix = gcrma(..., 'Method', *MethodValue*, ...) specifies the method to estimate the signal. Choices are MLE, a faster, ad hoc Maximum Likelihood Estimate method, or EB, a slower, more formal, empirical Bayes method. Default is MLE.

ExpressionMatrix = gcrma(..., 'TuningParam', *TuningParamValue*, ...) specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

Tip For information on determining a setting for this parameter, see Wu et al., 2004.

ExpressionMatrix = gcrma(..., 'GSBCorr', *GSBCorrValue*, ...) specifies whether to perform gene specific binding (GSB) correction using probe affinity data. Choices are true (default) or false. If there is no probe affinity information, this property is ignored.

ExpressionMatrix = gcrma(..., 'Normalize', *NormalizeValue*, ...) controls whether quantile normalization is performed on background adjusted data. Choices are true (default) or false.

ExpressionMatrix = gcrma(..., 'Verbose', *VerboseValue*, ...) controls the display of a progress report showing the number of each chip as it is completed. Choices are true (default) or false.

Examples

1 Load the MAT-file, included with the Bioinformatics Toolbox software, that contains Affymetrix data from a prostate cancer study. The variables in the MAT-file include `seqMatrix`, a matrix containing sequence information for PM probes, `pmMatrix` and `mmMatrix`, matrices containing PM and MM probe intensity values, and `probeIndices`, a column vector containing probe indexing information.

```
load prostatecancerrawdata
```

2 Compute the Affymetrix PM and MM probe affinities from their sequences and MM probe intensities.

```
[apm, amm] = affyprobeaffinities(seqMatrix, mmMatrix(:,1),...
              'ProbeIndices', probeIndices);
```

3 Perform GCRMA background adjustment, quantile normalization, and Robust Multi-array Average (RMA) summarization on the Affymetrix microarray probe-level data and create a matrix of expression values.

```
expdata = gcrma(pmMatrix, mmMatrix, probeIndices, seqMatrix);
```

The `prostatecancerrawdata.mat` file used in this example contains data from Best et al., 2005.

References

[1] Wu, Z., Irizarry, R.A., Gentleman, R., Murillo, F.M., and Spencer, F. (2004). A Model Based Background Adjustment for Oligonucleotide Expression Arrays. Journal of the American Statistical Association *99(468)*, 909–917.

[2] Wu, Z., and Irizarry, R.A. (2005). Stochastic Models Inspired by Hybridization Theory for Short Oligonucleotide Arrays. Proceedings of RECOMB 2004. J Comput Biol. *12(6)*, 882–93.

[3] Wu, Z., and Irizarry, R.A. (2005). A Statistical Framework for the Analysis of Microarray Probe-Level Data. Johns Hopkins University, Biostatistics Working Papers 73.

[4] Speed, T. (2006). Background models and GCRMA. Lecture 10, Statistics 246, University of California Berkeley. http://www.stat.berkeley.edu/users/terry/Classes/s246.2006/Week10/Week10L1.pdf.

[5] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

See Also

affygcrma | affyprobeseqread | affyread | affyrma | celintensityread | gcrmabackadj | quantilenorm | rmabackadj | rmasummary

gcrmabackadj

Perform GC Robust Multi-array Average (GCRMA) background adjustment on Affymetrix microarray probe-level data using sequence information

Syntax

PMMatrix_Adj = gcrmabackadj(*PMMatrix*, *MMMatrix*, *AffinPM*, *AffinMM*)
[*PMMatrix_Adj*, *nsbStruct*] = gcrmabackadj(*PMMatrix*, *MMMatrix*,
AffinPM, *AffinMM*)
... = gcrmabackadj(...'OpticalCorr', *OpticalCorrValue*, ...)
... = gcrmabackadj(...'CorrConst', *CorrConstValue*, ...)
... = gcrmabackadj(...'Method', *MethodValue*, ...)
... = gcrmabackadj(...'TuningParam', *TuningParamValue*, ...)
... = gcrmabackadj(...'AddVariance', *AddVarianceValue*, ...)
... = gcrmabackadj(...'GSBCorr', *GSBCorrValue*, ...)
... = gcrmabackadj(...'Showplot', *ShowplotValue*, ...)
... = gcrmabackadj(...'Verbose', *VerboseValue*, ...)

Input Arguments

PMMatrix	Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)
	Tip You can use the PMIntensities matrix returned by the celintensityread function.
MMMatrix	Matrix of intensity values where each row corresponds to a mismatch (MM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)
	Tip You can use the MMIntensities matrix returned by the celintensityread function.

AffinPM	Column vector of PM probe affinities, such as returned by the `affyprobeaffinities` function. Each row corresponds to a probe.
AffinMM	Column vector of MM probe affinities, such as returned by the `affyprobeaffinities` function. Each row corresponds to a probe.
OpticalCorrValue	Controls the use of optical background correction on the PM and MM probe intensity values in *PMMatrix* and *MMMatrix*. Choices are `true` (default) or `false`.
CorrConstValue	Value that specifies the correlation constant, rho, for log background intensity for each PM/MM probe pair. Choices are any value ≥ 0 and ≤ 1. Default is `0.7`.
MethodValue	String that specifies the method to estimate the signal. Choices are `MLE`, a faster, ad hoc Maximum Likelihood Estimate method, or `EB`, a slower, more formal, empirical Bayes method. Default is `MLE`.
TuningParamValue	Value that specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is `5` (MLE) or `0.5` (EB). **Tip** For information on determining a setting for this parameter, see Wu et al., 2004.
AddVarianceValue	Controls whether the signal variance is added to the weight function for smoothing low signal edge. Choices are `true` or `false` (default).
GSBCorrValue	Specifies whether to perform gene-specific binding (GSB) correction using probe affinity data. Choices are `true` (default) or `false`. If there is no probe affinity information, this property is ignored.

ShowplotValue	Controls the display of a plot showing the $\log_2$ of probe intensity values from a specified column (chip) in *MMMatrix*, versus probe affinities in *AffinMM*. Choices are true, false, or *I*, an integer specifying a column in *MMMatrix*. If set to true, the first column in *MMMatrix* is plotted. Default is: • false — When return values are specified. • true — When return values are not specified.
VerboseValue	Controls the display of a progress report showing the number of each chip as it is completed. Choices are true (default) or false.

Output Arguments

PMMatrix_Adj	Matrix of background adjusted PM (perfect match) intensity values.
nsbStruct	Structure containing nonspecific binding background parameters, estimated from the intensities and affinities of probes on an Affymetrix GeneChip array. *nsbStruct* includes the following fields: • sigma • mu_pm • mu_mm

Description

PMMatrix_Adj = gcrmabackadj(*PMMatrix*, *MMMatrix*, *AffinPM*, *AffinMM*) performs GCRMA background adjustment (including optical background correction and nonspecific binding correction) on Affymetrix microarray probe-level data, using probe sequence information and returns *PMMatrix_Adj*, a matrix of background adjusted PM (perfect match) intensity values.

Note: If *AffinPM* and *AffinMM* data are not available, you can still use the gcrmabackadj function by entering empty column vectors for both of these inputs in the syntax.

[*PMMatrix_Adj*, *nsbStruct*] = gcrmabackadj(*PMMatrix*, *MMMatrix*, *AffinPM*, *AffinMM*) returns *nsbStruct*, a structure containing nonspecific binding background parameters, estimated from the intensities and affinities of probes on an Affymetrix GeneChip array. *nsbStruct* includes the following fields:

- sigma

- mu_pm

- mu_mm

... = gcrmabackadj(...'*PropertyName*', *PropertyValue*, ...) calls gcrmabackadj with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = gcrmabackadj(...'OpticalCorr', *OpticalCorrValue*, ...) controls the use of optical background correction on the PM and MM probe intensity values in *PMMatrix* and *MMMatrix*. Choices are true (default) or false.

... = gcrmabackadj(...'CorrConst', *CorrConstValue*, ...) specifies the correlation constant, rho, for log background intensity for each PM/MM probe pair. Choices are any value ≥ 0 and ≤ 1. Default is 0.7.

... = gcrmabackadj(...'Method', *MethodValue*, ...) specifies the method to estimate the signal. Choices are MLE, a faster, ad hoc Maximum Likelihood Estimate method, or EB, a slower, more formal, empirical Bayes method. Default is MLE.

... = gcrmabackadj(...'TuningParam', *TuningParamValue*, ...) specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

Tip For information on determining a setting for this parameter, see Wu et al., 2004.

... = gcrmabackadj(...'AddVariance', *AddVarianceValue*, ...) controls whether the signal variance is added to the weight function for smoothing low signal edge. Choices are `true` or `false` (default).

... = gcrmabackadj(...'GSBCorr', *GSBCorrValue*, ...) specifies whether to perform gene specific binding (GSB) correction using probe affinity data. Choices are `true` (default) or `false`. If there is no probe affinity information, this property is ignored.

... = gcrmabackadj(...'Showplot', *ShowplotValue*, ...) controls the display of a plot showing the $\log_2$ of probe intensity values from a specified column (chip) in *MMMatrix*, versus probe affinities in *AffinMM*. Choices are `true`, `false`, or *I*, an integer specifying a column in *MMMatrix*. If set to `true`, the first column in *MMMatrix* is plotted. Default is:

- `false` — When return values are specified.
- `true` — When return values are not specified.

... = gcrmabackadj(...'Verbose', *VerboseValue*, ...) controls the display of a progress report showing the number of each chip as it is completed. Choices are `true` (default) or `false`.

Examples

1. Load the MAT-file, included with the Bioinformatics Toolbox software, that contains Affymetrix data from a prostate cancer study. The variables in the MAT-file include `seqMatrix`, a matrix containing sequence information for PM probes, `pmMatrix` and `mmMatrix`, matrices containing PM and MM probe intensity values, and `probeIndices`, a column vector containing probe indexing information.

   ```
   load prostatecancerrawdata
   ```

2. Compute the Affymetrix PM and MM probe affinities from their sequences and MM probe intensities.

   ```
   [apm, amm] = affyprobeaffinities(seqMatrix, mmMatrix(:,1),...
                   'ProbeIndices', probeIndices);
   ```

3. Perform GCRMA background adjustment on the Affymetrix microarray probe-level data, creating a matrix of background adjusted PM intensity values. Also, display a plot showing the $\log_2$ of probe intensity values from column 3 (chip 3) in `mmMatrix`, versus probe affinities in `amm`.

```
pms_adj = gcrmabackadj(pmMatrix, mmMatrix, apm, amm, 'showplot', 3);
```
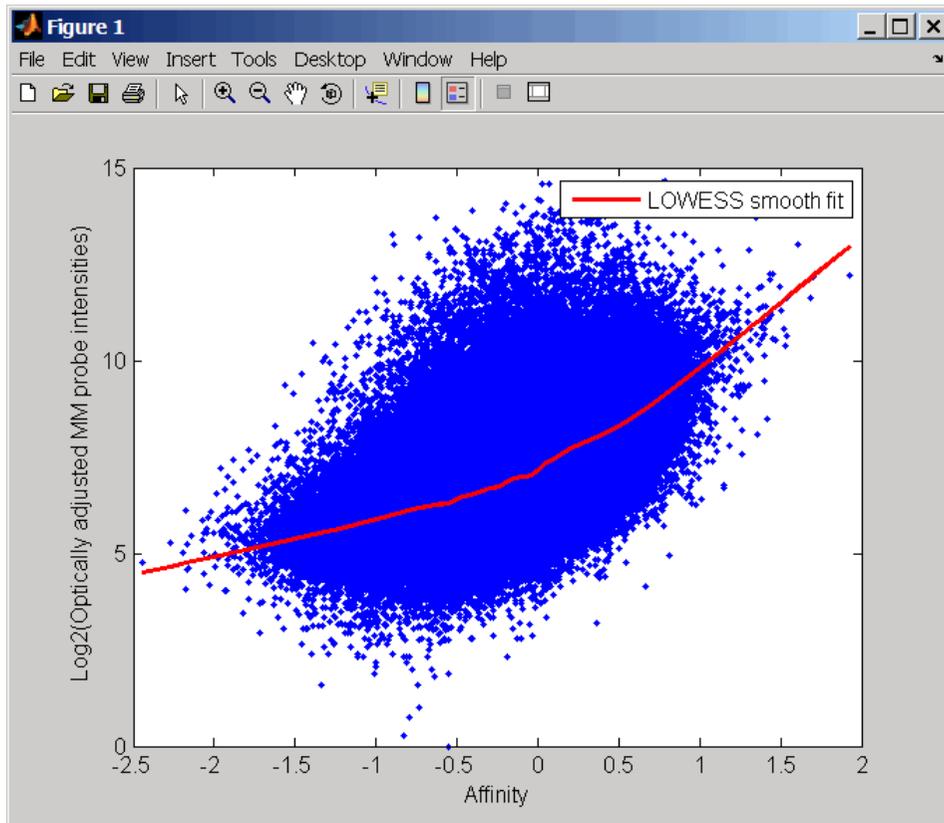

4 Perform GCRMA background adjustment again, using the slower, more formal, empirical Bayes method.

```
pms_adj2 = gcrmabackadj(pmMatrix, mmMatrix, apm, amm, 'method', 'EB');
```

The prostatecancerrawdata.mat file used in this example contains data from Best et al., 2005.

References

[1] Wu, Z., Irizarry, R.A., Gentleman, R., Murillo, F.M., and Spencer, F. (2004). A Model Based Background Adjustment for Oligonucleotide Expression Arrays. Journal of the American Statistical Association *99(468)*, 909–917.

[2] Wu, Z., and Irizarry, R.A. (2005). Stochastic Models Inspired by Hybridization Theory for Short Oligonucleotide Arrays. Proceedings of RECOMB 2004. J Comput Biol. *12(6)*, 882–93.

[3] Wu, Z., and Irizarry, R.A. (2005). A Statistical Framework for the Analysis of Microarray Probe-Level Data. Johns Hopkins University, Biostatistics Working Papers 73.

[4] Wu, Z., and Irizarry, R.A. (2003). A Model Based Background Adjustment for Oligonucleotide Expression Arrays. RSS Workshop on Gene Expression, Wye, England, http://biosun01.biostat.jhsph.edu/%7Eririzarr/Talks/gctalk.pdf.

[5] Abd Rabbo, N.A., and Barakat, H.M. (1979). Estimation Problems in Bivariate Lognormal Distribution. Indian J. Pure Appl. Math *10(7)*, 815–825.

[6] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

See Also
affygcrma | affyprobeseqread | affyread | celintensityread | probelibraryinfo

ge (DataMatrix)

Test DataMatrix objects for greater than or equal to

Syntax

T = ge(*DMObj1*, *DMObj2*)
T = *DMObj1* >= *DMObj2*
T = ge(*DMObj1*, *B*)
T = *DMObj1* >= *B*
T = ge(*B*, *DMObj1*)
T = *B* >= *DMObj1*

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

T	Logical matrix of the same size as *DMObj1* and *DMObj2* or *DMObj1* and *B*. It contains logical 1 (true) where elements in the first input are greater than or equal to the corresponding element in the second input, and logical 0 (false) otherwise.

Description

T = ge(*DMObj1*, *DMObj2*) or the equivalent *T* = *DMObj1* >= *DMObj2* compares each element in DataMatrix object *DMObj1* to the corresponding element in DataMatrix object *DMObj2*, and returns *T*, a logical matrix of the same size as *DMObj1* and *DMObj2*, containing logical 1 (true) where elements in *DMObj1* are greater than or equal to the corresponding element in *DMObj2*, and logical 0 (false) otherwise. *DMObj1* and *DMObj2*

must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). *DMObj1* and *DMObj2* can have different Name properties.

T = ge(*DMObj1, B*) or the equivalent *T* = *DMObj1* >= *B* compares each element in DataMatrix object *DMObj1* to the corresponding element in *B*, a numeric or logical array, and returns *T*, a logical matrix of the same size as *DMObj1* and *B*, containing logical 1 (true) where elements in *DMObj1* are greater than or equal to the corresponding element in *B*, and logical 0 (false) otherwise. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

T = ge(*B, DMObj1*) or the equivalent *T* = *B* >= *DMObj1* compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are greater than or equal to the corresponding element in *DMObj1*, and logical 0 (false) otherwise. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls *T* = ge(*X, Y*) for the syntax *T* = *X* >= *Y* when *X* or *Y* is a DataMatrix object.

More About

- DataMatrix object

See Also
DataMatrix | le

genbankread

Read data from GenBank file

Syntax

GenBankData = genbankread(*File*)

Arguments

File	Either of the following: • String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a GenBank-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. • MATLAB character array that contains the text of a GenBank-formatted file. **Tip** You can use the `getgenbank` function with the `'ToFile'` property to retrieve sequence information from the GenBank database and create an GenBank-formatted file.
GenBankData	MATLAB structure or array of structures containing fields corresponding to GenBank keywords.

Description

GenBankData = genbankread(*File*) reads a GenBank-formatted file, *File*, and creates *GenBankData*, a structure or array of structures, containing fields corresponding to the GenBank keywords. When *File* contains multiple entries, each entry is stored as a separate element in *GenBankData*. For a list of the GenBank keywords, see http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html.

Examples

1 Retrieve sequence information for the HEXA gene, store the data in a file, and then read into the MATLAB software.

```
getgenbank('nm_000520', 'ToFile', 'TaySachs_Gene.txt')
s = genbankread('TaySachs_Gene.txt')

s =

                     LocusName: 'NM_000520'
           LocusSequenceLength: '2437'
         LocusNumberofStrands: ''
                 LocusTopology: 'linear'
            LocusMoleculeType: 'mRNA'
         LocusGenBankDivision: 'PRI'
        LocusModificationDate: '18-FEB-2009'
                    Definition: [1x63 char]
                     Accession: 'NM_000520'
                       Version: 'NM_000520.4'
                            GI: '189181665'
                       Project: []
                        DBLink: []
                      Keywords: []
                       Segment: []
                        Source: 'Homo sapiens (human)'
                 SourceOrganism: [4x65 char]
                     Reference: {1x10 cell}
                       Comment: [32x67 char]
                      Features: [147x74 char]
                           CDS: [1x1 struct]
                      Sequence: [1x2437 char]
```

2 Display the source organism for this sequence.

```
s.SourceOrganism

ans =

Homo sapiens
Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
Mammalia; Eutheria; Euarchontoglires; Primates; Haplorrhini;
Catarrhini; Hominidae; Homo.
```

See Also
emblread | fastaread | genpeptread | getgenbank | scfread | seqviewer

geneentropyfilter

Remove genes with low entropy expression values

Syntax

```
Mask = geneentropyfilter(Data)
[Mask, FData] = geneentropyfilter(Data)
[Mask, FData, FNames] = geneentropyfilter(Data, Names)
geneentropyfilter(..., 'Percentile', PercentileValue)
```

Arguments

Data	DataMatrix object or numeric matrix where each row corresponds to the experimental results for one gene. Each column is the results for all genes from one experiment.
Names	Cell array with the name of a gene for each row of experimental data. *Names* has same number of rows as *Data* with each row containing the name or ID of the gene in the data set.
PercentileValue	Property to specify a percentile below which gene data is removed. Enter a value from 0 to 100.

Description

Mask = geneentropyfilter(*Data*) identifies gene expression profiles in *Data* with entropy values less than the 10th percentile.

Mask is a logical vector with one element for each row in *Data*. The elements of *Mask* corresponding to rows with a variance greater than the threshold have a value of 1, and those with a variance less than the threshold are 0.

[*Mask*, *FData*] = geneentropyfilter(*Data*) returns *FData*, a filtered data matrix. You can also create *FData* using *FData* = Data(*Mask*,:).

[*Mask*, *FData*, *FNames*] = geneentropyfilter(*Data*, *Names*) returns *FNames*, a filtered names array, where *Names* is a cell array of the names of the genes

corresponding to each row of *Data*. You can also create *FNames* using *FNames* = Names(*Mask*).

Note: If *Data* is a DataMatrix object with specified row names, you do not need to provide the second input *Names* to return the third output *FNames*.

---

geneentropyfilter(..., 'Percentile', *PercentileValue*) removes from *Data*, the experimental data, gene expression profiles with entropy values less than *PercentileValue*, the specified percentile.

Examples

1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: yeastvalues, a matrix of gene expression data, genes, a cell array of GenBank accession numbers for labeling the rows in yeastvalues, and times, a vector of time values for labeling the columns in yeastvalues

```
load yeastdata
```

2 Remove genes with low entropy expression values.

```
[fyeastvalues, fgenes] = geneentropyfilter(yeastvalues,genes);
```

References

[1] Kohane I.S., Kho A.T., Butte A.J. (2003), Microarrays for an Integrative Genomics, Cambridge, MA:MIT Press.

See Also

exprprofrange | exprprofvar | genelowvalfilter | generangefilter | genevarfilter

genelowvalfilter

Remove gene profiles with low absolute values

Syntax

```
Mask = genelowvalfilter(Data)
[Mask,FData] = genelowvalfilter(Data)
[Mask,FData,FNames] = genelowvalfilter(Data,geneNames)

[ ___ ] = genelowvalfilter( ___ ,Name,Value)
```

Description

`Mask = genelowvalfilter(Data)` returns a logical vector Mask identifying gene expression profiles in Data that have absolute expression levels in the lowest 10% of the data set.

Gene expression profile experiments have data where the absolute values are very low. The quality of this type of data is often bad due to large quantization errors or simply poor spot hybridization. Use this function to filter data.

`[Mask,FData] = genelowvalfilter(Data)` also returns FData, a data matrix containing filtered expression profiles.

`[Mask,FData,FNames] = genelowvalfilter(Data,geneNames)` also returns FNames, a cell array of filtered gene names or IDs. You have to specify geneNames to return FNames unless Data is a `DataMatrix` object with specified row names.

`[ ___ ] = genelowvalfilter( ___ ,Name,Value)` returns any of the previous output arguments using any input arguments from the previous syntaxes and additional options, specified as one or more optional name-value pair arguments.

Examples

Filter Out Genes with Low Absolute Expression Levels

Load the sample yeast data.

```
load yeastdata;
```

Retrieve the genes and corresponding expression data where absolute expression levels exceed the 10th percentile.

```
[mask,filteredData,filteredGenes] = genelowvalfilter(yeastvalues,genes);
```

Compare the number of filtered genes (`filteredGenes`) with the number of genes in the original data set (`genes`).

```
size (filteredGenes,1)

ans =

       6394

size (genes,1)

ans =

       6400
```

Filter Out Genes with Low Absolute Expression Levels Using a Logical Vector

Load the sample yeast data.

```
load yeastdata;
```

Mark the genes that have low absolute expression levels below the 10th percentile of the data set.

```
mask = genelowvalfilter(yeastvalues);
```

The variable `genes` contains every gene names in the yeast data set. Use the generated logical vector `mask` to retrieve the genes where expression levels exceed the 10th percentile.

```
filteredGenes = genes(mask);
```

Extract corresponding expression profile data for the selected genes from the variable `yeastvalues`, which contains expression profiles of every gene in the yeast data set.

```
filteredData = yeastvalues(mask,:);
```

Filter Out Genes with Absolute Expression Levels that are Lower Than a User-Defined Threshold

Load the sample yeast data.

```
load yeastdata;
```

Retrieve the genes and corresponding expression data where absolute expression levels exceed the 30th percentile of the data set.

```
[mask,filteredData,filteredGenes] = genelowvalfilter(yeastvalues,genes,'Percentile',30)
```

Compare the number of filtered genes (`filteredGenes`) with the number of genes in the original data set (`genes`).

```
size (filteredGenes,1)

ans =

       6384

size (genes,1)

ans =

       6400
```

Input Arguments

`Data` — Input data
DataMatrix object | numeric matrix

Input data, specified as a `DataMatrix` object or numeric matrix. Each row of the matrix corresponds to the experimental results for one gene. Each column represents the results for all genes from one experiment.

`geneNames` — Gene names or IDs
cell array of strings

Gene names or IDs, specified as a cell array of strings. The array has the same number of rows as Data. Each row contains the name or ID of the gene in the data set.

Note: If Data is a `DataMatrix` object with specified row names, you do not need to provide the second input geneNames to return the third output FNames.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'AbsValue',10.5` specifies `genelowvalfilter` to remove expression profiles with absolute values less than 10.5.

`'Percentile'` — Percentile value
10 (default) | scalar value in the range (0,100)

Percentile value, specified as a scalar value in the range (0 to 100). The function `genelowvalfilter` removes gene expression profiles with absolute values less than the percentile value, which is specified using `'Percentile'`.

Example: `'Percentile',50`

`'AbsValue'` — Absolute expression profile value
real number

Absolute expression profile value, specified as a real number. The function `genelowvalfilter` removes gene expression profiles with absolute values less than the absolute value, which is specified using `'AbsValue'`.

Example: `'AbsValue',10.5`

`'AnyVal'` — Logical indicator to select minimum or maximum absolute value
false (default) | true

Logical indicator to select the minimum or maximum absolute value, specified as `true` or `false`. Set the value to `true` to select the minimum absolute value. Set it to `false` to select the maximum absolute value.

Example: `'AnyVal',true`

Output Arguments

`Mask` — Logical vector
vector of 0s and 1s

Logical vector, returned as a vector of 0s and 1s for each row in Data. The elements of Mask with value 1 correspond to rows with absolute expression levels exceeding the threshold, and those with value 0 correspond to rows with absolute expression levels less than or equal to the threshold.

`FData` — Filtered data matrix
data matrix

Filtered data matrix, returned as a data matrix that contains gene expression profiles with absolute expression levels exceeding the threshold value. You can also create FData using `FData = Data(Mask,:)`.

`FNames` — Array of filtered gene names
cell array of strings

Array of filtered gene names, returned as a cell array of strings. It contains gene names or IDs corresponding to each row of `Data` that contains gene expression profiles with absolute expression levels exceeding the threshold value. You can also create FNames using `FNames = geneNames(Mask)`.

References

[1] Kohane, I.S., Kho, A.T., Butte, A.J. (2003). Microarrays for an Integrative Genomics, First Edition (Cambridge, MA: MIT Press).

See Also

`exprprofrange` | `exprprofvar` | `geneentropyfilter` | `generangefilter` | `genevarfilter`

geneont class

Data structure containing Gene Ontology (GO) information

Description

A geneont object is a data structure containing Gene Ontology information. You can explore and traverse Gene Ontology terms using "is_a" and "part_of" relationships.

Construction

.geneont

Create geneont object and term objects

Methods

getancestors

Find terms that are ancestors of specified Gene Ontology (GO) term

getdescendants

Find terms that are descendants of specified Gene Ontology (GO) term

getmatrix

Convert geneont object into relationship matrix

getrelatives

Find terms that are relatives of specified Gene Ontology (GO) term

Properties

date

Read-only string containing date and time OBO file was last updated

default_namespace	Read-only string containing namespace to which GO terms are assigned
format_version	Read-only string containing version of encoding of OBO file
terms	Read-only column vector with handles to term objects of geneont object

Instance Hierarchy

A geneont object contains term objects.

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Indexing

You can use parenthesis () indexing with either GO identifiers (numbers) or by GO terms (term objects) to create a subontology. See "Examples" on page 1-654 below.

Examples

Indexing into a geneont Object Using the GO Identifier

You can create a subontology by indexing into a geneont object by using the GO identifier.

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Create a subontology by returning the terms with GO identifiers of `GO:000005` through `GO:000010`.

```
subontology1 = GeneontObj(5:10)

Gene Ontology object with 6 Terms.
```

3 Create a subontology by returning the term with a GO identifier of `GO:000100`.

```
subontology2 = GeneontObj(100)

Gene Ontology object with 1 Terms.
```

Indexing into a geneont Object Using the GO Term

You can create a subontology by indexing into a geneont object by using the GO term.

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Create an array of term objects containing the fifth through tenth terms of the geneont object.

```
termObject = GeneontObj.terms(5:10)

6x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
```

```
obsolete
```

> **Note:** The GO term of 5 is the position of the term object in the geneont object, and is not necessarily the same as the term object with a GO identifier of GO:000005 used in the first example. This is because there are many terms that are obsolete and are not included as term objects in the geneont object.

3 Create a subontology by returning the fifth through tenth terms of the geneont object.

```
subontology3 = GeneontObj(termObject)

Gene Ontology object with 6 Terms.
```

See Also

goannotread | num2goid | term

geneont

Class: geneont

Create geneont object and term objects

Syntax

```
GeneontObj = geneont
GeneontObj = geneont('File', FileValue)
GeneontObj = geneont('Live', LiveValue)
GeneontObj = geneont('Live', LiveValue, 'ToFile', ToFileValue)
```

Description

GeneontObj = geneont creates *GeneontObj*, a geneont object, from the gene_ontology.obo file in the MATLAB current directory. It also creates multiple term objects, one for each term in the geneont object.

GeneontObj = geneont('File', *FileValue*) creates *GeneontObj*, a geneont object, from *FileValue*, a string specifying the file name of an Open Biomedical Ontology (OBO)-formatted file that is on the MATLAB search path.

GeneontObj = geneont('Live', *LiveValue*) controls the creation of *GeneontObj*, a geneont object, from the current version of the Gene Ontology database, which is the file at:

http://www.geneontology.org/ontology/gene_ontology.obo

Choices are true or false (default).

Note: The full Gene Ontology database may take several minutes to download when you run this function using the 'Live' property.

GeneontObj = geneont('Live', *LiveValue*, 'ToFile', *ToFileValue*), when *LiveValue* is true, creates *GeneontObj*, a geneont object, from the most recent version of the Gene Ontology database, which is the file at:

```
http://www.geneontology.org/ontology/gene_ontology.obo
```

and saves the contents of this file to *ToFileValue*, a string specifying a file name or a path and file name.

Input Arguments

FileValue	String specifying the file name of an OBO-formatted file that is on the MATLAB search path.
LiveValue	Controls the creation of the most up-to-date geneont object. Enter `true` to create *GeneontObj*, a geneont object, from the most recent version of the Gene Ontology database. Default is `false`.
ToFileValue	String specifying a file name or path and file name to which to save the contents of the current version of the Gene Ontology database.

Output Arguments

GeneontObj	MATLAB object containing gene ontology information.

Examples

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Display information about the geneont object.

```
get(GeneontObj)
```

```
default_namespace: 'gene_ontology'
```

```
                    format_version: '1.0'
                            version: '4.509'
                               date: '28:11:2008 19:30'
                           saved_by: 'gocvs'
                  auto_generated_by: 'OBO-Edit 1.101'
                          subsetdef: {7x1 cell}
                             import: ''
                     synonymtypedef: ''
                            idspace: ''
    default_relationship_id_prefix: ''
                         id_mapping: ''
                             remark: [1x31 char]
                             typeref: ''
                   unrecognized_tag: {1x2 cell}
                              Terms: [27769x1 geneont.term]
```

3 Search for all GO terms in the geneont object that contain the string `ribosome` in the field `name`, and use the `geneont.terms` property to create a MATLAB structure array of term objects containing those terms.

```
comparison = regexpi(get(GeneontObj.terms,'name'),'ribosome');
indices = find(~cellfun('isempty',comparison));
terms_with_ribosmome = GeneontObj.terms(indices)

22x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

Note: Although the `terms` property is a column vector with handles to term objects, in the MATLAB Command Window, it displays as a structure array, with one structure for each GO term in the geneont object.

See Also
goannotread | geneont.terms | num2goid | term

generangefilter

Remove gene profiles with small profile ranges

Syntax

```
Mask = generangefilter(Data)
[Mask, FData] = generangefilter(Data)
[Mask, FData, FNames] = generangefilter(Data, Names)
generangefilter(..., 'Percentile', PercentileValue, ...)
generangefilter(..., 'AbsValue', AbsValueValue, ...)
generangefilter(..., 'LogPercentile', LogPercentileValue, ...)
generangefilter(..., 'LogValue', LogValueValue, ...)
```

Arguments

Data	DataMatrix object or numeric matrix where each row corresponds to the experimental results for one gene. Each column is the results for all genes from one experiment.
Names	Cell array with the name of a gene for each row of experimental data. *Names* has same number of rows as *Data* with each row containing the name or ID of the gene in the data set.
PercentileValue	Property to specify a percentile below which gene expression profiles are removed. Enter a value from 0 to 100.
AbsValueValue	Property to specify an absolute value below which gene expression profiles are removed.
LogPercentileValue	Property to specify the logarithm of a percentile.
LogValueValue	Property to specify the logarithm of an absolute value.

Description

Mask = generangefilter(*Data*) calculates the range for each gene expression profile in *Data*, a DataMatrix object or matrix of the experimental data, and then identifies the expression profiles with ranges less than the 10th percentile.

Mask is a logical vector with one element for each row in *Data*. The elements of *Mask* corresponding to rows with a range greater than the threshold have a value of 1, and those with a range less than the threshold are 0.

[*Mask*, *FData*] = generangefilter(*Data*) returns *FData*, a filtered data matrix. You can also create *FData* using *FData* = Data(*Mask*,:).

[*Mask*, *FData*, *FNames*] = generangefilter(*Data*, *Names*) returns *FNames*, a filtered names array, where *Names* is a cell array of the names of the genes corresponding to each row in *Data*. You can also create *FNames* using *FNames* = Names(*Mask*).

Note: If *Data* is a DataMatrix object with specified row names, you do not need to provide the second input *Names* to return the third output *FNames*.

---

generangefilter(..., '*PropertyName*', *PropertyValue*, ...) calls generangefilter with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

generangefilter(..., 'Percentile', *PercentileValue*, ...) removes from the experimental data (*Data*) gene expression profiles with ranges less than a specified percentile (*PercentileValue*).

generangefilter(..., 'AbsValue', *AbsValueValue*, ...) removes from *Data* gene expression profiles with ranges less than *AbsValueValue*.

generangefilter(..., 'LogPercentile', *LogPercentileValue*, ...) filters genes with profile ranges in the lowest percent of the log range (*LogPercentileValue*).

generangefilter(..., 'LogValue', *LogValueValue*, ...) filters genes with profile log ranges lower than *LogValueValue*.

Examples

1   Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: `yeastvalues`, a matrix of gene expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`

    ```
    load yeastdata
    ```

2   Remove gene profiles with small profile ranges.

    ```
    [mask, fyeastvalues, fgenes] = generangefilter(yeastvalues,genes);
    ```

References

[1] Kohane I.S., Kho A.T., Butte A.J. (2003), Microarrays for an Integrative Genomics, Cambridge, MA:MIT Press.

See Also

```
exprprofrange | exprprofvar | geneentropyfilter | genelowvalfilter |
genevarfilter
```

geneticcode

Return nucleotide codon to amino acid mapping for genetic code

Syntax

Map = geneticcode
Map = geneticcode(*GeneticCode*)

Input Arguments

GeneticCode	Integer or string specifying a genetic code number or code name from the table Genetic Code. Default is 1 or 'Standard'.
	Tip If you use a code name, you can truncate the name to the first two letters of the name.

Output Arguments

Map	Structure containing the mapping of nucleotide codons to amino acids for the standard genetic code. The *Map* structure contains a field for each nucleotide codon.

Description

Map = geneticcode returns a structure containing the mapping of nucleotide codons to amino acids for the standard genetic code. The *Map* structure contains a field for each nucleotide codon.

Map = geneticcode(*GeneticCode*) returns a structure containing the mapping of nucleotide codons to amino acids for the specified genetic code. *GeneticCode* is either:

- An integer or string specifying a code number or code name from the table Genetic Code

- The `transl_table` (code) number from the NCBI Web page describing genetic codes:

  http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c

Tip If you use a code name, you can truncate the name to the first two letters of the name.

Genetic Code

Code Number	Code Name
1	`Standard`
2	`Vertebrate Mitochondrial`
3	`Yeast Mitochondrial`
4	`Mold`, `Protozoan`, `Coelenterate Mitochondrial`, and `Mycoplasma/Spiroplasma`
5	`Invertebrate Mitochondrial`
6	`Ciliate`, `Dasycladacean`, and `Hexamita Nuclear`
9	`Echinoderm Mitochondrial`
10	`Euplotid Nuclear`
11	`Bacterial` and `Plant Plastid`
12	`Alternative Yeast Nuclear`
13	`Ascidian Mitochondrial`
14	`Flatworm Mitochondrial`
15	`Blepharisma Nuclear`
16	`Chlorophycean Mitochondrial`
21	`Trematode Mitochondrial`
22	`Scenedesmus Obliquus Mitochondrial`
23	`Thraustochytrium Mitochondrial`

Examples

Return the mapping of nucleotide codons to amino acids for the Flatworm Mitochondrial genetic code.

```
wormmap = geneticcode('Flatworm Mitochondrial');
```

References

[1] NCBI Web page describing genetic codes:

```
http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c
```

See Also
aa2nt | aminolookup | baselookup | codonbias | dnds | dndsml | nt2aa | revgeneticcode | seqshoworfs | seqviewer

genevarfilter

Filter genes with small profile variance

Syntax

```
Mask = genevarfilter(Data)
[Mask, FData] = genevarfilter(Data)
[Mask, FData, FNames] = genevarfilter(Data, Names)
genevarfilter(..., 'Percentile', PercentileValue, ...)
genevarfilter(..., 'AbsValue', AbsValueValue, ...)
```

Arguments

Data	DataMatrix object or numeric matrix where each row corresponds to a gene. If a matrix, the first column is the names of the genes, and each additional column is the results from an experiment.
Names	Cell array with the name of a gene for each row of experimental data. *Names* has same number of rows as *Data* with each row containing the name or ID of the gene in the data set.
PercentileValue	Specifies a percentile below which gene expression profiles are removed. Choices are integers from 0 to 100. Default is 10.
AbsValueValue	Property to specify an absolute value below which gene expression profiles are removed.

Description

Gene profiling experiments typically include genes that exhibit little variation in their profile and are generally not of interest. These genes are commonly removed from the data.

Mask = genevarfilter(*Data*) calculates the variance for each gene expression profile in *Data* and returns *Mask*, which identifies the gene expression profiles with a variance

less than the 10th percentile. *Mask* is a logical vector with one element for each row in *Data*. The elements of *Mask* corresponding to rows with a variance greater than the threshold have a value of 1, and those with a variance less than the threshold are 0.

[*Mask*, *FData*] = genevarfilter(*Data*) calculates the variance for each gene expression profile in *Data* and returns *FData*, a filtered data matrix, in which the low-variation gene expression profiles are removed. You can also create *FData* using *FData* = Data(*Mask*,:).

[*Mask*, *FData*, *FNames*] = genevarfilter(*Data*, *Names*) returns *FNames*, a filtered names array, in which the names associated with low-variation gene expression profiles are removed. *Names* is a cell array of the names of the genes corresponding to each row in *Data*. You can also create *FNames* using *FNames* = Names(*Mask*).

Note: If *Data* is a DataMatrix object with specified row names, you do not need to provide the second input *Names* to return the third output *FNames*.

---

genevarfilter(..., '*PropertyName*', *PropertyValue*, ...) calls genevarfilter with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

genevarfilter(..., 'Percentile', *PercentileValue*, ...) removes from *Data*, the experimental data, the gene expression profiles with a variance less than the percentile specified by *PercentileValue*. Choices are integers from 0 to 100. Default is 10.

genevarfilter(..., 'AbsValue', *AbsValueValue*, ...) removes from *Data* , the experimental data, the gene expression profiles with a variance less than *AbsValueValue*.

Examples

1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: yeastvalues, a matrix of gene expression data, genes, a cell array of GenBank accession numbers for labeling the

rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`

```
load yeastdata
```

2 Filter genes with a small profile variance.

```
[fyeastvalues, fgenes] = genevarfilter(yeastvalues,genes);
```

More About

- DataMatrix object

References

[1] Kohane I.S., Kho A.T., Butte A.J. (2003), Microarrays for an Integrative Genomics, Cambridge, MA:MIT Press.

See Also

```
exprprofrange | exprprofvar | generangefilter | geneentropyfilter |
genelowvalfilter
```

genpeptread

Read data from GenPept file

Syntax

GenPeptData = genpeptread(*File*)

Arguments

File	Either of the following: • String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a GenPept-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. • MATLAB character array that contains the text of a GenPept-formatted file. **Tip** You can use the getgenpept function with the 'ToFile' property to retrieve sequence information from the GenPept database and create an GenPept-formatted file.
GenPeptData	MATLAB structure or array of structures containing fields corresponding to GenPept keywords.

Description

Note: NCBI has changed the name of their protein search engine from GenPept to Entrez Protein. However, the function names in the Bioinformatics Toolbox software (getgenpept and genpeptread) are unchanged representing the still-used GenPept report format.

GenPeptData = genpeptread(*File*) reads a GenPept-formatted file, *File*, and creates *GenPeptData*, a structure or array of structures, containing fields corresponding to the GenPept keywords. When *File* contains multiple entries, each entry is stored as a separate element in *GenPeptData*. For a list of the GenPept keywords, see http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html.

Examples

Retrieve sequence information for the protein coded by the HEXA gene, store the data in a file, and then read into the MATLAB software.

```
getgenpept('p06865', 'ToFile', 'TaySachs_Protein.txt')
genpeptread('TaySachs_Protein.txt')
```

See Also
fastaread | genbankread | getgenpept | pdbread | seqviewer

geoseriesread

Read Gene Expression Omnibus (GEO) Series (GSE) format data

Syntax

GEOData = geoseriesread(*File*)

Input Arguments

File	Either of the following: • String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a Gene Expression Omnibus (GEO) Series (GSE) format file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. • MATLAB character array that contains the text of a GEO Series (GSE) format file. **Tip** You can use the getgeodata function with the 'ToFile' property to retrieve GEO Series (GSE) format data from the GEO database and create a GEO Series (GSE) format file.

Output Arguments

GEOData	MATLAB structure containing the following fields: • Header — Header text from the GEO Series (GSE) format file, typically containing a description of the data or experiment information. • Data — DataMatrix object containing the data from a GEO Series (GSE) format file. The columns and rows of the DataMatrix object correspond to the sample IDs and Ref IDs, respectively, from the GEO Series (GSE) format file.

Description

GEOData = geoseriesread(*File*) reads a Gene Expression Omnibus (GEO) Series (GSE) format file, and then creates a MATLAB structure, *GEOData*, with the following fields.

Fields	Description
Header	Header text from the GEO Series (GSE) format file, typically containing a description of the data or experiment information.
Data	DataMatrix object containing the data from a GEO Series (GSE) format file. The columns and rows of the DataMatrix object correspond to the sample IDs and Ref IDs, respectively, from the GEO Series (GSE) format file.

Examples

1 Retrieve Series (GSE) data from the GEO Web site and save it to a file.

```
geodata = getgeodata('GSE11287','ToFile','GSE11287.txt');
```

2 In a subsequent MATLAB session, you can access the Series (GSE) data from your local file, instead of retrieving it from the GEO Web site.

```
geodata = geoseriesread('GSE11287.txt')

geodata =

    Header: [1x1 struct]
      Data: [45101x6 bioma.data.DataMatrix]
```

3 Access the sample IDs using the colnames property of a DataMatrix object.

```
sampleIDs = geodata.Data.colnames

sampleIDs =

  'GSM284935'  'GSM284936'  'GSM284937'  'GSM284938'  'GSM284939'  'GSM284940'
```

More About

•   DataMatrix object

See Also

affyread | agferead | galread | geosoftread | getgeodata | gprread |
ilmnbsread | sptread

geosoftread

Read Gene Expression Omnibus (GEO) SOFT format data

Syntax

GEOSOFTData = geosoftread(*File*)

Input Arguments

File	Either of the following: • String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a Gene Expression Omnibus (GEO) SOFT format Sample file (GSM), Data Set file (GDS), or Platform (GPL) file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. • MATLAB character array that contains the text of a GEO SOFT format file. **Tip** You can use the getgeodata function with the 'ToFile' property to retrieve GEO SOFT format data from the GEO database and create a GEO SOFT format file.

Output Arguments

GEOSOFTData	MATLAB structure containing information from a GEO SOFT format file.

Description

GEOSOFTData = geosoftread(*File*) reads a Gene Expression Omnibus (GEO) SOFT format Sample file (GSM), Data Set file (GDS), or Platform (GPL) file, and then creates a MATLAB structure, *GEOSOFTData*, with the following fields.

Fields	Description
Scope	Type of file read (SAMPLE, DATASET, or PLATFORM)
Accession	Accession number for record in GEO database.
Header	Microarray experiment information.
ColumnDescriptions	Cell array containing descriptions of columns in the data.
ColumnNames	Cell array containing names of columns in the data.
Data	Array containing microarray data.
Identifier (GDS files only)	Cell array containing probe IDs.
IDRef (GDS files only)	Cell array containing indices to probes.

Note: Currently, the geosoftread function supports Sample (GSM), Data Set (GDS), and Platform (GPL) records.

Examples

Retrieve GSM data from the GEO Web site and save it to a file.

```
geodata = getgeodata('GSM3258','ToFile','GSM3258.txt');
```

Use geosoftread to read a local copy of the GSM file, instead of accessing it from the GEO Web site.

```
geodata = geosoftread('GSM3258.txt')

geodata =

              Scope: 'SAMPLE'
          Accession: 'GSM3258'
             Header: [1x1 struct]
```

```
      ColumnDescriptions: {6x1 cell}
            ColumnNames: {6x1 cell}
                   Data: {5355x6 cell}
```

Read the GDS file for photosynthesis in proteobacteria.

```
gdsdata = geosoftread('GDS329.soft')

gdsdata =

                   Scope: 'DATASET'
               Accession: 'GDS329'
                  Header: [1x1 struct]
      ColumnDescriptions: {6x1 cell}
             ColumnNames: {6x1 cell}
                   IDRef: {5355x1 cell}
              Identifier: {5355x1 cell}
                    Data: [5355x6 double]
```

See Also
galread | getgeodata | geoseriesread | gprread | ilmnbsread | sptread

get (biograph)

Retrieve information about biograph object

Syntax

```
get(BGobj)
BGStruct = get(BGobj)
PropertyValue = get(BGobj, 'PropertyName')
[Property1Value, Property2Value, ...] = get(BGobj, 'Property1Name',
'Property2Name', ...)
```

Input Arguments

BGobj	Biograph object created with the function `biograph`.
PropertyName	Property name for a biograph object.

Output Arguments

BGStruct	Scalar structure, in which each field name is a property of a biograph object, and each field contains the value of that property.
PropertyValue	Value of the property specified by *PropertyName*.

Description

get(*BGobj*) displays all properties and their current values of *BGobj*, a biograph object.

BGStruct = get(*BGobj*) returns all properties of *BGobj*, a biograph object, to *BGStruct*, a scalar structure, in which each field name is a property of a biograph object, and each field contains the value of that property.

PropertyValue = get(*BGobj*, '*PropertyName*') returns the value of the specified property of *BGobj*, a biograph object.

[*Property1Value*, *Property2Value*, ...] = get(*BGobj*, '*Property1Name*', '*Property2Name*', ...) returns the values of the specified properties of *BGobj*, a biograph object.

Properties of a Biograph Object

Property	Description
ID	String to identify the biograph object. Default is ''.
Label	String to label the biograph object. Default is ''.
Description	String that describes the biograph object. Default is ''.
LayoutType	String that specifies the algorithm for the layout engine. Choices are: • 'hierarchical' (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges. • 'radial' — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges. • 'equilibrium' — Calculates layout by minimizing the energy in a dynamic spring system.
EdgeType	String that specifies how edges display. Choices are: • 'straight' • 'curved' (default) • 'segmented' **Note:** Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with LayoutType equal to 'equilibrium' or 'radial' cannot produce curved or segmented edges.
Scale	Positive number that post-scales the node coordinates. Default is 1.

Property	Description
LayoutScale	Positive number that scales the size of the nodes before calling the layout engine. Default is 1.
EdgeTextColor	Three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.
EdgeFontSize	Positive number that sets the size of the edge font in points. Default is 8.
ShowArrows	Controls the display of arrows with the edges. Choices are 'on' (default) or 'off'.
ArrowSize	Positive number that sets the size of the arrows in points. Default is 8.
ShowWeights	Controls the display of text indicating the weight of the edges. Choices are 'on' (default) or 'off'.
ShowTextInNodes	String that specifies the node property used to label nodes when you display a biograph object using the view method. Choices are: • 'Label' — Uses the Label property of the node object (default). • 'ID' — Uses the ID property of the node object. • 'None'
NodeAutoSize	Controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.
NodeCallback	User-defined callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph object in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, @(node) inspect(node), which displays the Property Inspector dialog box.
EdgeCallback	User-defined callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph object in the Biograph Viewer, you can double-

Property	Description
	click an edge to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, `@(edge) inspect(edge)`, which displays the Property Inspector dialog box.
`CustomNodeDrawFcn`	Function handle to a customized function to draw nodes. Default is `[]`.
`Nodes`	Read-only column vector with handles to node objects of a biograph object. The size of the vector is the number of nodes. For properties of node objects, see Properties of a Node Object.
`Edges`	Read-only column vector with handles to edge objects of a biograph object. The size of the vector is the number of edges. For properties of edge objects, see Properties of an Edge Object.

Examples

1 Create a biograph object and assign the node IDs.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
ids = {'M30931','L07625','K03454','M27323','M15390'};
bg = biograph(cm,ids);
```

2 Use the `get` function to display the node IDs.

```
get(bg.nodes,'ID')

ans =

    'M30931'
    'L07625'
    'K03454'
    'M27323'
    'M15390'
```

More About

· biograph object

See Also
biograph | set

get

Class: BioRead

Retrieve property of object

Syntax

```
Struct = get(BioObj)
PropValues = get(BioObj, PropertyName)
```

Description

Struct = get(*BioObj*) returns a MATLAB structure containing a field for each property of an object. Each field contains the current value of that property.

PropValues = get(*BioObj*, *PropertyName*) returns the value(s) of the property or properties specified by *PropertyName*, a string or cell array of strings specifying property names of *BioObj*. *PropValues* is a single property value or a cell array of property values.

Tips

- Use the get method to determine all the object properties and their current values.
- Specific get methods are also available for each property such as BioRead.getHeader, BioRead.getSequence, and BioRead.getQuality. Some of these specific get methods let you access all or a subset of a property.

Input Arguments

BioObj

Object of the BioRead or BioMap class.

Default:

`PropertyName`

Either of the following:

- String specifying the name of a property of the class
- Cell array of strings specifying the names of properties of the class

Default:

Output Arguments

`Struct`

MATLAB structure with a field for each property of an object. Each field contains the current value of that property.

`PropValues`

Single property value or a cell array of property values.

Examples

Retrieve properties from a `BioRead` object:

```
% Create variables containing sequences, quality scores, and headers
seqs = {randseq(10); randseq(15); randseq(20)};
quals = {repmat('!', 1, 10); repmat('%', 1, 15); repmat('&', 1, 20)};
headers = {'H1'; 'H2'; 'H3'};
% Construct a BioRead object from these three variables
BRObj = BioRead(seqs, quals, headers);
% Retrieve the values of the 'Header' property
get(BRObj, 'Header')

ans =

    'H1'
    'H2'
    'H3'
```

```
% Retrieve the values of the 'Sequence' and 'Quality' properties
get(BRObj, {'Sequence', 'Quality'});
```

Transform a `BioRead` object into a MATLAB structure:

```
% Return a MATLAB structure containing a field for each property of
% a BioRead object
BRStruct = get(BRObj)

BRStruct =

    Quality: {3x1 cell}
   Sequence: {3x1 cell}
     Header: {3x1 cell}
      NSeqs: 3
       Name: ''
```

See Also
getHeader | getQuality | BioRead | BioMap | getSequence | set

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive
· SAM format specification

get (clustergram)

Retrieve information about clustergram object

Syntax

```
get(CGobj)
CGStruct = get(CGobj)
PropertyValue = get(CGobj, 'PropertyName')
[Property1Value, Property2Value, ...] = get(CGobj, 'Property1Name',
'Property2Name', ...)
```

Arguments

CGobj	Clustergram object created with the function clustergram.
PropertyName	Property name for a clustergram object.

Description

get(*CGobj*) displays all properties and their current values of *CGobj*, a clustergram object.

CGStruct = get(*CGobj*) returns all properties of *CGobj*, a clustergram object, to *CGStruct*, a scalar structure, in which each field name is a property of a clustergram object, and each field contains the value of that property.

PropertyValue = get(*CGobj*, '*PropertyName*') returns the value of the specified property of *CGobj*, a clustergram object.

[*Property1Value*, *Property2Value*, ...] = get(*CGobj*, '*Property1Name*', '*Property2Name*', ...) returns the values of the specified properties of *CGobj*, a clustergram object.

Properties of a Clustergram Object

Property	Description
RowLabels	Vector of numbers or cell array of text strings to label the rows in the dendrogram and heat map. Default is a vector of values 1 through *M*, where *M* is the number of rows in *Data*, the matrix of data used by the clustergram function to create the clustergram object.
ColumnLabels	Vector of numbers or cell array of text strings to label the columns in the dendrogram and heat map. Default is a vector of values 1 through *N*, where *N* is the number of columns in *Data*, the matrix of data used by the clustergram function to create the clustergram object.
RowGroupNames	A cell array of text strings containing the names of the row groups exported to a clustergram object created using the **Export Group to Workspace** command in the Clustergram window.
RowNodeNames	A cell array of text strings containing the names of the row nodes exported to a clustergram object created using the **Export Group to Workspace** command in the Clustergram window.
ColumnGroupNames	A cell array of text strings containing the names of the column groups exported to a clustergram object created using the **Export Group to Workspace** command in the Clustergram window.
ColumnNodeNames	A cell array of text strings containing the names of the column nodes exported to a clustergram object created using the **Export Group to Workspace** command in the Clustergram window.
ExprValues	An *M*-by-*N* matrix of data, where *M* and *N* are the number of row nodes and column nodes respectively, exported to a clustergram object created using the **Export Group to Workspace** command in the Clustergram window. If the matrix contains gene expression data, typically each row corresponds to a gene and each column corresponds to a sample.
Standardize	Text string that specifies the dimension for standardizing the values in the data. The standardized values are transformed

Property	Description
	so that the mean is 0 and the standard deviation is 1 in the specified dimension. Possibilities are: • `'column'` or `1` — Standardize along the columns of data. • `'row'` or `2` — Standardize along the rows of data. • `'none'` or `3` (default) — Do not standardize.
Cluster	Text string that specifies the dimension for clustering the values in the data. Possibilities are: • `'Row (1)'` — Clustered rows of data only. • `'Column (2)'` — Clustered columns of data only. • `'All (3)'` — Clustered rows of data, then cluster columns of row-clustered data.
RowPdist	String or cell array that specifies the distance metric and optional arguments passed to the `pdist` function (Statistics and Machine Learning Toolbox software) used to calculate the pairwise distances between rows. For information on possibilities, see the `pdist` function.
ColumnPdist	String or cell array that specifies the distance metric and optional arguments passed to the `pdist` function (Statistics and Machine Learning Toolbox software) used to calculate the pairwise distances between columns. For information on possibilities, see the `pdist` function.
Linkage	String or two-element cell array of strings that specifies the linkage method passed to the `linkage` function (Statistics and Machine Learning Toolbox software) used to create the hierarchical cluster tree for rows and columns. If a two-element cell array of strings, the first element is for linkage between rows, and the second element is for linkage between columns. For information on possibilities, see the `linkage` function.
Dendrogram	Scalar or two-element numeric vector or cell array that specifies the `'colorthreshold'` property passed to the `dendrogram` function (Statistics and Machine Learning Toolbox software) used to create the dendrogram plot. If a

Property	Description
	two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns. For more information, see the dendrogram function.
OptimalLeafOrder	Property that enabled or disabled the optimal leaf ordering calculation, which determines the leaf order that maximizes the similarity between neighboring leaves. Possibilities are 1 (enabled) or 0 (disabled).
LogTrans	Controlled the $\log_2$ transform of the data from natural scale. Possibilities are 1 (true) or 0 (false).
Colormap	Either of the following: • *M*-by-3 matrix of RGB values • Name or function handle of a function that returns a colormap, such as redgreencmap or redbluecmap
DisplayRange	Positive scalar that specifies the display range of standardized values. For example, if you specify redgreencmap for the 'ColorMap' property, pure red represents values ≥ DisplayRange, and pure green represents values ≤ −DisplayRange.
Symmetric	Property to force the color scale of the heat map to be symmetric around zero. Possibilities are 1 (true) or 0 (false).
Ratio	Either of the following: • Scalar • Two-element vector It specifies the ratio of space that the row and column dendrograms occupy relative to the heat map. If Ratio is a scalar, it is the ratio for both dendrograms. If Ratio is a two-element vector, the first element is for the ratio of the row dendrogram width to the heat map width, and the second element is for the ratio of the column dendrogram height to the heat map height. The second element is ignored for one-dimensional clustergrams.

Property	Description
Impute	Any of the following: • Name of a function that imputes missing data. • Handle to a function that imputes missing data. • Cell array where the first element is the name of or handle to a function that imputes missing data and the remaining elements are property name/property value pairs used as inputs to the function.
RowMarkers	Optional structure array for annotating the groups (clusters) of rows determined by the `clustergram` function. Each structure in the array represents a group of rows and contains the following fields: • `GroupNumber` — Number to annotate the row group. • `Annotation` — String specifying text to annotate the row group. • `Color` — String or three-element vector of RGB values specifying a color, which is used to label the row group. For more information on specifying colors, see `colorspec`. If this field is empty, default is `'blue'`.
ColumnMarkers	Optional structure array for annotating groups (clusters) of columns determined by the `clustergram` function. Each structure in the array represents a group of rows and contains the following fields: • `GroupNumber` — Number to annotate the column group. • `Annotation` — String specifying text to annotate the column group. • `Color` — String or three-element vector of RGB values specifying a color, which is used to label the column group. For more information on specifying colors, see `colorspec`. If this field is empty, default is `'blue'`.

Examples

1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains `yeastvalues`, a matrix of gene expression data.

```
load filteredyeastdata
```

2 Create a clustergram object and display the dendrograms and heat map from the gene expression data in the first 30 rows of the `yeastvalues` matrix and standardize along the rows of data.

```
cgo = clustergram(yeastvalues(1:30,:),'Standardize', 'row')
```

```
Clustergram object with 30 rows of nodes and 7 columns of nodes.
```


3 Use the `get` method to display the properties of the clustergram object, `cgo`.

```
get(cgo)
```

```
                 Cluster: 'ALL'
                 RowPDist: {'Euclidean'}
              ColumnPDist: {'Euclidean'}
                  Linkage: {'Average'}
               Dendrogram: {}
          OptimalLeafOrder: 1
                 LogTrans: 0
             DisplayRatio: [0.2000 0.2000]
           RowGroupMarker: []
        ColumnGroupMarker: []
           ShowDendrogram: 'on'
              Standardize: 'ROW'
                Symmetric: 1
             DisplayRange: 3
                 Colormap: [11x3 double]
                ImputeFun: []
             ColumnLabels: {'2'  '1'  '3'  '4'  '5'  '7'  '6'}
                RowLabels: {30x1 cell}
        ColumnLabelsRotate: 90
           RowLabelsRotate: 0
      ColumnLabelsLocation: 'bottom'
         RowLabelsLocation: 'right'
                 Annotate: 'off'
            AnnotPrecision: 2
               AnnotColor: 'w'
         ColumnLabelsColor: []
            RowLabelsColor: []
         LabelsWithMarkers: 0
```

4 Export a clustergram object of a group (Group 19) of rows to the MATLAB Workspace by right-clicking a node in the row dendrogram, and then selecting **Export Group to Workspace**.
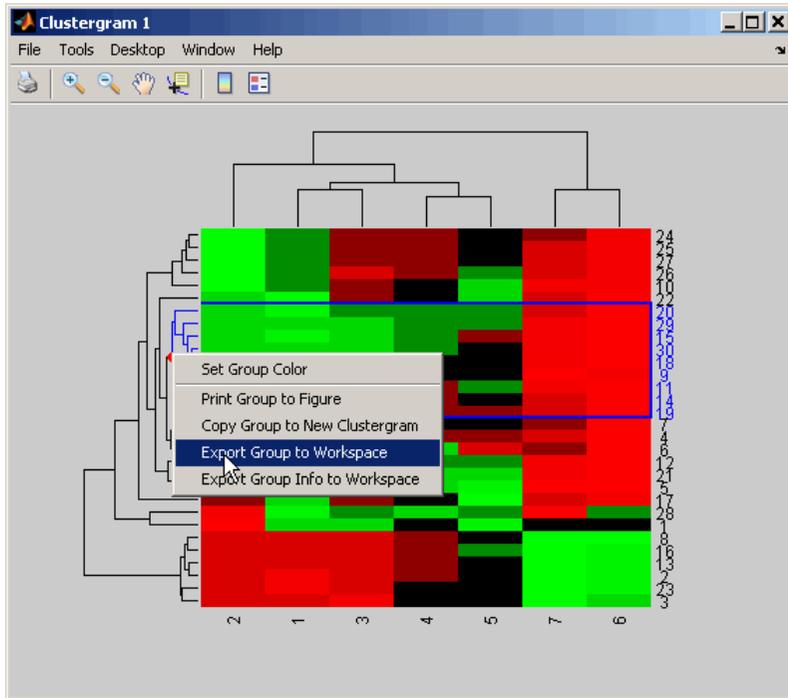
5 In the Export to Workspace dialog box, type **cgo2** for the Workspace variable name for the clustergram object, and then click **OK**.
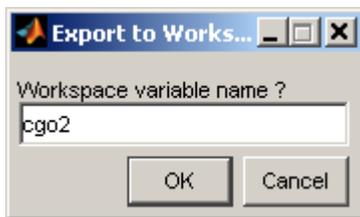
6 Use the `get` method to display the properties of `cgo2`, the clustergram object of the exported group.

```
get(cgo2)
```

```
         Cluster: 'ALL'
        RowPDist: {'Euclidean'}
```

```
         ColumnPDist: {'Euclidean'}
              Linkage: {'Average'}
           Dendrogram: {}
      OptimalLeafOrder: 1
             LogTrans: 0
          DisplayRatio: [0.2000 0.2000]
        RowGroupMarker: []
     ColumnGroupMarker: []
        ShowDendrogram: 'on'
           Standardize: 'ROW'
             Symmetric: 1
          DisplayRange: 3
              Colormap: [11x3 double]
             ImputeFun: []
          ColumnLabels: {'2'  '1'  '3'  '4'  '5'  '7'  '6'}
             RowLabels: {9x1 cell}
     ColumnLabelsRotate: 90
        RowLabelsRotate: 0
   ColumnLabelsLocation: 'bottom'
      RowLabelsLocation: 'right'
               Annotate: 'off'
         AnnotPrecision: 2
            AnnotColor: 'w'
      ColumnLabelsColor: []
         RowLabelsColor: []
      LabelsWithMarkers: 0
```

More About

·   clustergram object

See Also

clustergram | set | view | plot

get (DataMatrix)

Retrieve information about DataMatrix object

Syntax

```
get(DMObj)
DMStruct = get(DMObj)
PropertyValue = get(DMObj, 'PropertyName')
[Property1Value, Property2Value, ...] = get(DMObj, 'Property1Name',
'Property2Name', ...)
```

Input Arguments

DMObj	DataMatrix object, such as created by `DataMatrix` (object constructor).
PropertyName	Property name of a DataMatrix object.

Output Arguments

DMStruct	Scalar structure, in which each field name is a property of a DataMatrix object, and each field contains the value of that property.
PropertyValue	Value of the property specified by *PropertyName*.

Description

get(*DMObj*) displays all properties and their current values of *DMObj*, a DataMatrix object.

DMStruct = get(*DMObj*) returns all properties of *DMObj*, a DataMatrix object, to *DMStruct*, a scalar structure, in which each field name is a property of a DataMatrix object, and each field contains the value of that property.

PropertyValue = get(*DMObj*, '*PropertyName*') returns the value of the specified property of *DMObj*, a DataMatrix object.

[*Property1Value*, *Property2Value*, ...] = get(*DMObj*, '*Property1Name*', '*Property2Name*', ...) returns the values of the specified properties of *DMObj*, a DataMatrix object.

Properties of a DataMatrix Object

Property	Description
Name	String that describes the DataMatrix object. Default is ''.
RowNames	Empty array or cell array of strings that specifies the names for the rows, typically gene names or probe identifiers. The number of elements in the cell array must equal the number of rows in the matrix. Default is an empty array.
ColNames	Empty array or cell array of strings that specifies the names for the columns, typically sample identifiers. The number of elements in the cell array must equal the number of columns in the matrix.
NRows	Positive number that specifies the number of rows in the matrix.
NCols	Positive number that specifies the number of columns in the matrix.
NDims	Positive number that specifies the number of dimensions in the matrix.
ElementClass	String that specifies the class type, such as `single` or `double`.

Examples

1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: `yeastvalues`, a matrix of gene expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`.

```
load filteredyeastdata
```

2 Import the microarray object package so that the `DataMatrix` constructor function will be available.

```
import bioma.data.*
```

3 Create a DataMatrix object from the gene expression data in the first 30 rows of the `yeastvalues` matrix. Use the `genes` column vector and `times` row vector to specify the row names and column names.

```
dmo = DataMatrix(yeastvalues(1:30,:),genes(1:30,:),times);
```

4 Use the `get` method to display the properties of the DataMatrix object, `dmo`.

```
get(dmo)

        Name: ''
     RowNames: {30x1 cell}
     ColNames: {'   0'  ' 9.5'  '11.5'  '13.5'  '15.5'  '18.5'  '20.5'}
        NRows: 30
        NCols: 7
        NDims: 2
  ElementClass: 'double'
```

More About

• DataMatrix object

See Also

DataMatrix | set

get (phytree)

Retrieve information about phylogenetic tree object

Syntax

```
[Value1, Value2,...] = get(Tree, 'Property1','Property2',...)
get(Tree)
V = get(Tree)
```

Arguments

Tree	Phytree object created with the function `phytree`.
Name	Property name for a `phytree` object.

Description

[*Value1, Value2,...*] = get(*Tree, 'Property1','Property2',...*) returns the specified properties from a phytree object (`Tree`).

Properties for a `phytree` object are listed in the following table.

Property	Description
NumLeaves	Number of leaves
NumBranches	Number of branches
NumNodes	Number of nodes (NumLeaves + NumBranches)
Pointers	Branch to leaf/branch connectivity list
Distances	Edge length for every leaf/branch
LeafNames	Names of the leaves
BranchNames	Names of the branches
NodeNames	Names of all the nodes

get(*Tree*) displays all property names and their current values for a phytree object (*Tree*).

V = get(*Tree*) returns a structure where each field name is the name of a property of a phytree object (*Tree*) and each field contains the value of that property.

Examples

1   Read in a phylogenetic tree from a file.

```
tr = phytreeread('pf00002.tree')

Phylogenetic tree object with 33 leaves (32 branches)
```

2   Get the names of the leaves.

```
protein_names = get(tr,'LeafNames')

protein_names =

    'Q9YHC6_RANRI/126-382'
    'VIPR1_RAT/140-397'
    'VIPR_CARAU/100-359'
     ...
```

More About

·   phytree object

See Also

phytree | select | phytreeread | getbyname

getancestors (biograph)

Find ancestors in biograph object

Syntax

Nodes = getancestors(*BiographNode*)
Nodes = getancestors(*BiographNode, NumGenerations*)

Arguments

BiographNode	Node in a biograph object.
NumGenerations	Number of generations. Enter a positive integer.

Description

Nodes = getancestors(*BiographNode*) returns a node (BiographNode) and all of its direct ancestors.

Nodes = getancestors(*BiographNode, NumGenerations*) finds the node (*BiographNode*) and its direct ancestors up to a specified number of generations (*NumGenerations*).

Examples

1 Create a biograph object.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg = biograph(cm)
```

2 Find one generation of ancestors for node 2.

```
ancNodes = getancestors(bg.nodes(2));
set(ancNodes,'Color',[1 .7 .7]);
bg.view;
```
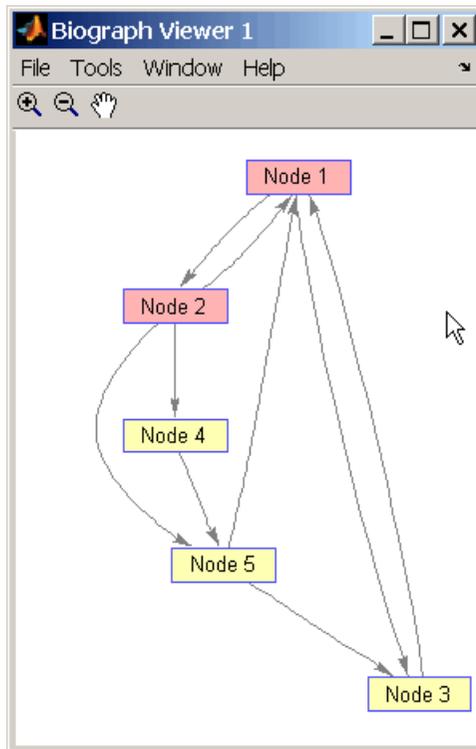
3 Find two generations of ancestors for node 2.

```
ancNodes = getancestors(bg.nodes(2),2);
set(ancNodes,'Color',[.7 1 .7]);
bg.view;
```
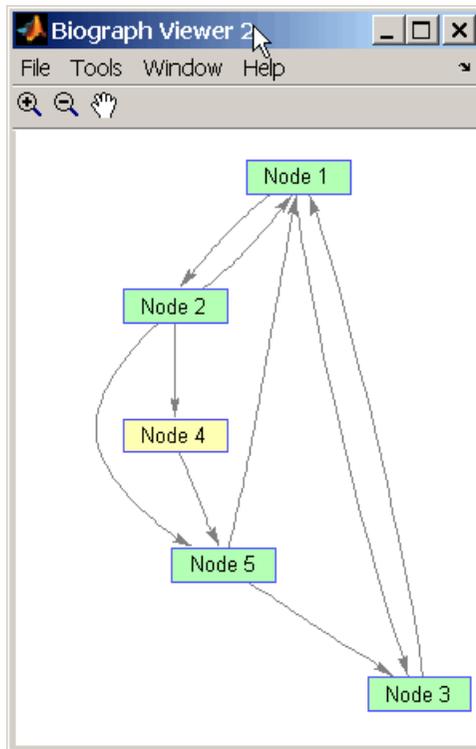
More About

·   biograph object

See Also

```
biograph | get | getdescendants | getnodesbyid | set | view | dolayout |
getancestors | getedgesbynodeid | getrelatives
```

getancestors

Class: geneont

Find terms that are ancestors of specified Gene Ontology (GO) term

Syntax

```
AncestorIDs = getancestors(GeneontObj, ID)
[AncestorIDs, Counts] = getancestors(GeneontObj, ID)
... = getancestors(..., 'Height', HeightValue, ...)
... = getancestors(..., 'Relationtype', RelationtypeValue, ...)
... = getancestors(..., 'Exclude', ExcludeValue, ...)
```

Description

AncestorIDs = getancestors(*GeneontObj*, *ID*) searches *GeneontObj*, a geneont object, for GO terms that are ancestors of the GO term(s) specified by *ID*, which is a GO term identifier or vector of identifiers. It returns *AncestorIDs*, a vector of GO term identifiers including *ID*. *ID* is a nonnegative integer or a vector containing nonnegative integers.

[*AncestorIDs*, *Counts*] = getancestors(*GeneontObj*, *ID*) also returns the number of times each ancestor is found. *Counts* is a column vector with the same number of elements as terms in *GeneontObj*.

Tip The *Counts* return value is useful when you tally counts in gene enrichment studies. For more information, see Gene Ontology Enrichment in Microarray Data.

... = getancestors(..., '*PropertyName*', *PropertyValue*, ...) calls getancestors with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = getancestors(..., 'Height', *HeightValue*, ...) searches up through a specified number of levels, *HeightValue*, in the gene ontology. *HeightValue* is a positive integer. Default is Inf.

... = getancestors(..., 'Relationtype', *RelationtypeValue*, ...) searches for specified relationship types, *RelationtypeValue*, in the gene ontology. *RelationtypeValue* is a string. Choices are 'is_a', 'part_of', or 'both' (default).

... = getancestors(..., 'Exclude', *ExcludeValue*, ...) controls excluding *ID*, the original queried term(s), from the output *AncestorIDs*, unless the term was reached while searching the gene ontology. Choices are true or false (default).

Input Arguments

GeneontObj	A geneont object, such as created by the geneont constructor function.
ID	GO term identifier or vector of identifiers.
HeightValue	Positive integer specifying the number of levels to search upward in the gene ontology.
RelationtypeValue	String specifying the relationship types to search for in the gene ontology. Choices are: • 'is_a' • 'part_of' • 'both' (default)
ExcludeValue	Controls excluding *ID*, the original queried term(s), from the output *AncestorIDs*, unless the term was reached while searching the gene ontology. Choices are true or false (default).

Output Arguments

AncestorIDs	Vector of GO term identifiers including *ID*.
Counts	Column vector with the same number of elements as terms in *GeneontObj*, indicating the number of times each ancestor is found.

Examples

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GO = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of terms in the database.

```
Gene Ontology object with 24316 Terms.
```

2 Retrieve the ancestors of the Gene Ontology term with an identifier of 46680.

```
ancestors = getancestors(GO,46680)

ancestors =
         8150
         9636
        17085
        42221
        46680
        50896
```

3 Create a subordinate Gene Ontology.

```
subontology = GO(ancestors)

Gene Ontology object with 6 Terms.
```

4 Create and display a report of the subordinate Gene Ontology terms, that includes the GO identifier and name.

```
rpt = get(subontology.terms,{'id','name'})

    [ 8150]    'biological_process'
    [ 9636]    'response to toxin'
    [17085]            [1x23 char]
    [42221]            [1x29 char]
    [46680]    'response to DDT'
    [50896]            [1x20 char]
```

5 View relationships of the subordinate Gene Ontology by using the `getmatrix` method to create a connection matrix to pass to the `biograph` function.

```
cm = getmatrix(subontology);
```

```
BG = biograph(cm, get(subontology.terms, 'name'));
view(BG)
```
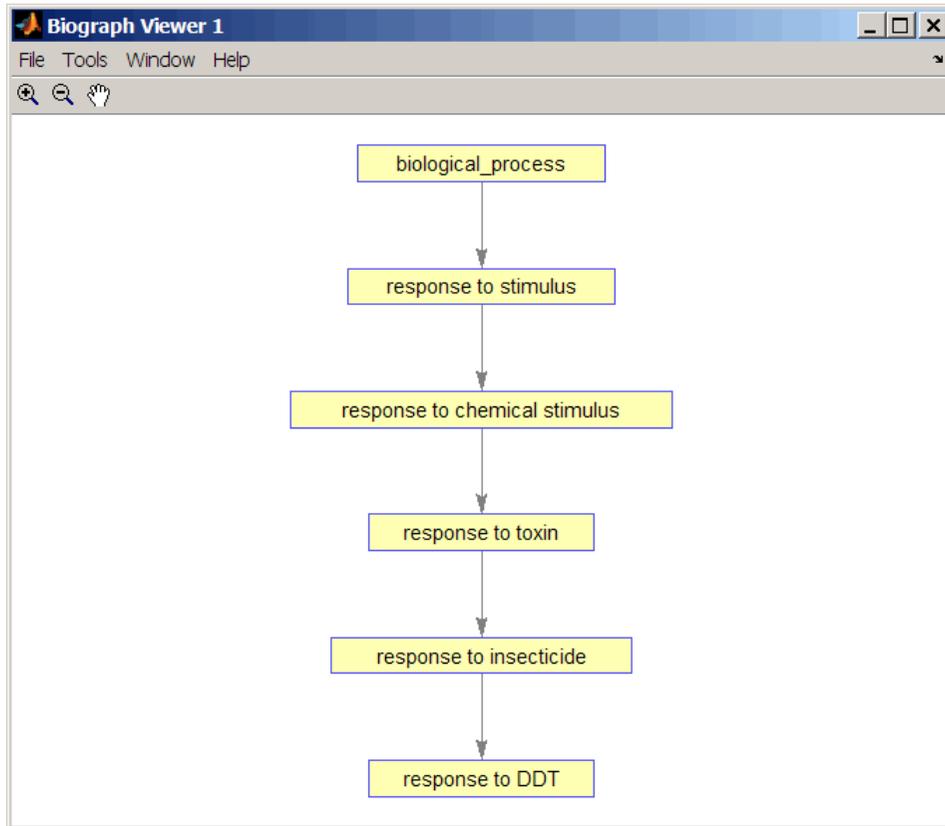

See Also
goannotread | num2goid | term

getblast

Retrieve BLAST report from NCBI Web site

Syntax

```
Data = getblast(RID)
Data = getblast(RID, ...'Descriptions', DescriptionsValue, ...)
Data = getblast(RID, ...'Alignments', AlignmentsValue, ...)
Data = getblast(RID, ...'ToFile', ToFileValue, ...)
Data = getblast(RID, ...'FileFormat', FileFormatValue, ...)
Data = getblast(RID, ...'WaitTime', WaitTimeValue, ...)
```

Input Arguments

RID	Request ID for the NCBI BLAST report, such as returned by the `blastncbi` function.
DescriptionsValue	Integer that specifies the number of descriptions in a report. Choices are any value ≥ 1 and ≤ 500. Default is 100.
AlignmentsValue	Integer that specifies the number of alignments to include in the report. Choices are any value ≥ 1 and ≤ 500. Default is 50. **Note:** This value must be ≤ the value you specified for the `'Alignments'` property when creating *RID* using the `blastncbi` function.
ToFileValue	String specifying a file name for saving report data.
FileFormatValue	String specifying the format of the file. Choices are `'text'` (default) or `'html'`.
WaitTimeValue	Positive value that specifies a time (in minutes) for the MATLAB software to wait for a report from the NCBI Web site to be available. If the report is still not available after the wait time, `getblast` returns an error message. Default behavior is to not wait for a report.

Tip Use the *RTOE* returned by the `blastncbi` function as the *WaitTimeValue*.	

Output Arguments

Data	MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from an NCBI BLAST report.

Description

The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of protein and nucleotide sequences against known sequences in online databases. `getblast` parses NCBI BLAST reports, including `blastn`, `blastp`, `psiblast`, `blastx`, `tblastn`, `tblastx`, and `megablast` reports.

Data = getblast(*RID*) reads *RID*, the Request ID for the NCBI BLAST report, and returns the report data in *Data*, a MATLAB structure or array of structures. The Request ID, *RID*, must be recently generated because NCBI purges reports after 24 hours.

Data = getblast(*RID*, ...'*PropertyName*', *PropertyValue*, ...) calls `getblast` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Data = getblast(*RID*, ...'Descriptions', *DescriptionsValue*, ...) specifies the number of descriptions in a report. Choices are any integer ≥ 1 and ≤ 500. Default is 100.

Data = getblast(*RID*, ...'Alignments', *AlignmentsValue*, ...) specifies the number of alignments to include in the report. Choices are any integer ≥ 1 and ≤ 500. Default is 50.

Note: This value must be $\leq$ the value you specified for the 'Alignments' property when creating *RID* using the `blastncbi` function.

Data = getblast(*RID*, ...'ToFile', *ToFileValue*, ...) saves the NCBI BLAST report data to a specified file. The default format for the file is 'text', but you can specify 'html' with the 'FileFormat' property.

Data = getblast(*RID*, ...'FileFormat', *FileFormatValue*, ...) specifies the format for the report. Choices are 'text' (default) or 'html'.

Data = getblast(*RID*, ...'WaitTime', *WaitTimeValue*, ...) pauses the MATLAB software and waits a specified time (in minutes) for a report from the NCBI Web site to be available. If the report is still unavailable after the wait time, getblast returns an error message. Choices are any positive value. Default behavior is to not wait for a report.

Tip Use the *RTOE* returned by the blastncbi function as the *WaitTimeValue*.

For more information about reading and interpreting BLAST reports, see:

http://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastDocs

Data contains the following fields.

Field	Description
RID	Request ID for retrieving results for a specific NCBI BLAST search.
Algorithm	NCBI algorithm used to do a BLAST search.
Query	Identifier of the query sequence submitted to a BLAST search.
Database	All databases searched.
Hits.Name	Name of a database sequence (subject sequence) that matched the query sequence.
Hits.Length	Length of a subject sequence.
Hits.HSPs.Score	Pairwise alignment score for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Expect	Expectation value for a high-scoring sequence pair between the query sequence and a subject sequence.

Field	Description
`Hits.HSPs.Identities`	Identities (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.Positives`	Identical or similar residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject amino acid sequence. **Note:** This field applies only to translated nucleotide or amino acid query sequences and/or databases.
`Hits.HSPs.Gaps`	Nonaligned residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.Frame`	Reading frame of the translated nucleotide sequence for a high-scoring sequence pair between the query sequence and a subject sequence. **Note:** This field applies only when performing translated searches, that is, when using `tblastx`, `tblastn`, and `blastx`.
`Hits.HSPs.Strand`	Sense (`Plus` = 5' to 3' and `Minus` = 3' to 5') of the DNA strands for a high-scoring sequence pair between the query sequence and a subject sequence. **Note:** This field applies only when using a nucleotide query sequence and database.
`Hits.HSPs.Alignment`	Three-row matrix showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence.

Field	Description
`Hits.HSPs.QueryIndices`	Indices of the query sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
`Hits.HSPs.SubjectIndices`	Indices of the subject sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
`Statistics`	Summary of statistical details about the performed search, such as lambda values, gap penalties, number of sequences searched, and number of hits.

Examples

1 Create an NCBI BLAST report request using a GenPept accession number.

```
RID = blastncbi('AAA59174','blastp','expect',1e-10)

RID =

    '1175088155-31624-126008617054.BLASTQ3'
```

2 Pass the Request ID for the report to the `getblast` function to parse the report, and return the report data in a MATLAB structure, and save the report data to a text file.

```
reportStruct = getblast(RID,'ToFile','AAA59174_BLAST.rpt')

reportStruct =

           RID: '1175093633-2786-174709873694.BLASTQ3'
     Algorithm: 'BLASTP 2.2.16 [Mar-11-2007]'
         Query: [1x63 char]
      Database: [1x96 char]
          Hits: [1x50 struct]
    Statistics: [1x1034 char]
```

Note: You may need to wait for the report to become available on the NCBI Web site before you can run the preceding command.

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990). Basic local alignment search tool. J. Mol. Biol. *215*, 403–410.

[2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. *25*, 3389–3402.

For more information about reading and interpreting NCBI BLAST reports, see:

```
http://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastDocs
```

See Also
blastformat | blastlocal | blastncbi | blastread | blastreadlocal

getbyname (phytree)

Branches and leaves from phytree object

Syntax

```
S = getbyname(Tree, Expression)
S = getbyname(Tree, String)
S = getbyname(Tree, String, 'Exact', ExactValue)
```

Arguments

Tree	phytree object created by `phytree` function (object constructor) or `phytreeread` function.
Expression	Regular expression or cell array of regular expressions to search for in *Tree*.
String	String or cell array of strings to search for in *Tree*.
ExactValue	Controls whether the full exact node name must match the string(s), ignoring case. Choices are `true` or `false` (default). When `true`, *S* is a numeric column vector indicating which node names match a query exactly, in full.

Description

S = getbyname(*Tree*, *Expression*) searches the nodes names in *Tree*, a phytree object, for the regular expression(s) specified by *Expression*. It returns *S*, a logical matrix of size NumNodes-by-M, where M is either 1 or the length of *Expression*. Each row in *S* corresponds to a node, and each column corresponds to a query in *Expression*. The logical matrix *S* indicates the node names that match *Expression*, ignoring case.

S = getbyname(*Tree*, *String*) searches the nodes names in *Tree*, a phytree object, for the string(s) specified by *String*. It returns *S*, a logical matrix of size NumNodes-by-M,

where M is either 1 or the length of *String*. Each row in *S* corresponds to a node, and each column corresponds to a query in *String*. The logical matrix *S* indicates the node names that match *String*, ignoring case.

S = getbyname(*Tree*, *String*, 'Exact', *ExactValue*) specifies whether the full exact node name must match the string(s), ignoring case. Choices are true or false (default). When true, *S* is a numeric column vector indicating which node names match a query exactly, in full.

Examples

1   Read a phylogenetic tree file created from a protein family into a phytree object.

```
tr = phytreeread('pf00002.tree');
```

2   Determine all the mouse and human proteins by searching for nodes that include the strings 'mouse' and 'human' in their names.

```
sel = getbyname(tr,{'mouse','human'});
view(tr,any(sel,2));
```
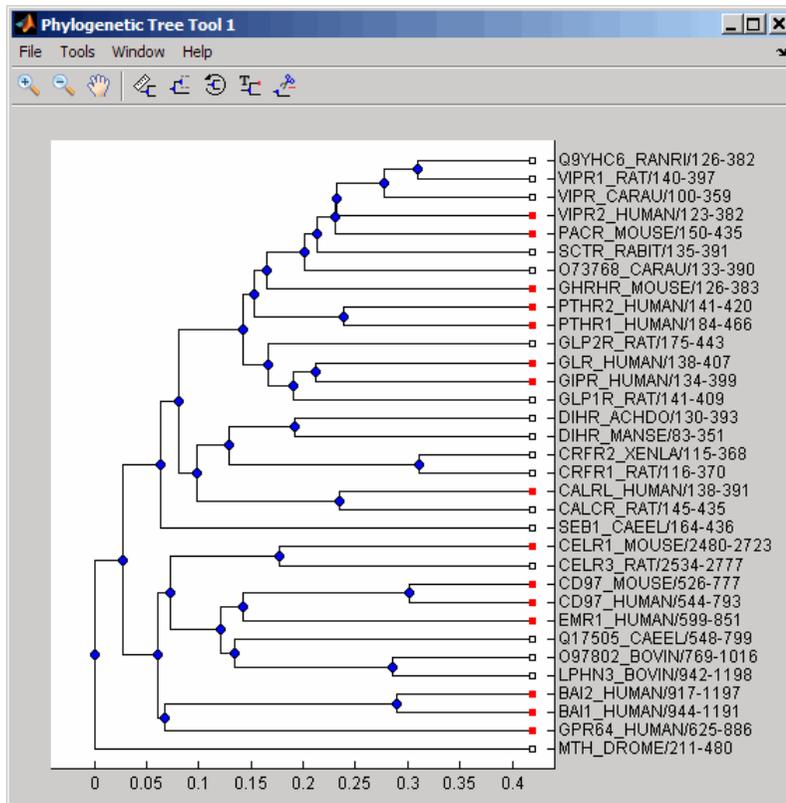
More About

- phytree object

See Also

phytree | prune | select | phytreeread | get

getcanonical (phytree)

Calculate canonical form of phylogenetic tree

Syntax

```
Pointers = getcanonical(Tree)
[Pointers, Distances, Names] = getcanonical(Tree)
```

Arguments

Tree	phytree object created by `phytree` function (object constructor).

Description

Pointers = getcanonical(*Tree*) returns the pointers for the canonical form of a phylogenetic tree (*Tree*). In a canonical tree the leaves are ordered alphabetically and the branches are ordered first by their width and then alphabetically by their first element. A canonical tree is isomorphic to all the trees with the same skeleton independently of the order of their leaves and branches.

[*Pointers, Distances, Names*] = getcanonical(*Tree*) returns, in addition to the pointers described above, the reordered distances (*Distances*) and node names (*Names*).

Examples

1   Create two phylogenetic trees with the same skeleton but slightly different distances.

```
b = [1 2; 3 4; 5 6; 7 8;9 10];
tr_1 = phytree(b,[.1 .2 .3 .3 .4 ]');
tr_2 = phytree(b,[.2 .1 .2 .3 .4 ]');
```

2   Plot the trees.

```
  plot(tr_1)
  plot(tr_2)
```

3 Check whether the trees have an isomorphic construction.

```
isequal(getcanonical(tr_1),getcanonical(tr_2))

ans =
    1
```

More About

•   phytree object

See Also

`phytree` | `select` | `phytreeread` | `getbyname` | `subtree`

getData

Class: GFFAnnotation

Create structure containing subset of data from `GFFAnnotation` object

Syntax

```
AnnotStruct = getData(AnnotObj)
AnnotStruct = getData(AnnotObj,StartPos,EndPos)
AnnotStruct = getData(AnnotObj,Subset)
AnnotStruct = getData( ___ ,Name,Value)
```

Description

`AnnotStruct = getData(AnnotObj)` returns AnnotStruct, an array of structures containing data from all elements in AnnotObj. The fields in the return structures are the same as the elements in the `FieldNames` property of AnnotObj.

`AnnotStruct = getData(AnnotObj,StartPos,EndPos)` returns AnnotStruct, an array of structures containing data from a subset of the elements in AnnotObj that falls within each reference sequence range specified by StartPos and EndPos.

`AnnotStruct = getData(AnnotObj,Subset)` returns AnnotStruct, an array of structures containing subset of data from AnnotObj specified by Subset, a vector of integers.

`AnnotStruct = getData( ___ ,Name,Value)` returns AnnotStruct, an array of structures, using any of the input arguments in the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

Tips

Using `getData` creates a structure, which provides better access to the annotation data than an object.

- You can access all field values in a structure.
- You can extract, assign, and delete field values.
- You can use linear indexing to access field values of specific annotations. For example, you can access the start value of only the fifth annotation.

Input Arguments

`AnnotObj`

Object of the `GFFAnnotation` class.

Default:

`StartPos`

Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

Default:

`EndPos`

Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

Default:

`Subset`

Vector of positive integers less than or equal to the number of entries in the object. Use the vector `Subset` to retrieve any element or subset of data from the object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'  '`). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'Reference'

String or cell array of strings specifying one or more reference sequences in AnnotObj. Only annotations whose reference field matches one of the strings are included in AnnotStruct.

Default:

'Feature'

String or cell array of strings specifying one or more features in AnnotObj. Only annotations whose feature field matches one of the strings are included in AnnotStruct.

Default:

'Overlap'

Minimum number of base positions that an annotation must overlap in the range, to be included in AnnotStruct. This value can be any of the following:

- Positive integer
- `'full'` — An annotation must be fully contained in the range to be included.
- `'start'` — An annotation's start position must lie within the range to be included.

Default: 1

Output Arguments

AnnotStruct

Array of structures containing data from elements in AnnotObj. The fields in the return structures are the same as the elements in the `FieldNames` property of AnnotObj, and specified by GFF (General Feature Format) specifications document. Specifically, these fields are:

- Reference
- Start
- Stop
- Feature

- Source
- Score
- Strand
- Frame
- Attributes

Examples

Retrieve Subsets of Data from a GFFAnnotation Object

Construct a `GFFAnnotation` object using a GFF-formatted file that is provided with Bioinformatics Toolbox.

```
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
```

Extract annotations for positions 10,000 through 20,000 from the reference sequence.

```
AnnotStruct1 = getData(GFFAnnotObj,10000,20000)

AnnotStruct1 =

9x1 struct array with fields:
    Reference
    Start
    Stop
    Feature
    Source
    Score
    Strand
    Frame
    Attributes
```

Extract the first five annotations from the object.

```
AnnotStruct2 = getData(GFFAnnotObj,[1:5])

AnnotStruct2 =

5x1 struct array with fields:
    Reference
    Start
```

```
Stop
Feature
Source
Score
Strand
Frame
Attributes
```

See Also
GTFAnnotation.getData

How To
·   "Store and Manage Feature Annotations in Objects"

Related Links
·   GFF (General Feature Format) specifications document

getData

Class: GTFAnnotation

Create structure containing subset of data from `GTFAnnotation` object

Syntax

```
AnnotStruct = getData(AnnotObj)
AnnotStruct = getData(AnnotObj,StartPos,EndPos)
AnnotStruct = getData(AnnotObj,Subset)
AnnotStruct = getData( ___ ,Name,Value)
```

Description

`AnnotStruct = getData(AnnotObj)` returns AnnotStruct, an array of structures containing data from all elements in AnnotObj. The fields in the return structures are the same as the elements in the `FieldNames` property of AnnotObj.

`AnnotStruct = getData(AnnotObj,StartPos,EndPos)` returns AnnotStruct, an array of structures containing data from a subset of the elements in AnnotObj that falls within each reference sequence range specified by StartPos and EndPos.

`AnnotStruct = getData(AnnotObj,Subset)` returns AnnotStruct, an array of structures containing subset of data from AnnotObj specified by Subset, a vector of integers.

`AnnotStruct = getData( ___ ,Name,Value)` returns AnnotStruct, an array of structures, using any of the input arguments from the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

Tips

Using `getdata` creates a structure, which provides better access to the annotation data than an object.

- You can access all field values in a structure.
- You can not only extract field values, but also assign and delete values.

- You can use linear indexing to access field values of specific annotations. For example, you can access the start value of only the fifth annotation.

Input Arguments

AnnotObj

Object of the GTFAnnotation class.

Default:

StartPos

Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

Default:

EndPos

Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

Default:

Subset

Vector of positive integers equal or less than the number of entries in the object. Use the vector Subset to retrieve any element or subset of data from the object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'Reference'

String or cell array of strings specifying one or more reference sequences in AnnotObj. Only annotations whose reference field matches one of the strings are included in AnnotStruct.

Default:

`'Feature'`

String or cell array of strings specifying one or more features in AnnotObj. Only annotations whose feature field matches one of the strings are included in AnnotStruct.

Default:

`'Gene'`

String or cell array of strings specifying one or more genes in AnnotObj. Only annotations whose gene field matches one of the strings are included in AnnotStruct.

`'Transcript'`

String or cell array of strings specifying one or more transcripts in AnnotObj. Only annotations whose transcript field matches one of the strings are included in AnnotStruct.

`'Overlap'`

Minimum number of base positions that an annotation must overlap in the range, to be included in AnnotStruct. This value can be any of the following:

- Positive integer
- `'full'` — An annotation must be fully contained in the range to be included.
- `'start'` — An annotation's start position must lie within the range to be included.

Default: 1

Output Arguments

`AnnotStruct`

Array of structures containing data from elements in AnnotObj. The fields in the return structures are the same as the elements in the `FieldNames` property of AnnotObj, and specified by GTF2.2: A Gene Annotation Format. Specifically, these fields are:

- Reference

- Start
- Stop
- Feature
- Gene
- Transcript
- Source
- Score
- Strand
- Frame
- Attributes

Examples

Retrieve Subsets of Data from a GTFAnnotation Object

Construct a `GTFAnnotation` object using a GTF-formatted file that is provided with Bioinformatics Toolbox.

```
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
```

Extract the annotation data for positions 668,000 through 680,000 from the reference sequence.

```
AnnotStruct1 = getData(GTFAnnotObj,668000,680000)

AnnotStruct1 =

18x1 struct array with fields:
    Reference
    Start
    Stop
    Feature
    Gene
    Transcript
    Source
    Score
    Strand
    Frame
```

```
    Attributes
```

Extract the first five annotations from the object.

```
AnnotStruct2 = getData(GTFAnnotObj,[1:5])

AnnotStruct2 =

5x1 struct array with fields:
    Reference
    Start
    Stop
    Feature
    Gene
    Transcript
    Source
    Score
    Strand
    Frame
    Attributes
```

See Also
GFFAnnotation.getData

How To
· "Store and Manage Feature Annotations in Objects"

Related Links
· GTF2.2: A Gene Annotation Format

getdescendants (biograph)

Find descendants in biograph object

Syntax

Nodes = getdescendants(*BiographNode*)
Nodes = getdescendants(*BiographNode,NumGenerations*)

Arguments

BiographNode	Node in a biograph object.
NumGenerations	Number of generations. Enter a positive integer.

Description

Nodes = getdescendants(*BiographNode*) finds a given node (*BiographNode*) all of its direct descendants.

Nodes = getdescendants(*BiographNode,NumGenerations*) finds the node (*BiographNode*) and all of its direct descendants up to a specified number of generations (*NumGenerations*).

Examples

1   Create a biograph object.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg = biograph(cm)
```

2   Find one generation of descendants for node 4.

```
desNodes = getdescendants(bg.nodes(4));
set(desNodes,'Color',[1 .7 .7]);
bg.view;
```
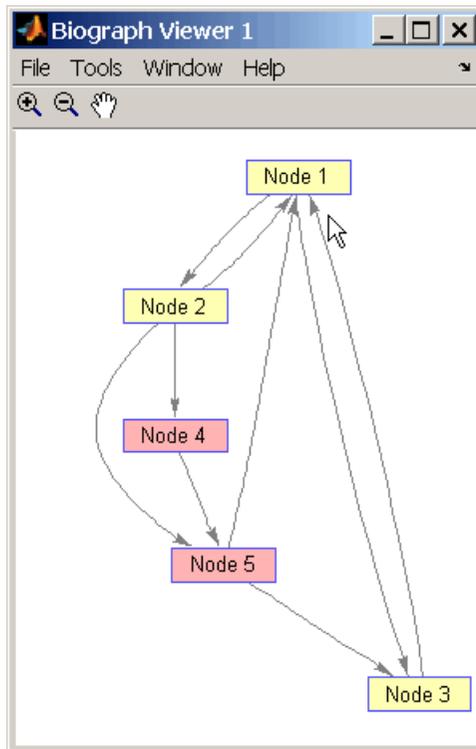
3   Find two generations of descendants for node 4.

```
desNodes = getdescendants(bg.nodes(4),2);
set(desNodes,'Color',[.7 1 .7]);
bg.view;
```
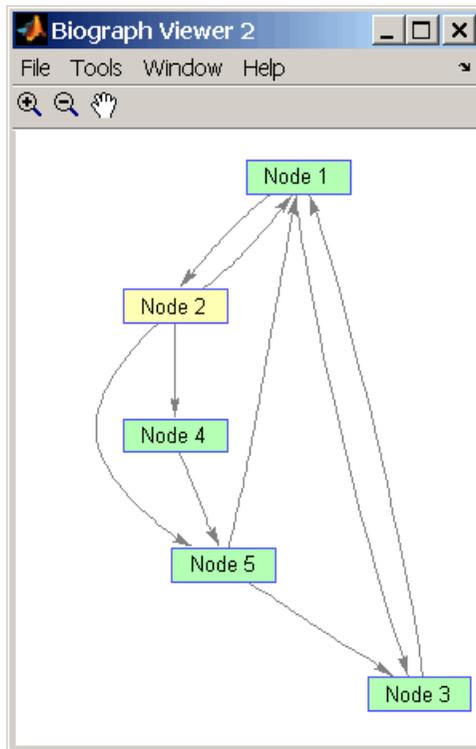
More About

- biograph object

See Also

```
biograph | get | getdescendants | getnodesbyid | set | view | dolayout |
getancestors | getedgesbynodeid | getrelatives
```

getdescendants

Class: geneont

Find terms that are descendants of specified Gene Ontology (GO) term

Syntax

```
DescendantIDs = getdescendants(GeneontObj, ID)
[DescendantIDs, Counts] = getdescendants(GeneontObj, ID)
... = getdescendants(..., 'Depth', DepthValue, ...)
... = getdescendants(..., 'Relationtype', RelationtypeValue, ...)
... = getdescendants(..., 'Exclude', ExcludeValue, ...)
```

Description

DescendantIDs = getdescendants(*GeneontObj*, *ID*) searches *GeneontObj*, a geneont object, for GO terms that are descendants of the GO term(s) specified by *ID*, which is a GO term identifier or vector of identifiers. It returns *DescendantIDs*, a vector of GO term identifiers including *ID*. *ID* is a nonnegative integer or a vector containing nonnegative integers.

[*DescendantIDs*, *Counts*] = getdescendants(*GeneontObj*, *ID*) also returns the number of times each descendant is found. *Counts* is a column vector with the same number of elements as terms in *GeneontObj*.

Tip The *Counts* return value is useful when you tally counts in gene enrichment studies. For more information, see Gene Ontology Enrichment in Microarray Data.

... = getdescendants(..., '*PropertyName*', *PropertyValue*, ...) calls getdescendants with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = getdescendants(..., 'Depth', *DepthValue*, ...) searches down through a specified number of levels, *DepthValue*, in the gene ontology. *DepthValue* is a positive integer. Default is Inf.

... = getdescendants(..., 'Relationtype', *RelationtypeValue*, ...) searches for specified relationship types, *RelationtypeValue*, in the gene ontology. *RelationtypeValue* is a string. Choices are 'is_a', 'part_of', or 'both' (default).

... = getdescendants(..., 'Exclude', *ExcludeValue*, ...) controls excluding *ID*, the original queried term(s), from the output *DescendantIDs*, unless the term was found while searching the gene ontology. Choices are true or false (default).

Input Arguments

GeneontObj	A geneont object, such as created by the geneont constructor function.
ID	GO term identifier or vector of identifiers.
DepthValue	Positive integer specifying the number of levels to search downward in the gene ontology.
RelationtypeValue	String specifying the relationship types to search for in the gene ontology. Choices are:
	• 'is_a'
	• 'part_of'
	• 'both' (default)
ExcludeValue	Controls excluding *ID*, the original queried term(s), from the output *DescendantIDs*, unless the term was reached while searching the gene ontology. Choices are true or false (default).

Output Arguments

DescendantIDs	Vector of GO term identifiers including *ID*.
Counts	Column vector with the same number of elements as terms in *GeneontObj*, indicating the number of times each descendant is found.

Examples

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GO = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of terms in the database.

```
Gene Ontology object with 27827 Terms.
```

2 Retrieve the descendants of the "aldo-keto reductase activity" GO term with a GO identifier of 4033.

```
descendants = getdescendants(GO,4033)

descendants =

        4032
        4033
        8106
       32018
       32866
       32867
       46568
       50112
       50236
```

3 Create a subordinate Gene Ontology.

```
subontology = GO(descendants)

Gene Ontology object with 9 Terms.
```

4 Create and display a report of the subordinate Gene Ontology terms, that includes the GO identifier and name.

```
rpt = [num2goid(cell2mat(get(subontology.terms,'id')))...
       get(subontology.terms,'name')]';
disp(sprintf('%s --> %s \n',rpt{:}))

GO:0004032 --> aldehyde reductase activity
GO:0004033 --> aldo-keto reductase activity
GO:0008106 --> alcohol dehydrogenase (NADP+) activity
GO:0032018 --> 2-methylbutanal reductase activity
GO:0032866 --> xylose reductase activity
GO:0032867 --> arabinose reductase activity
```

```
GO:0046568 --> 3-methylbutanal reductase activity
GO:0050112 --> inositol 2-dehydrogenase activity
GO:0050236 --> pyridoxine 4-dehydrogenase activity
```

5 View relationships of the subordinate Gene Ontology by using the `getmatrix` method to create a connection matrix to pass to the `biograph` function.

```
cm = getmatrix(subontology);
BG = biograph(cm,rpt(1,:));
view(BG)
```
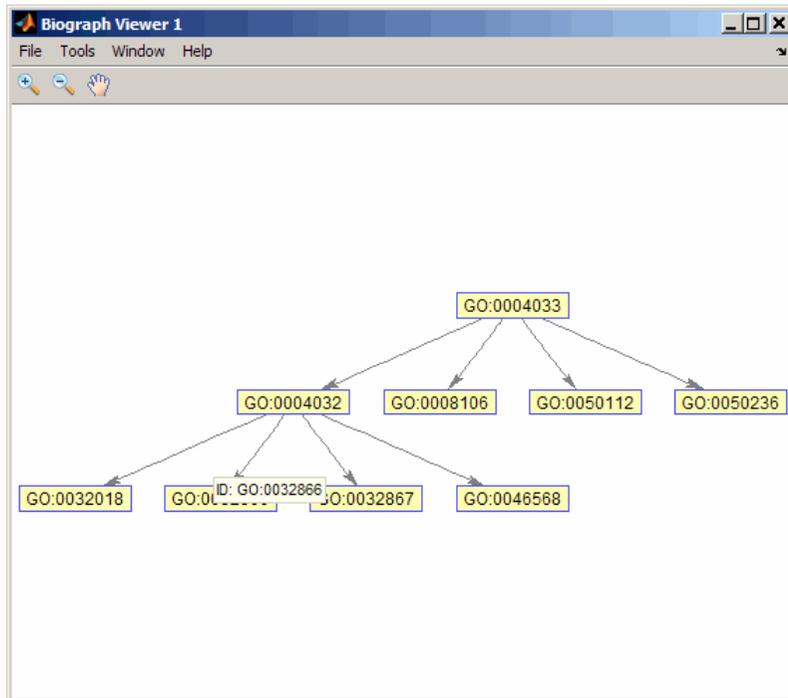

See Also

goannotread | num2goid | term

getDictionary

Class: BioIndexedFile

Retrieve reference sequence names from SAM-formatted source file associated with BioIndexedFile object

Syntax

Dict = getDictionary(*BioIFobj*)

Description

Dict = getDictionary(*BioIFobj*) returns *Dict*, a cell array of unique strings specifying the names of the reference sequences in the SAM-formatted source file associated with *BioIFobj*, a BioIndexedFile object.

Input Arguments

BioIFobj

Object of the `BioIndexedFile` class.

Default:

Output Arguments

Dict

Cell array of unique strings specifying the reference sequence names in the SAM-formatted source file associated with *BioIFobj*, a BioIndexedFile object.

See Also
BioIndexedFile.getSubset | BioIndexedFile | BioMap

How To

- "Work with Large Multi-Entry Text Files"
- "Manage Short-Read Sequence Data in Objects"

getedgesbynodeid (biograph)

Get handles to edges in biograph object

Syntax

Edges = getedgesbynodeid(*BGobj,SourceIDs,SinkIDs*)

Arguments

BGobj	Biograph object.
SourceIDs, *SinkIDs*	Enter a cell string, or an empty cell array (gets all edges).

Description

Edges = getedgesbynodeid(*BGobj,SourceIDs,SinkIDs*) gets the handles to the edges that connect the specified source nodes (*SourceIDs*) to the specified sink nodes (*SinkIDs*) in a biograph object.

Examples

1 Create a biograph object for the Hominidae family.

```
species = {'Homo','Pan','Gorilla','Pongo','Baboon',...
           'Macaca','Gibbon'};
cm = magic(7)>25 & 1-eye(7);
bg = biograph(cm, species);
```

2 Find all the edges that connect to the Homo node.

```
EdgesIn = getedgesbynodeid(bg,[],'Homo');
EdgesOut = getedgesbynodeid(bg,'Homo',[]);
set(EdgesIn,'LineColor',[0 1 0]);
set(EdgesOut,'LineColor',[1 0 0]);
bg.view;
```

3 Find all edges that connect members of the Cercopithecidae family to members of the Hominidae family.

```
Cercopithecidae = {'Macaca','Baboon'};
Hominidae = {'Homo','Pan','Gorilla','Pongo'};
edgesSel = getedgesbynodeid(bg,Cercopithecidae,Hominidae);
set(bg.edges,'LineColor',[.5 .5 .5]);
set(edgesSel,'LineColor',[O O 1]);
bg.view;
```

More About

· `biograph object`

See Also
`biograph` | `get` | `getdescendants` | `getnodesbyid` | `set` | `view` | `dolayout` | `getancestors` | `getedgesbynodeid` | `getrelatives`

getembl

Retrieve sequence information from EMBL database

Syntax

```
EMBLData = getembl(AccessionNumber)
EMBLData = getembl(..., 'ToFile', ToFileValue, ...)
EMBLSeq = getembl(..., 'SequenceOnly', SequenceOnlyValue, ...)
```

Input Arguments

AccessionNumber	Unique identifier for a sequence record. Enter a unique combination of letters and numbers.
ToFileValue	String specifying a file name or a path and file name to which to save the data. If you specify only a file name, the file is stored in the current folder.
SequenceOnlyValue	Controls the retrieving of only the sequence without the metadata. Choices are `true` or `false` (default).

Output Arguments

EMBLData	MATLAB structure with fields corresponding to EMBL data.
EMBLSeq	MATLAB character string representing the sequence.

Description

`getembl` retrieves information from the European Molecular Biology Laboratory (EMBL) database for nucleotide sequences. This database is maintained by the European Bioinformatics Institute (EBI). For more details about the EMBL database, see

`http://www.ebi.ac.uk/ena/about/formats`

EMBLData = getembl(*AccessionNumber*) searches for the accession number in the EMBL database (http://www.ebi.ac.uk/) and returns *EMBLData*, a MATLAB structure with fields corresponding to the EMBL two-character line type code. Each line type code is stored as a separate element in the structure.

EMBLData contains the following fields.

Field
Identification
Accession
SequenceVersion
DateCreated
DateUpdated
Description
Keyword
OrganismSpecies
OrganismClassification
Organelle
Reference
DatabaseCrossReference
Comments
Assembly
Feature
BaseCount
Sequence

EMBLData = getembl(..., '*PropertyName*', *PropertyValue*, ...) calls getembl with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

EMBLData = getembl(..., 'ToFile', *ToFileValue*, ...) saves the information to an EMBL-formatted file. *ToFileValue* is a string specifying a file name or a path and

1-739

file name to which to save the data. If you specify only a file name, the file is stored in the current folder.

Tip Read an EMBL-formatted file back into the MATLAB software using the `emblread` function.

EMBLSeq = `getembl(..., 'SequenceOnly', *SequenceOnlyValue*, ...)` controls the retrieving of only the sequence without the metadata. Choices are `true` or `false` (default).

Examples

Retrieve data for the rat liver apolipoprotein A-I.

```
emblout = getembl('X00558')
```

Retrieve data for the rat liver apolipoprotein A-I and save it to the file `rat_protein`. If you specify a file name without a path, the file is stored in the current folder.

```
emblout = getembl('X00558','ToFile','c:\project\rat_protein.txt')
```

Retrieve only the sequence for the rat liver apolipoprotein A-I.

```
Seq = getembl('X00558','SequenceOnly',true)
```

See Also
`emblread` | `getgenbank` | `getgenpept` | `getpdb` | `seqviewer`

getEntryByIndex

Class: BioIndexedFile

Retrieve entries from source file associated with BioIndexedFile object using numeric index

Syntax

Entries = getEntryByIndex(*BioIFobj, Indices*)

Description

Entries = getEntryByIndex(*BioIFobj, Indices*) extracts entries from the source file associated with *BioIFobj*, a BioIndexedFile object. It extracts and concatenates the entries specified by *Indices*, a numeric vector of positive integers. It returns *Entries*, a single string of concatenated entries. The value of each element in *Indices* must be less than or equal to the number of entries in the source file. A one-to-one relationship exists between the number and order of elements in *Indices* and the output *Entries*, even if *Indices* has repeated entries.

Tips

Use this method to visualize and explore a subset of the entries in the source file for validation purposes.

Input Arguments

BioIFobj

Object of the `BioIndexedFile` class.

Default:

Indices

Numeric vector of positive integers. The value of each element must be less than or equal to the number of entries in the source file associated with *BioIFobj*, the BioIndexedFile object.

Default:

Output Arguments

Entries

Single string of concatenated entries extracted from the source file associated with *BioIFobj*, the BioIndexedFile object.

Examples

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

```
% Create variable containing full absolute path of source file
sourcefile = which('yeastgenes.sgd');
% Create a BioIndexedFile object from the source file. Indicate
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with !
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
                            'KeyColumn', 3, 'HeaderPrefix','!')
```

Return the first, third, and fifth entries from the source file:

```
% Access 1st, 3rd, and 5th entries
subset_entries = getEntryByIndex(gene2goObj, [1 3 5]);
```

See Also
BioIndexedFile.getEntryByKey | BioIndexedFile.getSubset |
BioIndexedFile

How To

- "Work with Large Multi-Entry Text Files"

getEntryByKey

Class: BioIndexedFile

Retrieve entries from source file associated with BioIndexedFile object using alphanumeric key

Syntax

Entries = getEntryByKey(*BioIFobj*, *Key*)

Description

Entries = getEntryByKey(*BioIFobj*, *Key*) extracts entries from the source file associated with *BioIFobj*, a BioIndexedFile object. It extracts and concatenates the entries specified by *Key*, a string or cell array of strings specifying one or more alphanumeric keys. It returns *Entries*, a single string of concatenated entries. If the keys in the source file are not unique, it returns all entries that match a specified key, all at the position of the key in the *Key* cell array. If the keys in the source file are unique, there is a one-to-one relationship between the number and order of elements in *Key* and the output *Entries*.

Tips

Use this method to visualize and explore a subset of the entries in the source file for validation purposes.

Input Arguments

BioIFobj

Object of the BioIndexedFile class.

Default:

Key

String or cell array of strings specifying one or more keys in the source file associated with *BioIFobj*, the BioIndexedFile object.

Default:

Output Arguments

Entries

Single string of concatenated entries extracted from the source file associated with *BioIFobj*, the BioIndexedFile object.

Examples

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

```
% Create variable containing full absolute path of source file
sourcefile = which('yeastgenes.sgd');
% Create a BioIndexedFile object from the source file. Indicate
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with !
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
                            'KeyColumn', 3, 'HeaderPrefix','!')
```

Return the entries from the source file that are specified by the keys AAC1 and AAD10:

```
% Access entries that have the keys AAC1 and AAD10
subset_entries = getEntryByKey(gene2goObj, {'AAC1' 'AAD10'});
```

See Also

BioIndexedFile.getEntryByIndex | BioIndexedFile.getKeys | BioIndexedFile | BioIndexedFile.getSubset

How To

· "Work with Large Multi-Entry Text Files"

getExons

Class: GTFAnnotation

Return table of exons from `GTFAnnotation` object

Syntax

```
exons = getExons(AnnotObj)
[exons,junctions]= getExons(AnnotObj)
[ ___ ] = getExons(AnnotObj,'Reference',R)
[ ___ ] = getExons(AnnotObj,'Gene',G)
[ ___ ] = getExons(AnnotObj,'Transcript',T)
```

Description

`exons = getExons(AnnotObj)` returns exons, a table of existing exons in AnnotObj.

`[exons,junctions]= getExons(AnnotObj)` also returns junctions, a table of spliced junctions for each reference listed in AnnotObj.

`[ ___ ] = getExons(AnnotObj,'Reference',R)` returns the exons that belong to the reference(s) specified by R.

`[ ___ ] = getExons(AnnotObj,'Gene',G)` returns the exons that belong to the gene(s) specified by G.

`[ ___ ] = getExons(AnnotObj,'Transcript',T)` returns the exons that belong to the transcript(s) specified by T.

Input Arguments

`AnnotObj` — GTF annotation
GTFAnnotation object

GTF annotation, specified as a `GTFAnnotation` object.

R — Names of reference sequences
string | cell array of strings | categorical array

Names of reference sequences, specified as a string, cell array of strings, or categorical array.

The names must come from the `Reference` field of AnnotObj. If a name does not exist, the function provides a warning and ignores it.

G — Names of gene
string | cell array of strings | categorical array

Names of genes, specified as a string, cell array of strings, or categorical array.

The names must come from the `Gene` field of AnnotObj. If a name does not exist, the function provides a warning and ignores the name.

T — Names of transcripts
string | cell array of strings | categorical array

Names of transcripts, specified as a string, cell array of strings, or categorical array.

The names must come from the `Transcript` field of AnnotObj. If a name does not exist, the function gives a warning and ignores the name.

Output Arguments

exons — Exons in `AnnotObj`
table

Exons in AnnotObj, returned as a table. The table contains the following variables for each transcript.

Variable Name	Description
Transcript	Cell array of strings containing transcript IDs, obtained from the `Transcript` field of AnnotObj.
GeneName	Cell array of strings containing the names of expressed genes, obtained from the `Attributes` field of AnnotObj. This cell array can contain empty strings if the corresponding gene names are not found in `Attributes`.

Variable Name	Description
GeneID	Cell array of strings containing the expressed gene IDs, obtained from the Gene field of AnnotObj.
Reference	Categorical array representing the names of reference sequences to which the expressed genes belong. The reference names are from the Reference field of AnnotObj.
Start	Start location of each exon.
Stop	Stop location of each exon.
Strand	Categorical array containing the strand of expressed gene.

junctions — Spliced junctions for each reference
table

Spliced junctions for each reference, returned as a table. The table contains the following variables for each junction.

Variable Name	Description
Start	Start location of each junction.
Stop	Stop location of each junction.
Reference	Categorical array representing the names of reference sequences to which the junctions belong. The reference names are from the Reference field of AnnotObj.

Examples

Retrieve Exons from a GTF-formatted File

Create a GTFAnnotation object from a GTF-formatted file.

```
obj = GTFAnnotation('hum37_2_1M.gtf');
```

Get the list of gene names listed in the object.

```
gNames = getGeneNames(obj)
```

gNames =

```
                         'uc002qvu.2'
                         'uc002qvv.2'
                         'uc002qvw.2'
                         'uc002qvx.2'
                         'uc002qvy.2'
                         'uc002qvz.2'
                         'uc002qwa.2'
                         'uc002qwb.2'
                         'uc002qwc.1'
                         'uc002qwd.2'
                         'uc002qwe.3'
                         'uc002qwf.2'
                         'uc002qwg.2'
                         'uc002qwh.2'
                         'uc002qwi.3'
                         'uc002qwk.2'
                         'uc002qwl.2'
                         'uc002qwm.1'
                         'uc002qwn.1'
                         'uc002qwo.1'
                         'uc002qwp.2'
                         'uc002qwq.2'
                         'uc010ewe.2'
                         'uc010ewf.1'
                         'uc010ewg.2'
                         'uc010ewh.1'
                         'uc010ewi.2'
                         'uc010yim.1'
```

Get a table of exons which belong to the first gene `uc002qvu.2`.

```
exons = getExons(obj,'Gene',gNames{1})


exons =

    Transcript      GeneName        GeneID       Reference    Start      Stop      Strand
    _____    _____    _____    _____    _____    _____    _____

    'uc002qvu.2'      ''        'uc002qvu.2'      chr2       218138    219001    -
    'uc002qvu.2'      ''        'uc002qvu.2'      chr2       224864    224920    -
    'uc002qvu.2'      ''        'uc002qvu.2'      chr2       229966    230044    -
    'uc002qvu.2'      ''        'uc002qvu.2'      chr2       231023    231191    -
```

```
'uc002qvu.2'     ''             'uc002qvu.2'    chr2         233101    233229    -
'uc002qvu.2'     ''             'uc002qvu.2'    chr2         234160    234272    -
'uc002qvu.2'     ''             'uc002qvu.2'    chr2         247538    247602    -
'uc002qvu.2'     ''             'uc002qvu.2'    chr2         249731    249852    -
```

See Also

GTFAnnotation | GTFAnnotation.getData | GTFAnnotation.getFeatureNames
| GTFAnnotation.getGeneNames | GTFAnnotation.getIndex |
GTFAnnotation.getRange | GTFAnnotation.getReferenceNames
| GTFAnnotation.getSubset | GTFAnnotation.getGenes |
GTFAnnotation.getSegments | GTFAnnotation.getTranscripts

More About

·   "Store and Manage Feature Annotations in Objects"

External Web Sites

·   GTF2.2: A Gene Annotation Format

getFeatureNames

Class: GFFAnnotation

Retrieve unique feature names from `GFFAnnotation` object

Syntax

```
Features = getFeatureNames(AnnotObj)
```

Description

`Features = getFeatureNames(AnnotObj)` returns Features, a cell array of strings specifying the unique feature names associated with annotations in AnnotObj.

Input Arguments

AnnotObj

Object of the `GFFAnnotation` class.

Default:

Output Arguments

Features

Cell array of strings specifying the unique feature names associated with annotations in AnnotObj.

Examples

Construct a `GFFAnnotation` object from a GFF-formatted file that is provided with Bioinformatics Toolbox, and then retrieve the feature names from the annotation object:

```
% Construct a GFFAnnotation object from a GFF file
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
% Retrieve feature names for the annotation object
featureNames = getFeatureNames(GFFAnnotObj)

featureNames =

    'CDS'
    'exon'
    'five_prime_UTR'
    'gene'
    'mRNA'
    'miRNA'
    'ncRNA'
    'protein'
    'pseudogene'
    'pseudogenic_exon'
    'pseudogenic_transcript'
    'tRNA'
    'three_prime_UTR'
    'transposable_element_gene'
```

See Also

GTFAnnotation.getFeatureNames

How To

·   "Store and Manage Feature Annotations in Objects"

Related Links

·   GFF (General Feature Format) specifications document

getFeatureNames

Class: GTFAnnotation

Retrieve unique feature names from `GTFAnnotation` object

Syntax

```
Features = getFeatureNames(AnnotObj)
```

Description

`Features = getFeatureNames(AnnotObj)` returns Features, a cell array of strings specifying the unique feature names associated with annotations in AnnotObj.

Input Arguments

AnnotObj

Object of the `GTFAnnotation` class.

Default:

Output Arguments

Features

Cell array of strings specifying the unique feature names associated with annotations in AnnotObj.

Examples

Construct a `GTFAnnotation` object from a GTF-formatted file that is provided with Bioinformatics Toolbox, and then retrieve the feature names from the annotation object:

```
% Construct a GTFAnnotation object from a GTF file
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
% Retrieve feature names for the annotation object
featureNames = getFeatureNames(GTFAnnotObj)

featureNames =

    'CDS'
    'exon'
    'start_codon'
    'stop_codon'
```

See Also

GFFAnnotation.getFeatureNames

How To

·   "Store and Manage Feature Annotations in Objects"

Related Links

·   GTF2.2: A Gene Annotation Format

getgenbank

Retrieve sequence information from GenBank database

Syntax

```
Data = getgenbank(AccessionNumber)
getgenbank(AccessionNumber)
Data = getgenbank(..., 'PartialSeq', PartialSeqValue, ...)
Data = getgenbank(..., 'ToFile', ToFileValue, ...)
Data = getgenbank(..., 'FileFormat', FileFormatValue, ...)
Data = getgenbank(..., 'SequenceOnly', SequenceOnlyValue, ...)
```

Arguments

AccessionNumber	String specifying a unique alphanumeric identifier for a sequence record.
PartialSeqValue	Two-element array of integers containing the start and end positions of the subsequence [*StartBP*, *EndBP*] that specifies a subsequence to retrieve. *StartBP* is an integer between 1 and *EndBP*. *EndBP* is an integer between *StartBP* and the length of the sequence.
ToFileValue	String specifying either a file name or a path and file name for saving the GenBank data. If you specify only a file name, the file is saved to the MATLAB Current Folder.
FileFormatValue	String specifying the format for the sequence information. Choices are: • `'GenBank'` — Default when *SequenceOnlyValue* is `false`. • `'FASTA'` — Default when *SequenceOnlyValue* is `true`. When `'FASTA'`, then *Data* contains only two fields, `Header` and `Sequence`.

SequenceOnlyValue	Controls the return of only the sequence as a character array. Choices are `true` or `false` (default).

Description

`getgenbank` retrieves nucleotide information from the GenBank database. This database is maintained by the National Center for Biotechnology Information (NCBI). For more details about the GenBank database, see

`http://www.ncbi.nlm.nih.gov/Genbank/`

Data = `getgenbank`(*AccessionNumber*) searches for the accession number in the GenBank database and returns *Data*, a MATLAB structure containing information for the sequence.

Tip If an error occurs while retrieving the GenBank-formatted information, try rerunning the query. Errors can occur due to Internet connectivity issues that are unrelated to the GenBank record.

`getgenbank`(*AccessionNumber*) displays information in the MATLAB Command Window without returning data to a variable. The displayed information is only hyperlinks to the URLs used to search for and retrieve the data.

`getgenbank`(..., '*PropertyName*', *PropertyValue*, ...) calls `getgenbank` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Data = `getgenbank`(..., 'PartialSeq', *PartialSeqValue*, ...) returns the specified subsequence in the `Sequence` field of the MATLAB structure. *PartialSeqValue* is a two-element array of integers containing the start and end positions of the subsequence [*StartBP*, *EndBP*]. *StartBP* is an integer between 1 and *EndBP*. *EndBP* is an integer between *StartBP* and the length of the sequence.

Data = `getgenbank`(..., 'ToFile', *ToFileValue*, ...) saves the data returned from the GenBank database to a file. *ToFileValue* is a string specifying either a file

name or a path and file name for saving the GenBank data. If you specify only a file name, the file is saved to the MATLAB Current Folder.

Tip You can read a GenBank-formatted file back into MATLAB using the `genbankread` function.

Tip To append GenBank data to an existing file, specify that file name, and the data will be added to the end of the file.

If you are using `getgenbank` in a script, you can disable the append warning message by entering the following command lines before the `getgenbank` command:

```
warnState = warning %Save the current warning state
warning('off','Bioinfo:getncbidata:AppendToFile');
```
Then enter the following command line after the `getgenbank` command:

```
warning(warnState) %Reset warning state to previous settings
```

Data = getgenbank(..., 'FileFormat', *FileFormatValue*, ...) returns the sequence in the specified format. Choices are `'GenBank'` or `'FASTA'`. When `'FASTA'`, then *Data* contains only two fields, `Header` and `Sequence`. `'GenBank'` is the default when *SequenceOnlyValue* is `false`. `'FASTA'` is the default when *SequenceOnlyValue* is `true`.

Data = getgenbank(..., 'SequenceOnly', *SequenceOnlyValue*, ...) returns only the sequence in *Data*, a character array. Choices are `true` or `false` (default).

Note: If you use the `'SequenceOnly'` and `'ToFile'` properties together, the output is always a FASTA-formatted file.

Examples

Retrieving an RNA Sequence

To retrieve the sequence from chromosome 19 that codes for the human insulin receptor and store it in a structure, S, in the MATLAB Command Window, type:

```
S = getgenbank('M10051')
```

```
S =

                  LocusName: 'HUMINSR'
        LocusSequenceLength: '4723'
        LocusNumberofStrands: ''
               LocusTopology: 'linear'
           LocusMoleculeType: 'mRNA'
        LocusGenBankDivision: 'PRI'
      LocusModificationDate: '06-JAN-1995'
                  Definition: 'Human insulin receptor mRNA, complete cds.'
                   Accession: 'M10051'
                     Version: 'M10051.1'
                          GI: '186439'
                     Project: []
                      DBLink: []
                    Keywords: 'insulin receptor; tyrosine kinase.'
                     Segment: []
                      Source: 'Homo sapiens (human)'
              SourceOrganism: [4x65 char]
                   Reference: {[1x1 struct]}
                     Comment: [14x67 char]
                    Features: [51x74 char]
                         CDS: [1x1 struct]
                    Sequence: [1x4723 char]
                   SearchURL: [1x67 char]
                 RetrieveURL: [1x101 char]
```

Retrieving a Partial RNA Sequence

By looking at the `Features` field of the structure returned in Retrieving an RNA
Sequence, you can determine that the coding sequence is positions 139 through 4287. To
retrieve only the coding sequence from chromosome 19 that codes for the human insulin
receptor and store it in a structure, `CDS`, in the MATLAB Command Window, type:

```
CDS = getgenbank('M10051','PARTIALSEQ',[139,4287]);
```

See Also
genbankread | getembl | getgenpept | getpdb | seqviewer

getGenes

Class: GTFAnnotation

Return table of unique genes in `GTFAnnotation` object

Syntax

```
genes = getGenes(AnnotObj)
genes = getGenes(AnnotObj,'Reference',R)
genes = getGenes(AnnotObj,'Gene',G)
genes = getGenes(AnnotObj,'Transcript',T)
```

Description

`genes = getGenes(AnnotObj)` returns genes, a table of genes referenced by exons in AnnotObj.

`genes = getGenes(AnnotObj,'Reference',R)` returns the gene(s) that belong to the reference(s) specified by R.

`genes = getGenes(AnnotObj,'Gene',G)` returns the gene(s) specified by G.

`genes = getGenes(AnnotObj,'Transcript',T)` returns the gene(s) that contains the transcript(s) specified by T.

Input Arguments

AnnotObj — GTF annotation
GTFAnnotation object

GTF annotation, specified as a `GTFAnnotation` object.

R — Names of reference sequences
string | cell array of strings | categorical array

Names of reference sequences, specified as a string, cell array of strings, or categorical array.

The names must come from the `Reference` field of AnnotObj. If a name does not exist, the function provides a warning and ignores it.

G — Names of gene
string | cell array of strings | categorical array

Names of genes, specified as a string, cell array of strings, or categorical array.

The names must come from the `Gene` field of AnnotObj. If a name does not exist, the function provides a warning and ignores the name.

T — Names of transcripts
string | cell array of strings | categorical array

Names of transcripts, specified as a string, cell array of strings, or categorical array.

The names must come from the `Transcript` field of AnnotObj. If a name does not exist, the function gives a warning and ignores the name.

Output Arguments

genes — Genes referenced by exons in `AnnotObj`
table

Genes referenced by exons in AnnotObj, returned as a table. The table contains the following variables for each gene.

Variable Name	Description
`GeneID`	Cell array of strings containing gene IDs as listed in AnnotObj, obtained from the `Gene` field of AnnotObj.
`GeneName`	Cell array of strings containing gene names, obtained from the `Attributes` field of AnnotObj. This cell array can contain empty strings if the corresponding gene names are not found in `Attributes`.
`Reference`	Categorical array representing the names of reference sequences to which the genes belong, obtained from the `Reference` field of AnnotObj.
`Start`	Start location of the first exon in each gene.
`Stop`	Stop location of the last exon in each gene.

Variable Name	Description
Strand	Categorical array containing the strand of each gene.
NumTranscript	Integer array listing the number of transcripts in each gene.

Examples

Retrieve Genes from a GTF-formatted File

Create a GTFAnnotation object from a GTF-formatted file.

```
obj = GTFAnnotation('hum37_2_1M.gtf');
```

Retrieve unique reference names. In this case, there is only one reference sequence, which is chromosome 2 (chr2).

```
ref = getReferenceNames(obj)
```

```
ref =

    'chr2'
```

Get a table of all genes which belong to chr2.

```
genes = getGenes(obj,'Reference',ref)
```

```
genes =

      GeneID        GeneName    Reference    Start     Stop      Strand    NumTranscr:
    _____    _____    _____    _____    _____    _____    _____

    'uc010yim.1'      ''          chr2         41609     46385      -           1
    'uc002qvu.2'      ''          chr2        218138    249852      -           1
    'uc002qvv.2'      ''          chr2        218138    256690      -           1
    'uc002qvw.2'      ''          chr2        218138    260702      -           1
    'uc002qvx.2'      ''          chr2        218138    264068      -           1
    'uc002qvy.2'      ''          chr2        218138    264068      -           1
    'uc002qvz.2'      ''          chr2        218138    264392      -           1
    'uc002qwa.2'      ''          chr2        218138    264743      -           1
    'uc010ewe.2'      ''          chr2        218138    264810      -           1
```

```
'uc002qwb.2'     ''          chr2      239563     242178      -          1
'uc002qwc.1'     ''          chr2      243503     262786      -          1
'uc002qwd.2'     ''          chr2      264869     272481      +          1
'uc002qwe.3'     ''          chr2      264869     273148      +          1
'uc002qwg.2'     ''          chr2      264869     278280      +          1
'uc002qwh.2'     ''          chr2      264869     278280      +          1
'uc002qwf.2'     ''          chr2      264869     278280      +          1
'uc002qwi.3'     ''          chr2      279563     288308      -          1
'uc010ewf.1'     ''          chr2      279929     287892      -          1
'uc002qwk.2'     ''          chr2      667975     676158      -          1
'uc002qwl.2'     ''          chr2      667975     677439      -          1
'uc002qwm.1'     ''          chr2      677186     682802      +          1
'uc002qwn.1'     ''          chr2      780347     863877      -          1
'uc002qwo.1'     ''          chr2      780347     863877      -          1
'uc010ewg.2'     ''          chr2      902783     905582      -          1
'uc010ewh.1'     ''          chr2      902823     906011      -          1
'uc002qwp.2'     ''          chr2      946555    1157355      +          1
'uc002qwq.2'     ''          chr2      946555    1371382      +          1
'uc010ewi.2'     ''          chr2      946555    1371382      +          1
```

See Also

```
GTFAnnotation | GTFAnnotation.getData | GTFAnnotation.getFeatureNames
| GTFAnnotation.getGeneNames | GTFAnnotation.getIndex |
GTFAnnotation.getRange | GTFAnnotation.getReferenceNames
| GTFAnnotation.getSubset | GTFAnnotation.getTranscripts |
GTFAnnotation.getSegments | GTFAnnotation.getExons
```

More About

- "Store and Manage Feature Annotations in Objects"

External Web Sites

- GTF2.2: A Gene Annotation Format

getGeneNames

Class: GTFAnnotation

Retrieve unique gene names from `GTFAnnotation` object

Syntax

```
Genes = getGeneNames(AnnotObj)
```

Description

`Genes = getGeneNames(AnnotObj)` returns Genes, a cell array of strings specifying the unique gene names associated with annotations in AnnotObj.

Input Arguments

AnnotObj

Object of the `GTFAnnotation` class.

Default:

Output Arguments

Genes

Cell array of strings specifying the unique gene names associated with annotations in AnnotObj.

Examples

Construct a `GTFAnnotation` object from a GTF-formatted file that is provided with Bioinformatics Toolbox, and then retrieve a list of the unique gene names from the object:

```
% Construct a GTFAnnotation object from a GTF file
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
% Get gene names from object
geneNames = getGeneNames(GTFAnnotObj)

geneNames =

    'uc002qvu.2'
    'uc002qvv.2'
    'uc002qvw.2'
    'uc002qvx.2'
    'uc002qvy.2'
    'uc002qvz.2'
    'uc002qwa.2'
    'uc002qwb.2'
    'uc002qwc.1'
    'uc002qwd.2'
    'uc002qwe.3'
    'uc002qwf.2'
    'uc002qwg.2'
    'uc002qwh.2'
    'uc002qwi.3'
    'uc002qwk.2'
    'uc002qwl.2'
    'uc002qwm.1'
    'uc002qwn.1'
    'uc002qwo.1'
    'uc002qwp.2'
    'uc002qwq.2'
    'uc010ewe.2'
    'uc010ewf.1'
    'uc010ewg.2'
    'uc010ewh.1'
    'uc010ewi.2'
    'uc010yim.1'
```

How To

• "Store and Manage Feature Annotations in Objects"

Related Links

• GTF2.2: A Gene Annotation Format

getgenpept

Retrieve sequence information from GenPept database

Syntax

Data = getgenpept(*AccessionNumber*)
getgenpept(*AccessionNumber*)
Data = getgenpept(..., 'PartialSeq', *PartialSeqValue*, ...)
Data = getgenpept(..., 'ToFile', *ToFileValue*, ...)
Data = getgenpept(..., 'FileFormat', *FileFormatValue*, ...)
Data = getgenpept(..., 'SequenceOnly', *SequenceOnlyValue*, ...)

Arguments

AccessionNumber	String specifying a unique alphanumeric identifier for a sequence record.
PartialSeqValue	Two-element array of integers containing the start and end positions of the subsequence [*StartAA*, *EndAA*] that specifies a subsequence to retrieve. *StartAA* is an integer between 1 and *EndAA*; *EndAA* is an integer between *StartAA* and the length of the sequence.
ToFileValue	String specifying either a file name or a path and file name for saving the GenPept data. If you specify only a file name, the file is saved to the MATLAB Current Folder.
FileFormatValue	String specifying the format for the sequence information. Choices are: • 'Genpept' — Default when *SequenceOnlyValue* is false. • 'FASTA' — Default when *SequenceOnlyValue* is true. When 'FASTA', then *Data* contains only two fields, Header and Sequence.
SequenceOnlyValue	Controls the return of only the sequence as a character array. Choices are true or false (default).

Description

getgenpept retrieves a protein (amino acid) sequence information from the GenPept database, which is a translation of the nucleotide sequences in the GenBank database and is maintained by the National Center for Biotechnology Information (NCBI).

Note: NCBI has changed the name of their protein search engine from GenPept to Entrez Protein. However, the function names in the Bioinformatics Toolbox software (getgenpept and genpeptread) are unchanged representing the still-used GenPept report format.

Data = getgenpept(*AccessionNumber*) searches for the accession number in the GenPept database and returns *Data*, a MATLAB structure containing information for the sequence.

Tip If an error occurs while retrieving the GenPept-formatted information, try rerunning the query. Errors can occur due to Internet connectivity issues that are unrelated to the GenPept record.

---

getgenpept(*AccessionNumber*) displays information in the MATLAB Command Window without returning data to a variable. The displayed information is only hyperlinks to the URLs used to search for and retrieve the data.

getgenpept(..., '*PropertyName*', *PropertyValue*, ...) calls getgenpept with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Data = getgenpept(..., 'PartialSeq', *PartialSeqValue*, ...) returns the specified subsequence in the Sequence field of the MATLAB structure. *PartialSeqValue* is a two-element array of integers containing the start and end positions of the subsequence [*StartAA*, *EndAA*]. *StartAA* is an integer between 1 and *EndAA*; *EndAA* is an integer between *StartAA* and the length of the sequence.

Data = getgenpept(..., 'ToFile', *ToFileValue*, ...) saves the data returned from the GenPept database to a file. *ToFileValue* is a string specifying either a file

name or a path and file name for saving the GenPept data. If you specify only a file name, the file is saved to the MATLAB Current Folder.

Tip You can read a GenPept-formatted file back into MATLAB using the `genpeptread` function.

Tip To append GenPept data to an existing file, specify that file name, and the data will be added to the end of the file.

If you are using `getgenpept` in a script, you can disable the append warning message by entering the following command lines before the `getgenpept` command:

```
warnState = warning %Save the current warning state
warning('off','Bioinfo:getncbidata:AppendToFile');
```
Then enter the following command line after the `getgenpept` command:

```
warning(warnState) %Reset warning state to previous settings
```

Data = getgenpept(..., 'FileFormat', *FileFormatValue*, ...) returns the sequence in the specified format. Choices are `'GenPept'` or `'FASTA'`. When `'FASTA'`, then *Data* contains only two fields, `Header` and `Sequence`. `'GenPept'` is the default when *SequenceOnlyValue* is `false`. `'FASTA'` is the default when *SequenceOnlyValue* is `true`.

Data = getgenpept(..., 'SequenceOnly', *SequenceOnlyValue*, ...) returns only the sequence in *Data*, a character array. Choices are `true` or `false` (default).

Note: If you use the `'SequenceOnly'` and `'ToFile'` properties together, the output is always a FASTA-formatted file.

Examples

Retrieving a Peptide Sequence

To retrieve the sequence for the human insulin receptor and store it in a structure, `Seq`, in the MATLAB Command Window, type:

```
Seq = getgenpept('AAA59174')

Seq =

                    LocusName: 'AAA59174'
          LocusSequenceLength: '1382'
        LocusNumberofStrands: ''
                LocusTopology: 'linear'
            LocusMoleculeType: ''
         LocusGenBankDivision: 'PRI'
       LocusModificationDate: '06-JAN-1995'
                   Definition: 'insulin receptor precursor.'
                    Accession: 'AAA59174'
                      Version: 'AAA59174.1'
                           GI: '307070'
                      Project: []
                     DBSource: 'locus HUMINSR accession M10051.1'
                     Keywords: ''
                       Source: 'Homo sapiens (human)'
                SourceOrganism: [4x65 char]
                    Reference: {[1x1 struct]}
                      Comment: [14x67 char]
                     Features: [40x64 char]
                     Sequence: [1x1382 char]
                    SearchURL: [1x104 char]
                  RetrieveURL: [1x92 char]
```

Retrieving a Partial Peptide Sequence

By looking at the `Features` field of the structure returned in Retrieving a Peptide
Sequence, you can determine that the furin-like repeats domain is positions 234 through
281. To retrieve only the furin-like repeats domain from the sequence for the human
insulin receptor and store it in a structure, `Fur`, in the MATLAB Command Window,
type:

```
Fur = getgenpept('AAA59174','PARTIALSEQ',[234,281]);
```

See Also
genpeptread | getembl | getgenbank | getpdb

getgeodata

Retrieve Gene Expression Omnibus (GEO) format data

Syntax

```
GEOData = getgeodata(AccessionNumber)
getgeodata(AccessionNumber, 'ToFile', ToFileValue)
```

Input Arguments

AccessionNumbe	String specifying a unique identifier for a GEO Sample (GSM), Data Set (GDS), Platform (GPL), or Series (GSE) record in the GEO database.
	Tip Recently submitted data sets may not be available for immediate download. There can be a one- to two-day delay between an experiment being submitted to the GEO database and its availability on the FTP site.
	Tip If you are unable to retrieve data for an accession number, increase your Java® heap space:
	• If you have MATLAB version 7.10 (R2010a) or later, see "Java Heap Memory Preferences"
	• If you have MATLAB version 7.9 (R2009b) or earlier, see http://www.mathworks.com/support/solutions/data/1-18I2C.html
ToFileValue	String specifying a file name or path and file name for saving the data. If you specify only a file name, that file will be saved in the MATLAB Current Folder.

Output Arguments

GEOData	MATLAB structure containing information for a GEO record retrieved from the GEO database.

Description

GEOData = getgeodata(*AccessionNumber*) searches the Gene Expression Omnibus database for the specified accession number of a Sample (GSM), Data Set (GDS), Platform (GPL), or Series (GSE) record and returns a MATLAB structure containing the following fields:

Field	Description
Scope	Type of data retrieved (SAMPLE, DATASET, PLATFORM, or SERIES)
Accession	Accession number for record in GEO database.
Header	Microarray experiment information.
ColumnDescriptions	Cell array containing descriptions of columns in the data.
ColumnNames	Cell array containing names of columns in the data.
Data	Array containing microarray data.
Identifier (GDS files only)	Cell array containing probe IDs.
IDRef (GDS files only)	Cell array containing indices to probes.

Note: Currently, the getgeodata function supports Sample (GSM), Data Set (GDS), Platform (GPL), and Series (GSE) records.

getgeodata(*AccessionNumber*, 'ToFile', *ToFileValue*) saves the data returned from the database to a file.

Note: You can read a GEO SOFT-formatted file back into the MATLAB software using the geosoftread function. You can read a GEO SERIES-formatted file back into the MATLAB software using the geoseriesread function.

For more information, see

http://www.ncbi.nlm.nih.gov/About/disclaimer.html

Examples

geoStruct = getgeodata('GSM1768')

See Also
geoseriesread | geosoftread | getgenbank | getgenpept

getHeader

Class: BioRead

Retrieve sequence headers from object

Syntax

```
Headers = getHeader(BioObj)
Headers = getHeader(BioObj, Subset)
```

Description

Headers = getHeader(*BioObj*) returns *Headers*, a cell array of strings containing sequence headers from an object.

Headers = getHeader(*BioObj*, *Subset*) returns header strings for only object elements specified by *Subset*.

Input Arguments

BioObj

Object of the `BioRead` or `BioMap` class.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

`Headers`

Cell array of strings containing the sequence headers specified by *Subset* in *BioObj*.

Examples

Retrieve the headers from different elements of a `BioRead` object:

```
% Create variables containing sequences, quality scores, and headers
seqs = {randseq(10); randseq(15); randseq(20)};
quals = {repmat('!', 1, 10); repmat('%', 1, 15); repmat('&', 1, 20)};
headers = {'H1'; 'H2'; 'H3'};
% Construct a BioRead object from these three variables
BRObj = BioRead(seqs, quals, headers);
% Retrieve the Header value of the second element in the object
getHeader(BRObj, 2);
getHeader(BRObj, [false true false]);
% Retrieve the Header values of the first and third elements in the
% object
getHeader(BRObj, [1 3]);
getHeader(BRObj, [true false true]);
% Retrieve the Header value of all the elements in the object
getHeader(BRObj);
getHeader(BRObj, 1:3);
```

Alternatives

An alternative to using the `getHeader` method is to use dot indexing with the `Header` property:

```
BioObj.Header(Indices)
```

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers.

See Also
`setHeader` | `BioRead` | `BioMap`

How To
- "Manage Short-Read Sequence Data in Objects"

Related Links
- Sequence Read Archive
- SAM format specification

gethmmalignment

Retrieve multiple sequence alignment associated with hidden Markov model (HMM) profile from PFAM database

Syntax

```
AlignStruct = gethmmalignment(PFAMName)
AlignStruct = gethmmalignment(PFAMAccessNumber)
AlignStruct = gethmmalignment(PFAMNumber)
AlignStruct = gethmmalignment(..., 'ToFile', ToFileValue, ...)
AlignStruct = gethmmalignment(..., 'Type', TypeValue, ...)
AlignStruct = gethmmalignment(..., 'Mirror', MirrorValue, ...)
AlignStruct = gethmmalignment(..., 'IgnoreGaps', IgnoreGaps, ...)
```

Input Arguments

PFAMName	String specifying a protein family name (unique identifier) of an HMM profile record in the PFAM database. For example, `'7tm_2'`.
PFAMAccessNumber	String specifying a protein family accession number of an HMM profile record in the PFAM database. For example, `'PF00002'`.
PFAMNumber	Integer specifying a protein family number of an HMM profile record in the PFAM database. For example, `2` is the protein family number for the protein family `PF00002`.
ToFileValue	String specifying a file name or a path and file name for saving the data. If you specify only a file name, that file will be saved in the MATLAB Current Folder.
TypeValue	String that specifies the set of alignments returned. Choices are: • `'full'` — Default. Returns all alignments that fit the HMM profile. • `'seed'` — Returns only the alignments used to generate the HMM profile.

MirrorValue	String that specifies a Web database. Choices are: • `'Sanger'` (default) • `'Janelia'`
IgnoreGapsValue	Controls the removal of the symbols - and . from the sequence. Choices are `true` or `false` (default).

Output Arguments

AlignStruct	MATLAB structure array containing the multiple sequence alignment associated with an HMM profile.

Description

AlignStruct = gethmmalignment(*PFAMName*) searches the PFAM database for the HMM profile record represented by *PFAMName*, a protein family name, retrieves the multiple sequence alignment associated with the HMM profile, and returns *AlignStruct*, a MATLAB structure array, with each structure containing the following fields:

Field	Description
`Header`	Protein name
`Sequence`	Protein sequence

AlignStruct = gethmmalignment(*PFAMAccessNumber*) searches the PFAM database for the HMM profile record represented by *PFAMAccessNumber*, a protein family accession number, retrieves the multiple sequence alignment associated with the HMM profile, and returns *AlignStruct*, a MATLAB structure array.

AlignStruct = gethmmalignment(*PFAMNumber*) determines a protein family accession number from *PFAMNumber*, an integer, searches the PFAM database for the associated HMM profile record, retrieves the multiple sequence alignment associated with the HMM profile, and returns *AlignStruct*, a MATLAB structure array.

AlignStruct = gethmmalignment(..., '*PropertyName*', *PropertyValue*, ...) calls gethmmalignment with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

AlignStruct = gethmmalignment(..., 'ToFile', *ToFileValue*, ...) saves the data returned from the PFAM database to a file specified by *ToFileValue*.

Note: You can read a FASTA-formatted file containing PFAM data back into the MATLAB software using the fastaread function.

AlignStruct = gethmmalignment(..., 'Type', *TypeValue*, ...) specifies the set of alignments returned. Choices are:

- 'full' — Default. Returns all sequences that fit the HMM profile.
- 'seed' — Returns only the sequences used to generate the HMM profile.

AlignStruct = gethmmalignment(..., 'Mirror', *MirrorValue*, ...) specifies a Web database. Choices are:

- 'Sanger' (default)
- 'Janelia'

You can reach other mirror sites by passing the complete URL to the fastaread function.

Note: These mirror sites are maintained separately and may have slight variations.

For more information about the PFAM database, see:

```
http://pfam.sanger.ac.uk
http://pfam.janelia.org/
```

AlignStruct = gethmmalignment(..., 'IgnoreGaps', *IgnoreGaps*, ...) controls the removal of the symbols - and . from the sequence. Choices are true or false (default).

Examples

To retrieve a multiple alignment of the sequences used to train the HMM profile for global alignment to the 7-transmembrane receptor protein in the secretin family, enter either of the following:

```
pfamalign = gethmmalignment(2,'Type','seed')

pfamalign = gethmmalignment('PF00002','Type','seed')

pfamalign =

32x1 struct array with fields:
    Header
    Sequence
```

See Also

fastaread | gethmmprof | gethmmtree | multialignread | multialignwrite | pfamhmmread

gethmmprof

Retrieve hidden Markov model (HMM) profile from PFAM database

Syntax

HMMStruct = gethmmprof(*PFAMName*)
HMMStruct = gethmmprof(*PFAMNumber*)
HMMStruct = gethmmprof(..., 'ToFile', *ToFileValue*, ...)
HMMStruct = gethmmprof(..., 'Mode', *ModeValue*, ...)
HMMStruct = gethmmprof(..., 'Mirror', *MirrorValue*, ...)

Input Arguments

PFAMName	String specifying a protein family name (unique identifier) of an HMM profile record in the PFAM database. For example, '7tm_2'.
PFAMNumber	Integer specifying a protein family number of an HMM profile record in the PFAM database. For example, 2 is the protein family number for the protein family 'PF00002'.
ToFileValue	String specifying a file name or a path and file name for saving the data. If you specify only a file name, that file will be saved in the MATLAB Current Folder.
ModeValue	String that specifies the returned alignment mode. Choices are: • 'ls' — Default. Global alignment mode. • 'fs' — Local alignment mode.
MirrorValue	String that specifies a Web database. Choices are: • 'Sanger' (default) • 'Janelia'

Output Arguments

HMMStruct	MATLAB structure containing information for an HMM profile retrieved from the PFAM database.

Description

Note: gethmmprof retrieves information from PFAM-HMM profiles, from file format version HMMER2.0 to HMMER3/b.

HMMStruct = gethmmprof(*PFAMName*) searches the PFAM database for the record represented by *PFAMName* (a protein family name), retrieves the HMM profile information, and stores it in *HMMStruct*, a MATLAB structure containing the following fields corresponding to parameters of an HMM profile.

Field	Description
Name	The protein family name (unique identifier) of the HMM profile record in the PFAM database.
PfamAccessionNumber	The protein family accession number of the HMM profile record in the PFAM database.
ModelDescription	Description of the HMM profile.
ModelLength	The length of the profile (number of MATCH states).
Alphabet	The alphabet used in the model, 'AA' or 'NT'. **Note:** AlphaLength is 20 for 'AA' and 4 for 'NT'.
MatchEmission	Symbol emission probabilities in the MATCH states. The format is a matrix of size ModelLength-by-AlphaLength, where each row corresponds to the emission distribution for a specific MATCH state.
InsertEmission	Symbol emission probabilities in the INSERT state.

Field	Description
	The format is a matrix of size `ModelLength`-by-`AlphaLength`, where each row corresponds to the emission distribution for a specific INSERT state.
`NullEmission`	Symbol emission probabilities in the MATCH and INSERT states for the NULL model.
	The format is a 1-by-`AlphaLength` row vector.
	Note: NULL probabilities are also known as the background probabilities.
`BeginX`	BEGIN state transition probabilities.
	Format is a 1-by-(`ModelLength + 1`) row vector:
	`[B->D1 B->M1 B->M2 B->M3 .... B->Mend]`
`MatchX`	MATCH state transition probabilities.
	Format is a 4-by-(`ModelLength - 1`) matrix:

	`[M1->M2 M2->M3 ... M[end-1]->Mend;`
	` M1->I1 M2->I2 ... M[end-1]->I[end-1];`
	` M1->D2 M2->D3 ... M[end-1]->Dend;`
	` M1->E  M2->E  ... M[end-1]->E  ]`

`InsertX`	INSERT state transition probabilities.
	Format is a 2-by-(`ModelLength - 1`) matrix:
	`[ I1->M2 I2->M3 ... I[end-1]->Mend;` `  I1->I1 I2->I2 ... I[end-1]->I[end-1] ]`
`DeleteX`	DELETE state transition probabilities.
	Format is a 2-by-(`ModelLength - 1`) matrix:
	`[ D1->M2 D2->M3 ... D[end-1]->Mend ;` `  D1->D2 D2->D3 ... D[end-1]->Dend ]`

Field	Description

FlankingInsertX	Flanking insert states (N and C) used for LOCAL profile alignment.
	
	Format is a 2-by-2 matrix:
	
	`[N->B  C->T ;`
	` N->N  C->C]`

LoopX	Loop states transition probabilities used for multiple hits alignment.
	
	Format is a 2-by-2 matrix:
	
	`[E->C  J->B ;`
	` E->J  J->J]`

NullX	Null transition probabilities used to provide scores with log-odds values also for state transitions. Format is a 2-by-1 column vector: `[G->F ; G->G]`

HMMStruct = gethmmprof(*PFAMNumber*) determines a protein family accession number from *PFAMNumber* (an integer), searches the PFAM database for the associated record, retrieves the HMM profile information, and stores it in *HMMStruct*, a MATLAB structure.

HMMStruct = gethmmprof(..., '*PropertyName*', *PropertyValue*, ...) calls gethmmprof with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

HMMStruct = gethmmprof(..., 'ToFile', *ToFileValue*, ...) saves the data returned from the PFAM database in a file specified by *ToFileValue*.

Note: You can read an HMM-formatted file back into the MATLAB software using the pfamhmmread function.

HMMStruct = gethmmprof(..., 'Mode', *ModeValue*, ...) specifies the returned alignment mode. Choices are:

- 'ls' (default) — Global alignment mode.
- 'fs' — Local alignment mode.

HMMStruct = gethmmprof(..., 'Mirror', *MirrorValue*, ...) specifies a Web database. Choices are:

- 'Sanger' (default)
- 'Janelia'

You can reach other mirror sites by passing the complete URL to the pfamhmmread function.

Note: These mirror sites are maintained separately and may have slight variations.

For more information about the PFAM database, see:

```
http://pfam.sanger.ac.uk
http://pfam.janelia.org/
```

For more information on HMM profile models, see "HMM Profile Model" on page 1-976.

Examples

To retrieve a hidden Markov model (HMM) profile for the global alignment of the 7-transmembrane receptor protein in the secretin family, enter either of the following:

```
hmm = gethmmprof(2)

hmm = gethmmprof('7tm_2')

hmm =

                   Name: '7tm_2'
    PfamAccessionNumber: 'PF00002.14'
        ModelDescription: [1x42 char]
```

```
       ModelLength: 296
          Alphabet: 'AA'
     MatchEmission: [296x20 double]
    InsertEmission: [296x20 double]
      NullEmission: [1x20 double]
            BeginX: [297x1 double]
            MatchX: [295x4 double]
           InsertX: [295x2 double]
           DeleteX: [295x2 double]
   FlankingInsertX: [2x2 double]
             LoopX: [2x2 double]
             NullX: [2x1 double]
```

See Also

gethmmalignment | hmmprofalign | hmmprofstruct | pfamhmmread |
showhmmprof

gethmmtree

Retrieve phylogenetic tree data from PFAM database

Syntax

Tree = gethmmtree(*PFAMName*)
Tree = gethmmtree(*PFAMAccessionNumber*)
Tree = gethmmtree(*PFAMNumber*)
Tree = gethmmtree(...'ToFile', *ToFileValue*, ...)
Tree = gethmmtree(...'Type', *TypeValue*, ...)

Input Arguments

PFAMName	String specifying a protein family name (unique identifier) of an HMM profile record in the PFAM database. For example, '7tm_2'.
PFAMAccessionNumber	String specifying a protein family accession number of an HMM profile record in the PFAM database. For example, 'PF00002'.
PFAMNumber	Integer specifying a protein family number of an HMM profile record in the PFAM database. For example, 2 is the protein family number for the protein family PF0002.
ToFileValue	Property to specify the location and file name for saving data. Enter either a file name or a path and file name supported by your system (ASCII text file).
TypeValue	String that specifies which alignments to include in the tree. Choices are: • 'seed' — Returns a tree with only the alignments used to generate the HMM model. • 'full' (default) — Returns a tree with all of the alignments that match the model.

	Warning Setting the `'Type'` name-value pair argument to `'seed'` or `'full'` is now ignored since the PFAM database no longer provides trees for the `full` alignment. This name-value pair argument will be removed in a future release. To download the `'seed'` tree use `gethmmtree` without any extra input arguments. To obtain the `'full'` tree you may use the `gethmmalignment` function to download the `'full'` alignment and build a tree using the `seqpdist` and `seqneighjoin` functions as illustrated in the following example.

Output Arguments

Tree	An object containing a phylogenetic tree representative of the protein family.

Description

Tree = gethmmtree(*PFAMName*) searches the PFAM database for the record represented by *PFAMName*, a protein family name, retrieves information, and returns `Tree`, an object containing a phylogenetic tree representative of the protein family.

Tree = gethmmtree(*PFAMAccessionNumber*) searches the PFAM database for the record represented by *PFAMAccessionNumber*, a protein family accession number, retrieves information, and returns `Tree`, an object containing a phylogenetic tree representative of the protein family.

Tree = gethmmtree(*PFAMNumber*) determines a protein family accession number from *PFAMNumber*, an integer, searches the PFAM database for the associated record, retrieves information, and returns `Tree`, an object containing a phylogenetic tree representative of the protein family.

Tree = gethmmtree(...'*PropertyName*', *PropertyValue*, ...) calls gethmmtree with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in

single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Tree = gethmmtree(...'ToFile', *ToFileValue*, ...) saves the data returned from the PFAM database in the file *ToFileValue*.

Tree = gethmmtree(...'Type', *TypeValue*, ...) specifies which alignments to include in the tree. Choices for *TypeValue* are:

- 'seed' — Returns a tree with only the alignments used to generate the HMM model.
- 'full' (default) — Returns a tree with all of the alignments that match the model.

Warning Setting the 'Type' name-value pair argument to 'seed' or 'full' is now ignored since the PFAM database no longer provides trees for the full alignment. This name-value pair argument will be removed in a future release. To download the 'seed' tree use gethmmtree without any extra input arguments. To obtain the 'full' tree you may use the gethmmalignment function to download the 'full' alignment and build a tree using the seqpdist and seqneighjoin functions as illustrated in the following example.

Examples

Retrieve phylogenetic tree built from the multiple-aligned sequences used to train the HMM profile model for global alignment. The PFAM accession number PF00002 is for the 7-transmembrane receptor protein in the secretin family.

```
tree  = gethmmtree('PF00002');
```

Recover the 'full' tree for the same family by downloading the full multiple sequence alignment and building the tree using the seqdist and seqneighjoin functions. It may take some considerable amount of time to calculate the tree for large families.

```
seqs = gethmmalignment('PF00002','type','full');
dis = seqpdist(seqs);
tree = seqneighjoin(dis,'equivar',seqs);
```

See Also
gethmmalignment | phytreeread

getIndex

Class: GFFAnnotation

Return index array of annotations from `GFFAnnotation` object

Syntax

```
Idx = getIndex(AnnotObj)
Idx = getIndex(AnnotObj,StartPos,EndPos)
Idx = getIndex( ___ ,Name,Value)
```

Description

`Idx = getIndex(AnnotObj)` returns an index array Idx, an array of integers containing the index of each annotation in AnnotObj.

`Idx = getIndex(AnnotObj,StartPos,EndPos)` returns an index array Idx for a subset of elements that falls within each reference sequence range specified by `StartPos` and `EndPos`.

`Idx = getIndex( ___ ,Name,Value)` returns an index array Idx, using any of the input arguments from the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

Input Arguments

AnnotObj

Object of the `GFFAnnotation` class.

Default:

StartPos

Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

Default:

EndPos

Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`'Reference'`

String or cell array of strings specifying one or more reference sequences in AnnotObj. Only indices of annotations whose reference field matches one of the strings are included in Idx.

Default:

`'Feature'`

String or cell array of strings specifying one or more features in AnnotObj. Only indices of annotations whose feature field matches one of the strings are included in Idx.

Default:

`'Overlap'`

Minimum number of base positions that an annotation must overlap in the range, to have its index included in Idx. This value can be any of the following:

- Positive integer
- `'full'` — An annotation must be fully contained in the range to be included.
- `'start'` — An annotation's start position must lie within the range to be included.

Default: 1

Output Arguments

`Idx`

Array of integers representing indices of elements in AnnotObj.

Examples

Retrieve Indices of Annotations from a GFFAnnotation Object

Construct a `GFFAnnotation` object using a GFF-formatted file that is provided with Bioinformatics Toolbox.

```
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
```

Extract indices of annotations or features for positions 10,000 through 20,000 from the reference sequence.

```
Idx = getIndex(GFFAnnotObj,10000,20000)

Idx =

    61
    62
    63
    64
    65
    66
    67
    68
    69
```

See Also
GTFAnnotation.getData

How To
·   "Store and Manage Feature Annotations in Objects"

Related Links
·   GFF (General Feature Format) specifications document

getIndex

Class: GTFAnnotation

Return index array of annotations from `GTFAnnotation` object

Syntax

```
Idx = getIndex(AnnotObj)
Idx = getIndex(AnnotObj,StartPos,EndPos)
Idx = getIndex( ___ ,Name,Value)
```

Description

`Idx = getIndex(AnnotObj)` returns an index array Idx, an array of integers containing the index of each annotation in AnnotObj.

`Idx = getIndex(AnnotObj,StartPos,EndPos)` returns an index array Idx for a subset of elements that falls within each reference sequence range specified by `StartPos` and `EndPos`.

`Idx = getIndex( ___ ,Name,Value)` returns an index array Idx, using any of the input arguments from the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

Input Arguments

AnnotObj

Object of the `GTFAnnotation` class.

Default:

StartPos

Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

Default:

`EndPos`

Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`'Reference'`

String or cell array of strings specifying one or more reference sequences in AnnotObj. Only indices of annotations whose reference field matches one of the strings are included in Idx.

Default:

`'Feature'`

String or cell array of strings specifying one or more features in AnnotObj. Only indices of annotations whose feature field matches one of the strings are included in Idx.

Default:

`'Gene'`

String or cell array of strings specifying one or more genes in AnnotObj. Only annotations whose gene field matches one of the strings are included in AnnotStruct.

`'Transcript'`

String or cell array of strings specifying one or more transcripts in AnnotObj. Only annotations whose transcript field matches one of the strings are included in AnnotStruct.

'Overlap'

Minimum number of base positions that an annotation must overlap in the range, to have its index included in Idx. This value can be any of the following:

- Positive integer
- 'full' — An annotation must be fully contained in the range to be included.
- 'start' — An annotation's start position must lie within the range to be included.

Default: 1

Output Arguments

Idx

Array of integers representing indices of elements in AnnotObj.

Examples

Retrieve Indices of Annotations from a GTFAnnotation Object

Construct a GTFAnnotation object using a GTF-formatted file that is provided with Bioinformatics Toolbox.

```
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
```

Extract indices of annotations for positions 210,000 through 220,000 from the reference sequence.

```
Idx = getIndex(GTFAnnotObj,210000,220000)

Idx =

     7
    15
    16
    17
    36
    47
    48
```

```
     49
     69
     70
     71
     89
     99
    111
    112
    113
```

See Also
`GTFAnnotation.getData`

How To
·   "Store and Manage Feature Annotations in Objects"

Related Links
·   GTF2.2: A Gene Annotation Format

getIndexByKey

Class: BioIndexedFile

Retrieve indices from source file associated with BioIndexedFile object using alphanumeric key

Syntax

Indices = getIndexByKey(*BioIFobj*, *Key*)
[*Indices*, *LogicalVals*] = getIndexByKey(*BioIFobj*, *Key*)

Description

Indices = getIndexByKey(*BioIFobj*, *Key*) returns the indices of entries in the source file associated with *BioIFobj*, a BioIndexedFile object. It returns the indices of entries that have the keys specified by *Key*, a string or cell array of strings specifying one or more alphanumeric keys. It returns *Indices*, a numeric vector of the indices of entries that have the alphanumeric keys specified by *Key*. If the keys in the source file are not unique, it returns all indices of entries that match a specified key, all at the position of the key in the *Key* cell array. If the keys in the source file are unique, there is a one-to-one relationship between the number and order of elements in *Key* and the output *Indices*.

[*Indices*, *LogicalVals*] = getIndexByKey(*BioIFobj*, *Key*) returns a logical vector that indicates only the last match for each key, such that there is a one-to-one relationship between the number and order of elements in *Key* and *Indices*(*LogicalVals*).

Tips

Use this method to determine the indices of specific entries with known keys.

Input Arguments

BioIFobj

Object of the `BioIndexedFile` class.

Default:

Key

String or cell array of strings specifying one or more keys in the source file associated with *BioIFobj*, the BioIndexedFile object.

Default:

Output Arguments

Indices

Numeric vector of the indices of entries in source file that have the alphanumeric keys specified by *Key*.

LogicalVals

Logical vector containing the same number of elements as *Indices*. The vector indicates only the last match for each key specified in *Key*, such that there is a one-to-one relationship between the number and order of elements in *Key* and *Indices(LogicalVals)*.

Tip Some files contain repeated keys. For example, SAM-formatted files use the same key for entries that are paired end reads. Use the *Indices(LogicalVals)* syntax to return only the last index of a repeated key. For more information, see "Examples" on page 1-796.

Examples

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

```
% Create variable containing full absolute path of source file
sourcefile = which('yeastgenes.sgd');
% Create a BioIndexedFile object from the source file. Indicate
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with !
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
                            'KeyColumn', 3, 'HeaderPrefix','!')
```

Return the indices for the entries in the source file that are specified by the keys AAC1 and AAD10.

```
% Access indices for entries that have the keys AAC1 and AAD10
indices = getIndexByKey(gene2goObj, {'AAC1' 'AAD10'})

indices =

        3
    5
```

Construct a BioIndexedFile object to access a SAM-formatted file that has repeated keys.

```
% Create variable containing full absolute path of source file
samsourcefile = which('ex1.sam');
% Create a BioIndexedFile object from the source file. Store the
% index file in the Current Folder.
samObj = BioIndexedFile('sam', samsourcefile, '.')
```

Return only the last indices for the entries in the source file that are specified by two keys,'B7_593:7:15:244:876 and EAS56_65:4:296:78:421, both of which are repeated keys.

```
% Return all indices for entries that have two specific keys
[Indices, LogicalVal] = getIndexByKey(samObj, ...
                {'B7_593:7:15:244:876', 'EAS56_65:4:296:78:421'})

Indices =

        3058
        3238
        3292
        3293

LogicalVal =
```

```
       0
       1
       0
       1

% Return only the last index for each key
LastIndices = Indices(LogicalVal)

LastIndices =

       3238
       3293
```

See Also
`BioIndexedFile.getEntryByKey` | `BioIndexedFile.getKeys` | `BioIndexedFile` | `BioIndexedFile.getSubset`

How To
- "Work with Large Multi-Entry Text Files"

getKeys

Class: BioIndexedFile

Retrieve alphanumeric keys from source file associated with BioIndexedFile object

Syntax

Keys = getKeys(*BioIFobj*)

Description

Keys = getKeys(*BioIFobj*) returns *Keys*, a cell array of strings specifying all the keys to the entries in the source file associated with *BioIFobj*, a BioIndexedFile object. The keys appear in the same order as they do in the source file, even if they are not unique.

Tips

Use this method to see a complete list of the alphanumeric keys, in the order they occur in the source file from which the BioIndexedFile object was created.

Input Arguments

BioIFobj

Object of the `BioIndexedFile` class.

Default:

Output Arguments

Keys

Cell array of strings specifying all the keys to the entries in the source file. The keys appear in the same order as they do in the source file, even if they are not unique.

Examples

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

```
% Create variable containing full absolute path of source file
sourcefile = which('yeastgenes.sgd');
% Create a BioIndexedFile object from the source file. Indicate
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with !
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
                            'KeyColumn', 3, 'HeaderPrefix','!')
```

Retrieve all the keys for the entries in the source file, then view the first 12 keys:

```
% Retrieve all keys for entries in gene2goObj
keys = getKeys(gene2goObj);
% View the first 12 keys
keys(1:12)

ans =

    '15S_RRNA'
    '21S_RRNA'
    'AAC1'
    'AAC3'
    'AAD10'
    'AAD14'
    'AAD15'
    'AAD16'
    'AAD3'
    'AAD4'
    'AAD6'
    'AAH1'
```

See Also

BioIndexedFile.getEntryByKey | BioIndexedFile.getIndexByKey | BioIndexedFile | BioIndexedFile.getSubset

How To

- "Work with Large Multi-Entry Text Files"

getmatrix (biograph)

Get connection matrix from biograph object

Syntax

[*Matrix, ID, Distances*] = getmatrix(*BGObj*)

Arguments

BGObj	Biograph object created by biograph (object constructor).

Description

[*Matrix, ID, Distances*] = getmatrix(*BGObj*) converts the biograph object, *BiographObj*, into a logical sparse matrix, *Matrix*, in which 1 indicates that a node (row index) is connected to another node (column index). *ID* is a cell array of strings listing the ID properties for each node, and corresponds to the rows and columns of *Matrix*. *Distances* is a column vector with one entry for every nonzero entry in *Matrix* traversed column-wise and representing the respective Weight property for each edge.

Examples

```
cm = [0 1 1 0 0;2 0 0 4 4;4 0 0 0 0;0 0 0 0 2;4 0 5 0 0];
bg = biograph(cm);
[cm, IDs, dist] = getmatrix(bg)
```

More About

- biograph object

See Also

biograph | getancestors | getedgesbynodeid | getrelatives | dolayout | getdescendants | getnodesbyid | view

getmatrix

Class: geneont

Convert geneont object into relationship matrix

Syntax

[*Matrix*, *ID*, *Relationship*] = getmatrix(*GeneontObj*)

Description

[*Matrix*, *ID*, *Relationship*] = getmatrix(*GeneontObj*) converts a geneont object, *GeneontObj*, into *Matrix*, a matrix of relationship values between nodes (row and column indices), in which 0 indicates no relationship, 1 indicates an "is_a" relationship, and 2 indicates a "part_of" relationship. *ID* is a column vector listing Gene Ontology IDs that correspond to the rows and columns of *Matrix*. *Relationship* is a cell array of strings defining the types of relationships.

Input Arguments

GeneontObj	A geneont object, such as created by the geneont constructor function.

Output Arguments

Matrix	Matrix of relationship values between nodes (row and column indices), in which 0 indicates no relationship, 1 indicates an "is_a" relationship, and 2 indicates a "part_of" relationship.
ID	Column vector listing Gene Ontology IDs that correspond to the rows and columns of *Matrix*.
Relationship	Cell array of strings defining the types of relationships.

Examples

1   Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GO = geneont('LIVE',true)
```

The MATLAB software creates a geneont object and displays the number of terms in the database.

```
Gene Ontology object with 27595 Terms.
```

2   Convert this geneont object into a relationship matrix.

```
[MATRIX, ID, REL] = getmatrix(GO);
```

See Also
goannotread | num2goid | term

getmatrix (phytree)

Convert phytree object into relationship matrix

Syntax

[*Matrix, ID, Distances*] = getmatrix(*PhytreeObj*)

Arguments

PhytreeObj	phytree object created by phytree (object constructor).

Description

[*Matrix, ID, Distances*] = getmatrix(*PhytreeObj*) converts a phytree object, *PhytreeObj*, into a logical sparse matrix, *Matrix*, in which 1 indicates that a branch node (row index) is connected to its child (column index). The child can be either another branch node or a leaf node. *ID* is a column vector of strings listing the labels that correspond to the rows and columns of *Matrix*, with the labels from 1 to *Number of Leaves* being the leaf nodes, then the labels from *Number of Leaves* + 1 to *Number of Leaves* + *Number of Branches* being the branch nodes, and the label for the last branch node also being the root node. *Distances* is a column vector with one entry for every nonzero entry in *Matrix* traversed column-wise and representing the distance between the branch node and the child.

Examples

```
T = phytreeread('pf00002.tree')
[MATRIX, ID, DIST] = getmatrix(T);
```

More About

· phytree object

See Also

phytree | pdist | prune | phytreeviewer | get

getnewickstr (phytree)

Create Newick-formatted string

Syntax

```
String = getnewickstr(Tree)
getnewickstr(..., 'PropertyName', PropertyValue,...)
getnewickstr(..., 'Distances', DistancesValue)
getnewickstr(..., 'BranchNames', BranchNamesValue)
```

Arguments

Tree	Phytree object created with the function `phytree`.
DistancesValue	Property to control including or excluding distances in the output. Enter either `true` (include distances) or `false` (exclude distances). Default is `true`.
BranchNamesValue	Property to control including or excluding branch names in the output. Enter either `true` (include branch names) or `false` (exclude branch names). Default is `false`.

Description

String = getnewickstr(*Tree*) returns the Newick formatted string of a phylogenetic tree object (*Tree*).

getnewickstr(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

getnewickstr(..., 'Distances', *DistancesValue*), when *DistancesValue* is `false`, excludes the distances from the output.

getnewickstr(..., 'BranchNames', *BranchNamesValue*), when *BranchNamesValue* is `true`, includes the branch names in the output.

Examples

1   Create some random sequences.

    ```
    seqs = int2nt(ceil(rand(10)*4));
    ```
2   Calculate pairwise distances.

    ```
    dist = seqpdist(seqs,'alpha','nt');
    ```
3   Construct a phylogenetic tree.

    ```
    tree = seqlinkage(dist);
    ```
4   Get the Newick string.

    ```
    str  = getnewickstr(tree)
    ```

More About

·   phytree object

References

Information about the Newick tree format.

http://evolution.genetics.washington.edu/phylip/newicktree.html

See Also
phytree | get | getcanonical | phytreeread | phytreeviewer | phytreewrite
| seqlinkage | getbyname

getnodesbyid (biograph)

Get handles to nodes

Syntax

NodesHandles = getnodesbyid(*BGobj,NodeIDs*)

Arguments

BGobj	Biograph object.
NodeIDs	Enter a cell string of node identifications.

Description

NodesHandles = getnodesbyid(*BGobj,NodeIDs*) gets the handles for the specified nodes (*NodeIDs*) in a biograph object.

Examples

1 Create a biograph object.

```
species = {'Homosapiens','Pan','Gorilla','Pongo','Baboon',...
          'Macaca','Gibbon'};
cm = magic(7)>25 & 1-eye(7);
bg = biograph(cm, species)
```

2 Find the handles to members of the Cercopithecidae family and members of the Hominidae family.

```
Cercopithecidae = {'Macaca','Baboon'};
Hominidae = {'Homosapiens','Pan','Gorilla','Pongo'};
CercopithecidaeNodes = getnodesbyid(bg,Cercopithecidae);
HominidaeNodes = getnodesbyid(bg,Hominidae);
```

3 Color the families differently and draw a graph.

More About

•   biograph object

See Also

biograph | get | getdescendants | getnodesbyid | set | view | dolayout |
getancestors | getedgesbynodeid | getrelatives

getpdb

Retrieve protein structure data from Protein Data Bank (PDB) database

Syntax

PDBStruct = getpdb(*PDBid*)
PDBStruct = getpdb(*PDBid*, ...'ToFile', *ToFileValue*, ...)
PDBStruct = getpdb(*PDBid*, ...'SequenceOnly', *SequenceOnlyValue*, ...)

Input Arguments

PDBid	String specifying a unique identifier for a protein structure record in the PDB database.
	Note: Each structure in the PDB database is represented by a four-character alphanumeric identifier. For example, 4hhb is the identifier for hemoglobin.
ToFileValue	String specifying a file name or a path and file name for saving the PDB-formatted data. If you specify only a file name, that file will be saved in the MATLAB Current Folder.
	Tip After you save the protein structure record to a local PDB-formatted file, you can use the pdbread function to read the file into the MATLAB software offline or use the molviewer function to display and manipulate a 3-D image of the structure.
SequenceOnlyValue	Controls the return of the protein sequence only. Choices are true or false (default).
	If there is one sequence, it is returned as a character array. If there are multiple sequences, they are returned as a cell array.

Output Arguments

PDBStruct	MATLAB structure containing a field for each PDB record.

Description

The Protein Data Bank (PDB) database is an archive of experimentally determined 3-D biological macromolecular structure data. For more information about the PDB format, see:

http://www.wwpdb.org/documentation/format23/v2.3.html

getpdb retrieves protein structure data from the Protein Data Bank (PDB) database, which contains 3-D biological macromolecular structure data.

PDBStruct = getpdb(*PDBid*) searches the PDB database for the protein structure record specified by the identifier *PDBid* and returns the MATLAB structure *PDBStruct*, which contains a field for each PDB record. The following table summarizes the possible PDB records and the corresponding fields in the MATLAB structure *PDBStruct*:

PDB Database Record	Field in the MATLAB Structure
HEADER	Header
OBSLTE	Obsolete
TITLE	Title
CAVEAT	Caveat
COMPND	Compound
SOURCE	Source
KEYWDS	Keywords
EXPDTA	ExperimentData
AUTHOR	Authors
REVDAT	RevisionDate
SPRSDE	Superseded
JRNL	Journal
REMARK 1	Remark1

PDB Database Record	Field in the MATLAB Structure
REMARK *N* **Note:** *N* equals 2 through 999.	Remark*n* **Note:** *n* equals 2 through 999.
DBREF	DBReferences
SEQADV	SequenceConflicts
SEQRES	Sequence
FTNOTE	Footnote
MODRES	ModifiedResidues
HET	Heterogen
HETNAM	HeterogenName
HETSYN	HeterogenSynonym
FORMUL	Formula
HELIX	Helix
SHEET	Sheet
TURN	Turn
SSBOND	SSBond
LINK	Link
HYDBND	HydrogenBond
SLTBRG	SaltBridge
CISPEP	CISPeptides
SITE	Site
CRYST1	Cryst1
ORIGXn	OriginX
SCALEn	Scale
MTRIXn	Matrix
TVECT	TranslationVector
MODEL	Model

PDB Database Record	Field in the MATLAB Structure
ATOM	Atom
SIGATM	AtomSD
ANISOU	AnisotropicTemp
SIGUIJ	AnisotropicTempSD
TER	Terminal
HETATM	HeterogenAtom
CONECT	Connectivity

PDBStruct = getpdb(*PDBid*, ...'*PropertyName*', *PropertyValue*, ...) calls getpdb with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

PDBStruct = getpdb(*PDBid*, ...'ToFile', *ToFileValue*, ...) saves the data returned from the database to a PDB-formatted file, *ToFileValue*.

Tip After you save the protein structure record to a local PDB-formatted file, you can use the pdbread function to read the file into the MATLAB software offline or use the molviewer function to display and manipulate a 3-D image of the structure.

PDBStruct = getpdb(*PDBid*, ...'SequenceOnly', *SequenceOnlyValue*, ...) controls the return of the protein sequence only. Choices are true or false (default). If there is one sequence, it is returned as a character array. If there are multiple sequences, they are returned as a cell array.

The Sequence Field

The Sequence field is also a structure containing sequence information in the following subfields:

- NumOfResidues
- ChainID
- ResidueNames — Contains the three-letter codes for the sequence residues.

- `Sequence` — Contains the single-letter codes for the sequence residues.

Note: If the sequence has modified residues, then the `ResidueNames` subfield might not correspond to the standard three-letter amino acid codes. In this case, the `Sequence` subfield will contain the modified residue code in the position corresponding to the modified residue. The modified residue code is provided in the `ModifiedResidues` field.

The Model Field

The `Model` field is also a structure or an array of structures containing coordinate information. If the MATLAB structure contains one model, the `Model` field is a structure containing coordinate information for that model. If the MATLAB structure contains multiple models, the `Model` field is an array of structures containing coordinate information for each model. The `Model` field contains the following subfields:

- `Atom`
- `AtomSD`
- `AnisotropicTemp`
- `AnisotropicTempSD`
- `Terminal`
- `HeterogenAtom`

The Atom Field

The `Atom` field is also an array of structures containing the following subfields:

- `AtomSerNo`
- `AtomName`
- `altLoc`
- `resName`
- `chainID`
- `resSeq`
- `iCode`
- `X`

- Y
- Z
- occupancy
- tempFactor
- segID
- element
- charge
- AtomNameStruct — Contains three subfields: chemSymbol, remoteInd, and branch.

Examples

Retrieve the structure information for the electron transport (heme) protein that has a PDB identifier of 5CYT, read the information into a MATLAB structure pdbstruct, and save the information to a PDB-formatted file electron_transport.pdb in the MATLAB Current Folder.

```
pdbstruct = getpdb('5CYT', 'ToFile', 'electron_transport.pdb')
```

See Also

getembl | getgenbank | getgenpept | molviewer | pdbdistplot | pdbread | pdbsuperpose | pdbtransform | pdbwrite

getQuality

Class: BioRead

Retrieve sequence quality scores from object

Syntax

```
Quality = getQuality(BioObj)
Quality = getQuality(BioObj, Subset)
```

Description

Quality = getQuality(*BioObj*) returns *Quality*, a cell array of strings containing the ASCII representations of per-base quality scores for nucleotide sequences from an object.

Quality = getQuality(*BioObj*, *Subset*) returns quality strings for only object elements specified by *Subset*.

Input Arguments

BioObj

Object of the `BioRead` or `BioMap` class.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

`Quality`

`Quality` property of a subset of elements in *BioObj*.*Quality* is a cell array of strings containing the quality scores for sequences specified by *Subset*.

Examples

Retrieve the quality scores from different elements of a `BioRead` object:

```
% Construct a BioRead object from a FASTQ file
BRObj = BioRead('SRR005164_1_50.fastq');
% Retrieve the Quality value of the second element in the object
getQuality(BRObj, 2);
getQuality(BRObj, [false true false]);
% Retrieve the Quality values of the first and third elements
% in the object
getQuality(BRObj, [1 3]);
getQuality(BRObj, [true false true]);
% Retrieve the Quality value of all the elements in the object
getQuality(BRObj, 1:50);
```

Alternatives

An alternative to using the `getQuality` method is to use dot indexing with the `Quality` property:

BioObj.Quality(*Indices*)

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers.

See Also
setQuality | BioRead | BioMap

How To
- "Manage Short-Read Sequence Data in Objects"

Related Links
- Sequence Read Archive
- SAM format specification

getRange

Class: GFFAnnotation

Retrieve range of annotations from `GFFAnnotation` object

Syntax

```
Range = getRange(AnnotObj)
```

Description

Range = getRange(AnnotObj) returns Range, a 1-by-2 numeric array specifying the minimum and maximum positions in the reference sequence covered by annotations in AnnotObj.

Tips

- Use the `GFFAnnotation.getSubset` method with the `Reference` name-value pair to return a `GFFAnnotation` object containing only one reference sequence. Then use this subsetted object as input to the `getRange` method.

Input Arguments

AnnotObj

Object of the `GFFAnnotation` class.

Default:

Output Arguments

Range

1-by-2 numeric array specifying the minimum and maximum positions in the reference sequence covered by annotations in AnnotObj

Examples

Construct a `GFFAnnotation` object from a GFF-formatted file that is provided with Bioinformatics Toolbox, and then return the range of the feature annotations:

```
% Construct a GFFAnnotation object from a GFF file
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
% Return first and last positions of reference associated with feature annotations
range = getRange(GFFAnnotObj)

range =

      3631      498516
```

See Also
`GFFAnnotation.getSubset` | `GTFAnnotation.getRange`

How To
- "Store and Manage Feature Annotations in Objects"

Related Links
- GFF (General Feature Format) specifications document

getRange

Class: GTFAnnotation

Retrieve range of annotations from `GTFAnnotation` object

Syntax

```
Range = getRange(AnnotObj)
```

Description

`Range = getRange(AnnotObj)` returns Range, a 1-by-2 numeric array specifying the minimum and maximum positions in the reference sequence covered by annotations in AnnotObj.

Tips

- Use the `GTFAnnotation.getSubset` method with the `Reference` name-value pair to return a `GFFAnnotation` object containing only one reference sequence. Then use this subsetted object as input to the `getRange` method.

Input Arguments

AnnotObj

Object of the `GTFAnnotation` class.

Default:

Output Arguments

Range

1-by-2 numeric array specifying the minimum and maximum positions in the reference sequence covered by annotations in AnnotObj

Examples

Construct a GTFAnnotation object from a GTF-formatted file that is provided with Bioinformatics Toolbox, and then return the range of the feature annotations:

```
% Construct a GTFAnnotation object from a GTF file
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
% Return first and last positions of reference associated with feature annotations
range = getRange(GTFAnnotObj)

range =

     41609    1371382
```

See Also

GTFAnnotation.getSubset | GFFAnnotation.getRange

How To

· "Store and Manage Feature Annotations in Objects"

Related Links

· GTF2.2: A Gene Annotation Format

getReferenceNames

Class: GFFAnnotation

Retrieve reference names from `GFFAnnotation` object

Syntax

```
References = getReferenceNames(AnnotObj)
```

Description

`References = getReferenceNames(AnnotObj)` returns References, a cell array of strings specifying the names of all reference sequences in AnnotObj.

Input Arguments

AnnotObj

Object of the `GFFAnnotation` class.

Default:

Output Arguments

References

Cell array of strings specifying the names of all reference sequences in AnnotObj.

Examples

Construct a `GFFAnnotation` object from a GFF-formatted file that is provided with Bioinformatics Toolbox, and then return the names of the reference sequences from the annotation object:

```
% Construct a GFFAnnotation object from a GFF file
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
% Return reference names for the annotation object
refNames = getReferenceNames(GFFAnnotObj)

refNames =

    'Chr1'
```

See Also
GTFAnnotation.getReferenceNames

How To
·   "Store and Manage Feature Annotations in Objects"

Related Links
·   GFF (General Feature Format) specifications document

getReferenceNames

Class: GTFAnnotation

Retrieve reference names from `GTFAnnotation` object

Syntax

```
References = getReferenceNames(AnnotObj)
```

Description

`References = getReferenceNames(AnnotObj)` returns References, a cell array of strings specifying the names of all reference sequences in AnnotObj.

Input Arguments

AnnotObj

Object of the `GTFAnnotation` class.

Default:

Output Arguments

References

Cell array of strings specifying the names of all reference sequences in AnnotObj

Examples

Construct a `GTFAnnotation` object from a GTF-formatted file that is provided with Bioinformatics Toolbox, and then return the names of the reference sequences from the annotation object:

1-825

```
% Construct a GTFAnnotation object from a GTF file
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
% Return reference names for the annotation object
refNames = getReferenceNames(GTFAnnotObj)

refNames =

    'chr2'
```

See Also

GFFAnnotation.getReferenceNames

How To

- "Store and Manage Feature Annotations in Objects"

Related Links

- GTF2.2: A Gene Annotation Format

getrelatives (biograph)

Find relatives in biograph object

Syntax

Nodes = getrelatives(*BiographNode*)
Nodes = getrelatives(*BiographNode,NumGenerations*)

Arguments

BiographNode	Node in a biograph object.
NumGenerations	Number of generations. Enter a positive integer.

Description

Nodes = getrelatives(*BiographNode*) finds all the direct relatives for a given node (*BiographNode*).

Nodes = getrelatives(*BiographNode,NumGenerations*) finds the direct relatives for a given node (*BiographNode*) up to a specified number of generations (*NumGenerations*).

Examples

1   Create a biograph object.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg = biograph(cm)
```

2   Find all nodes interacting with node 1.

```
intNodes = getrelatives(bg.nodes(1));
set(intNodes,'Color',[.7 .7 1]);
bg.view;
```

More About

- biograph object

See Also

```
biograph | get | getdescendants | getnodesbyid | set | view | dolayout |
getancestors | getedgesbynodeid | getrelatives
```

getrelatives

Class: geneont

Find terms that are relatives of specified Gene Ontology (GO) term

Syntax

```
RelativeIDs = getrelatives(GeneontObj, ID)
[RelativeIDs, Counts] = getrelatives(GeneontObj, ID)
... = getrelatives(..., 'Height', HeightValue, ...)
... = getrelatives(..., 'Depth', DepthValue, ...)
... = getrelatives(..., 'Levels', LevelsValue, ...)
... = getrelatives(..., 'Relationtype', RelationtypeValue, ...)
... = getrelatives(..., 'Exclude', ExcludeValue, ...)
```

Description

RelativeIDs = getrelatives(*GeneontObj*, *ID*) searches *GeneontObj*, a geneont object, for GO terms that are relatives of the GO term(s) specified by *ID*, which is a GO term identifier or vector of identifiers. It returns *RelativeIDs*, a vector of GO term identifiers including *ID*. *ID* is a nonnegative integer or a vector containing nonnegative integers.

[*RelativeIDs*, *Counts*] = getrelatives(*GeneontObj*, *ID*) also returns the number of times each relative is found. *Counts* is a column vector with the same number of elements as terms in *GeneontObj*.

Tip The *Counts* return value is useful when you tally counts in gene enrichment studies. For more information, see Gene Ontology Enrichment in Microarray Data.

... = getrelatives(..., '*PropertyName*', *PropertyValue*, ...) calls getrelatives with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = getrelatives(..., 'Height', *HeightValue*, ...) searches up through a specified number of levels, *HeightValue*, in the gene ontology. *HeightValue* is a positive integer. Default is 1.

... = getrelatives(..., 'Depth', *DepthValue*, ...) searches down through a specified number of levels, *DepthValue*, in the gene ontology. *DepthValue* is a positive integer. Default is 1.

... = getrelatives(..., 'Levels', *LevelsValue*, ...) searches up and down through a specified number of levels, *LevelsValue*, in the gene ontology. *LevelsValue* is a positive integer. When specified, it overrides *HeightValue* and *DepthValue*.

... = getrelatives(..., 'Relationtype', *RelationtypeValue*, ...) searches for specified relationship types, *RelationtypeValue*, in the gene ontology. *RelationtypeValue* is a string. Choices are 'is_a', 'part_of', or 'both' (default).

... = getrelatives(..., 'Exclude', *ExcludeValue*, ...) controls excluding *ID*, the original queried term(s), from the output *RelativeIDs*, unless a term was found while searching the gene ontology. Choices are true or false (default).

Input Arguments

GeneontObj	A geneont object, such as created by the geneont constructor function.
ID	GO term identifier or vector of identifiers.
HeightValue	Positive integer specifying the number of levels to search upward in the gene ontology.
DepthValue	Positive integer specifying the number of levels to search downward in the gene ontology.
LevelsValue	Positive integer specifying the number of levels up and down to search in the gene ontology. When specified, it overrides *HeightValue* and *DepthValue*.
RelationtypeValue	String specifying the relationship types to search for in the gene ontology. Choices are: • 'is_a'

	• `'part_of'`
	• `'both'` (default)
ExcludeValue	Controls excluding *ID*, the original queried term(s), from the output *RelativeIDs*, unless the term was reached while searching the gene ontology. Choices are `true` or `false` (default).

Output Arguments

RelativeIDs	Vector of GO term identifiers including *ID*.
Counts	Column vector with the same number of elements as terms in *GeneontObj*, indicating the number of times each relative is found.

Examples

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GO = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of terms in the database.

```
Gene Ontology object with 27769 Terms.
```

2 Retrieve the immediate relatives for the mitochondrial membrane GO term with a GO identifier of 31966.

```
relatives = getrelatives(GO,31966,'levels',1)

relatives =

      5741
      5743
     31090
     31966
     44429
```

3 Create a subordinate Gene Ontology.

```
subontology = GO(relatives)

Gene Ontology object with 5 Terms.
```

4   Create a report of the subordinate Gene Ontology terms, that includes the GO identifier and name.

```
rpt = get(subontology.terms,{'id','name'})

rpt =

    [ 5741]              [1x28 char]
    [ 5743]              [1x28 char]
    [31090]     'organelle membrane'
    [31966]              [1x22 char]
    [44429]     'mitochondrial part'
```

5   View relationships of the subordinate Gene Ontology by using the `getmatrix` method to create a connection matrix to pass to the `biograph` function, and color the mitochondrial membrane GO term red.

```
[cm acc rels] = getmatrix(subontology);
BG = biograph(cm, get(subontology.terms, 'name'));
BG.nodes(acc==31966).Color = [1 0 0];
view(BG)
```
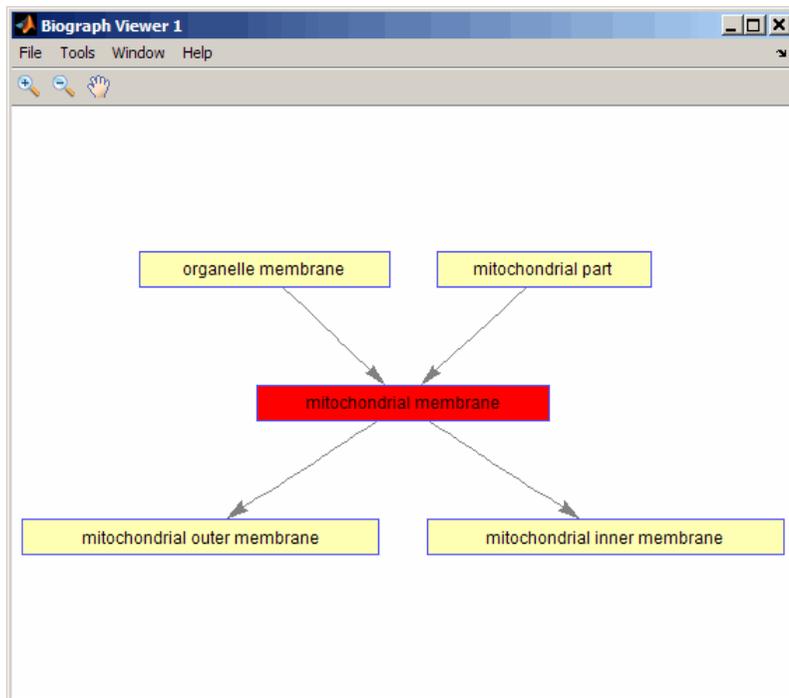
6 Retrieve all relatives for the mithocondrial outer membrane GO term with an identifier of `5741`.

```
relatives = getrelatives(GO,5741,'levels',inf);
```

7 Create a subordinate Gene Ontology.

```
subontology = GO(relatives)

Gene Ontology object with 13 Terms.
```

8 View relationships of the subordinate Gene Ontology by using the `getmatrix` method to create a connection matrix to pass to the `biograph` function and methods, and color the mitochondrial outer membrane GO terms red.

```
[cm acc rels] = getmatrix(subontology);
BG = biograph(cm, get(subontology.terms, 'name'));
BG.nodes(acc==5741).Color = [1 0 0];
view(BG)
```
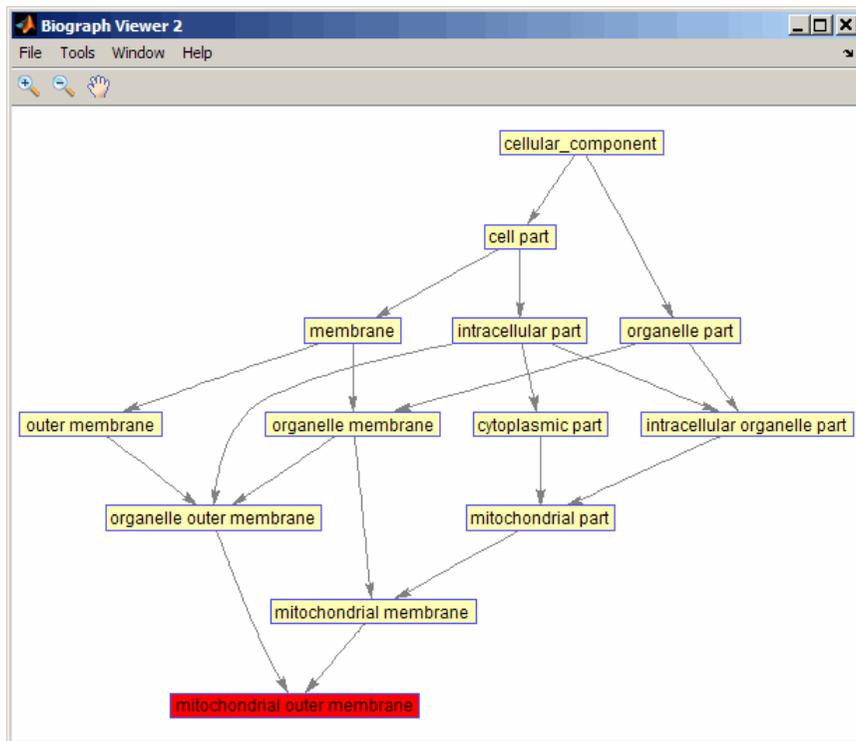
See Also

goannotread | num2goid | term

getSegments

Class: GTFAnnotation

Return table of non-overlapping segments from `GTFAnnotation` object

Syntax

```
segments = getSegments(AnnotObj)
[segments,transcriptIDs] = getSegments(AnnotObj)
[ ___ ] = getSegments(AnnotObj,'Reference',R)
[ ___ ] = getSegments(AnnotObj,'Gene',G)
[ ___ ] = getSegments(AnnotObj,'Transcript',T)
```

Description

`segments = getSegments(AnnotObj)` returns segments, a table of non-overlapping segments of nucleotide sequences built by flattening the transcripts in AnnotObj. If an exon boundary is not the same in two or more transcripts of a gene, then the function creates two or more non-overlapping segments which cover all exons in the transcript.

`[segments,transcriptIDs] = getSegments(AnnotObj)` returns transcriptIDs, a cell array of strings containing all unique transcript IDs in AnnotObj.

`[ ___ ] = getSegments(AnnotObj,'Reference',R)` returns the segments that belong to the reference(s) specified by R.

`[ ___ ] = getSegments(AnnotObj,'Gene',G)` returns the segments that belong to the gene(s) specified by G.

`[ ___ ] = getSegments(AnnotObj,'Transcript',T)` returns the segments that belong to the transcript(s) specified by T.

Input Arguments

AnnotObj — GTF annotation
GTFAnnotation object

GTF annotation, specified as a `GTFAnnotation` object.

R — Names of reference sequences
string | cell array of strings | categorical array

Names of reference sequences, specified as a string, cell array of strings, or categorical array.

The names must come from the `Reference` field of AnnotObj. If a name does not exist, the function provides a warning and ignores it.

G — Names of gene
string | cell array of strings | categorical array

Names of genes, specified as a string, cell array of strings, or categorical array.

The names must come from the `Gene` field of AnnotObj. If a name does not exist, the function provides a warning and ignores the name.

T — Names of transcripts
string | cell array of strings | categorical array

Names of transcripts, specified as a string, cell array of strings, or categorical array.

The names must come from the `Transcript` field of AnnotObj. If a name does not exist, the function gives a warning and ignores the name.

Output Arguments

segments — Non-overlapping segments
table

Non-overlapping segments, returned as a table. The table contains the following variables for each segments.

Variable Name	Description
Start	Start location of each segment.
Stop	Stop location of each segment.

Variable Name	Description
Reference	Categorical array representing the names of reference sequences to which the segments belong, obtained from the `Reference` field of AnnotObj.
ExonIndicator	Logical sparse matrix of segment versus exon. The rows represent segments. The columns are exons. If the ith segment is part of the jth exon, the element at position (i,j) is 1. Otherwise, it is 0.
TranscriptInd	Logical sparse matrix of segment versus transcript. The rows represent segments and the columns are transcripts. The element at position (i,j) is 1 if the ith segment is part of the jth transcript, and 0 otherwise.

`transcriptIDs` — Unique transcript IDs
cell array of strings

Unique transcript IDs, returned as a cell array of strings. The transcript IDs correspond to columns of the `TranscriptIndicator` variable of segments. For instance, the first element of transcriptIDs is the ID of the first column of `TranscriptIndicator` matrix.

Examples

Retrieve Segments from a GTF-formatted File

Create a GTFAnnotation object from a GTF-formatted file.

```
obj = GTFAnnotation('hum37_2_1M.gtf');
```

Retrieve unique reference names. In this case, there is only one reference sequence, which is chromosome 2 (`chr2`).

```
ref = getReferenceNames(obj)


ref =

    'chr2'
```

Get a table of all non-overlapping segments of nucleotide sequences which belong to `chr2`.

```
segments = getSegments(obj,'Reference',ref);
```

See Also

GTFAnnotation | GTFAnnotation.getData | GTFAnnotation.getFeatureNames
| GTFAnnotation.getGeneNames | GTFAnnotation.getIndex |
GTFAnnotation.getRange | GTFAnnotation.getReferenceNames
| GTFAnnotation.getSubset | GTFAnnotation.getGenes |
GTFAnnotation.getTranscripts | GTFAnnotation.getExons

More About

· "Store and Manage Feature Annotations in Objects"

External Web Sites

· GTF2.2: A Gene Annotation Format

getSequence

Class: BioRead

Retrieve sequences from object

Syntax

```
Sequences = getSequence(BioObj)
Sequences = getSequence(BioObj, Subset)
```

Description

Sequences = getSequence(*BioObj*) returns *Sequences*, a cell array of strings containing the letter representations of nucleotide sequences from an object.

Sequences = getSequence(*BioObj*, *Subset*) returns sequence strings for only object elements specified by *Subset*.

Input Arguments

BioObj

Object of the `BioRead` or `BioMap` class.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

`Sequences`

Cell array of strings containing the sequences specified by *Subset* in *BioObj*.

Examples

Retrieve the sequences from different elements of a `BioRead` object:

```
% Construct a BioRead object from a FASTQ file
BRObj = BioRead('SRR005164_1_50.fastq');
% Retrieve the sequence of the second element in the object
getSequence(BRObj, 2);
getSequence(BRObj, {'SRR005164.2'});
getSequence(BRObj, [false true false]);
% Retrieve the sequences of the first and third elements in the object
getSequence(BRObj, [1 3]);
getSequence(BRObj, {'SRR005164.1', 'SRR005164.3'});
getSequence(BRObj, [true false true]);
% Retrieve the sequences of all the elements in the object
getSequence(BRObj, 1:50);
```

Alternatives

An alternative to using the `getSequence` method is to use dot indexing with the `Sequence` property:

BioObj.Sequence(*Indices*)

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers.

See Also
setSequence | BioRead | BioMap

How To

- "Manage Short-Read Sequence Data in Objects"

Related Links

- Sequence Read Archive
- SAM format specification

getSubsequence

Class: BioRead

Retrieve partial sequences from object

Syntax

Subsequences = getSubsequence(*BioObj*, *Subset*, *Positions*)

Description

Subsequences = getSubsequence(*BioObj*, *Subset*, *Positions*) returns *Subsequences*, a cell array of strings containing the letter representations of partial nucleotide sequences from an object. getSubsequence returns sequence strings for only object elements specified by *Subset* and for only sequence positions specified by *Positions*.

Input Arguments

BioObj

Object of the BioRead or BioMap class.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Positions

One of the following to specify a subset of positions in a sequence:

- Vector of positive integers
- Logical vector

Note: The last position specified by *Positions* must be within the range of positions for each sequence specified by *Subset*.

Default:

Output Arguments

SubSequences

Cell array of strings containing the partial sequences specified by *Subset* and *Positions* in *BioObj*.

Examples

Retrieve subsequences from different elements of a `BioRead` object:

```
% Construct a BioRead object from a FASTQ file
BRObj = BioRead('SRR005164_1_50.fastq');
% Retrieve the first five positions of the first three sequences in
% the object
getSubsequence(BRObj, 1:3, 1:5)
% Retrieve the first five positions of the sequence with a header
% of SRR005164.3
getSubsequence(BRObj, 'SRR005164.3', 1:5)
```

See Also
`BioRead` | `BioMap` | `setSubsequence`

How To

- "Manage Short-Read Sequence Data in Objects"

Related Links

- Sequence Read Archive
- SAM format specification

getSubset

Class: BioIndexedFile

Create object containing subset of elements from BioIndexedFile object

Syntax

```
NewObj = getSubset(BioIFObj, Indices)
NewObj = getSubset(BioIFObj, Keys)
```

Description

NewObj = getSubset(*BioIFObj*, *Indices*) returns *NewObj*, a new BioIndexedFile object that accesses a subset of entries in the source file associated with *BioIFObj*, a BioIndexedFile object. The entries are specified by *Indices*, a vector containing unique positive integers.

NewObj = getSubset(*BioIFObj*, *Keys*) returns *NewObj*, a new BioIndexedFile object that accesses a subset of entries in the source file associated with *BioIFObj*, a BioIndexedFile object. The entries are specified by *Keys*, a string or cell array of unique strings specifying keys.

Tips

Use this method to create a smaller, more manageable BioIndexedFile object.

Input Arguments

BioIFObj

Object of the `BioIndexedFile` class.

Default:

Indices

Vector containing unique positive integers that specify the entries in the source file to access with *NewObj*. The number of elements in *Indices* cannot exceed the number of entries indexed by *BioIFObj*. There is a one-to-one relationship between the elements in *Indices* and the entries that *NewObj* accesses.

Keys

String or cell array of unique strings specifying keys that specify the entries in the source file to access with *NewObj*. The number of elements in *Keys* is less than or equal to the number of entries indexed by *BioIFObj*. If the keys in the source file are not unique, then all entries that match a given key are indexed by *NewObj*. In this case, there is not a one-to-one relationship between the elements in *Keys* and the entries that *NewObj* accesses. If the keys in the source file are unique, then there is a one-to-one relationship between the elements in *Keys* and the entries that *NewObj* accesses.

Output Arguments

NewObj

Object of the `BioIndexedFile` class.

Examples

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

```
% Create a variable containing the full absolute path of the source file.
sourcefile = which('yeastgenes.sgd');
% Create a BioIndexedFile object from the source file. Indicate
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with !
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
                            'KeyColumn', 3, 'HeaderPrefix','!')
```

Create a new BioIndexedFile object that accesses only the first 1,000 cross-references and reuses the same index file as `gene2goObj`:

```
% Create a new BioIndexedFile object.
gene2goSubset = getSubset(gene2goObj,1:1000);
```

See Also

BioIndexedFile.getEntryByKey | BioIndexedFile.getIndexByKey
| BioIndexedFile.getKeys | BioIndexedFile |
BioIndexedFile.getEntryByIndex | BioIndexedFile.read

How To

· "Work with Large Multi-Entry Text Files"

getSubset

Class: BioRead

Create object containing subset of elements from object

Syntax

```
NewObj = getSubset(BioObj, Subset)
NewObj = getSubset(BioObj, 'SelectReference', R)
NewObj = getSubset(..., 'ParameterName', ParameterValue)
```

Description

NewObj = getSubset(*BioObj*, *Subset*) returns *NewObj*, a new object containing a subset of the elements from *BioObj*. getSubset returns object elements specified by *Subset*. If *BioObj* is indexed, then *NewObj* is indexed. If *BioObj* is in memory, then *NewObj* is in memory.

NewObj = getSubset(*BioObj*, 'SelectReference', *R*) for *BioObj* objects of the BioMap class creates a subset object with only the short reads mapped to *R*.

NewObj = getSubset(..., '*ParameterName*', *ParameterValue*) accepts one or more comma-separated parameter name/value pairs. Specify *ParameterName* inside single quotes.

Input Arguments

BioObj

Object of the BioRead or BioMap class.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

R

Vector of positive integers indexing the `SequenceDictionary` property of *BioObj* or a cell array of strings specifying the actual names of references in objects of the `BioMap` class.

Parameter Name/Value Pairs

'Name'

String specifying a name for *NewObj*. This information populates the `Name` property of *NewObj*.

Default: Empty string

'InMemory'

Logical specifying whether `getSubset` uses indexed access to the source file or loads the contents of the source file into memory. If the data specified for the subset is still large, set to `false` to use indexed access and be memory efficient. If the data specified for the subset fits in memory, set to `true` to load the data into memory, which lets you access *NewObj* faster and update its properties.

If *BioObj* was not constructed using indexed access and is already in memory, the `InMemory` name/value pair is ignored, and the data is automatically placed in memory.

Default: `false`

Output Arguments

NewObj

Object of the `BioRead` or `BioMap` class. If *BioObj* is in memory, then *NewObj* is in memory. If *BioObj* is indexed, then *NewObj* is indexed, unless you set the `InMemory` parameter name/value pair to `true`.

Examples

Retrieve a subset of elements from a `BioRead` object:

```
% Construct a BioRead object from a FASTQ file
BRObj = BioRead('SRR005164_1_50.fastq');
% Retrieve the information associated with the second and third
% elements in the object
getSubset(BRObj, [2 3]);
getSubset(BRObj, [2:3]);
getSubset(BRObj, {'SRR005164.2', 'SRR005164.3'});
getSubset(BRObj, [false true true]);
% Create a new BioRead object containing the first and third elements
% from the object
NewBRObj = getSubset(BRObj, [1 3]);
```

See Also
setSubset | BioRead | BioMap

How To
• "Manage Short-Read Sequence Data in Objects"

Related Links
• Sequence Read Archive
• SAM format specification

getSubset

Class: GFFAnnotation

Retrieve subset of elements from `GFFAnnotation` object

Syntax

```
NewObj = getSubset(AnnotObj,StartPos,EndPos)
NewObj = getSubset(AnnotObj,Subset)
NewObj = getSubset( ___ ,Name,Value)
```

Description

`NewObj = getSubset(AnnotObj,StartPos,EndPos)` returns NewObj, a new object containing a subset of the elements from AnnotObj that falls within each reference sequence range specified by StartPos and EndPos.

`NewObj = getSubset(AnnotObj,Subset)` returns NewObj, a new object containing a subset of elements specified by Subset, a vector of integers.

`NewObj = getSubset( ___ ,Name,Value)` returns NewObj, a new object containing a subset of the elements from AnnotObj, using any of the input arguments from the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

Tips

- The `getSubset` method selects annotations from the range specified by StartPos and EndPos for all reference sequences in AnnotObj unless you use the Reference name-value pair argument to limit the reference sequences.

- After creating a subsetted object, you can access the number of entries, range of reference sequence covered by annotations, field names, and reference names. To access the values of all fields, create a structure of the data using the `GFFAnnotation.getData` method.

Input Arguments

AnnotObj

Object of the `GFFAnnotation` class.

Default:

StartPos

Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

Default:

EndPos

Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

Default:

Subset

Vector of positive integers equal or less than the number of entries in the object. Use the vector `Subset` to retrieve any element or subset of the object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'Reference'

String or cell array of strings specifying one or more reference sequences in AnnotObj. Only annotations whose reference field matches one of the strings are included in NewObj.

Default:

'Feature'

String or cell array of strings specifying one or more features in AnnotObj. Only annotations whose feature field matches one of the strings are included in NewObj.

Default:

'Overlap'

Minimum number of base positions that an annotation must overlap in the range, to be included in NewObj. This value can be any of the following:

- Positive integer
- 'full' — An annotation must be fully contained in the range to be included.
- 'start' — An annotation's start position must lie within the range to be included.

Default: 1

Output Arguments

NewObj

Object of the GFFAnnotation class.

Examples

Create a Subset of Data Containing Only Protein Features from a GFF-formatted File

Construct a GFFAnnotation object using a GFF-formatted file that is provided with Bioinformatics Toolbox.

```
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
```

Create a subset of data containing only protein features.

```
subsetGFF1 = getSubset(GFFAnnotObj,'Feature','protein')

subsetGFF1 =

  GFFAnnotation with properties:
```

```
                FieldNames: {1x9 cell}
                NumEntries: 200
```

Retrieve Subsets of Data from a GFFAnnotation Object

Construct a `GFFAnnotation` object using a GFF-formatted file that is provided with Bioinformatics Toolbox.

```
GFFAnnotObj = GFFAnnotation('tair8_1.gff');
```

Retrieve a subset of data from the first to fifth elements of `GFFAnnotObj`.

```
subsetGFF2 = getSubset(GFFAnnotObj,[1:5])

subsetGFF2 =

  GFFAnnotation with properties:

    FieldNames: {1x9 cell}
    NumEntries: 5
```

Retrieve only the first, fifth and eighth elements of `GFFAnnotObj`.

```
subsetGFF3 = getSubset(GFFAnnotObj,[1 5 8])

subsetGFF3 =

  GFFAnnotation with properties:

    FieldNames: {1x9 cell}
    NumEntries: 3
```

See Also
GTFAnnotation.getSubset | GFFAnnotation.getData

How To
· "Store and Manage Feature Annotations in Objects"

Related Links
· GFF (General Feature Format) specifications document

getSubset

Class: GTFAnnotation

Create object containing subset of elements from `GTFAnnotation` object

Syntax

```
NewObj = getSubset(AnnotObj,StartPos,EndPos)
NewObj = getSubset(AnnotObj,Subset)
NewObj = getSubset( ___ ,Name,Value)
```

Description

`NewObj = getSubset(AnnotObj,StartPos,EndPos)` returns NewObj, a new object containing a subset of the elements from AnnotObj that falls within each reference sequence range specified by StartPos and EndPos.

`NewObj = getSubset(AnnotObj,Subset)` returns NewObj, a new object containing a subset of elements specified by Subset, a vector of integers.

`NewObj = getSubset( ___ ,Name,Value)` returns NewObj, a new object containing a subset of the elements from AnnotObj, using any of the input arguments from the previous syntaxes and additional options specified by one or more Name,Value pair arguments.

Tips

- The `getSubset` method selects annotations from the range specified by StartPos and EndPos for each reference sequence in AnnotObj unless you use the 'Reference' name-value pair argument to limit the reference sequences.

- After creating a subsetted object, you can access the number of entries, range of reference sequences covered by annotations, field names, and reference names. To access the values of all fields, create a structure of the data using the `GTFAnnotation.getData` method.

Input Arguments

AnnotObj

Object of the GTFAnnotation class.

Default:

StartPos

Nonnegative integer specifying the start of a range in each reference sequence in AnnotObj. The integer StartPos must be less than or equal to EndPos.

Default:

EndPos

Nonnegative integer specifying the end of a range in each reference sequence in AnnotObj. The integer EndPos must be greater than or equal to StartPos.

Default:

Subset

Vector of positive integers less than or equal to the number of entries in the object. Use the vector Subset to retrieve any element or subset of the object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'Reference'

String or cell array of strings specifying one or more reference sequences in AnnotObj. Only annotations whose reference field matches one of the strings are included in NewObj.

Default:

'Feature'

String or cell array of strings specifying one or more features in AnnotObj. Only annotations whose feature field matches one of the strings are included in NewObj.

Default:

'Gene'

String or cell array of strings specifying one or more genes in AnnotObj. Only annotations whose gene field matches one of the strings are included in NewObj.

'Transcript'

String or cell array of strings specifying one or more transcripts in AnnotObj. Only annotations whose transcript field matches one of the strings are included in NewObj.

'Overlap'

Minimum number of base positions that an annotation must overlap in the range, to be included in NewObj. This value can be any of the following:

- Positive integer
- `'full'` — An annotation must be fully contained in the range to be included.
- `'start'` — An annotation's start position must lie within the range to be included.

Default: 1

Output Arguments

NewObj

Object of the `GTFAnnotation` class.

Examples

Create a Subset of Data Containing Only CDS Features from a GTF-formatted File

Construct a `GTFAnnotation` object using a GTF-formatted file that is provided with Bioinformatics Toolbox.

```
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
```

Create a subset of the data containing only CDS features.

```
subsetGTF = getSubset(GTFAnnotObj,'Feature','CDS')

subsetGTF =

  GTFAnnotation with properties:

    FieldNames: {1x11 cell}
    NumEntries: 92
```

Retrieve Subsets of Data from a GTFAnnotation Object

Construct a `GTFAnnotation` object using a GTF-formatted file that is provided with Bioinformatics Toolbox.

```
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
```

Retrieve a subset of data from the first to fifth elements of `GTFAnnotObj`.

```
subsetGTF1 = getSubset(GTFAnnotObj,[1:5])

subsetGTF1 =

  GTFAnnotation with properties:

    FieldNames: {1x11 cell}
    NumEntries: 5
```

Retrieve only the first, fifth and eighth elements of `GTFAnnotObj`.

```
subsetGTF2 = getSubset(GTFAnnotObj,[1 5 8])

subsetGTF2 =

  GTFAnnotation with properties:

    FieldNames: {1x11 cell}
    NumEntries: 3
```

See Also
GFFAnnotation.getSubset | GTFAnnotation.getData

How To

· "Store and Manage Feature Annotations in Objects"

Related Links

· GTF2.2: A Gene Annotation Format

getTranscripts

Class: GTFAnnotation

Return table of unique transcripts in `GTFAnnotation` object

Syntax

```
transcriptsTable = getTranscripts(AnnotObj)
transcriptsTable = getTranscripts(AnnotObj,'Reference',R)
transcriptsTable = getTranscripts(AnnotObj,'Gene',G)
transcriptsTable = getTranscripts(AnnotObj,'Transcript',T)
```

Description

`transcriptsTable = getTranscripts(AnnotObj)` returns transcriptsTable, a table of transcripts referenced by exons in AnnotObj.

`transcriptsTable = getTranscripts(AnnotObj,'Reference',R)` returns the transcript(s) that belong to the reference(s) specified by R.

`transcriptsTable = getTranscripts(AnnotObj,'Gene',G)` returns the transcript(s) that belong to the gene(s) specified by G.

`transcriptsTable = getTranscripts(AnnotObj,'Transcript',T)` returns the transcript(s) specified by T.

Input Arguments

`AnnotObj` — GTF annotation
GTFAnnotation object

GTF annotation, specified as a `GTFAnnotation` object.

R — Names of reference sequences
string | cell array of strings | categorical array

Names of reference sequences, specified as a string, cell array of strings, or categorical array.

The names must come from the `Reference` field of AnnotObj. If a name does not exist, the function provides a warning and ignores it.

G — Names of gene
string | cell array of strings | categorical array

Names of genes, specified as a string, cell array of strings, or categorical array.

The names must come from the `Gene` field of AnnotObj. If a name does not exist, the function provides a warning and ignores the name.

T — Names of transcripts
string | cell array of strings | categorical array

Names of transcripts, specified as a string, cell array of strings, or categorical array.

The names must come from the `Transcript` field of AnnotObj. If a name does not exist, the function gives a warning and ignores the name.

Output Arguments

transcriptsTable — Transcripts
table

Transcripts, returned as a table. The table contains the following variables for each transcript.

Variable Name	Description
Transcript	Cell array of strings containing transcript IDs, obtained from the `Transcript` field of AnnotObj.
GeneName	Cell array of strings containing the names of expressed genes, obtained from the `Attributes` field of AnnotObj. This cell array can contain empty strings if the corresponding gene names are not found in `Attributes`.
GeneID	Cell array of strings containing the expressed gene IDs, obtained from the `Gene` field of AnnotObj.

Variable Name	Description
Reference	Categorical array representing the names of reference sequences to which the expressed genes belong. The reference names are from the Reference field of AnnotObj.
Start	Start location of the first exon in each transcript.
Stop	Stop location of the last exon in each transcript.
Strand	Categorical array containing the strand of expressed gene.

Examples

Retrieve Transcripts from a GTF-formatted File

Create a GTFAnnotation object from a GTF-formatted file.

```
obj = GTFAnnotation('hum37_2_1M.gtf');
```

Get the list of gene names listed in the object.

```
gNames = getGeneNames(obj)
```

```
gNames =

    'uc002qvu.2'
    'uc002qvv.2'
    'uc002qvw.2'
    'uc002qvx.2'
    'uc002qvy.2'
    'uc002qvz.2'
    'uc002qwa.2'
    'uc002qwb.2'
    'uc002qwc.1'
    'uc002qwd.2'
    'uc002qwe.3'
    'uc002qwf.2'
    'uc002qwg.2'
    'uc002qwh.2'
    'uc002qwi.3'
    'uc002qwk.2'
    'uc002qwl.2'
```

```
       'uc002qwm.1'
       'uc002qwn.1'
       'uc002qwo.1'
       'uc002qwp.2'
       'uc002qwq.2'
       'uc010ewe.2'
       'uc010ewf.1'
       'uc010ewg.2'
       'uc010ewh.1'
       'uc010ewi.2'
       'uc010yim.1'
```

Get a table of transcripts which belong to the first gene uc002qvu.2.

```
transcripts = getTranscripts(obj,'Gene',gNames{1})

transcripts =

    Transcript      GeneName       GeneID       Reference    Start      Stop      Strand
    _____    _____    _____    _____    _____    _____    _____

    'uc002qvu.2'       ''       'uc002qvu.2'      chr2       218138    249852      -
```

See Also

GTFAnnotation | GTFAnnotation.getData | GTFAnnotation.getFeatureNames
| GTFAnnotation.getGeneNames | GTFAnnotation.getIndex |
GTFAnnotation.getRange | GTFAnnotation.getReferenceNames
| GTFAnnotation.getSubset | GTFAnnotation.getGenes |
GTFAnnotation.getSegments | GTFAnnotation.getExons

More About

- "Store and Manage Feature Annotations in Objects"

External Web Sites

- GTF2.2: A Gene Annotation Format

getTranscriptNames

Class: GTFAnnotation

Retrieve unique transcript names from `GTFAnnotation` object

Syntax

```
Transcripts = getTranscriptNames(AnnotObj)
```

Description

`Transcripts = getTranscriptNames(AnnotObj)` returns Transcripts, a cell array of strings specifying the unique transcript names associated with annotations in AnnotObj.

Input Arguments

AnnotObj

Object of the `GTFAnnotation` class.

Default:

Output Arguments

Transcripts

Cell array of strings specifying the unique transcript names associated with annotations in AnnotObj.

Examples

Construct a `GTFAnnotation` object from a GTF-formatted file that is provided with Bioinformatics Toolbox, and then retrieve a list of the unique transcript names from the object:

```
% Construct a GTFAnnotation object from a GTF file
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
% Get transcript names from object
transcriptNames = getTranscriptNames(GTFAnnotObj)

transcriptNames =

    'uc002qvu.2'
    'uc002qvv.2'
    'uc002qvw.2'
    'uc002qvx.2'
    'uc002qvy.2'
    'uc002qvz.2'
    'uc002qwa.2'
    'uc002qwb.2'
    'uc002qwc.1'
    'uc002qwd.2'
    'uc002qwe.3'
    'uc002qwf.2'
    'uc002qwg.2'
    'uc002qwh.2'
    'uc002qwi.3'
    'uc002qwk.2'
    'uc002qwl.2'
    'uc002qwm.1'
    'uc002qwn.1'
    'uc002qwo.1'
    'uc002qwp.2'
    'uc002qwq.2'
    'uc010ewe.2'
    'uc010ewf.1'
    'uc010ewg.2'
    'uc010ewh.1'
    'uc010ewi.2'
    'uc010yim.1'
```

How To

- "Store and Manage Feature Annotations in Objects"

Related Links

- GTF2.2: A Gene Annotation Format

GFFAnnotation class

Represent General Feature Format (GFF) annotations

Description

The GFFAnnotation class contains annotations for one or more reference sequences, conforming to the GFF file format.

You construct a GFFAnnotation object from a GFF- or GTF-formatted file. Each element in the object represents an annotation. Use the object properties and methods to filter annotations by feature, reference sequence, or reference sequence position. Use object methods to extract data for a subset of annotations into an array of structures.

Construction

Annotobj = GFFAnnotation(*File*) constructs *Annotobj*, a GFFAnnotation object, from *File*, a GFF- or GTF-formatted file.

Input Arguments

File

String specifying a GFF- or GTF-formatted file.

Default:

Properties

FieldNames

Cell array of strings specifying the names of the available data fields for each annotation in the GFFAnnotation object. This property is read only.

NumEntries

Integer specifying number of annotations in the GFFAnnotation object. This property is read only.

Methods

getData

           Create structure containing subset of data from GFFAnnotation object

getFeatureNames

           Retrieve unique feature names from GFFAnnotation object

getIndex

           Return index array of annotations from GFFAnnotation object

getRange

           Retrieve range of annotations from GFFAnnotation object

getReferenceNames

           Retrieve reference names from GFFAnnotation object

getSubset

           Retrieve subset of elements from GFFAnnotation object

Copy Semantics

Value. To learn how value classes affect copy operations, see Copying Objects in the MATLAB documentation.

Indexing

GFFAnnotation objects support dot . indexing to extract properties.

Examples

Construct a `GFFAnnotation` object from a GFF-formatted file that is provided with Bioinformatics Toolbox:

```
GFFAnnotObj = GFFAnnotation('tair8_1.gff')

GFFAnnotObj =

  GFFAnnotation with properties:

    FieldNames: {'Reference'  'Start'  'Stop'  'Feature'  'Source'  'Score'  'Strand'
    NumEntries: 3331
```

Construct a `GFFAnnotation` object from a GTF-formatted file that is provided with Bioinformatics Toolbox:

```
GFFAnnotObj = GFFAnnotation('hum37_2_1M.gtf')

GFFAnnotObj =

  GFFAnnotation with properties:

    FieldNames: {'Reference'  'Start'  'Stop'  'Feature'  'Source'  'Score'  'Strand'
    NumEntries: 308
```

See Also
GTFAnnotation

How To
- "Store and Manage Feature Annotations in Objects"

Related Links
- GFF (General Feature Format) specifications document

goannotread

Read annotations from Gene Ontology annotated file

Syntax

Annotation = goannotread(*File*)
Annotation = goannotread(*File*, ...'Fields', *FieldsValue*, ...)
Annotation = goannotread(*File*, ...'Aspect', *AspectValue*, ...)

Input Arguments

File	String specifying a file name of a Gene Ontology (GO) annotated format (GAF) file.
FieldsValue	String or cell array of strings specifying one or more fields to read from the Gene Ontology annotated file. Default is to read all fields. Valid fields are listed below.
AspectValue	Character array specifying one or more characters. Valid aspects are: • `P` — Biological process • `F` — Molecular function • `C` — Cellular component Default is `'CFP'`, which specifies to read all aspects.

Output Arguments

Annotation	MATLAB array of structures containing annotations from a Gene Ontology annotated file.

Description

Note: The `goannotread` function supports GAF 1.0 and 2.0 file formats.

Annotation = goannotread(*File*) converts the contents of *File*, a Gene Ontology annotated file, into *Annotation*, an array of structures. Files should have the structure specified in:

`http://www.geneontology.org/GO.annotation.shtml#file`

A list with some annotated files can be found at:

`http://www.geneontology.org/GO.current.annotations.shtml`

Annotation = goannotread(*File*, ...'*PropertyName*', *PropertyValue*, ...) calls goannotread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Annotation = goannotread(*File*, ...'Fields', *FieldsValue*, ...) specifies the fields to read from the Gene Ontology annotated file. *FieldsValue* is a string or cell array of strings specifying one or more fields. Default is to read all fields. Valid fields are:

- `Database`
- `DB_Object_ID`
- `DB_Object_Symbol`
- `Qualifier`
- `GOid`
- `DBReference`
- `Evidence`
- `WithFrom`
- `Aspect`
- `DB_Object_Name`
- `Synonym`
- `DB_Object_Type`
- `Taxon`
- `Date`
- `Assigned_by`

For more information on these fields, see:

```
http://www.geneontology.org/GO.format.annotation.shtml
```

Annotation = goannotread(*File*, ...'Aspect', *AspectValue*, ...) specifies the aspects to read from the Gene Ontology annotated file. *AspectValue* is a character array specifying one or more characters. Valid aspects are:

- P — Biological process
- F — Molecular function
- C — Cellular component

Default is 'CFP', which specifies to read all aspects.

Examples

Reading All Annotations from a Gene Ontology Annotated File

1 Open a Web browser to

```
http://www.geneontology.org/GO.current.annotations.shtml
```

2 Download gene_association.sgd.gz, the file containing GO annotations for the gene products of *Saccharomyces cerevisiae*, to your MATLAB Current Folder.

3 Uncompress the file using the gunzip function.

```
gunzip('gene_association.sgd.gz')
```

4 Read the file into the MATLAB software.

```
SGDGenes = goannotread('gene_association.sgd');
```

5 Create a structure with GO annotations and display a list of the first five genes.

```
S = struct2cell(SGDGenes);
genes = S(3,1:5)'

genes =

    '15S_RRNA'
    '15S_RRNA'
    '15S_RRNA'
    '15S_RRNA'
    '21S_RRNA'
```

Reading a Subset of Annotations from a Gene Ontology Annotated File

1 Open a Web browser to

http://www.geneontology.org/GO.current.annotations.shtml

2 Download gene_association.goa_human.gz, the file containing GO annotations for the gene products of *Homo sapiens*, to your MATLAB Current Folder.

3 Uncompress the file using the gunzip function.

gunzip('gene_association.goa_human.gz')

4 Read the file into the MATLAB software, but limit the annotations to genes related to molecular function (F), and to the fields for the gene symbol and the associated ID, that is, DB_Object_Symbol and GOid.

HumanStruct = goannotread('gene_association.goa_human', ...
                'Aspect','F','Fields',{'DB_Object_Symbol','GOid'});

5 Create a list of the *Homo sapiens* genes and a list of the associated GO terms.

Humangenes = {HumanStruct.DB_Object_Symbol};
HumanGO = [HumanStruct.GOid];

See Also

geneont.getancestors | geneont.getmatrix | geneont | num2goid | geneont |
geneont.getdescendants | geneont.getrelatives

gonnet

Return Gonnet scoring matrix

Syntax

```
gonnet
```

Description

`gonnet` returns the Gonnet matrix.

The Gonnet matrix is the recommended mutation matrix for initially aligning protein sequences. Matrix elements are ten times the logarithmic of the probability that the residues are aligned divided by the probability that the residues are aligned by chance, and then matrix elements are normalized to 250 PAM units.

Expected score = `-0.6152`, Entropy = `1.6845` bits, Lowest score = `-8`, Highest score = `14.2`

Order:

```
A R N D C Q E G H I L K M F P S T W Y V B Z X *
```

References

[1] Gaston, H., Gonnet, M., Cohen, A., and Benner, S. (1992). Exhaustive matching of the entire protein sequence database. Science. *256*, 1443–1445.

See Also

```
blosum | dayhoff | localalign | nuc44 | nwalign | pam | swalign
```

gprread

Read microarray data from GenePix Results (GPR) file

Syntax

```
GPRData = gprread('File')
gprread(..., 'PropertyName', PropertyValue,...)
gprread(..., 'CleanColNames', CleanColNamesValue)
```

Arguments

File	GenePix Results (GPR) formatted file. Enter a file name or a path and file name.
CleanColNamesValue	Controls the creation of column names that can be used as variable names.

Description

GPRData = gprread('*File*') reads GenePix results data from *File* and creates a MATLAB structure (*GPRData*).

gprread(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

gprread(..., 'CleanColNames', *CleanColNamesValue*) controls the creation of column names that can be used as variable names. A GPR file may contain column names with spaces and some characters that the MATLAB software cannot use in MATLAB variable names. If *CleanColNamesValue* is true, gprread returns names in the field ColumnNames that are valid MATLAB variable names and names that you can use in functions. By default, *CleanColNamesValue* is false and the field ColumnNames may contain characters that are invalid for MATLAB variable names.

The field Indices of the structure contains indices that can be used for plotting heat maps of the data.

For more details on the GPR format, see

`http://mdc.custhelp.com/app/answers/detail/a_id/18883/kw/file%20format`

The function supports versions 3, 4, and 5 of the GenePix Results Format.

Examples

Read and display data from GenePix® result (GPR) file

This example shows how to read and display data from a GenePix® result (GPR) file.

Read in a sample GPR file.

```
gprStruct = gprread('mouse_a1pd.gpr')


gprStruct =

         Header: [1x1 struct]
           Data: [9504x38 double]
         Blocks: [9504x1 double]
        Columns: [9504x1 double]
           Rows: [9504x1 double]
          Names: {9504x1 cell}
            IDs: {9504x1 cell}
    ColumnNames: {38x1 cell}
        Indices: [132x72 double]
          Shape: [1x1 struct]
```

Plot the median foreground intensity for the 635 nm channel.

```
maimage(gprStruct,'F635 Median')
```
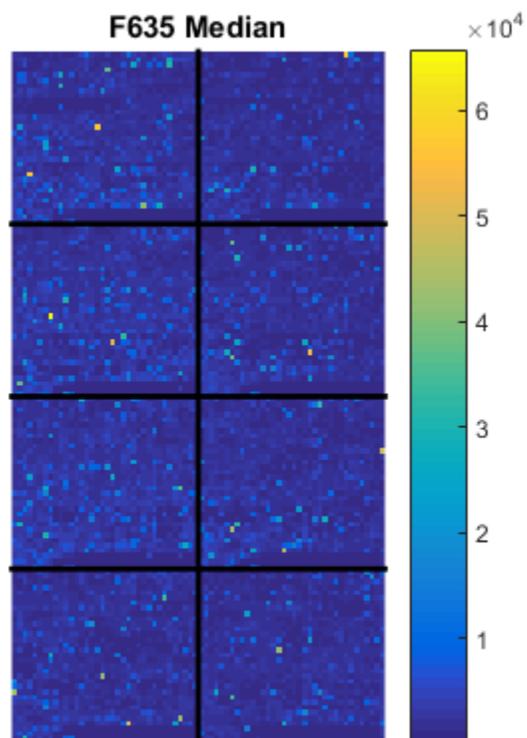
See Also

affyread | agferead | celintensityread | galread | geoseriesread | geosoftread | ilmnbsread | imageneread | magetfield | sptread

graphallshortestpaths

Find all shortest paths in graph

Syntax

```
[dist] = graphallshortestpaths(G)
[dist] = graphallshortestpaths(G, ...'Directed', DirectedValue, ...)
[dist] = graphallshortestpaths(G, ...'Weights', WeightsValue, ...)
```

Arguments

G	N-by-N sparse matrix that represents a graph. Nonzero entries in matrix *G* represent the weights of the edges.
DirectedValue	Property that indicates whether the graph is directed or undirected. Enter `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`.
WeightsValue	Column vector that specifies custom weights for the edges in matrix *G*. It must have one entry for every nonzero value (edge) in matrix *G*. The order of the custom weights in the vector must match the order of the nonzero values in matrix *G* when it is traversed column-wise. This property lets you use zero-valued weights. By default, `graphallshortestpaths` gets weight information from the nonzero entries in matrix *G*.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*dist*] = `graphallshortestpaths(G)` finds the shortest paths between every pair of nodes in the graph represented by matrix *G*, using Johnson's algorithm. Input *G* is an

N-by-N sparse matrix that represents a graph. Nonzero entries in matrix *G* represent the weights of the edges.

Output *dist* is an N-by-N matrix where *dist*(S,T) is the distance of the shortest path from source node S to target node T. Elements in the diagonal of this matrix are always 0, indicating the source node and target node are the same. A 0 not in the diagonal indicates that the distance between the source node and target node is 0. An Inf indicates there is no path between the source node and the target node.

Johnson's algorithm has a time complexity of O(N*log(N)+N*E), where N and E are the number of nodes and edges respectively.

[...] = graphallshortestpaths (*G*, '*PropertyName*', *PropertyValue*, ...) calls graphallshortestpaths with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[*dist*] = graphallshortestpaths(*G*, ...'Directed', *DirectedValue*, ...) indicates whether the graph is directed or undirected. Set *DirectedValue* to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

[*dist*] = graphallshortestpaths(*G*, ...'Weights', *WeightsValue*, ...) lets you specify custom weights for the edges. *WeightsValue* is a column vector having one entry for every nonzero value (edge) in matrix *G*. The order of the custom weights in the vector must match the order of the nonzero values in matrix *G* when it is traversed column-wise. This property lets you use zero-valued weights. By default, graphallshortestpaths gets weight information from the nonzero entries in matrix *G*.

Examples

Finding All Shortest Paths in a Directed Graph

1 Create and view a directed graph with 6 nodes and 11 edges.

```
W = [.41 .99 .51 .32 .15 .45 .38 .32 .36 .29 .21];
DG = sparse([6 1 2 2 3 4 4 5 5 6 1],[2 6 3 5 4 1 6 3 4 3 5],W)

DG =
```

```
(4,1)        0.4500
(6,2)        0.4100
(2,3)        0.5100
(5,3)        0.3200
(6,3)        0.2900
(3,4)        0.1500
(5,4)        0.3600
(1,5)        0.2100
(2,5)        0.3200
(1,6)        0.9900
(4,6)        0.3800
```

```
view(biograph(DG,[],'ShowWeights','on'))
```
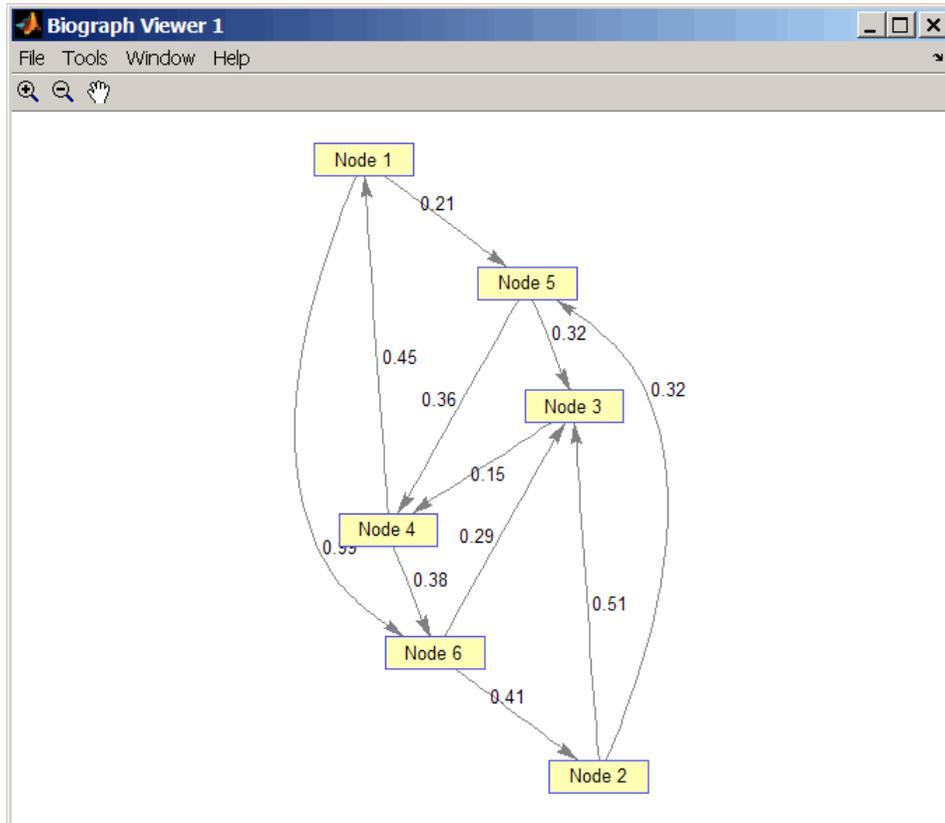

2   Find all the shortest paths between every pair of nodes in the directed graph.

```
graphallshortestpaths(DG)

ans =

         0    1.3600    0.5300    0.5700    0.2100    0.9500
    1.1100         0    0.5100    0.6600    0.3200    1.0400
    0.6000    0.9400         0    0.1500    0.8100    0.5300
    0.4500    0.7900    0.6700         0    0.6600    0.3800
    0.8100    1.1500    0.3200    0.3600         0    0.7400
    0.8900    0.4100    0.2900    0.4400    0.7300         0
```

The resulting matrix shows the shortest path from node 1 (first row) to node 6 (sixth column) is 0.95. You can see this in the graph by tracing the path from node 1 to node 5 to node 4 to node 6 (0.21 + 0.36 + 0.38 = 0.95).

Finding All Shortest Paths in an Undirected Graph

1   Create and view an undirected graph with 6 nodes and 11 edges.

```
UG = tril(DG + DG')

UG =

    (4,1)        0.4500
    (5,1)        0.2100
    (6,1)        0.9900
    (3,2)        0.5100
    (5,2)        0.3200
    (6,2)        0.4100
    (4,3)        0.1500
    (5,3)        0.3200
    (6,3)        0.2900
    (5,4)        0.3600
    (6,4)        0.3800

view(biograph(UG,[],'ShowArrows','off','ShowWeights','on'))
```
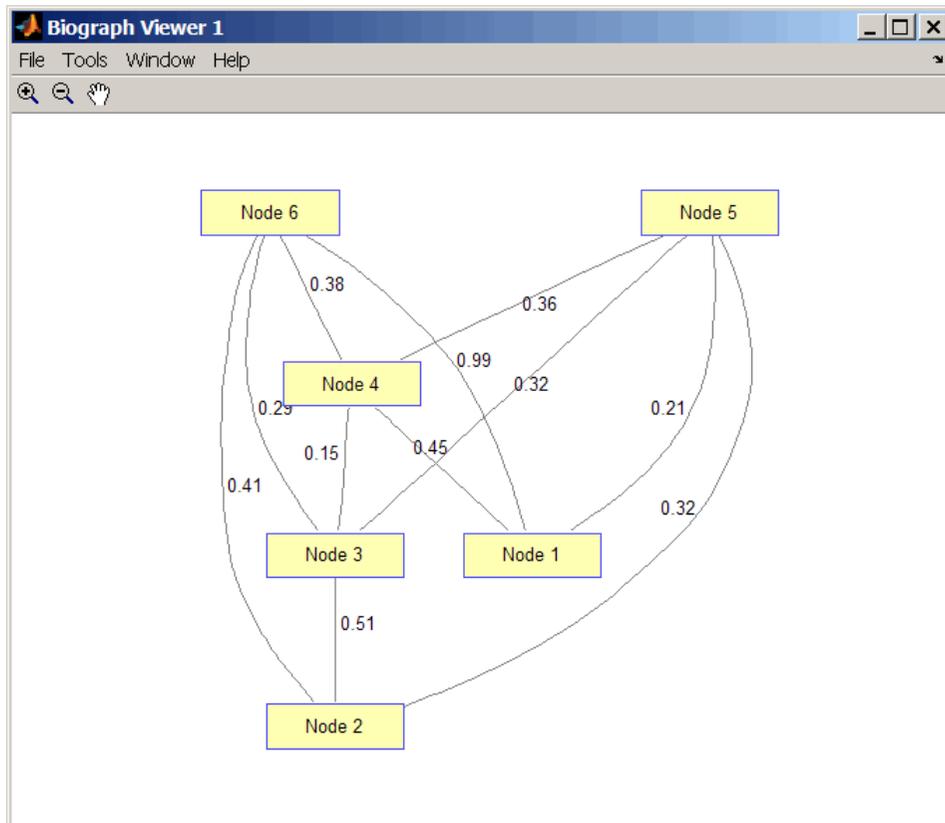
2 Find all the shortest paths between every pair of nodes in the undirected graph.

```
graphallshortestpaths(UG,'directed',false)

ans =

         0    0.5300    0.5300    0.4500    0.2100    0.8300
    0.5300         0    0.5100    0.6600    0.3200    0.7000
    0.5300    0.5100         0    0.1500    0.3200    0.5300
    0.4500    0.6600    0.1500         0    0.3600    0.3800
    0.2100    0.3200    0.3200    0.3600         0    0.7400
    0.8300    0.7000    0.5300    0.3800    0.7400         0
```

The resulting matrix is symmetrical because it represents an undirected graph. It shows the shortest path from node 1 (first row) to node 6 (sixth column) is 0.83. You can see this in the graph by tracing the path from node 1 to node 4 to node 6 (0.45 + 0. 38 = 0.83). Because `UG` is an undirected graph, we can use the edge between node 1 and node 4, which we could not do in the directed graph `DG`.

References

[1] Johnson, D.B. (1977). Efficient algorithms for shortest paths in sparse networks. Journal of the ACM *24(1)*, 1-13.

[2] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

graphconncomp | allshortestpaths | graphisdag | graphisomorphism | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphshortestpath | graphtopoorder | graphtraverse

graphconncomp

Find strongly or weakly connected components in graph

Syntax

```
[S, C] = graphconncomp(G)
[S, C] = graphconncomp(G, ...'Directed', DirectedValue, ...)
[S, C] = graphconncomp(G, ...'Weak', WeakValue, ...)
```

Arguments

G	N-by-N sparse matrix that represents a graph. Nonzero entries in matrix *G* indicate the presence of an edge.
DirectedValue	Property that indicates whether the graph is directed or undirected. Enter `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`. A DFS-based algorithm computes the connected components. Time complexity is `O(N+E)`, where `N` and `E` are number of nodes and edges respectively.
WeakValue	Property that indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set *WeakValue* to `true` to find weakly connected components. Default is `false`, which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is `O(N+E)`, where `N` and `E` are number of nodes and edges respectively.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*S*, *C*] = graphconncomp(*G*) finds the strongly connected components of the graph represented by matrix *G* using Tarjan's algorithm. A strongly connected component is a maximal group of nodes that are mutually reachable without violating the edge directions. Input *G* is an N-by-N sparse matrix that represents a graph. Nonzero entries in matrix *G* indicate the presence of an edge.

The number of components found is returned in *S*, and *C* is a vector indicating to which component each node belongs.

Tarjan's algorithm has a time complexity of O(N+E), where N and E are the number of nodes and edges respectively.

[*S*, *C*] = graphconncomp(*G*, ...'*PropertyName*', *PropertyValue*, ...) calls graphconncomp with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[*S*, *C*] = graphconncomp(*G*, ...'Directed', *DirectedValue*, ...) indicates whether the graph is directed or undirected. Set *directedValue* to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true. A DFS-based algorithm computes the connected components. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.

[*S*, *C*] = graphconncomp(*G*, ...'Weak', *WeakValue*, ...) indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set *WeakValue* to true to find weakly connected components. Default is false, which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.

Note: By definition, a single node can be a strongly connected component.

Note: A directed acyclic graph (DAG) cannot have any strongly connected components larger than one.

Examples

1 Create and view a directed graph with 10 nodes and 17 edges.

```
DG = sparse([1 1 1 2 2 3 3 4 5 6 7 7 8 9 9  9 9], ...
             [2 6 8 3 1 4 2 5 4 7 6 4 9 8 10 5 3],true,10,10)

DG =

   (2,1)        1
   (1,2)        1
   (3,2)        1
   (2,3)        1
   (9,3)        1
   (3,4)        1
   (5,4)        1
   (7,4)        1
   (4,5)        1
   (9,5)        1
   (1,6)        1
   (7,6)        1
   (6,7)        1
   (1,8)        1
   (9,8)        1
   (8,9)        1
   (9,10)       1

h = view(biograph(DG));
```
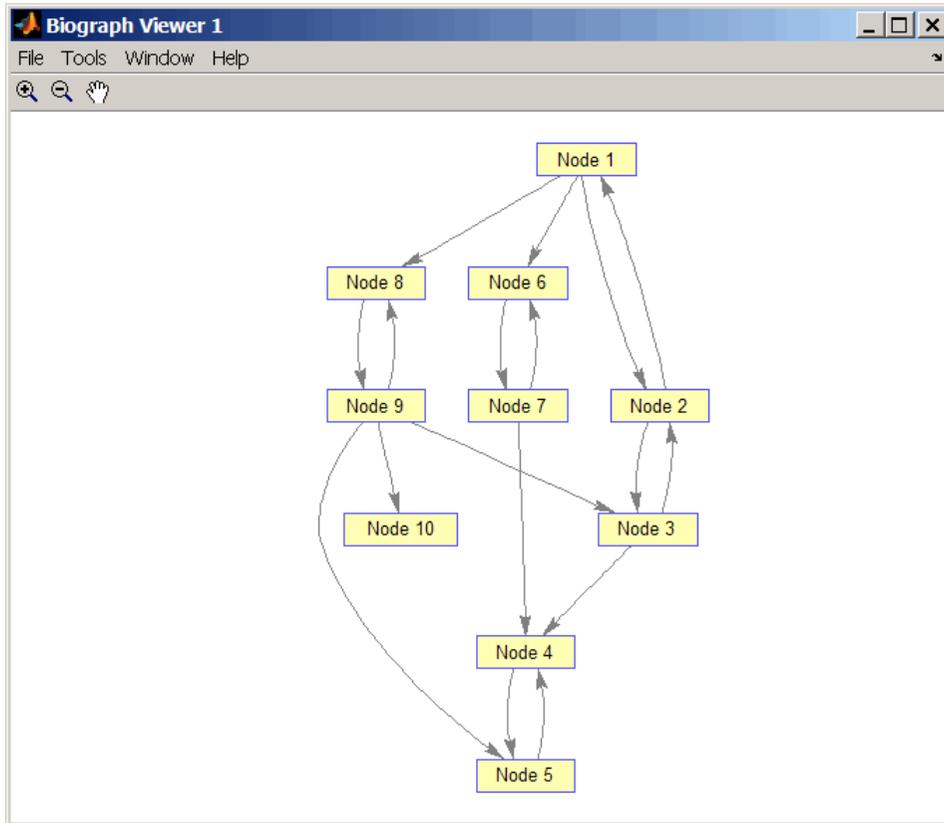
2 Find the number of strongly connected components in the directed graph and determine to which component each of the 10 nodes belongs.

```
[S,C] = graphconncomp(DG)

S =

    4


C =

    4    4    4    1    1    2    2    4    4    3
```

3 Color the nodes for each component with a different color.

```
colors = jet(S);
for i = 1:numel(h.nodes)
  h.Nodes(i).Color = colors(C(i),:);
end
```
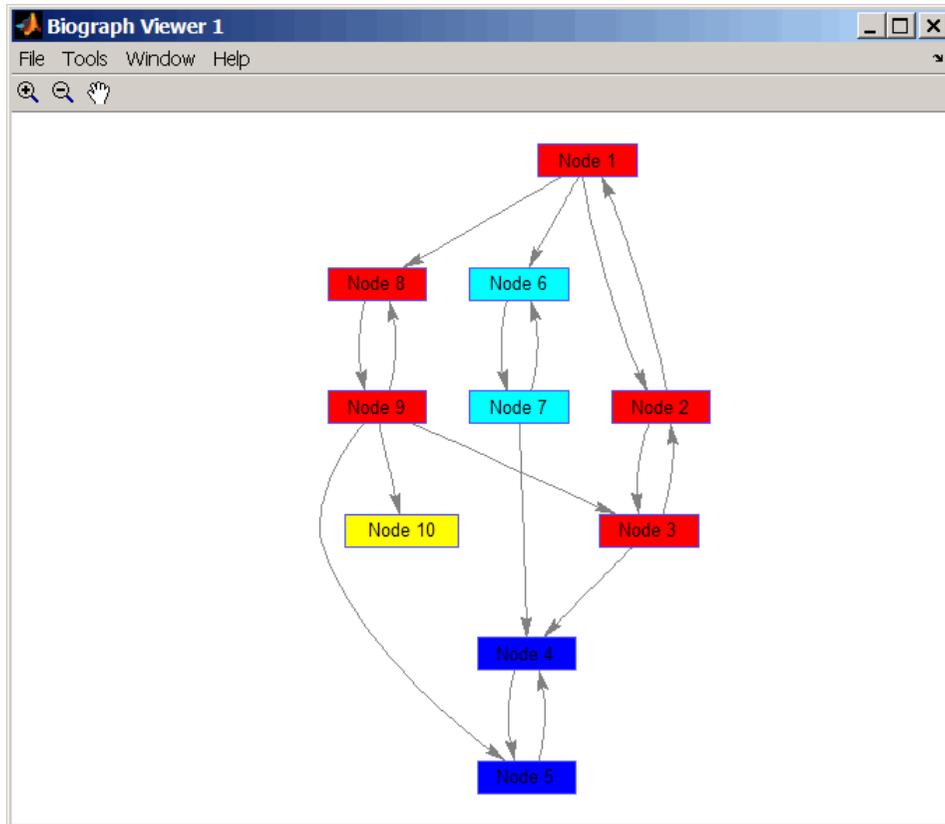

References

[1] Tarjan, R.E., (1972). Depth first search and linear graph algorithms. SIAM Journal on Computing *1(2)*, 146–160.

[2] Sedgewick, R., (2002). Algorithms in C++, Part 5 Graph Algorithms (Addison-Wesley).

[3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

graphallshortestpaths | conncomp | graphisdag | graphisomorphism | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphshortestpath | graphtopoorder | graphtraverse

graphisdag

Test for cycles in directed graph

Syntax

```
graphisdag(G)
```

Arguments

G	N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix *G* indicate the presence of an edge.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

`graphisdag(G)` returns logical 1 (`true`) if the directed graph represented by matrix *G* is a directed acyclic graph (DAG) and logical 0 (`false`) otherwise. *G* is an N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix *G* indicate the presence of an edge.

Examples

Testing for Cycles in Directed Graphs

1 Create and view a directed acyclic graph (DAG) with six nodes and eight edges.

```
DG = sparse([1 1 1 2 2 3 4 6],[2 4 6 3 5 4 6 5],true,6,6)

DG =
```

```
(1,2)        1
(2,3)        1
(1,4)        1
(3,4)        1
(2,5)        1
(6,5)        1
(1,6)        1
(4,6)        1
```

```
view(biograph(DG))
```
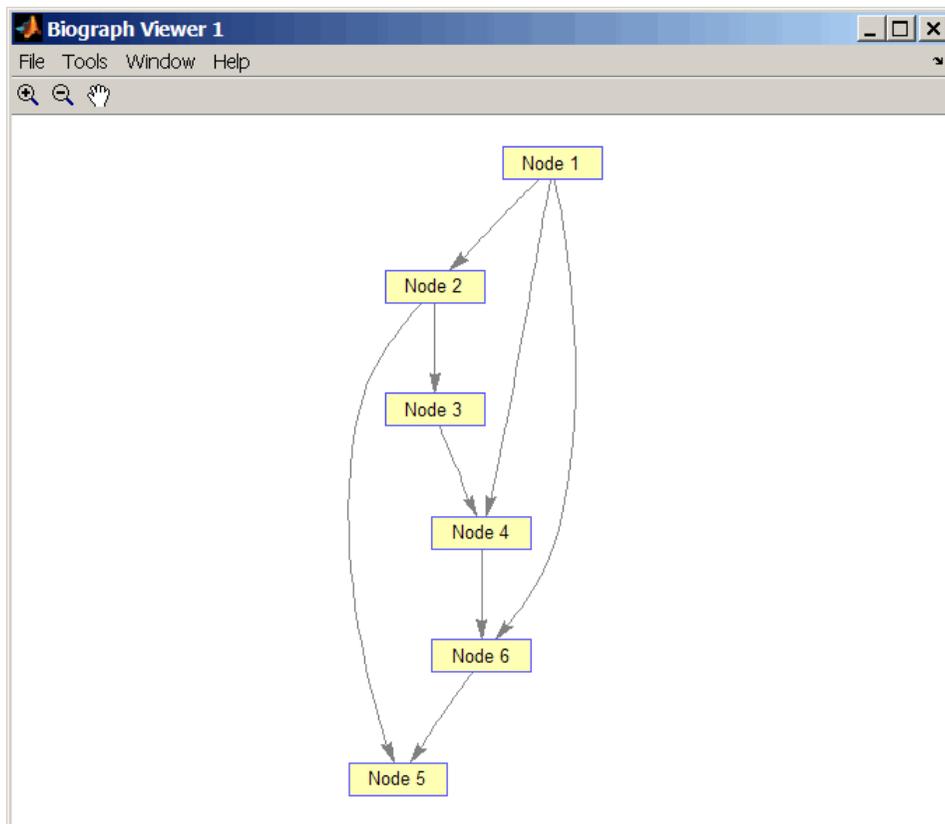

2 Test for cycles in the DAG.

```
graphisdag(DG)
```

```
ans =

     1
```

3 Add an edge to the DAG to make it cyclic, and then view the directed graph.

```
DG(5,1) = true

DG =

   (5,1)        1
   (1,2)        1
   (2,3)        1
   (1,4)        1
   (3,4)        1
   (2,5)        1
   (6,5)        1
   (1,6)        1
   (4,6)        1

view(biograph(DG))
```
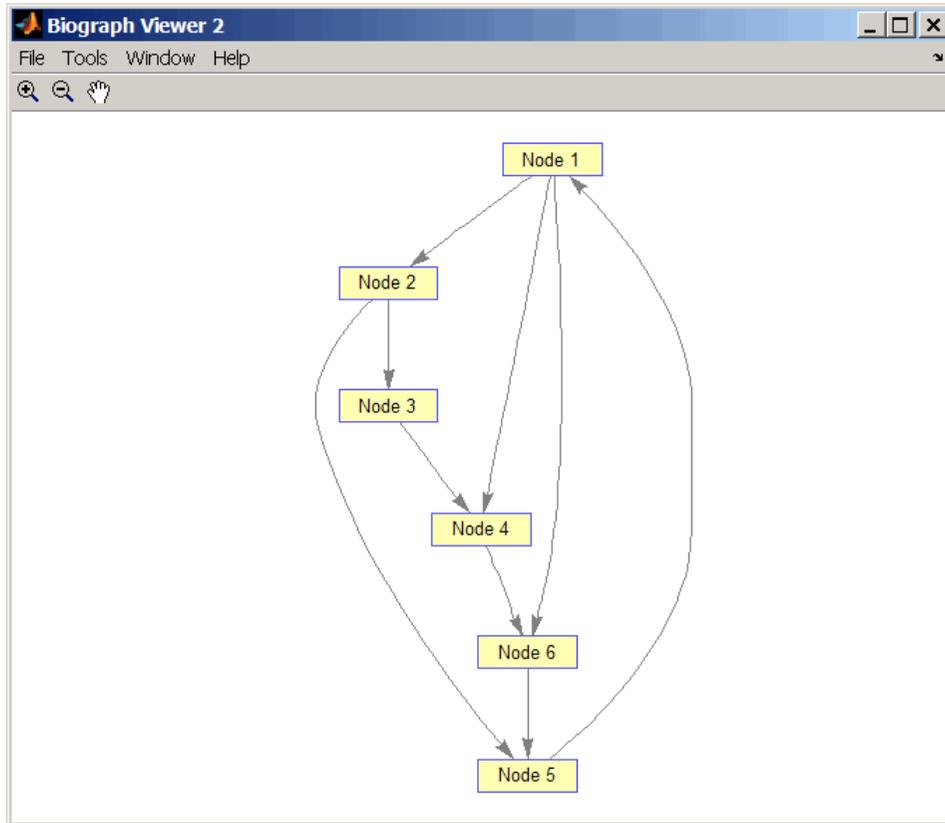
4 Test for cycles in the new graph.

```
graphisdag(DG)

ans =

      0
```

Testing for Cycles in a Very Large Graph (Greater Than 20,000 Nodes and 30,000 Edges)

1 Download the Gene Ontology database to a geneont object.

```
GO = geneont('live',true);
```

2 Convert the geneont object to a matrix.

```
CM = getmatrix(GO);
```

3 Test for cycles in the graph.

```
graphisdag(CM)
```

Creating a Random DAG

1 Create and view a random directed acyclic graph (DAG) with 15 nodes and 20 edges.

```
g = sparse([],[],true,15,15);
while nnz(g) < 20
  edge = randsample(15*15,1); % get a random edge
  g(edge) = true;
  g(edge) = graphisdag(g);
end
view(biograph(g))
```

2 Test for cycles in the graph.

```
graphisdag(g)
```

References

[1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

graphallshortestpaths | isdag | graphconncomp | graphisomorphism | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphshortestpath | graphtopoorder | graphtraverse

graphisomorphism

Find isomorphism between two graphs

Syntax

```
[Isomorphic, Map] = graphisomorphism(G1, G2)
[Isomorphic, Map] = graphisomorphism(G1,
G2,'Directed', DirectedValue)
```

Arguments

G1	N-by-N sparse matrix that represents a directed or undirected graph. Nonzero entries in matrix *G1* indicate the presence of an edge.
G2	N-by-N sparse matrix that represents a directed or undirected graph. *G2* must be the same (directed or undirected) as *G1*.
DirectedValue	Property that indicates whether the graphs are directed or undirected. Enter false when both *G1* and *G2* are undirected graphs. In this case, the upper triangles of the sparse matrices *G1* and *G2* are ignored. Default is true, meaning that both graphs are directed.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*Isomorphic*, *Map*] = graphisomorphism(*G1*, *G2*) returns logical 1 (true) in *Isomorphic* if *G1* and *G2* are isomorphic graphs, and logical 0 (false) otherwise. A graph isomorphism is a 1-to-1 mapping of the nodes in the graph *G1* and the nodes in the graph *G2* such that adjacencies are preserved. *G1* and *G2* are both N-by-N sparse

matrices that represent directed or undirected graphs. Return value *Isomorphic* is Boolean. When *Isomorphic* is true, *Map* is a row vector containing the node indices that map from *G2* to *G1* for one possible isomorphism. When *Isomorphic* is false, *Map* is empty. The worst-case time complexity is O(N!), where N is the number of nodes.

[*Isomorphic*, *Map*] = graphisomorphism(*G1*, *G2*,'Directed', *DirectedValue*) indicates whether the graphs are directed or undirected. Set *DirectedValue* to false when both *G1* and *G2* are undirected graphs. In this case, the upper triangles of the sparse matrices *G1* and *G2* are ignored. Default is true, meaning that both graphs are directed.

Examples

1   Create and view a directed graph with 8 nodes and 11 edges.

```
m('ABCDEFGH') = [1 2 3 4 5 6 7 8];
g1 = sparse(m('ABDCDCGEFFG'),m('BCBDGEEFHGH'),true,8,8)

g1 =

    (1,2)        1
    (4,2)        1
    (2,3)        1
    (3,4)        1
    (3,5)        1
    (7,5)        1
    (5,6)        1
    (4,7)        1
    (6,7)        1
    (6,8)        1
    (7,8)        1

view(biograph(g1,'ABCDEFGH'))
```
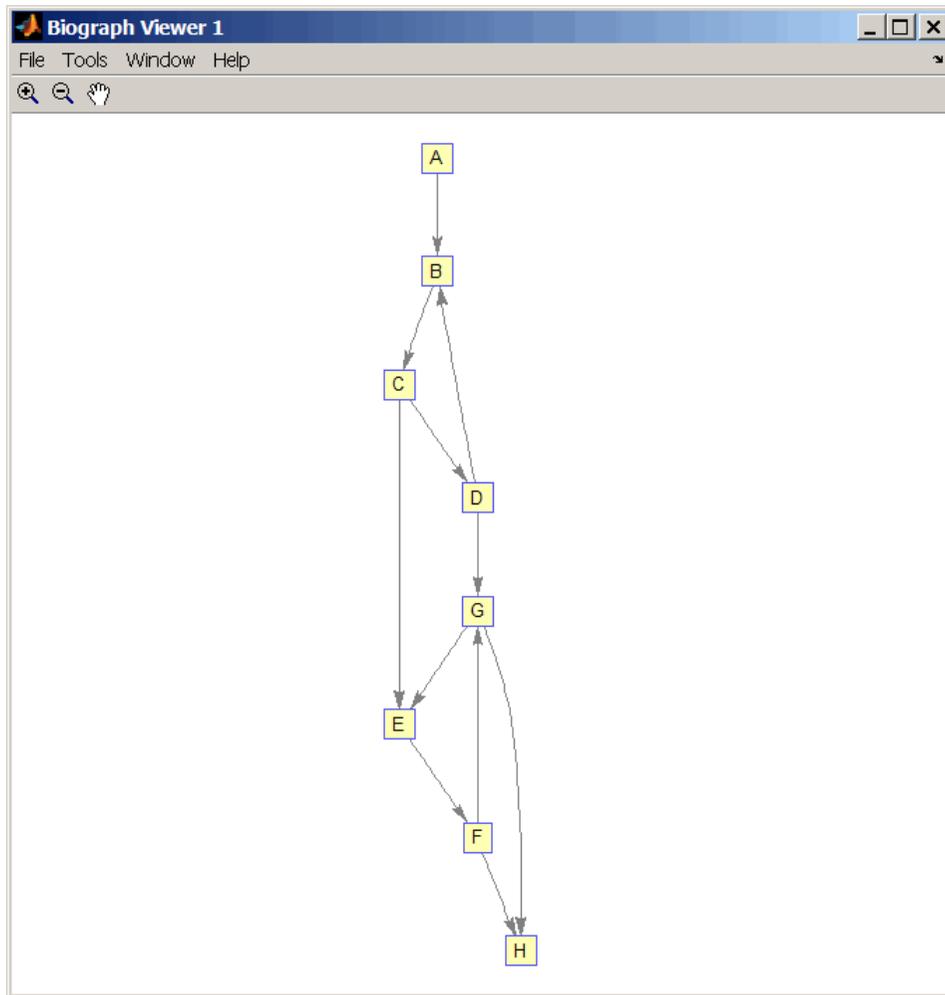
2 Set a random permutation vector and then create and view a new permuted graph.

```
p = randperm(8)

p =

      7     8     2     3     6     4     1     5
```

```
g2 = g1(p,p);
view(biograph(g2,'12345678'))
```
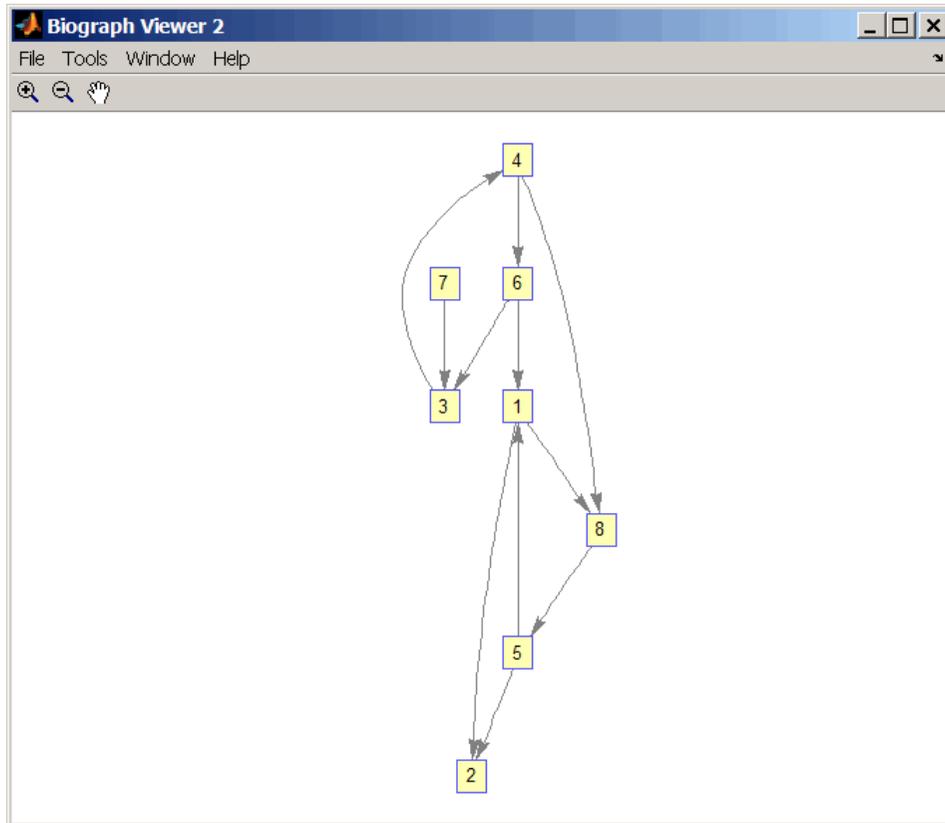

3 Check if the two graphs are isomorphic.

```
[F,Map] = graphisomorphism(g2,g1)

F =

     1


Map =

     7     8     2     3     6     4     1     5
```

Note that the `Map` row vector containing the node indices that map from `g2` to `g1` is the same as the permutation vector you created in step 2.

4 Reverse the direction of the D-G edge in the first graph, and then check for isomorphism again.

```
g1(m('DG'),m('GD')) = g1(m('GD'),m('DG'));
view(biograph(g1,'ABCDEFGH'))
```
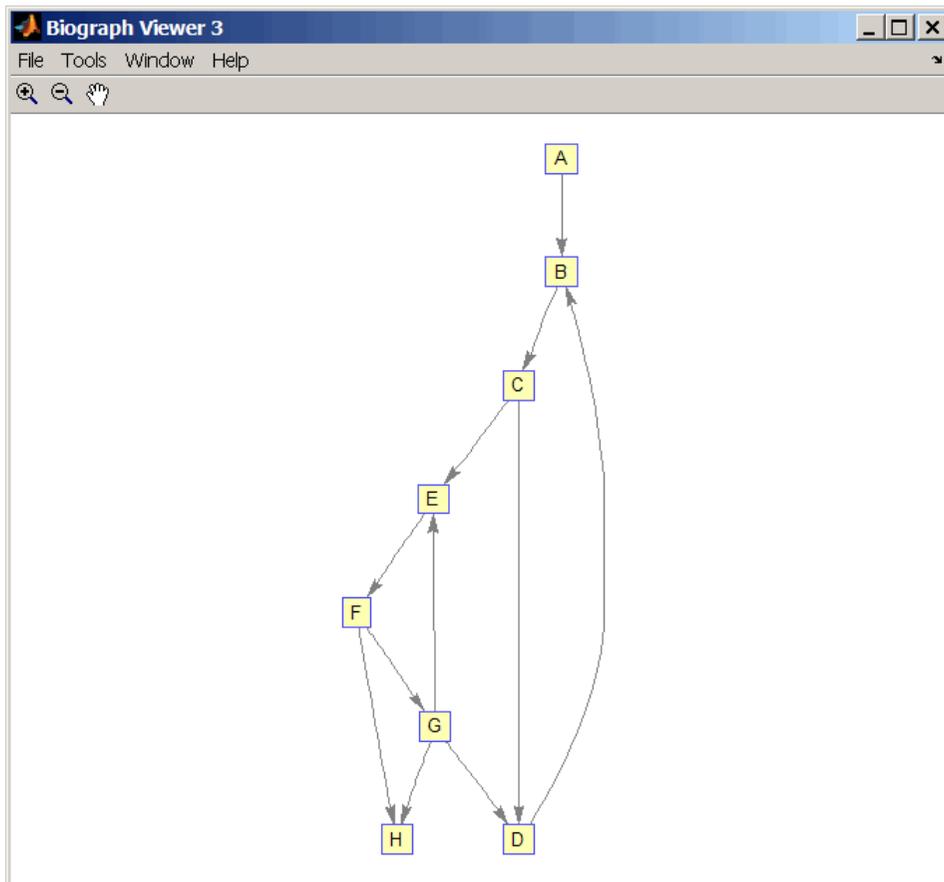


```
[F,M] = graphisomorphism(g2,g1)

F =
```

```
        0


M =

     []
```

5 Convert the graphs to undirected graphs, and then check for isomorphism.

```
[F,M] = graphisomorphism(g2+g2',g1+g1','directed',false)

F =

     1


M =

     7    8    2    3    6    4    1    5
```

References

[1] Fortin, S. (1996). The Graph Isomorphism Problem. Technical Report, 96-20, Dept. of Computer Science, University of Alberta, Edomonton, Alberta, Canada.

[2] McKay, B.D. (1981). Practical Graph Isomorphism. Congressus Numerantium *30*, 45-87.

[3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

graphallshortestpaths | isomorphism | graphconncomp | graphisdag | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphshortestpath | graphtopoorder | graphtraverse

graphisspantree

Determine if tree is spanning tree

Syntax

TF = graphisspantree(*G*)

Arguments

G	N-by-N sparse matrix whose lower triangle represents an undirected graph. Nonzero entries in matrix *G* indicate the presence of an edge.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

TF = graphisspantree(*G*) returns logical 1 (true) if *G* is a spanning tree, and logical 0 (false) otherwise. A spanning tree must touch all the nodes and must be acyclic. *G* is an N-by-N sparse matrix whose lower triangle represents an undirected graph. Nonzero entries in matrix *G* indicate the presence of an edge.

## Examples

1 Create a phytree object from a phylogenetic tree file.

   ```
   tr = phytreeread('pf00002.tree')

   Phylogenetic tree object with 33 leaves (32 branches)
   ```

2 Create a connection matrix from the phytree object.

   ```
   [CM,labels,dist] = getmatrix(tr);
   ```

3 Determine if the connection matrix is a spanning tree.

```
graphisspantree(CM)

ans =

     1
```

4 Add an edge between the root and the first leaf in the connection matrix.

```
CM(end,1) = 1;
```

5 Determine if the modified connection matrix is a spanning tree.

```
graphisspantree(CM)

ans =

     0
```

References

[1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

graphallshortestpaths | isspantree | graphconncomp | graphisdag |
graphisomorphism | graphmaxflow | graphminspantree | graphpred2path |
graphshortestpath | graphtopoorder | graphtraverse

graphmaxflow

Calculate maximum flow in directed graph

Syntax

```
[MaxFlow, FlowMatrix, Cut] = graphmaxflow(G, SNode, TNode)
[...] = graphmaxflow(G, SNode, TNode, ...'Capacity',
CapacityValue, ...)
[...] = graphmaxflow(G, SNode, TNode, ...'Method', MethodValue, ...)
```

Arguments

G	N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix G represent the capacities of the edges.
SNode	Node in *G*.
TNode	Node in *G*.
CapacityValue	Column vector that specifies custom capacities for the edges in matrix *G*. It must have one entry for every nonzero value (edge) in matrix *G*. The order of the custom capacities in the vector must match the order of the nonzero values in matrix *G* when it is traversed column-wise. By default, graphmaxflow gets capacity information from the nonzero entries in matrix *G*.
MethodValue	String that specifies the algorithm used to find the minimal spanning tree (MST). Choices are: • 'Edmonds' — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the *labeling algorithm*. Time complexity is O(N*E^2), where N and E are the number of nodes and edges respectively. • 'Goldberg' — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as *preflow-push*. Time complexity is O(N^2*sqrt(E)), where N and E are the number of nodes and edges respectively.

Description

[*MaxFlow*, *FlowMatrix*, *Cut*] = graphmaxflow(*G*, *SNode*, *TNode*)calculates the maximum flow of directed graph *G* from node *SNode* to node *TNode*. Input *G* is an N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix *G* represent the capacities of the edges. Output *MaxFlow* is the maximum flow, and *FlowMatrix* is a sparse matrix with all the flow values for every edge. *FlowMatrix*(*X*,*Y*) is the flow from node *X* to node *Y*. Output *Cut* is a logical row vector indicating the nodes connected to *SNode* after calculating the minimum cut between *SNode* and *TNode*. If several solutions to the minimum cut problem exist, then *Cut* is a matrix.

Tip The algorithm that determines *Cut*, all minimum cuts, has a time complexity of O(2^*N*), where *N* is the number of nodes. If this information is not needed, use the graphmaxflow function without the third output.

[...] = graphmaxflow(*G*, *SNode*, *TNode*, ...'*PropertyName*', *PropertyValue*, ...) calls graphmaxflow with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[...] = graphmaxflow(*G*, *SNode*, *TNode*, ...'Capacity', *CapacityValue*, ...) lets you specify custom capacities for the edges. *CapacityValue* is a column vector having one entry for every nonzero value (edge) in matrix *G*. The order of the custom capacities in the vector must match the order of the nonzero values in matrix *G* when it is traversed column-wise. By default, graphmaxflow gets capacity information from the nonzero entries in matrix *G*.

[...] = graphmaxflow(*G*, *SNode*, *TNode*, ...'Method', *MethodValue*, ...) lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- 'Edmonds' — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the *labeling algorithm*. Time complexity is O(N*E^2), where N and E are the number of nodes and edges respectively.

- `'Goldberg'` — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as *preflow-push*. Time complexity is $O(N^2*sqrt(E))$, where N and E are the number of nodes and edges respectively.

Examples

1 Create a directed graph with six nodes and eight edges.

```
cm = sparse([1 1 2 2 3 3 4 5],[2 3 4 5 4 5 6 6],...
    [2 3 3 1 1 1 2 3],6,6)cm =

   (1,2)        2
   (1,3)        3
   (2,4)        3
   (3,4)        1
   (2,5)        1
   (3,5)        1
   (4,6)        2
   (5,6)        3
```

2 Calculate the maximum flow in the graph from node 1 to node 6.

```
[M,F,K] = graphmaxflow(cm,1,6)

M =

      4


F =

   (1,2)        2
   (1,3)        2
   (2,4)        1
   (3,4)        1
   (2,5)        1
   (3,5)        1
   (4,6)        2
   (5,6)        2


K =
```

```
1    1    1    1    0    0
1    0    1    0    0    0
```

Notice that K is a two-row matrix because there are two possible solutions to the minimum cut problem.

3 View the graph with the original capacities.

```
h = view(biograph(cm,[],'ShowWeights','on'))
```
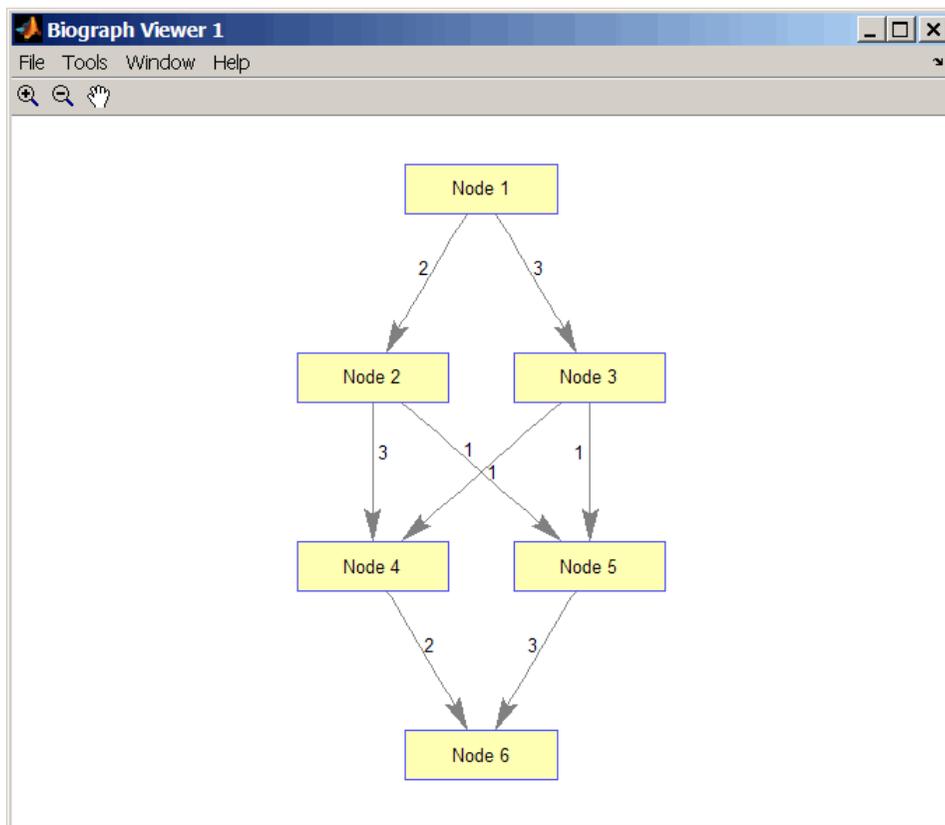

4 View the graph with the calculated maximum flows.

```
view(biograph(F,[],'ShowWeights','on'))
```
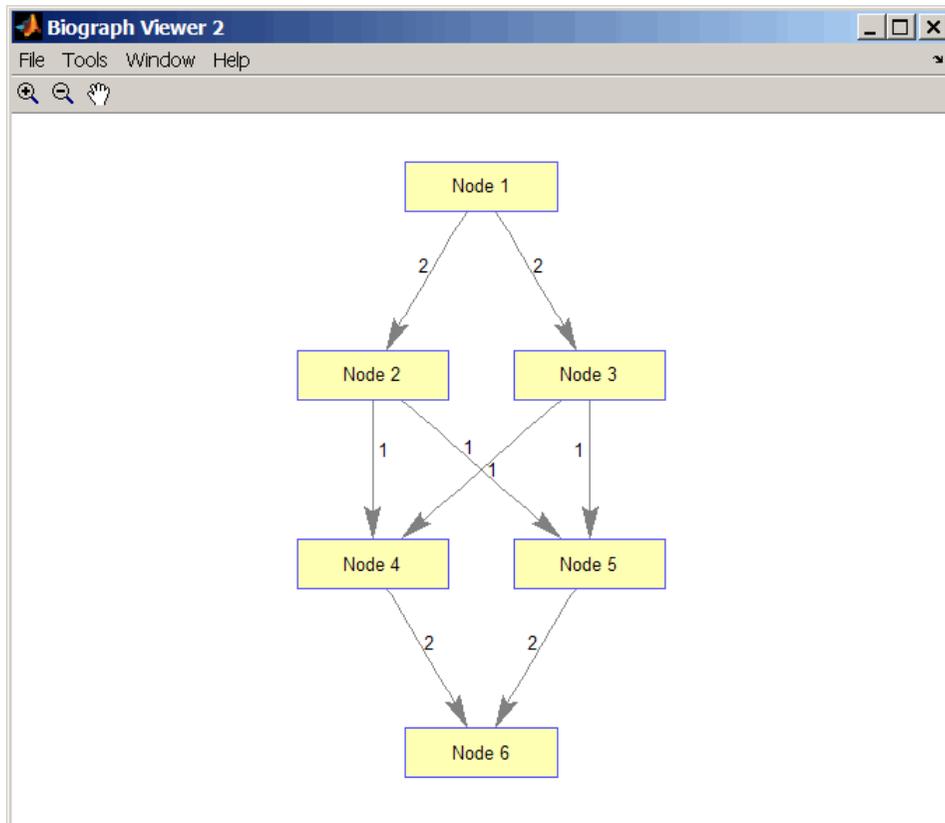
5 Show one solution to the minimum cut problem in the original graph.

```
set(h.Nodes(K(1,:)),'Color',[1 0 0])
```
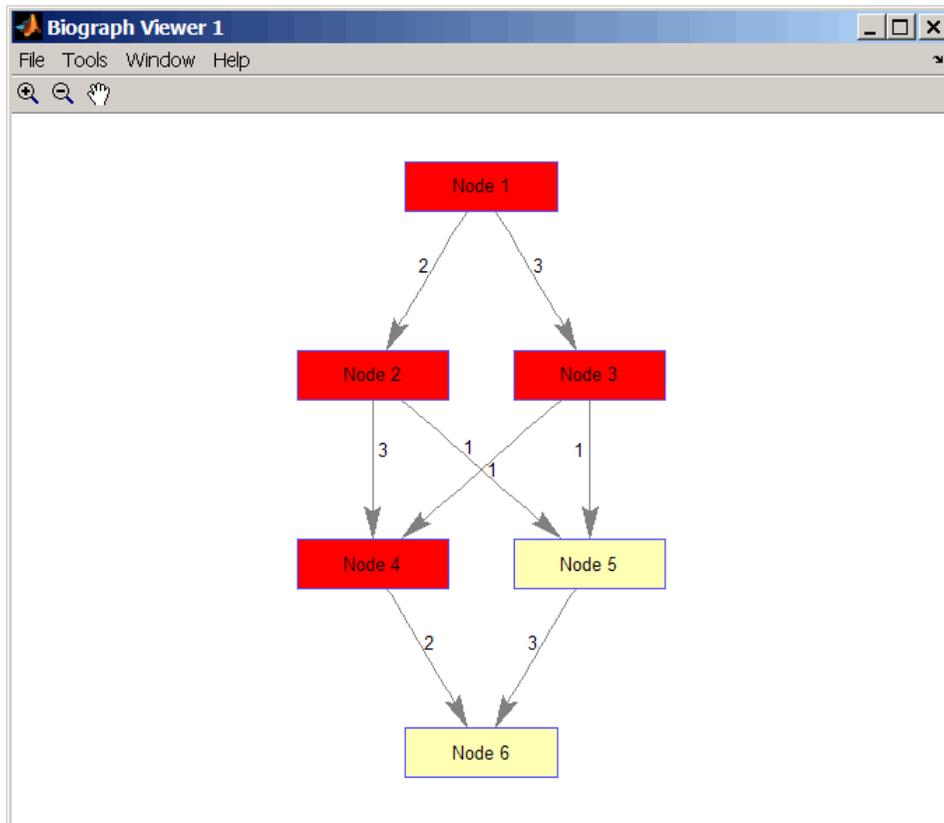
Notice that in the three edges that connect the source nodes (red) to the destination nodes (yellow), the original capacities and the calculated maximum flows are the same.

References

[1] Edmonds, J. and Karp, R.M. (1972). Theoretical improvements in the algorithmic efficiency for network flow problems. Journal of the ACM *19*, 248-264.

[2] Goldberg, A.V. (1985). A New Max-Flow Algorithm. MIT Technical Report MIT/LCS/ TM-291, Laboratory for Computer Science, MIT.

[3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

graphallshortestpaths | maxflow | graphconncomp | graphisdag |
graphisomorphism | graphisspantree | graphminspantree | graphpred2path |
graphshortestpath | graphtopoorder | graphtraverse

graphminspantree

Find minimal spanning tree in graph

Syntax

```
[Tree, pred] = graphminspantree(G)
[Tree, pred] = graphminspantree(G, R)
[Tree, pred] = graphminspantree(..., 'Method', MethodValue, ...)
[Tree, pred] = graphminspantree(..., 'Weights', WeightsValue, ...)
```

Arguments

G	N-by-N sparse matrix that represents an undirected graph. Nonzero entries in matrix *G* represent the weights of the edges.
R	Scalar between 1 and the number of nodes.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*Tree*, *pred*] = graphminspantree(*G*) finds an acyclic subset of edges that connects all the nodes in the undirected graph *G* and for which the total weight is minimized. Weights of the edges are all nonzero entries in the lower triangle of the N-by-N sparse matrix *G*. Output *Tree* is a spanning tree represented by a sparse matrix. Output *pred* is a vector containing the predecessor nodes of the minimal spanning tree (MST), with the root node indicated by 0. The root node defaults to the first node in the largest connected component. This computation requires an extra call to the graphconncomp function.

[*Tree*, *pred*] = graphminspantree(*G*, *R*) sets the root of the minimal spanning tree to node R.

[*Tree*,
pred] = graphminspantree(..., '*PropertyName*', *PropertyValue*, ...)
calls graphminspantree with optional properties that use property name/property
value pairs. You can specify one or more properties in any order. Each *PropertyName*
must be enclosed in single quotes and is case insensitive. These property name/property
value pairs are as follows:

[*Tree*, *pred*] = graphminspantree(..., 'Method', *MethodValue*, ...) lets
you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- 'Kruskal' — Grows the minimal spanning tree (MST) one edge at a time by
  finding an edge that connects two trees in a spreading forest of growing MSTs. Time
  complexity is O(E+X*log(N)), where X is the number of edges no longer than the
  longest edge in the MST, and N and E are the number of nodes and edges respectively.

- 'Prim' — Default algorithm. Grows the minimal spanning tree (MST) one edge at
  a time by adding a minimal edge that connects a node in the growing MST with any
  other node. Time complexity is O(E*log(N)), where N and E are the number of nodes
  and edges respectively.

Note: When the graph is unconnected, Prim's algorithm returns only the tree that
contains R, while Kruskal's algorithm returns an MST for every component.

[*Tree*, *pred*] = graphminspantree(..., 'Weights', *WeightsValue*, ...)
lets you specify custom weights for the edges. *WeightsValue* is a column vector having
one entry for every nonzero value (edge) in matrix *G*. The order of the custom weights in
the vector must match the order of the nonzero values in matrix *G* when it is traversed
column-wise. By default, graphminspantree gets weight information from the nonzero
entries in matrix *G*.

Examples

1 Create and view an undirected graph with 6 nodes and 11 edges.

```
W = [.41 .29 .51 .32 .50 .45 .38 .32 .36 .29 .21];
DG = sparse([1 1 2 2 3 4 4 5 5 6 6],[2 6 3 5 4 1 6 3 4 2 5],W);

UG = tril(DG + DG')
UG =
```

```
  (2,1)        0.4100
  (4,1)        0.4500
  (6,1)        0.2900
  (3,2)        0.5100
  (5,2)        0.3200
  (6,2)        0.2900
  (4,3)        0.5000
  (5,3)        0.3200
  (5,4)        0.3600
  (6,4)        0.3800
  (6,5)        0.2100
```

```
view(biograph(UG,[],'ShowArrows','off','ShowWeights','on'))
```
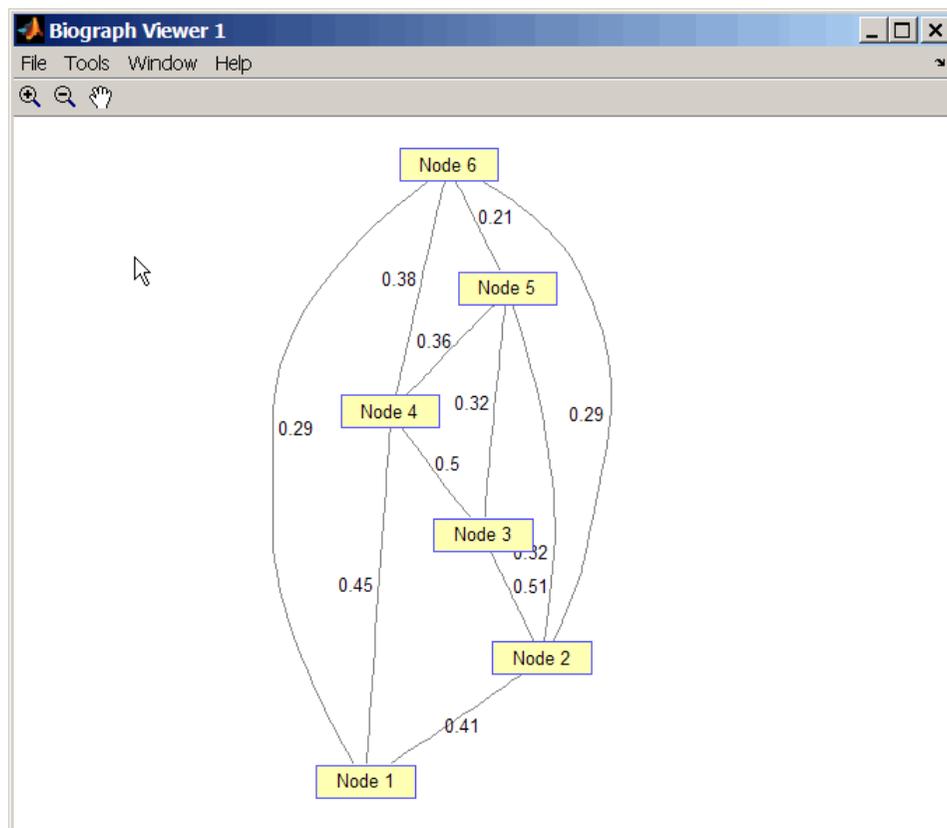
2 Find and view the minimal spanning tree of the undirected graph.

```
[ST,pred] = graphminspantree(UG)

ST =

    (6,1)        0.2900
    (6,2)        0.2900
    (5,3)        0.3200
    (5,4)        0.3600
    (6,5)        0.2100


pred =

     0     6     5     5     6     1

view(biograph(ST,[],'ShowArrows','off','ShowWeights','on'))
```
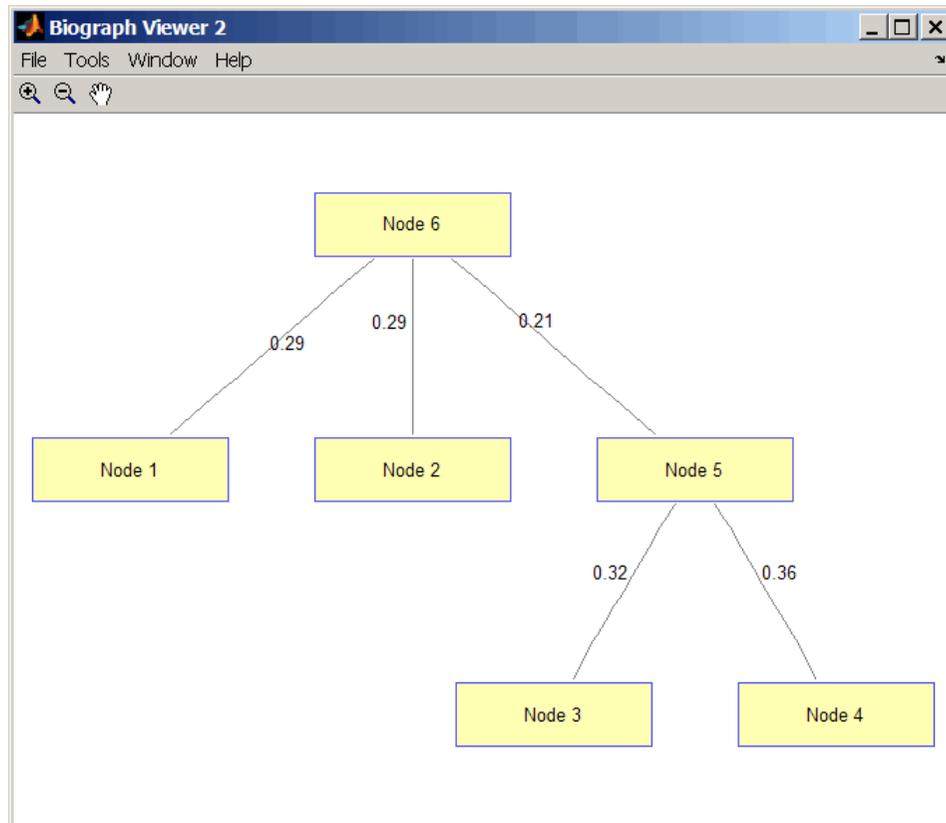
References

[1] Kruskal, J.B. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society *7*, 48-50.

[2] Prim, R. (1957). Shortest Connection Networks and Some Generalizations. Bell System Technical Journal *36*, 1389-1401.

[3] Siek, J.G. Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

graphallshortestpaths | minspantree | graphconncomp | graphisdag |
graphisomorphism | graphisspantree | graphmaxflow | graphpred2path |
graphshortestpath | graphtopoorder | graphtraverse

graphpred2path

Convert predecessor indices to paths

Syntax

path = graphpred2path(*pred*, *D*)

Arguments

pred	Row vector or matrix of predecessor node indices. The value of the root (or source) node in *pred* must be 0.
D	Destination node in *pred*.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

---

path = graphpred2path(*pred*, *D*) traces back a path by following the predecessor list in *pred* starting at destination node *D*.

The value of the root (or source) node in *pred* must be 0. If a NaN is found when following the predecessor nodes, graphpred2path returns an empty path.

If *pred* is a ...	And *D* is a ...	Then *path* is a ...
row vector of predecessor node indices	scalar	row vector listing the nodes from the root (or source) to *D*.
	row vector	row cell array with every column containing the path to the destination for every element in *D*.

If *pred* is a ...	And *D* is a ...	Then *path* is a ...
matrix	scalar	column cell array with every row containing the path for every row in *pred*.
	row vector	matrix cell array with every row containing the paths for the respective row in *pred*, and every column containing the paths to the respective destination in *D*.

Note: If *D* is omitted, the paths to all the destinations are calculated for every predecessor listed in *pred*.

Examples

1   Create a phytree object from the phylogenetic tree file for the GLR_HUMAN protein.

```
tr = phytreeread('pf00002.tree')
    Phylogenetic tree object with 33 leaves (32 branches)
```

2   View the phytree object.

```
view(tr)
```
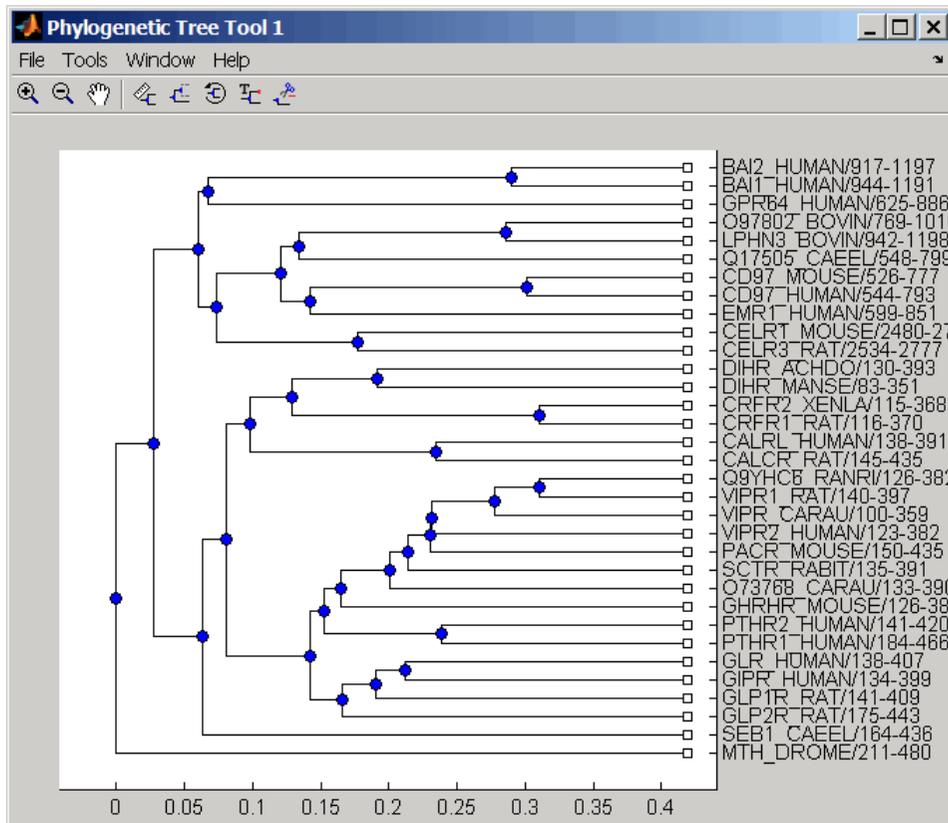
3 From the phytree object, create a connection matrix to represent the phylogenetic tree.

```
[CM,labels,dist] = getmatrix(tr);
```

4 Find the nodes from the root to one leaf in the phylogenetic tree created from the phylogenetic tree file for the GLR_HUMAN protein.

```
root_loc = size(CM,1)

root_loc =

    65

glr_loc = strncmp('GLR',labels,3);
```

```
 glr_loc_ind = find(glr_loc)

glr_loc_ind =

    12

[T,PRED]=graphminspantree(CM,root_loc);
PATH = graphpred2path(PRED,glr_loc_ind)

PATH =

    65    64    53    52    46    45    44    43    12
```

References

[1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

graphallshortestpaths | graphconncomp | graphisdag | graphisomorphism | graphisspantree | graphmaxflow | graphminspantree | graphshortestpath | graphtopoorder | graphtraverse

graphshortestpath

Solve shortest path problem in graph

Syntax

```
[dist, path, pred] = graphshortestpath(G, S)
[dist, path, pred] = graphshortestpath(G, S, T)
[...] = graphshortestpath(..., 'Directed', DirectedValue, ...)
[...] = graphshortestpath(..., 'Method', MethodValue, ...)
[...] = graphshortestpath(..., 'Weights', WeightsValue, ...)
```

Arguments

G	N-by-N sparse matrix that represents a graph. Nonzero entries in matrix *G* represent the weights of the edges.
S	Node in *G*.
T	Node in *G*.
DirectedValue	Property that indicates whether the graph is directed or undirected. Enter `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`.
MethodValue	String that specifies the algorithm used to find the shortest path. Choices are: • `'Bellman-Ford'` — Assumes weights of the edges to be nonzero entries in sparse matrix `G`. Time complexity is `O(N*E)`, where `N` and `E` are the number of nodes and edges respectively. • `'BFS'` — Breadth-first search. Assumes all weights to be equal, and nonzero entries in sparse matrix `G` to represent edges. Time complexity is `O(N+E)`, where `N` and `E` are the number of nodes and edges respectively. • `'Acyclic'` — Assumes `G` to be a directed acyclic graph and that weights of the edges are nonzero entries in sparse matrix

	G. Time complexity is `O(N+E)`, where `N` and `E` are the number of nodes and edges respectively.
	• `'Dijkstra'` — Default algorithm. Assumes weights of the edges to be positive values in sparse matrix `G`. Time complexity is `O(log(N)*E)`, where `N` and `E` are the number of nodes and edges respectively.
WeightsValue	Column vector that specifies custom weights for the edges in matrix *G*. It must have one entry for every nonzero value (edge) in matrix *G*. The order of the custom weights in the vector must match the order of the nonzero values in matrix *G* when it is traversed column-wise. This property lets you use zero-valued weights. By default, `graphshortestpaths` gets weight information from the nonzero entries in matrix *G*.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*dist*, *path*, *pred*] = graphshortestpath(*G*, *S*) determines the single-source shortest paths from node *S* to all other nodes in the graph represented by matrix *G*. Input *G* is an N-by-N sparse matrix that represents a graph. Nonzero entries in matrix *G* represent the weights of the edges. *dist* are the N distances from the source to every node (using `Inf`s for nonreachable nodes and `0` for the source node). *path* contains the winning paths to every node. *pred* contains the predecessor nodes of the winning paths.

[*dist*, *path*, *pred*] = graphshortestpath(*G*, *S*, *T*) determines the single source-single destination shortest path from node *S* to node *T*.

[...] = graphshortestpath(..., '*PropertyName*', *PropertyValue*, ...) calls graphshortestpath with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[...] = graphshortestpath(..., 'Directed', *DirectedValue*, ...) indicates whether the graph is directed or undirected. Set *DirectedValue* to false

for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`.

`[...] = graphshortestpath(..., 'Method', MethodValue, ...)` lets you specify the algorithm used to find the shortest path. Choices are:

- `'Bellman-Ford'` — Assumes weights of the edges to be nonzero entries in sparse matrix `G`. Time complexity is `O(N*E)`, where `N` and `E` are the number of nodes and edges respectively.
- `'BFS'` — Breadth-first search. Assumes all weights to be equal, and nonzero entries in sparse matrix `G` to represent edges. Time complexity is `O(N+E)`, where `N` and `E` are the number of nodes and edges respectively.
- `'Acyclic'` — Assumes `G` to be a directed acyclic graph and that weights of the edges are nonzero entries in sparse matrix `G`. Time complexity is `O(N+E)`, where `N` and `E` are the number of nodes and edges respectively.
- `'Dijkstra'` — Default algorithm. Assumes weights of the edges to be positive values in sparse matrix `G`. Time complexity is `O(log(N)*E)`, where `N` and `E` are the number of nodes and edges respectively.

`[...] = graphshortestpath(..., 'Weights', WeightsValue, ...)` lets you specify custom weights for the edges. *WeightsValue* is a column vector having one entry for every nonzero value (edge) in matrix *G*. The order of the custom weights in the vector must match the order of the nonzero values in matrix *G* when it is traversed column-wise. This property lets you use zero-valued weights. By default, `graphshortestpath` gets weight information from the nonzero entries in matrix *G*.

Examples

Finding the Shortest Path in a Directed Graph

1 Create and view a directed graph with 6 nodes and 11 edges.

```
W = [.41 .99 .51 .32 .15 .45 .38 .32 .36 .29 .21];
DG = sparse([6 1 2 2 3 4 4 5 5 6 1],[2 6 3 5 4 1 6 3 4 3 5],W)

DG =

    (4,1)        0.4500
    (6,2)        0.4100
    (2,3)        0.5100
```

```
(5,3)        0.3200
(6,3)        0.2900
(3,4)        0.1500
(5,4)        0.3600
(1,5)        0.2100
(2,5)        0.3200
(1,6)        0.9900
(4,6)        0.3800
```

```
h = view(biograph(DG,[],'ShowWeights','on'))
Biograph object with 6 nodes and 11 edges.
```
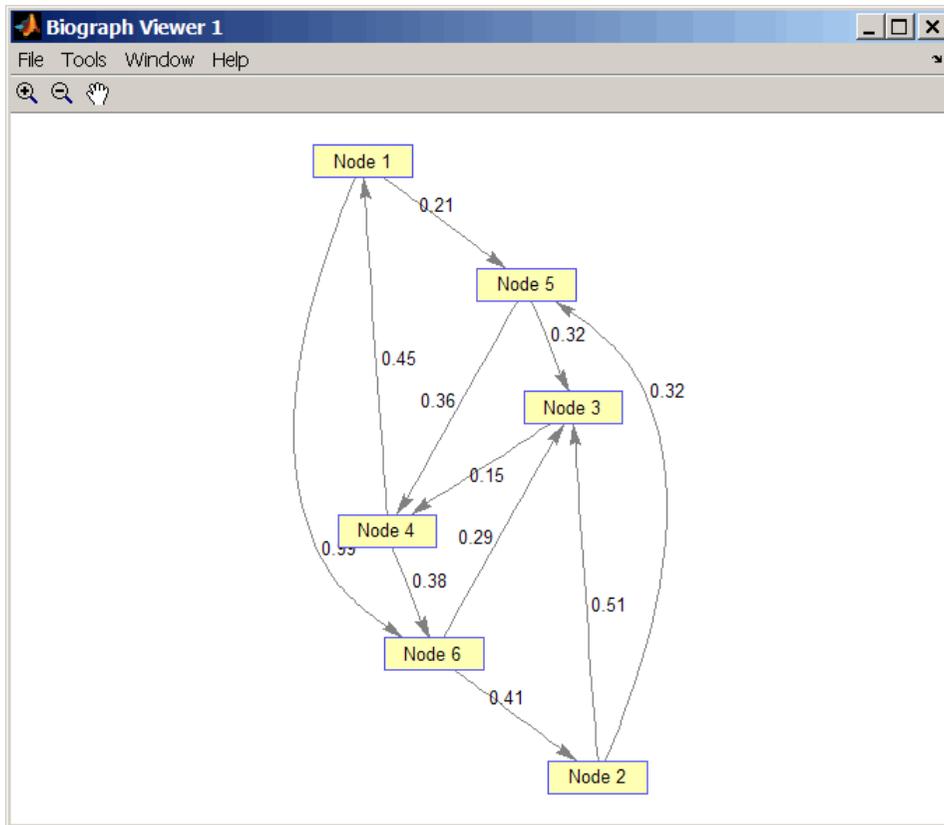

2   Find the shortest path in the graph from node 1 to node 6.

```
[dist,path,pred] = graphshortestpath(DG,1,6)
```

```
dist =

    0.9500


path =

    1     5     4     6


pred =

    0     6     5     5     1     4
```

3 Mark the nodes and edges of the shortest path by coloring them red and increasing the line width.

```
set(h.Nodes(path),'Color',[1 0.4 0.4])
edges = getedgesbynodeid(h,get(h.Nodes(path),'ID'));
set(edges,'LineColor',[1 0 0])
set(edges,'LineWidth',1.5)
```
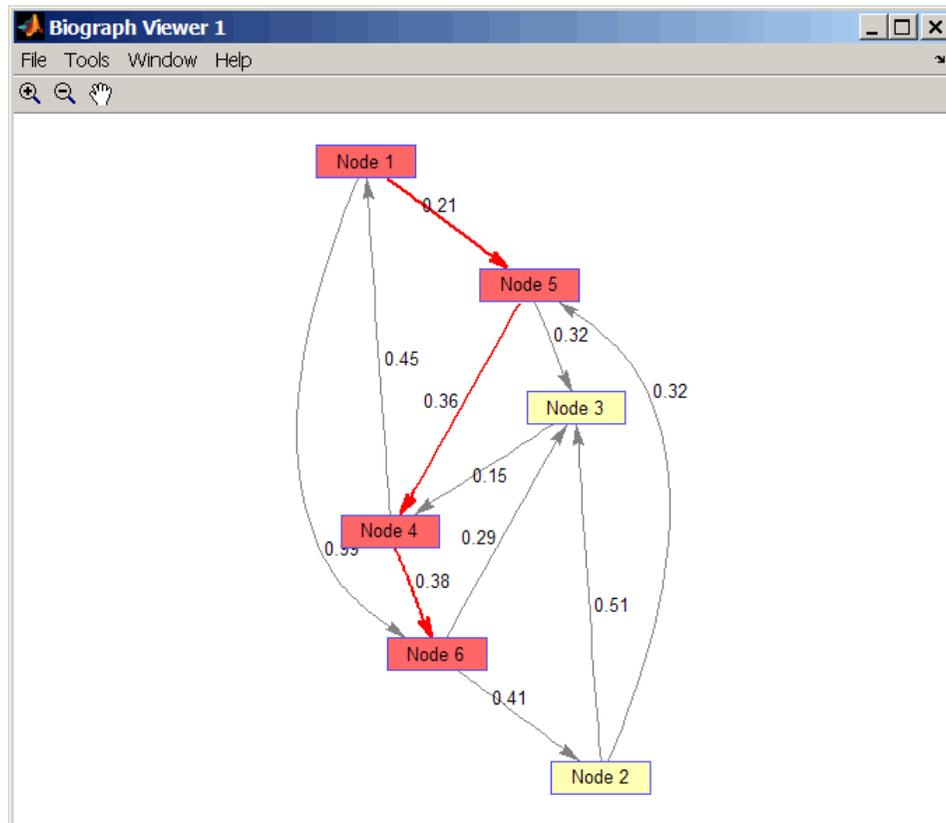
Finding the Shortest Path in an Undirected Graph

1 Create and view an undirected graph with 6 nodes and 11 edges.

```
UG = tril(DG + DG')

UG =

    (4,1)        0.4500
    (5,1)        0.2100
    (6,1)        0.9900
    (3,2)        0.5100
    (5,2)        0.3200
    (6,2)        0.4100
```

```
(4,3)          0.1500
(5,3)          0.3200
(6,3)          0.2900
(5,4)          0.3600
(6,4)          0.3800

h = view(biograph(UG,[],'ShowArrows','off','ShowWeights','on'))
Biograph object with 6 nodes and 11 edges.
```
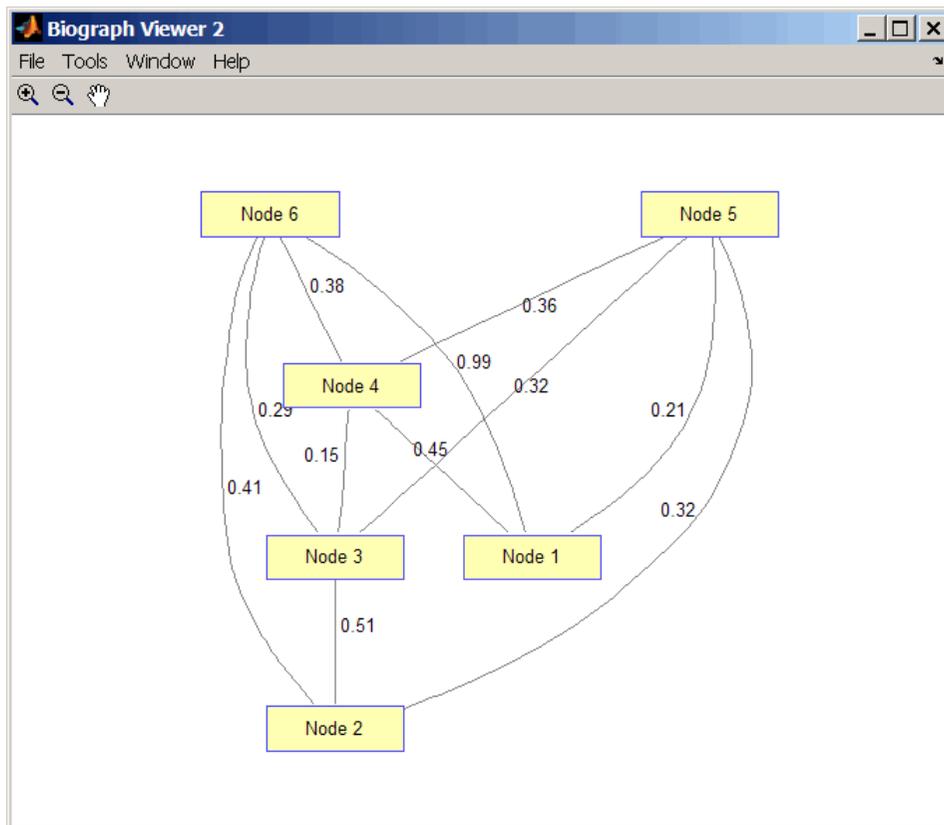

2 Find the shortest path in the graph from node 1 to node 6.

```
[dist,path,pred] = graphshortestpath(UG,1,6,'directed',false)

dist =
```

```
    0.8200


path =

    1    5    3    6


pred =

    0    5    5    1    1    3
```

3 Mark the nodes and edges of the shortest path by coloring them red and increasing the line width.

```
set(h.Nodes(path),'Color',[1 0.4 0.4])
 fowEdges = getedgesbynodeid(h,get(h.Nodes(path),'ID'));
 revEdges = getedgesbynodeid(h,get(h.Nodes(fliplr(path)),'ID'));
 edges = [fowEdges;revEdges];
 set(edges,'LineColor',[1 0 0])
 set(edges,'LineWidth',1.5)
```
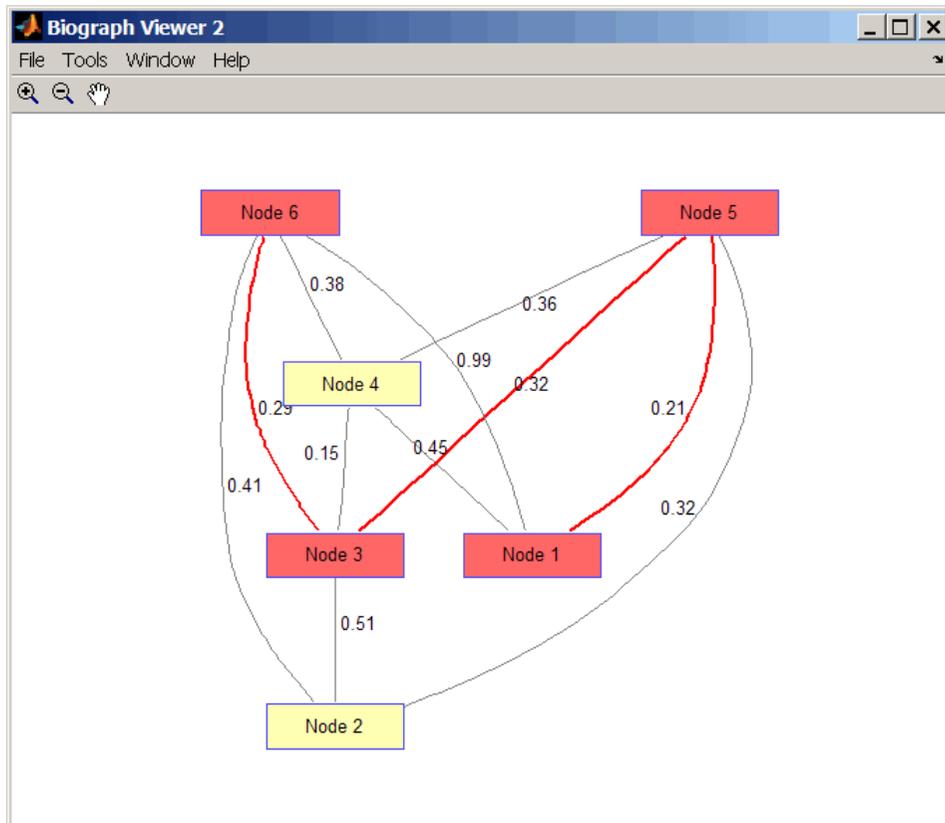
References

[1] Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik *1*, 269-271.

[2] Bellman, R. (1958). On a Routing Problem. Quarterly of Applied Mathematics *16(1)*, 87-90.

[3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also
graphallshortestpaths | shortestpath | graphconncomp | graphisdag | graphisomorphism | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphtopoorder | graphtraverse

graphtopoorder

Perform topological sort of directed acyclic graph

Syntax

order = graphtopoorder(*G*)

Arguments

G	N-by-N sparse matrix that represents a directed acyclic graph. Nonzero entries in matrix *G* indicate the presence of an edge.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

---

order = graphtopoorder(*G*) returns an index vector with the order of the nodes sorted topologically. In topological order, an edge can exist between a source node u and a destination node v, if and only if u appears before v in the vector *order*. *G* is an N-by-N sparse matrix that represents a directed acyclic graph (DAG). Nonzero entries in matrix *G* indicate the presence of an edge.

Examples

1 Create and view a directed acyclic graph (DAG) with six nodes and eight edges.

```
DG = sparse([6 6 6 2 2 3 5 1],[2 5 1 3 4 5 1 4],true,6,6)

DG =

    (5,1)        1
```

```
(6,1)        1
(6,2)        1
(2,3)        1
(1,4)        1
(2,4)        1
(3,5)        1
(6,5)        1
```

```
view(biograph(DG))
```
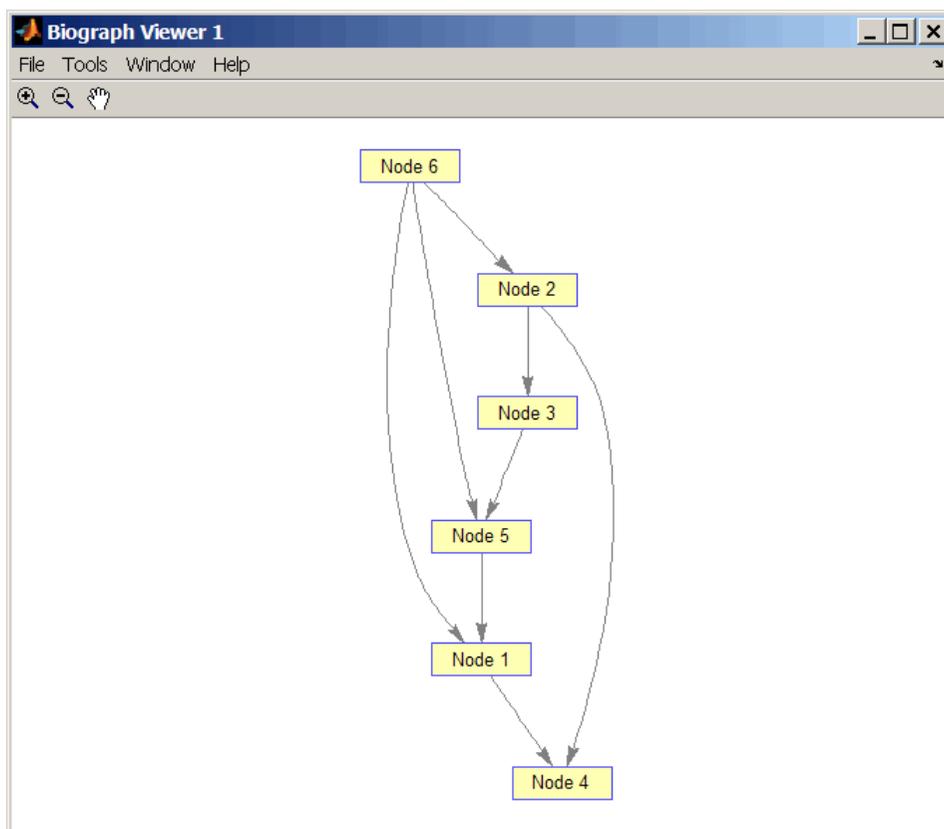

2 Find the topological order of the DAG.

```
order = graphtopoorder(DG)
```

```
order =
```

```
        6    2    3    5    1    4
```

3 Permute the nodes so that they appear ordered in the graph display.

```
DG = DG(order,order)

DG =

   (1,2)        1
   (2,3)        1
   (1,4)        1
   (3,4)        1
   (1,5)        1
   (4,5)        1
   (2,6)        1
   (5,6)        1

view(biograph(DG))
```
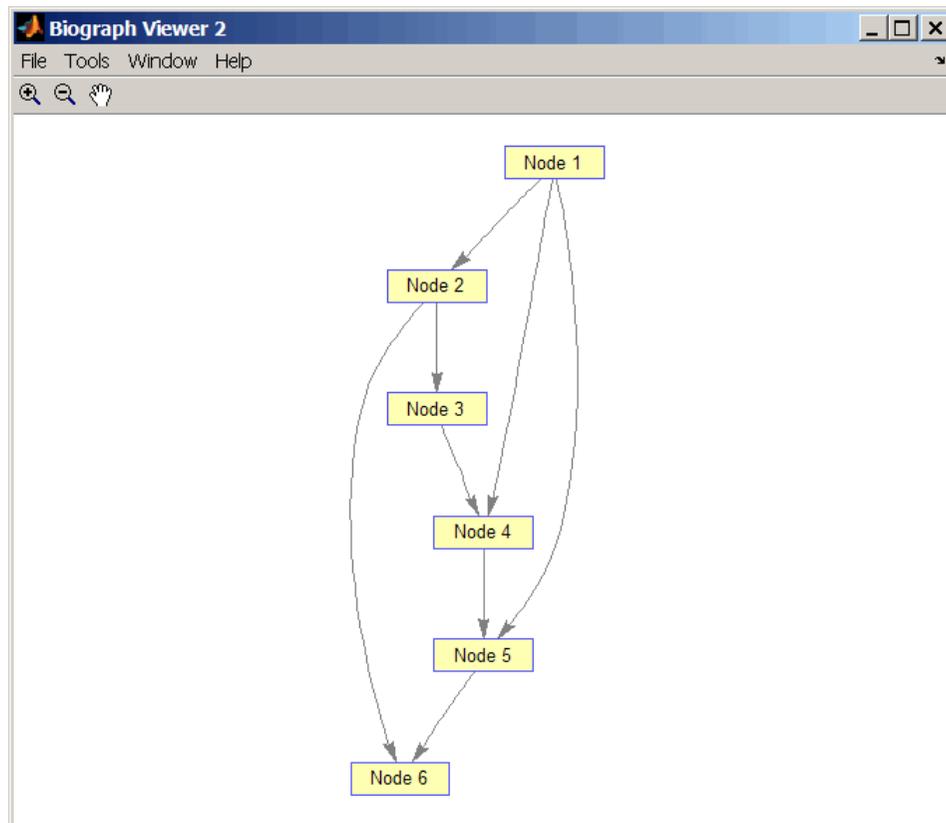
References

[1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

graphallshortestpaths | topoorder | graphconncomp | graphisdag | graphisomorphism | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphshortestpath | graphtraverse

graphtraverse

Traverse graph by following adjacent nodes

Syntax

```
[disc, pred, closed] = graphtraverse(G, S)
[...] = graphtraverse(G, S, ...'Depth', DepthValue, ...)
[...] = graphtraverse(G, S, ...'Directed', DirectedValue, ...)
[...] = graphtraverse(G, S, ...'Method', MethodValue, ...)
```

Arguments

G	N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix *G* indicate the presence of an edge.
S	Integer that indicates the source node in graph *G*.
DepthValue	Integer that indicates a node in graph *G* that specifies the depth of the search. Default is Inf (infinity).
DirectedValue	Property that indicates whether graph *G* is directed or undirected. Enter false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.
MethodValue	String that specifies the algorithm used to traverse the graph. Choices are: • 'BFS' — Breadth-first search. Time complexity is O(N+E), where N and E are number of nodes and edges respectively. • 'DFS' — Default algorithm. Depth-first search. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*disc*, *pred*, *closed*] = graphtraverse(*G*, *S*) traverses graph G starting from the node indicated by integer S. *G* is an N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix *G* indicate the presence of an edge. *disc* is a vector of node indices in the order in which they are discovered. *pred* is a vector of predecessor node indices (listed in the order of the node indices) of the resulting spanning tree. *closed* is a vector of node indices in the order in which they are closed.

[...] = graphtraverse(*G*, *S*, ...'*PropertyName*', *PropertyValue*, ...) calls graphtraverse with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[...] = graphtraverse(*G*, *S*, ...'Depth', *DepthValue*, ...) specifies the depth of the search. *DepthValue* is an integer indicating a node in graph *G*. Default is Inf (infinity).

[...] = graphtraverse(*G*, *S*, ...'Directed', *DirectedValue*, ...) indicates whether the graph is directed or undirected. Set *DirectedValue* to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

[...] = graphtraverse(*G*, *S*, ...'Method', *MethodValue*, ...) lets you specify the algorithm used to traverse the graph. Choices are:

- 'BFS' — Breadth-first search. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.
- 'DFS' — Default algorithm. Depth-first search. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.

Examples

1 Create a directed graph with 10 nodes and 12 edges.

```
DG = sparse([1 2 3 4 5 5 5 6 7 8 8 9],...
             [2 4 1 5 3 6 7 9 8 1 10 2],true,10,10)

DG =

   (3,1)        1
   (8,1)        1
   (1,2)        1
   (9,2)        1
   (5,3)        1
   (2,4)        1
   (4,5)        1
   (5,6)        1
   (5,7)        1
   (7,8)        1
   (6,9)        1
   (8,10)       1

h = view(biograph(DG))
Biograph object with 10 nodes and 12 edges.
```
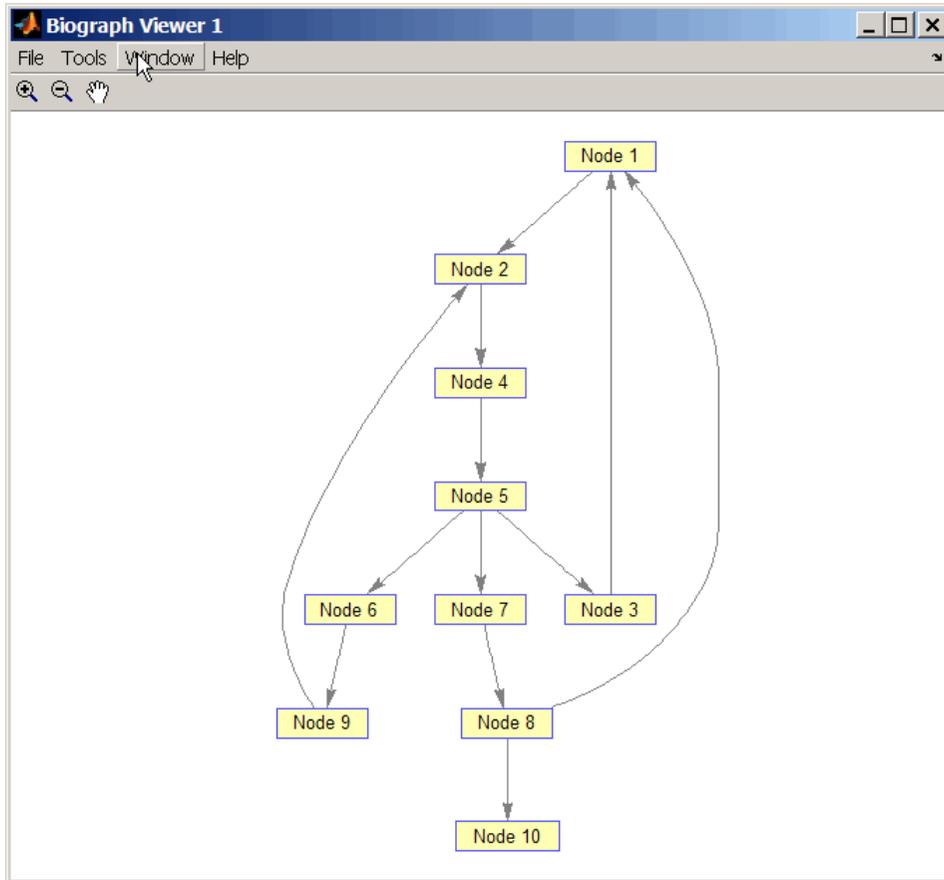
2   Traverse the graph to find the depth-first search (DFS) discovery order starting at node 4.

```
order = graphtraverse(DG,4)

order =

     4     5     3     1     2     6     9     7     8    10
```

3   Label the nodes with the DFS discovery order.

```
for i = 1:10
    h.Nodes(order(i)).Label =...
```

```
    sprintf('%s:%d',h.Nodes(order(i)).ID,i);
end
h.ShowTextInNodes = 'label'
dolayout(h)
```
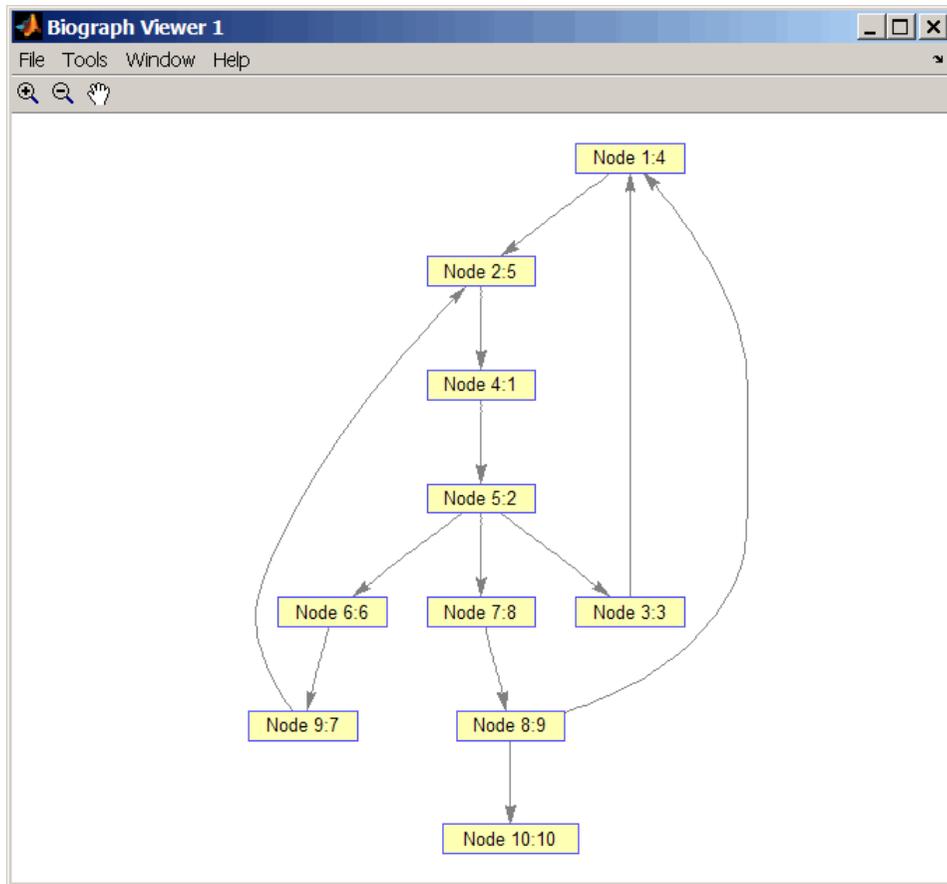

4   Traverse the graph to find the breadth-first search (BFS) discovery order starting at node 4.

```
order = graphtraverse(DG,4,'Method','BFS')

order =

    4    5    3    6    7    1    9    8    2    10
```

5 Label the nodes with the BFS discovery order.

```
for i = 1:10
    h.Nodes(order(i)).Label =...
    sprintf('%s:%d',h.Nodes(order(i)).ID,i);
end
h.ShowTextInNodes = 'label'
dolayout(h)
```
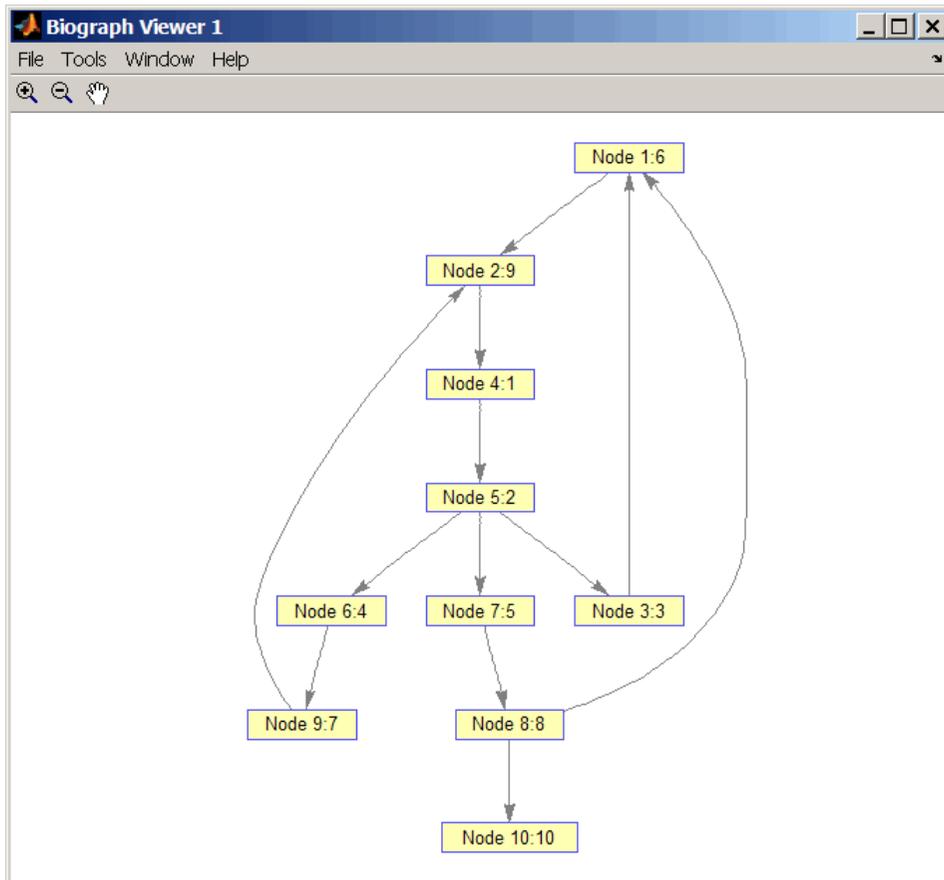

6 Find and color nodes that are close to (within two edges of) node 4.

```
node_idxs = graphtraverse(DG,4,'depth',2)
```

```
node_idxs =

     4     5     3     6     7

set(h.nodes(node_idxs),'Color',[1 0 0])
```
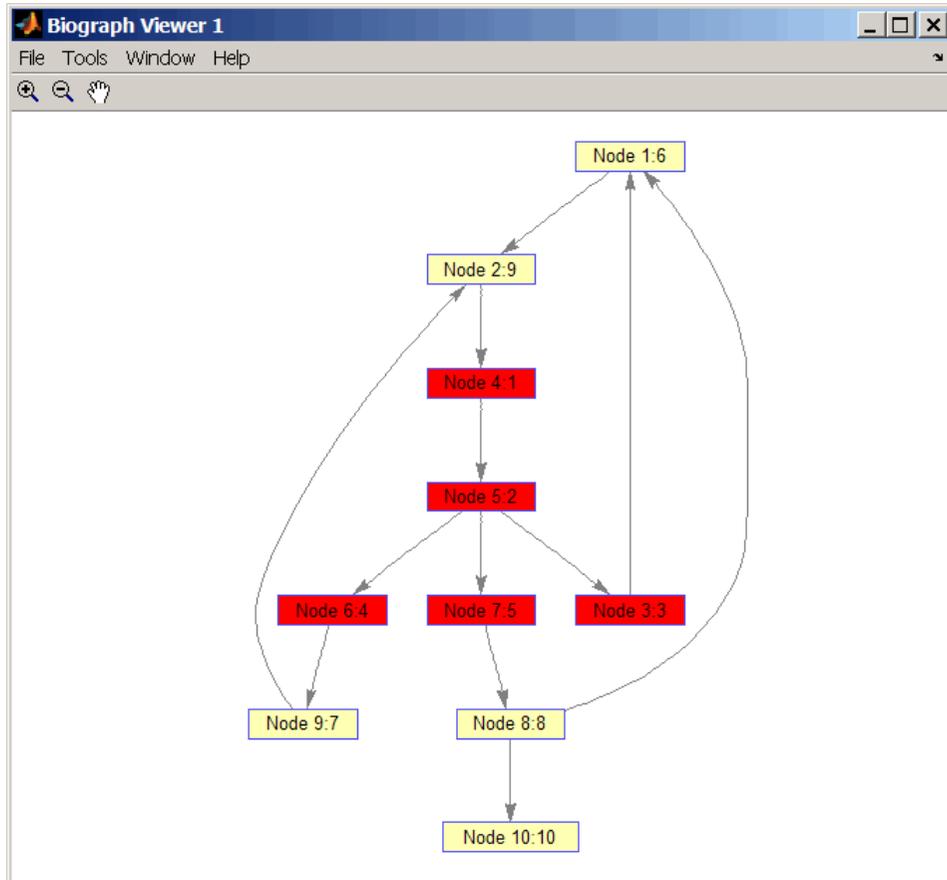

References

[1] Sedgewick, R., (2002). Algorithms in C++, Part 5 Graph Algorithms (Addison-Wesley).

[2] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

graphallshortestpaths | traverse | graphconncomp | graphisdag | graphisomorphism | graphisspantree | graphmaxflow | graphminspantree | graphpred2path | graphshortestpath | graphtopoorder

gt (DataMatrix)

Test DataMatrix objects for greater than

Syntax

T = gt(*DMObj1*, *DMObj2*)
T = *DMObj1* > *DMObj2*
T = gt(*DMObj1*, *B*)
T = *DMObj1* > *B*
T = gt(*B*, *DMObj1*)
T = *B* > *DMObj1*

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

T	Logical matrix of the same size as *DMObj1* and *DMObj2* or *DMObj1* and *B*. It contains logical 1 (true) where elements in the first input are greater than the corresponding element in the second input, and logical 0 (false) otherwise.

Description

T = gt(*DMObj1*, *DMObj2*) or the equivalent *T* = *DMObj1* > *DMObj2* compares each element in DataMatrix object *DMObj1* to the corresponding element in DataMatrix object *DMObj2*, and returns *T*, a logical matrix of the same size as *DMObj1* and *DMObj2*, containing logical 1 (true) where elements in *DMObj1* are greater than the corresponding element in *DMObj2*, and logical 0 (false) otherwise. *DMObj1* and *DMObj2* must have

the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). *DMObj1* and *DMObj2* can have different `Name` properties.

T = `gt(`*DMObj1*, *B*`)` or the equivalent *T* = *DMObj1* > *B* compares each element in DataMatrix object *DMObj1* to the corresponding element in *B*, a numeric or logical array, and returns *T*, a logical matrix of the same size as *DMObj1* and *B*, containing logical `1` (true) where elements in *DMObj1* are greater than the corresponding element in *B*, and logical `0` (false) otherwise. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

T = `gt(`*B*, *DMObj1*`)` or the equivalent *T* = *B* > *DMObj1* compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical `1` (true) where elements in *B* are greater than the corresponding element in *DMObj1*, and logical `0` (false) otherwise. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls *T* = `gt(`*X*, *Y*`)` for the syntax *T* = *X* > *Y* when *X* or *Y* is a DataMatrix object.

More About

- DataMatrix object

See Also
`DataMatrix` | `lt`

GTFAnnotation class

Represent Gene Transfer Format (GTF) annotations

Description

The `GTFAnnotation` class contains annotations for one or more reference sequences, conforming to the GTF file format.

You construct a `GTFAnnotation` object from a GTF-formatted file. Each element in the object represents an annotation. Use the object properties and methods to filter annotations by feature, reference sequence, or reference sequence position. Use object methods to extract data for a subset of annotations into an array of structures.

Construction

Annotobj = GTFAnnotation(*File*) constructs *Annotobj*, a GTFAnnotation object, from *File*, a GTF-formatted file.

Input Arguments

`File`

String specifying a GTF-formatted file.

Default:

Properties

`FieldNames`

Cell array of strings specifying the names of the available data fields for each annotation in the `GTFAnnotation` object. This property is read only.

`NumEntries`

Integer specifying number of annotations in the `GTFAnnotation` object. This property is read only.

Methods

getData

Create structure containing subset of data from `GTFAnnotation` object

getExons

Return table of exons from `GTFAnnotation` object

getFeatureNames

Retrieve unique feature names from `GTFAnnotation` object

getGenes

Return table of unique genes in `GTFAnnotation` object

getGeneNames

Retrieve unique gene names from `GTFAnnotation` object

getIndex

Return index array of annotations from `GTFAnnotation` object

getRange

Retrieve range of annotations from `GTFAnnotation` object

getReferenceNames

Retrieve reference names from `GTFAnnotation` object

getSegments

Return table of non-overlapping segments from `GTFAnnotation` object

getSubset

Create object containing subset of elements from `GTFAnnotation` object

getTranscripts

Return table of unique transcripts in `GTFAnnotation` object

getTranscriptNames

Retrieve unique transcript names from `GTFAnnotation` object

Copy Semantics

Value. To learn how value classes affect copy operations, see Copying Objects in the MATLAB documentation.

Indexing

`GTFAnnotation` objects support dot . indexing to extract properties.

Examples

Construct a `GTFAnnotation` object from a GTF-formatted file that is provided with Bioinformatics Toolbox:

```
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf')

GTFAnnotObj =

  GTFAnnotation with properties:

    FieldNames: {1x11 cell}
    NumEntries: 308
```

See Also
GFFAnnotation

How To
·   "Store and Manage Feature Annotations in Objects"

Related Links
·   GTF2.2: A Gene Annotation Format

HeatMap object

Object containing matrix and heat map display properties

Description

A HeatMap object contains data and display properties that you can view in a heat map (2-D color image).

Create a HeatMap object using the object constructor function `HeatMap`. View a graphical representation of the HeatMap object in a heat map using the `view` method.

The HeatMap class is a superclass of the clustergram class.

Method Summary

Following are methods of a HeatMap object:

addTitle (HeatMap)

           Add title to heat map

addXLabel (HeatMap)

           Label *x*-axis of heat map

addYLabel (HeatMap)

           Label *y*-axis of heat map

plot (HeatMap)

           Render heat map for HeatMap object

view (HeatMap)

           View heat map of HeatMap object

Property Summary

Properties for Heat Map Creation

Property Name	Description
Standardize	String or number specifying the dimension for standardizing the data values. The standardized values are transformed

Property Name	Description
	so that the mean is 0 and the standard deviation is 1 in the specified dimension. Choices are: • `'column'` or 1 — Standardize along the columns of data. • `'row'` or 2 — Standardize along the rows of data. • `'none'` or 3 (default) — Do not standardize.
Colormap	Either of the following: • *M*-by-3 matrix of RGB values • Name or function handle of a function that returns a colormap, such as `redgreencmap` or `redbluecmap` Default is `redgreencmap`.
DisplayRange	Positive scalar specifying the display range of standardized values. Default is the maximum absolute value in the input matrix. For example, if you specify 3, there is a color variation for values between –3 and 3, but values >3 are the same color as 3, and values < –3 are the same color as –3. For example, if you specify `redgreencmap` for the `'Colormap'` property, pure red represents values ≥ *DisplayRangeValue*, and pure green represents values ≤ –*DisplayRangeValue*.
Symmetric	Forces the color scale of the heat map to be symmetric around zero. Choices are `true` (default) or `false`.
ImputeFun	One of the following: • Name of a function that imputes missing data. • Handle to a function that imputes missing data. • Cell array where the first element is the name of or handle to a function that imputes missing data. The remaining elements are property name/property value pairs used as inputs to the function.

Properties for Row and Column Labels

Property Name	Description
RowLabels	Vector of numbers or cell array of text strings to label the rows in the heat map. Default is a vector of values 1 through *M*, where *M* is the number of rows in *Data*, the matrix of data used by the HeatMap function to create the HeatMap object.
ColumnLabels	Vector of numbers or cell array of text strings to label the columns in the heat map. Default is a vector of values 1 through *M*, where *M* is the number of columns in *Data*, the matrix of data used by the HeatMap function to create the HeatMap object.
ColumnLabelsLocation	String specifying the location of the column labels. Choices are 'top' or 'bottom' (default).
RowLabelsLocation	String specifying the location of the row labels. Choices are 'left' (default) or 'right'.
RowLabelsColor	Structure or structure array containing color information for labeling the rows (*y*-axis) of the heat map. If a single structure, then the fields contain a cell array of elements. If a structure array, then the fields contain one element: • Labels — String specifying a row label listed in the RowLabels vector. • Colors — String or three-element vector of RGB values specifying a color for the row label specified in the Labels field. For more information on specifying colors, see ColorSpec. If this field is empty, default colors are assigned to the row label.
ColumnLabelsColor	Structure or structure array containing color information for labeling the columns (*x*-axis) of the heat map. If a single structure, then the fields contain a cell array of elements. If a structure array, then the fields contain one element: • Labels — String specifying a column label listed in the ColumnLabels vector. • Colors — String or three-element vector of RGB values specifying a color for the column label specified in the

Property Name	Description
	`Labels` field. For more information on specifying colors, see `ColorSpec`. If this field is empty, default colors are assigned to the column label.
LabelsWithMarkers	Controls the display of colored markers instead of colored text for the row labels and column labels. Choices are `true` or `false` (default).
RowLabelsRotate	Numeric value in degrees rotation specifying the orientation of row (*y*-axis) labels. Default is 0 degrees, which is horizontal. Positive values cause counterclockwise rotation.
ColumnLabelsRotate	Numeric value in degrees rotation specifying the orientation of column (*x*-axis) labels. Default is 90 degrees, which is vertical. Values greater than 90 degrees cause counterclockwise rotation.

Properties for Annotating Data

Property Name	Description
Annotate	Controls the display of intensity values on each area of the heat map. Choices are `true` or `false` (default).
AnnotPrecision	Positive integer specifying the precision of the intensity values when displayed on the heat map. Default is 2.
AnnotColor	String or three-element vector of RGB values specifying a color for the text of the intensity values when displayed on the heat map. Default is `'white'`. For more information on specifying colors, see `ColorSpec`.

Examples

Note: The following examples use the `get` and `set` methods with property names and values of a HeatMap object. When supplying a *PropertyName*, be aware that it is case sensitive.

Determining Properties and Property Values of a HeatMap Object

Display all properties and their current values of a HeatMap object, *HMobj*:

get(*HMobj*)

Return all properties and their current values of *HMobj*, a HeatMap object, to *CGstruct*, a scalar structure in which each field name is a property of a HeatMap object, and each field contains the value of that property.

CGstruct = get(*HMobj*)

Return the value of a specific property of a HeatMap object, *HMobj*, using either:

PropertyValue = get(*HMobj*, '*PropertyName*')

PropertyValue = *HMobj.PropertyName*

Return the value of specific properties of a HeatMap object, *HMobj*:

[*Property1Value*, *Property2Value*, ...] = get(*HMobj*, ...
'*Property1Name*', '*Property2Name*', ...)

Determining Possible Values of HeatMap Object Properties

Display possible values for all properties that have a fixed set of property values in a HeatMap object, *HMobj*:

set(*HMobj*)

Display possible values for a specific property that has a fixed set of property values in a HeatMap object, *HMobj*:

set(*HMobj*, '*PropertyName*')

Specifying Properties of a HeatMap Object

Set a specific property of a HeatMap object, *HMobj*, using either:

set(*HMobj*, '*PropertyName*', *PropertyValue*)

HMobj.PropertyName = *PropertyValue*

Set multiple properties of a HeatMap object, *HMobj*:

set(*HMobj*, '*Property1Name*', *Property1Value*, ...

```
'Property2Name', Property2Value, ...)
```

See Also
HeatMap | addXLabel | plot | view | addTitle | addYLabel | display

HeatMap

Display heat map of matrix data and create HeatMap object

Syntax

```
HMobj = HeatMap(Data)
HeatMap(Data, ...'RowLabels', RowLabelsValue, ...)
HeatMap(Data, ...'ColumnLabels', ColumnLabelsValue, ...)
HeatMap(Data, ...'Standardize', StandardizeValue, ...)
HeatMap(Data, ...'Colormap', ColormapValue, ...)
HeatMap(Data, ...'DisplayRange', DisplayRangeValue, ...)
HeatMap(Data, ...'Symmetric', SymmetricValue, ...)
HeatMap(Data, ...'ImputeFun', ImputeFunValue, ...)
HeatMap(Data, ...'RowLabelsColor', RowLabelsColorValue, ...)
HeatMap(Data, ...'ColumnLabelsColor', ColumnLabelsColorValue, ...)
HeatMap(Data, ...'LabelsWithMarkers', LabelsWithMarkersValue, ...)
```

Arguments

Data	DataMatrix object or numeric matrix of data.
RowLabelsValue	Vector of numbers or cell array of text strings to label the rows in the heat map. Default is a vector of values 1 through *M*, where *M* is the number of rows in *Data*.
ColumnLabelsValue	Vector of numbers or cell array of text strings to label the columns in the heat map. Default is a vector of values 1 through *N*, where *N* is the number of columns in *Data*.
StandardizeValue	String or number specifying the dimension for standardizing the values in *Data*. The HeatMap function transforms the standardized values so that the mean is 0 and the standard deviation is 1 in the specified dimension. Choices are: • `'column'` or 1 — Standardize along the columns of data.

	• `'row'` or `2` — Standardize along the rows of data. • `'none'` or `3` (default) — Do not standardize.
ColormapValue	Either of the following: • *M*-by-3 matrix of RGB values • Name of or handle to a function that returns a colormap, such as `redgreencmap` or `redbluecmap` Default is `redgreencmap`, in which red represents values above the mean, black represents the mean, and green represents values below the mean of a row (gene) across all columns (samples).
DisplayRangeValue	Positive scalar that specifies the display range of standardized values. Default is the maximum absolute value in the input *Data*. For example, if you specify `3`, there is a color variation for values between –3 and 3, but values >3 are the same color as 3, and values < –3 are the same color as –3. For example, if you specify `redgreencmap` for the `'Colormap'` property, pure red represents values ≥ *DisplayRangeValue*, and pure green represents values ≤ –*DisplayRangeValue*.
SymmetricValue	Forces the color scale of the heat map to be symmetric around zero. Choices are `true` (default) or `false`.

ImputeFunValue	One of the following: • Name of a function that imputes missing data. • Handle to a function that imputes missing data. • Cell array where the first element is the name of or handle to a function that imputes missing data. The remaining elements are property name/property value pairs used as inputs to the function. --- **Tip** If data points are missing, you can use the `'ImputeFun'` property to impute the missing values.
RowLabelsColorValue	Structure or structure array containing color information for labeling the rows (*y*-axis) of the heat map. The structure or structures contain the following fields. If a single structure, then the fields contain a cell array of elements. If a structure array, then the fields contain a single element. • `Labels` — String specifying a row label listed in the `RowLabels` vector. • `Colors` — String or three-element vectors of RGB values specifying a color for the row label you specified in the `Labels` field. For more information on specifying colors, see `ColorSpec`. If this field is empty, default colors are assigned to the row label.

ColumnLabelsColorValue	Structure or structure array containing color information for labeling the columns (*x*-axis) of the heat map. The structure or structures contain the following fields. If a single structure, then the fields contain a cell array of elements. If a structure array, then the fields contain a single element. • `Labels` — String specifying a column label listed in the `ColumnLabels` vector. • `Colors` — String or three-element vector of RGB values specifying a color for the column label you specified in the `Labels` field. For more information on specifying colors, see `ColorSpec`. If this field is empty, default colors are assigned to the column label.
LabelsWithMarkersValue	Controls the display of colored markers instead of colored text for the row labels and column labels. Choices are `true` or `false` (default).

Description

HMobj = `HeatMap`(*Data*) displays a heat map (2-D color image) of the data and returns an object containing the data and display properties.

`HeatMap`(*Data*, ...'*PropertyName*', *PropertyValue*, ...) calls `HeatMap` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

`HeatMap`(*Data*, ...'RowLabels', *RowLabelsValue*, ...) uses the contents of *RowLabelsValue*, a vector of numbers or cell array of text strings, as labels for the rows in the heat map. Default is a vector of values 1 through *M*, where *M* is the number of rows in *Data*.

`HeatMap`(*Data*, ...'ColumnLabels', *ColumnLabelsValue*, ...) uses the contents of *ColumnLabelsValue*, a vector of numbers or cell array of text strings, as labels for the columns in the heat map. Default is a vector of values 1 through *M*, where *M* is the number of columns in *Data*.

HeatMap(*Data*, ...'Standardize', *StandardizeValue*, ...) specifies the dimension for standardizing the values in *Data*. The HeatMap function transforms the standardized values are so that the mean is 0 and the standard deviation is 1 in the specified dimension. *StandardizeValue* can be:

- 'column' or 1 — Standardize along the columns of data.
- 'row' or 2 — Standardize along the rows of data.
- 'none' or 3 (default) — Do not standardize.

HeatMap(*Data*, ...'Colormap', *ColormapValue*, ...) specifies the colormap to use to create the heat map. The colormap controls the colors used to display the heat map. *ColormapValue* is either an *M*-by-3 matrix of RGB values or the name of or handle to a function that returns a colormap, such as redgreencmap or redbluecmap. Default is redgreencmap.

Note: In redgreencmap, red represents values above the mean, black represents the mean, and green represents values below the mean of a row (gene) across all columns (samples). In redbluecmap, red represents values above the mean, white represents the mean, and blue represents values below the mean of a row (gene) across all columns (samples).

HeatMap(*Data*, ...'DisplayRange', *DisplayRangeValue*, ...) specifies the display range of standardized values. *DisplayRangeValue* must be a positive scalar. Default is the maximum absolute value in the input *Data*. For example, if you specify 3, there is a color variation for values between –3 and 3, but values >3 are the same color as 3, and values < –3 are the same color as –3.

For example, if you specify redgreencmap for the 'Colormap' property, pure red represents values $\geq$ *DisplayRangeValue*, and pure green represents values $\leq$ –*DisplayRangeValue*.

HeatMap(*Data*, ...'Symmetric', *SymmetricValue*, ...) controls whether the color scale of the heat map is symmetric around zero. *SymmetricValue* can be true (default) or false.

HeatMap(*Data*, ...'ImputeFun', *ImputeFunValue*, ...) specifies a function and optional inputs that impute missing data. *ImputeFunValue* can be any of the following:

- Name of a function that imputes missing data.

- Handle to a function that imputes missing data.

- Cell array where the first element is the name of or handle to a function that imputes missing data. The remaining elements are property name/property value pairs used as inputs to the function.

Tip If data points are missing, you can use the `'ImputeFun'` property to impute the missing values.

HeatMap(*Data*, ...'RowLabelsColor', *RowLabelsColorValue*, ...) specifies color information for labeling the rows (*y*-axis) of the heat map.

HeatMap(*Data*, ...'ColumnLabelsColor', *ColumnLabelsColorValue*, ...) specifies color information for labeling the columns (*x*-axis) of the heat map.

HeatMap(*Data*, ...'LabelsWithMarkers', *LabelsWithMarkersValue*, ...) controls the display of colored markers instead of colored text for the row labels and column labels. Choices are `true` or `false` (default).

Examples

Plot a heatmap of a data matrix

This example shows plot a heatmap of a data matrix

Create a matrix of data.

```
data = gallery('invhess',20);
```

Display a 2-D color image of the data.

```
hmo = HeatMap(data)
```

HeatMap object with 20 rows and 20 columns.
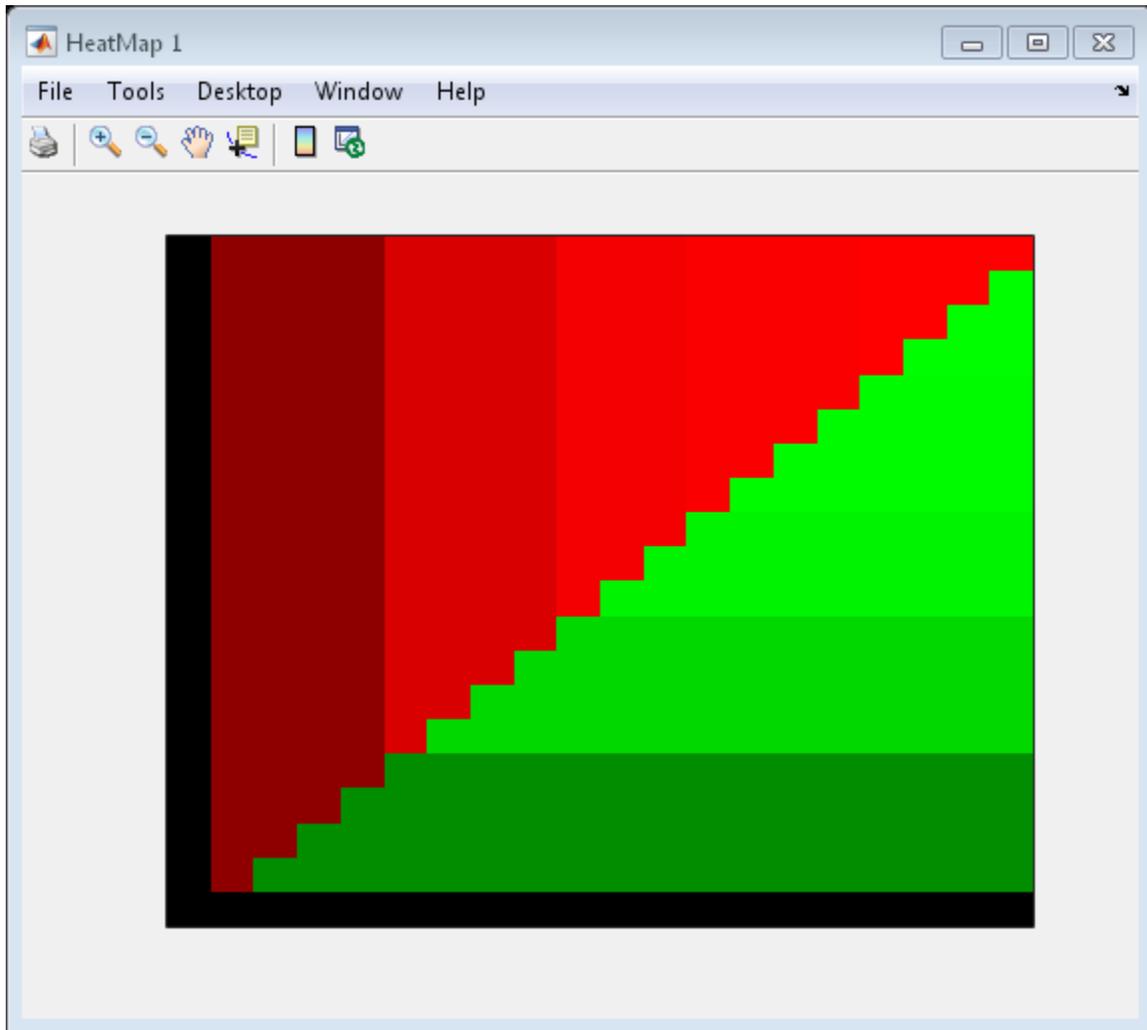
More About

- HeatMap object

See Also

redbluecmap | addXLabel | plot | view | redgreencmap | addTitle | addYLabel

hmmprofalign

Align query sequence to profile using hidden Markov model alignment

Syntax

```
Score = hmmprofalign(Model, Seq)
[Score, Alignment] = hmmprofalign(Model, Seq)
[Score, Alignment, Pointer] = hmmprofalign(Model, Seq)
hmmprofalign(..., 'ShowScore', ShowScoreValue, ...)
hmmprofalign(..., 'Flanks', FlanksValue, ...)
hmmprofalign(..., 'ScoreFlanks', ScoreFlanksValue, ...)
hmmprofalign(..., 'ScoreNullTransitions',
ScoreNullTransitionsValue, ...)
```

Arguments

Model	Hidden Markov model created with the function `hmmprofstruct`.
Seq	Amino acid or nucleotide sequence. You can also enter a structure with the field `Sequence`.
ShowScoreValue	Controls the display of the scoring space and the winning path. Choices are `true` or `false` (default).
FlanksValue	Controls the inclusion of the symbols generated by the FLANKING INSERT states in the output sequence. Choices are `true` or `false` (default).
ScoreFlanksValue	Controls the inclusion of the transition probabilities for the flanking states in the raw score. Choices are `true` or `false` (default).
ScoreNullTransitionsValue	Controls the adjustment of the raw score using the null model for transitions (`Model.NullX`). Choices are `true` or `false` (default).

Description

Score = hmmprofalign(*Model, Seq*) returns the score for the optimal alignment of the query amino acid or nucleotide sequence (Seq) to the profile hidden Markov model (*Model*). Scores are computed using log-odd ratios for emission probabilities and log probabilities for state transitions.

[*Score*, *Alignment*] = hmmprofalign(*Model, Seq*) also returns a string showing the optimal profile alignment.

Uppercase letters and dashes correspond to MATCH and DELETE states respectively (the combined count is equal to the number of states in the model). Lowercase letters are emitted by the INSERT states. For more information about the HMM profile, see hmmprofstruct.

[*Score, Alignment, Pointer*] = hmmprofalign(*Model, Seq*) also returns a vector of the same length as the profile model with indices pointing to the respective symbols of the query sequence. Null pointers (NaN) mean that such states did not emit a symbol in the aligned sequence because they represent model jumps from the BEGIN state of a MATCH state, model jumps from the from a MATCH state to the END state, or because the alignment passed through DELETE states.

hmmprofalign(..., '*PropertyName*', *PropertyValue*, ...) calls hmmprofalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

hmmprofalign(..., 'ShowScore', *ShowScoreValue*, ...), when *ShowScoreValue* is true, displays the scoring space and the winning path.

hmmprofalign(..., 'Flanks', *FlanksValue*, ...), when *FlanksValue* is true, includes the symbols generated by the FLANKING INSERT states in the output sequence.

hmmprofalign(..., 'ScoreFlanks', *ScoreFlanksValue*, ...), when *ScoreFlanksValue* is true, includes the transition probabilities for the flanking states in the raw score.

hmmprofalign(..., 'ScoreNullTransitions', *ScoreNullTransitionsValue*, ...), when *ScoreNullTransitionsValue* is true, adjusts the raw score using the null model for transitions (Model.NullX).

Note: Multiple target alignment is not supported in this implementation. All the `Model.LoopX` probabilities are ignored.

Examples

Align query sequence to profile using HMM model alignment

This example shows how to align a query sequence to a HMM model profile using HMM model alignment.

Load the HMM profile structure of the 7 transmembrane receptor (Secretin family).

```
load('hmm_model_examples','model_7tm_2');
```

Load an example sequence and align it to the profile structure using the HMM alignment.

```
load('hmm_model_examples','sequences');
humanSeq = sequences(1).Sequence;
[a,s]=hmmprofalign(model_7tm_2,humanSeq,'showscore',true)
```

```
a =

  483.7231


s =

YILVKAIYTLGYSVS-LMSLATGSIILCLFRKLHCTRNYIHLNLFLSFILRAISVLVKDDVLYSSSgtlhcpdqpsswvgCKLSLVF
```

log-odds score for best path
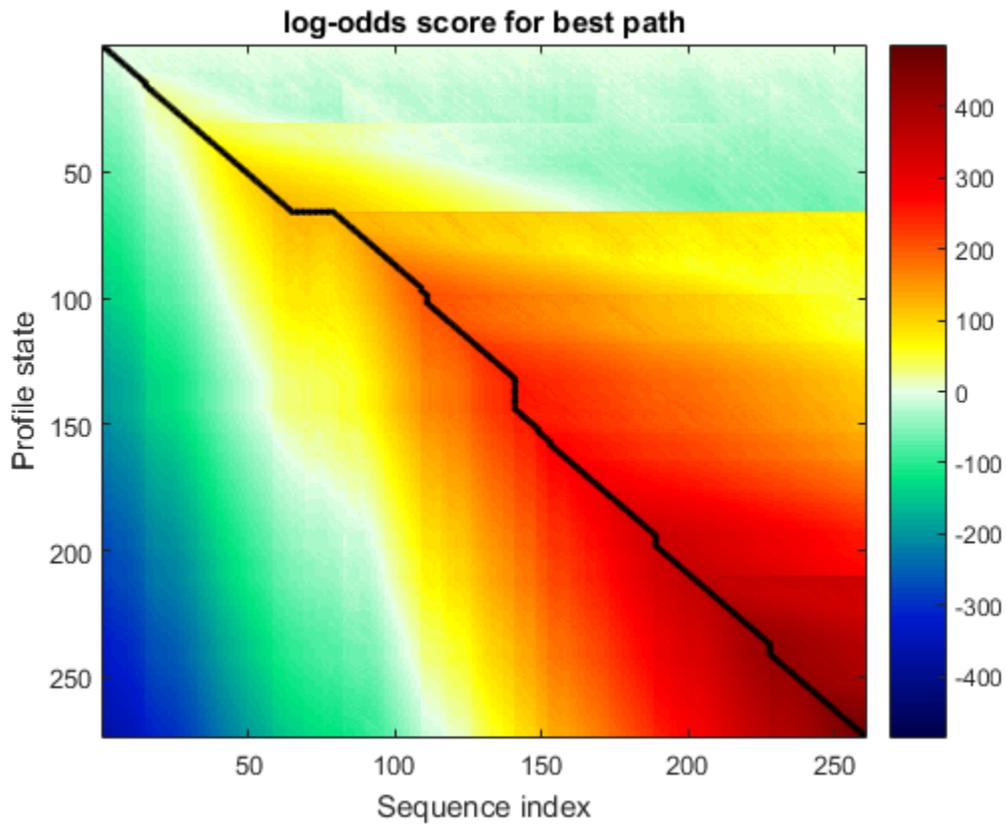
See Also

gethmmprof | hmmprofestimate | hmmprofgenerate | hmmprofgenerate |
hmmprofstruct | pfamhmmread | showhmmprof | multialign | profalign

hmmprofestimate

Estimate profile hidden Markov model (HMM) parameters using pseudocounts

Syntax

```
hmmprofestimate(Model, MultipleAlignment,'PropertyName',PropertyValue,...)

hmmprofestimate(..., 'A', AValue)

hmmprofestimate(..., 'Ax', AxValue)

hmmprofestimate(..., 'BE', BEValue)

hmmprofestimate(..., 'BDx', BDxValue)
```

Arguments

`Model`	Hidden Markov model created with the function `hmmprofstruc`.
`MultipleAlignment`	Array of sequences. `Sequences` can also be a structured array with the aligned sequences in a field `Aligned` or `Sequences` and the optional names in a field `Header` or `Name`.
`A`	Property to set the pseudocount weight A. Default value is `20`.
`Ax`	Property to set the pseudocount weight Ax. Default value is 20.
`BE`	Property to set the background symbol emission probabilities. Default values are taken from `Model.NullEmission`.
`BMx`	Property to set the background transition probabilities from any MATCH state ([M->M M->I M->D]). Default values are taken from `hmmprofstruct`.
`BDx`	Property to set the background transition probabilities from any DELETE state ([D->M D->D]). Default values are taken from `hmmprofstruct`.

Description

hmmprofestimate(Model, MultipleAlignment, '*PropertyName*', *PropertyValue*...) returns a structure with the fields containing the updated estimated parameters of a profile HMM. Symbol emission and state transition probabilities are estimated using the real counts and weighted pseudocounts obtained with the background probabilities. Default weight is A=20, the default background symbol emission for match and insert states is taken from Model.NullEmission, and the default background transition probabilities are the same as default transition probabilities returned by hmmprofstruct.

Model Construction: Multiple aligned sequences should contain uppercase letters and dashes indicating the model MATCH and DELETE states agreeing with Model.ModelLength. If model state annotation is missing, but MultipleAlignment is space aligned, then a "maximum entropy" criteria is used to select Model.ModelLength states.

Note: Insert and flank insert transition probabilities are not estimated, but can be modified afterwards using hmmprofstruct.

---

hmmprofestimate(..., 'A', *AValue*) sets the pseudocount weight A = Avalue when estimating the symbol emission probabilities. Default value is 20.

hmmprofestimate(...,'Ax', *AxValue*) sets the pseudocount weight Ax = Axvalue when estimating the transition probabilities. Default value is 20.

hmmprofestimate(...,'BE', *BEValue*) sets the background symbol emission probabilities. Default values are taken from Model.NullEmission.

hmmprofestimate(...,'BMx', *BMxValue*) sets the background transition probabilities from any MATCH state ([M->M M->I M->D]). Default values are taken from hmmprofstruct.

hmmprofestimate(..., 'BDx', *BDxValue*) sets the background transition probabilities from any DELETE state ([D->M D->D]). Default values are taken from hmmprofstruct.

See Also
hmmprofalign | hmmprofstruct | showhmmprof

hmmprofgenerate

Generate random sequence drawn from profile hidden Markov model (HMM)

Syntax

```
Sequence = hmmprofgenerate(Model)
[Sequence, Profptr] = hmmprofgenerate(Model)
... = hmmprofgenerate(Model, ...'Align', AlignValue, ...)
... = hmmprofgenerate(Model, ...'Flanks', FlanksValue, ...)
... = hmmprofgenerate(Model, ...'Signature', SignatureValue, ...)
```

Arguments

Model	Hidden Markov model created with the `hmmprofstruct` function.
AlignValue	Controls the use of uppercase letters for matches and lowercase letters for inserted letters. Choices are `true` or `false` (default).
FlanksValue	Controls the inclusion of the symbols generated by the FLANKING INSERT states in the output sequence. Choices are `true` or `false` (default).
SignatureValue	Controls the return of the most likely path and symbols. Choices are `true` or `false` (default).

Description

Sequence = hmmprofgenerate(*Model*) returns the string *Sequence* showing a sequence of amino acids or nucleotides drawn from the profile *Model*. The length, alphabet, and probabilities of the *Model* are stored in a structure. For more information about this structure, see `hmmprofstruct`.

[*Sequence*, *Profptr*] = hmmprofgenerate(*Model*) returns a vector of the same length as the profile model pointing to the respective states in the output sequence. Null

pointers (0) mean that such states do not exist in the output sequence, either because they are never touched (i.e., jumps from the BEGIN state to MATCH states or from MATCH states to the END state), or because DELETE states are not in the output sequence (not aligned output; see below).

`... = hmmprofgenerate(`*`Model`*`, ...'`*`PropertyName`*`', `*`PropertyValue`*`, ...)` calls `hmmprofgenerate` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

`... = hmmprofgenerate(`*`Model`*`, ...'Align', `*`AlignValue`*`, ...)` if `Align` is `true`, the output sequence is aligned to the model as follows: uppercase letters and dashes correspond to MATCH and DELETE states respectively (the combined count is equal to the number of states in the model). Lowercase letters are emitted by the INSERT or FLANKING INSERT states. If *AlignValue* is `false`, the output is a sequence of uppercase symbols. The default value is `true`.

`... = hmmprofgenerate(`*`Model`*`, ...'Flanks', `*`FlanksValue`*`, ...)` if `Flanks` is `true`, the output sequence includes the symbols generated by the FLANKING INSERT states. The default value is `false`.

`... = hmmprofgenerate(`*`Model`*`, ...'Signature', `*`SignatureValue`*`, ...)` if *SignatureValue* is `true`, returns the most likely path and symbols. The default value is `false`.

Examples

```
load('hmm_model_examples','model_7tm_2') % load a model example
rand_sequence = hmmprofgenerate(model_7tm_2)
```

See Also
hmmprofalign | hmmprofstruct | showhmmprof

hmmprofmerge

Concatenate prealigned strings of several sequences to profile hidden Markov model (HMM)

Syntax

```
hmmprofmerge(Sequences)
hmmprofmerge(Sequences, Names)
hmmprofmerge(Sequences, Names, Scores)
```

Arguments

Sequences	Array of sequences. *Sequences* can also be a structured array with the aligned sequences in a field `Aligned` or `Sequences`, and the optional names in a field `Header` or `Name`.
Names	Names for the sequences. Enter a vector of names.
Scores	Pairwise alignment scores from the function `hmmprofalign`. Enter a vector of values with the same length as the number of sequences in *Sequences*.

Description

hmmprofmerge(*Sequences*) opens your default Web browser and displays a set of prealigned sequences to an HMM model profile. The output is aligned corresponding to the HMM states.

- **Match states** — Uppercase letters
- **Insert states** — Lowercase letters or asterisks (*)
- **Delete states** — Dashes

Periods (.) are added at positions corresponding to inserts in other sequences. The input sequences must have the same number of profile states, that is, the joint count of capital letters and dashes must be the same.

hmmprofmerge(*Sequences*, *Names*) labels the sequences with Names.

hmmprofmerge(*Sequences*, *Names*, *Scores*) sorts the displayed sequences using Scores.

Examples

```
load('hmm_model_examples','model_7tm_2')  %load model
load('hmm_model_examples','sequences')  %load sequences

for ind =1:length(sequences)
    [scores(ind),sequences(ind).Aligned] =...
        hmmprofalign(model_7tm_2,sequences(ind).Sequence);
    end
hmmprofmerge(sequences, scores)
```

See Also
hmmprofalign | hmmprofstruct

hmmprofstruct

Create or edit hidden Markov model (HMM) profile structure

Syntax

Model = hmmprofstruct(*Length*)
Model = hmmprofstruct(*Length*, *Field1*, *Field1Value*, *Field2*,
Field2Value, ...)
NewModel = hmmprofstruct(*Model*, *Field1*, *Field1Value*, *Field2*,
Field2Value, ...)

Input Arguments

Length	Number of match states in the model.
Model	MATLAB structure containing fields for the parameters of an HMM profile created with the hmmprofstruct function.
Field	String containing a field name in the structure *Model*. See the table below for field names.
FieldValue	Value associated with *Field*. See the table below for descriptions.

Output Arguments

Model	MATLAB structure containing fields for the parameters of an HMM profile.

Description

Model = hmmprofstruct(*Length*) returns *Model*, a MATLAB structure containing fields for the parameters of an HMM profile. *Length* specifies the number of match states in the model. All other required parameters are set to the default values.

Model = hmmprofstruct(*Length*, *Field1*, *Field1Value*, *Field2*, *Field2Value*, ...) returns an HMM profile structure using the specified parameters. All other required parameters are set to default values.

NewModel = hmmprofstruct(*Model*, *Field1*, *Field1Value*, *Field2*, *Field2Value*, ...) returns an updated HMM profile structure using the specified parameters. All other parameters are taken from the input *Model*.

HMM Profile Structure

The MATLAB structure *Model* contains the following fields, which are the required and optional parameters of an HMM profile. All probability values are in the [0 1] range.

Field	Description
ModelLength	Integer specifying the length of the profile (number of MATCH states).
Alphabet	String specifying the alphabet used in the model. Choices are `'AA'` (default) or `'NT'`. **Note:** AlphaLength is 20 for `'AA'` and 4 for `'NT'`.
MatchEmission	Symbol emission probabilities in the MATCH states. Either of the following: • A matrix of size ModelLength-by-AlphaLength, where each row corresponds to the emission distribution for a specific MATCH state. Defaults to uniform distributions. • A structure containing residue counts, such as returned by aacount or basecount.
InsertEmission	Symbol emission probabilities in the INSERT state. Either of the following: • A matrix of size ModelLength-by-AlphaLength, where each row corresponds to the emission distribution for a specific INSERT state. Defaults to uniform distributions. • A structure containing residue counts, such as returned by aacount or basecount.
NullEmission	Symbol emission probabilities in the MATCH and INSERT states for the NULL model.

Field	Description
	Either of the following: • A 1-by-`AlphaLength` row vector. Defaults to a uniform distribution. • A structure containing residue counts, such as returned by `aacount` or `basecount`. **Note:** The NULL model is used to compute the log-odds ratio at every state and avoid overflow when propagating the probabilities through the model. **Note:** NULL probabilities are also known as the background probabilities.

BeginX	BEGIN state transition probabilities.
	
	Format is a 1-by-(`ModelLength` + 1) row vector:
	
	`[B->D1 B->M1 B->M2 B->M3 .... B->Mend]`
	
	Note: If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from the BEGIN state equals 1:
	
	`sum(Model.BeginX) = 1`
	
	For fragment profiles:
	
	`sum(Model.BeginX(3:end)) = 0`
	
	Default is `[0.01 0.99 0 0 ... 0]`.

Field	Description

MatchX	MATCH state transition probabilities.
	
	Format is a 4-by-`(ModelLength - 1)` matrix:
	
	`[M1->M2 M2->M3 ... M[end-1]->Mend;`
	` M1->I1 M2->I2 ... M[end-1]->I[end-1];`
	` M1->D2 M2->D3 ... M[end-1]->Dend;`
	` M1->E  M2->E  ... M[end-1]->E  ]`
	
	--- 	
	Note: If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from every MATCH state equals 1:
	
	`sum(`*`Model`*`.MatchX) = [ 1 1 ... 1 ]`
	
	For fragment profiles:
	
	`sum(`*`Model`*`.MatchX(4,:)) = 0`
	
	--- 	
	Default is `repmat([0.998 0.001 0.001 0],ModelLength-1,1)`.

InsertX	INSERT state transition probabilities.
	
	Format is a 2-by-`(ModelLength - 1)` matrix:
	
	`[ I1->M2 I2->M3 ... I[end-1]->Mend;`
	`  I1->I1 I2->I2 ... I[end-1]->I[end-1] ]`
	
	--- 	
	Note: If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from every INSERT state equals 1:
	
	`sum(`*`Model`*`.InsertX) = [ 1 1 ... 1 ]`
	
	--- 	
	Default is `repmat([0.5 0.5],ModelLength-1,1)`.

Field	Description

DeleteX	DELETE state transition probabilities.
	
	Format is a 2-by-`(ModelLength - 1)` matrix:
	
	`[ D1->M2 D2->M3 ... D[end-1]->Mend ;`
	`  D1->D2 D2->D3 ... D[end-1]->Dend ]`
	
	--- 	
	Note: If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from every DELETE state equals 1:
	
	`sum(`*Model*`.DeleteX) = [ 1 1 ... 1 ]`
	
	--- 	
	Default is `repmat([0.5 0.5],ModelLength-1,1)`.

FlankingInsertX	Flanking insert states (N and C) used for LOCAL profile alignment.
	
	Format is a 2-by-2 matrix:
	
	`[N->B  C->T ;`
	` N->N  C->C]`
	
	--- 	
	Note: If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from Flanking Insert states equals 1:
	
	`sum(`*Model*`.FlankingInsertsX) = [1 1]`
	
	--- 	
	Note: To force global alignment use:
	
	Model`.FlankingInsertsX = [1 1; 0 0]`
	
	--- 	
	Default is `[0.01 0.01; 0.99 0.99]`.

Field	Description

LoopX	Loop states transition probabilities used for multiple hits alignment.
	
	 Format is a 2-by-2 matrix:
	
	 `[E->C  J->B ;`
	` E->J  J->J]`
	
	 Note: If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from Loop states equals 1:
	
	 `sum(`*`Model`*`.LoopX) = [1 1]`
	
	 Default is `[0.5 0.01; 0.5 0.99]`.

NullX	Null transition probabilities used to provide scores with log-odds values also for state transitions.
	
	 Format is a 2-by-1 column vector:
	
	 `[G->F ; G->G]`
	
	 Note: If necessary, `hmmprofstruct` will normalize the data such that the sum of the transition probabilities from Null states equals 1:
	
	 `sum(`*`Model`*`.NullX) = 1`
	
	 Default is `[0.01; 0.99]`.

IDNumber	Optional. User-assigned identification number.
Description	Optional. User-assigned description of the model.

HMM Profile Model

An HMM profile model is a common statistical tool for modeling structured sequences composed of symbols. These symbols include randomness in both the output (emission of symbols) and the state transitions of the process. Markov models are generally represented by state diagrams.

The following figure is a state diagram for an HMM profile of length four. INSERT, MATCH, and DELETE states are in the center section.

- INSERT state represents the excess of one or more symbols in the target sequence that are not included in the profile.
- MATCH state means that the target sequence is aligned to the profile at the specific location.
- DELETE state represents a gap or symbol absence in the target sequence (also known as a silent state because it does not emit any symbols).

Flanking states (S, N, B, E, C, T) are used for proper modeling of the ends of the sequence, either for global, local or fragment alignment of the profile. S, B, E, and T are silent, while N and C are used to insert symbols at the flanks.
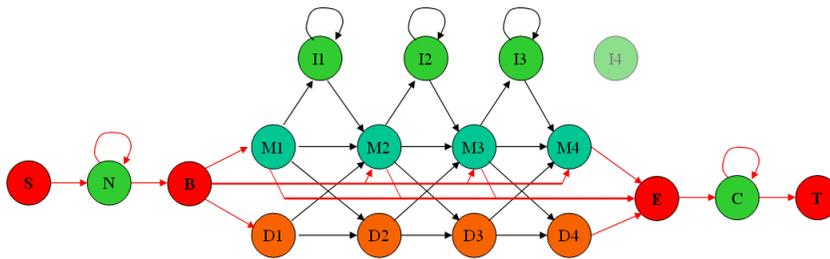
Examples

Creating an HMM Profile Structure

Create an HMM profile structure with 100 MATCH states, using the amino acid alphabet.

```
hmmProfile = hmmprofstruct(100,'Alphabet','AA')
```

```
hmmProfile =

          ModelLength: 100
             Alphabet: 'AA'
        MatchEmission: [100x20 double]
       InsertEmission: [100x20 double]
         NullEmission: [1x20 double]
               BeginX: [101x1 double]
               MatchX: [99x4 double]
              InsertX: [99x2 double]
              DeleteX: [99x2 double]
      FlankingInsertX: [2x2 double]
                LoopX: [2x2 double]
                NullX: [2x1 double]
```

Editing an HMM Profile Structure

1 Use the `pfamhmmread` function to create an HMM profile structure from `pf00002.ls`, a PFAM HMM-formatted file included with the software.

```
hmm02 = pfamhmmread('pf00002.ls');
```

2 Modify the HMM profile structure to force a global alignment by setting the looping transition probabilities in the flanking insert states to zero.

```
hmm02 = hmmprofstruct(hmm02,'FlankingInsertX',[0 0;1 1]);
hmm02.FlankingInsertX

ans =

     0     0
     1     1
```

See Also

aacount | basecount | gethmmprof | hmmprofalign | hmmprofestimate | hmmprofgenerate | hmmprofmerge | pfamhmmread | showhmmprof

horzcat (DataMatrix)

Concatenate DataMatrix objects horizontally

Syntax

```
DMObjNew = horzcat(DMObj1, DMObj2, ...)
DMObjNew = (DMObj1, DMObj2, ...)
DMObjNew = horzcat(DMObj1, B, ...)
DMObjNew = (DMObj1, B, ...)
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

DMObjNew	DataMatrix object created by horizontal concatenation.

Description

DMObjNew = horzcat(*DMObj1*, *DMObj2*, ...) or the equivalent *DMObjNew* = (*DMObj1*, *DMObj2*, ...) horizontally concatenates the DataMatrix objects *DMObj1* and *DMObj2* into *DMObjNew*, another DataMatrix object. *DMObj1* and *DMObj2* must have the same number of rows. The row names and the order of rows for *DMObjNew* are the same as *DMObj1*. The row names of *DMObj2* and any other DataMatrix object input arguments are not preserved. The columns names for *DMObjNew* are the column names of *DMObj1*, *DMObj2*, and other DataMatrix object input arguments.

DMObjNew = horzcat(*DMObj1*, *B*, ...) or the equivalent *DMObjNew* = (*DMObj1*, *B*, ...) horizontally concatenates the DataMatrix object *DMObj1* and a numeric or

logical array *B* into *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same number of rows. The row names for *DMObjNew* are the same as *DMObj1*. The row names of *DMObj2* and any other DataMatrix object input arguments are not preserved. The column names for *DMObjNew* are the column names of *DMObj1* and empty for the columns from *B*.

MATLAB calls *DMObjNew* = horzcat(*X1*, *X2*, *X3*, ...) for the syntax *DMObjNew* = [*X1*, *X2*, *X3*, ...] when any one of *X1*, *X2*, *X3*, etc. is a DataMatrix object.

More About

- DataMatrix object

See Also
DataMatrix | vertcat

ilmnbslookup

Look up Illumina BeadStudio target (probe) sequence and annotation information

Syntax

```
AnnotStruct = ilmnbslookup(AnnotationFile, ID)
AnnotStruct = ilmnbslookup(AnnotationFile, ID, 'LookUpField',
LookUpFieldValue)
```

Input Arguments

AnnotationFile	String specifying a file name or a path and file name of an Illumina® annotation file (CSV, BGX, or TXT format). If you specify only a file name, that file must be on the MATLAB search path or in the current folder.
	Tip You can download Illumina annotation files, such as `HumanRef-8_V3_0_R0_11282963_A.bgx`, from the Illumina Web site.
ID	String or cell array of strings representing a unique identifier(s) for one or more targets (probes) on an Illumina microarray.
	Tip By default, *ID* must match the `Search_key` field in *AnnotationFile*. However, you can use an identifier that corresponds to any of the fields in *AnnotationFile*, then set the `'LookUpField'` property appropriately. For example, if you want to look up annotation information for the targets (probes) on chromosome 7 only, set *ID* to `'7'`, then set *LookUpFieldValue* to `'Chromosome'`. For a list of all fields in *AnnotationFile*, see the following tables.
LookUpFieldValue	Field in *AnnotationFile* where `ilmnbslookup` looks for the specified *ID*. Default is the `Search_key` field.

<table>
<tr><td></td><td>Tip Set this property so that it corresponds to the ID you use as input.</td></tr>
</table>

Output Arguments

AnnotStruct	Structure containing the probe sequence and annotation information for one or more targets (probes) specified by *ID*, and by *AnnotationFile*, an Illumina annotation file. *AnnotStruct* contains the same fields as *AnnotationFile*. The fields are described in the following two tables.

Description

AnnotStruct = ilmnbslookup(*AnnotationFile*, *ID*) returns *AnnotStruct*, a structure containing probe sequence and annotation information for one or more targets (probes) specified by *ID*, and by *AnnotationFile*, an Illumina annotation file (CSV, BGX, or TXT format).

AnnotStruct contains the same fields as *AnnotationFile*. The fields are described in the following two tables.

Structure Created from Illumina CSV Annotation File

Field	Description
Search_key	Internal identifier for the target, useful for custom design array
Target	Unique identifier for the target
ProbeId	Illumina probe identifier
Gid	GenBank identifier for the gene
Transcript	Illumina internal transcript identifier
Accession	GenBank accession number for the gene
Symbol	Typically, the gene symbol
Type	Probe type
Start	Starting position of the probe sequence in the GenBank record

Field	Description
`Probe_Sequence`	Sequence of the probe
`Definition`	Definition field from the GenBank record
`Ontology`	Gene Ontology terms associated with the gene
`Synonym`	Synonyms for the gene (from the GenBank record)

Structure Created from a BGX or TXT Annotation File

Field	Description
`Accession`	GenBank accession number for the gene
`Array_Address_Id`	Decoder identifier
`Chromosome`	Chromosome on which the gene is located
`Cytoband`	Cytogenetic banding region of the chromosome on which the gene associated with the target is located
`Definition`	Definition field from the GenBank record
`Entrez_Gene_ID`	Entrez Gene database identifier for the gene
`GI`	GenBank identifier for the gene
`ILMN_Gene`	Illuminainternal gene symbol
`Obsolete_Probe_Id`	Probe identifier before BGX annotation files
`Ontology_Component`	Gene Ontology cellular components associated with the gene
`Ontology_Function`	Gene Ontology molecular functions associated with the gene
`Ontology_Process`	Gene Ontology biological processes associated with the gene
`Probe_Chr_Orientation`	Orientation of the probe on the NCBI genome build
`Probe_Coordinates`	Genomic position of the probe on the NCBI genome build
`Probe_Id`	Illuminaprobe identifier
`Probe_Sequence`	Sequence of the probe
`Probe_Start`	Start position of the probe relative to the 5' end of the source transcript sequence

Field	Description
Probe_Type	Information about what the probe is targeting
Protein_Product	NCBI protein accession number
RefSeq_ID	Identifier from the NCBI RefSeq database
Reporter_Composite_map	Information associated with control probes
Reporter_Group_Name	Information associated with control probes
Reporter_Group_id	Information associated with control probes
Search_Key	Internal identifier for the target, useful for custom design array
Source	Source from which the transcript sequence was obtained
Source_Reference_ID	Source's identifier
Species	Species associated with the gene
Symbol	Typically, the gene symbol
Synonyms	Synonyms for the gene (from the GenBank record)
Transcript	Illuminainternal transcript identifier
Unigene_ID	Identifier from the NCBI UniGene database

AnnotStruct = ilmnbslookup(*AnnotationFile*, *ID*, 'LookUpField', *LookUpFieldValue*) looks for *ID* in the annotation file in the field specified by *LookUpFieldValue*. Default is the Search_key field.

Examples

Note: The gene expression file, TumorAdjacent-probe-raw.txt, and the annotation file, HumanRef-8_V3_0_R0_11282963_A.bgx, used in the following examples are not provided with the Bioinformatics Toolbox software.

Look Up Annotation Information for a Single Target (Probe)

1   Read the contents of a tab-delimited file exported from the Illumina BeadStudio™ software into a MATLAB structure.

```
ilmnStruct = ilmnbsread('TumorAdjacent-probe-raw.txt')

ilmnStruct =

            Header: [1x1 struct]
          TargetID: {22184x1 cell}
       ColumnNames: {1x37 cell}
              Data: [22184x37 double]
   TextColumnNames: {1x23 cell}
          TextData: {22184x23 cell}
```

2 Find the number of the Search_key column in the TextColumnNames cell array, which is returned in the ilmnStruct structure by the ilmnbsread function.

```
srchCol = find(strcmpi('Search_Key',ilmnStruct.TextColumnNames))

srchCol =

     1
```

3 Use the output from step 2 to look up the probe sequence and annotation information for the 10th entry in the annotation file, HumanRef-8_V3_0_R0_11282963_A.bgx.

```
annotation = ilmnbslookup('HumanRef-8_V3_0_R0_11282963_A.bgx',...
                          ilmnStruct.TextData{10,srchCol})
annotation =

                  Accession: 'NM_144670.2'
           Array_Address_Id: '0004050154'
                 Chromosome: '12'
                   Cytoband: '12p13.31b'
                 Definition: 'Homo sapiens alpha-2-macroglobulin-like 1 (A2ML1), mRNA.'
              Entrez_Gene_ID: '144568'
                         GI: '74271844'
                  ILMN_Gene: 'A2ML1'
           Obsolete_Probe_Id: ''
          Ontology_Component: ''
           Ontology_Function: 'endopeptidase inhibitor activity [goid 4866] [evidence IEA]'
            Ontology_Process: ''
       Probe_Chr_Orientation: '+'
           Probe_Coordinates: '8920412-8920461'
                    Probe_Id: 'ILMN_2136495'
              Probe_Sequence: 'TGTAATCGCAGCCCCTTGGAAGGCCAAGGCAGGAGAATCGCCTCAACACT'
                 Probe_Start: '4889'
                  Probe_Type: 'S'
             Protein_Product: 'NP_653271.2'
                   RefSeq_ID: 'NM_144670.2'
      Reporter_Composite_map: ''
          Reporter_Group_Name: ''
            Reporter_Group_id: ''
                  Search_Key: 'ILMN_17375'
                      Source: 'RefSeq'
          Source_Reference_ID: 'NM_144670.2'
                     Species: 'Homo sapiens'
```

```
                    Symbol: 'A2ML1'
                  Synonyms: [1x141 char]
                 Transcript: 'ILMN_17375'
                 Unigene_ID: ''
```

Look Up Annotation Information for a Subset of Targets (Probes)

Use the `ilmnbslookup` function with the `'LookUpField'` property to look up the annotation information for all targets located on chromosome 12 in the annotation file, `HumanRef-8_V3_0_R0_11282963_A.bgx`.

```
chr12annotation = ilmnbslookup('HumanRef-8_V3_0_R0_11282963_A.bgx',...
                               '12','LookUpField','Chromosome')

chr12annotation =

                   Accession: {1x1186 cell}
            Array_Address_Id: {1x1186 cell}
                  Chromosome: {1x1186 cell}
                    Cytoband: {1x1186 cell}
                  Definition: {1x1186 cell}
               Entrez_Gene_ID: {1x1186 cell}
                          GI: {1x1186 cell}
                    ILMN_Gene: {1x1186 cell}
            Obsolete_Probe_Id: {1x1186 cell}
           Ontology_Component: {1x1186 cell}
            Ontology_Function: {1x1186 cell}
             Ontology_Process: {1x1186 cell}
        Probe_Chr_Orientation: {1x1186 cell}
            Probe_Coordinates: {1x1186 cell}
                     Probe_Id: {1x1186 cell}
               Probe_Sequence: {1x1186 cell}
                  Probe_Start: {1x1186 cell}
                   Probe_Type: {1x1186 cell}
              Protein_Product: {1x1186 cell}
                    RefSeq_ID: {1x1186 cell}
        Reporter_Composite_map: ''
          Reporter_Group_Name: ''
            Reporter_Group_id: ''
                   Search_Key: {1x1186 cell}
                       Source: {1x1186 cell}
          Source_Reference_ID: {1x1186 cell}
                      Species: {1x1186 cell}
                       Symbol: {1x1186 cell}
                     Synonyms: {1x1186 cell}
                   Transcript: {1x1186 cell}
```

```
Unigene_ID: {1x1186 cell}
```

The output structure indicates that there are 1,186 targets located on chromosome 12.

See Also

```
ilmnbsread
```

ilmnbsread

Read gene expression data exported from Illumina BeadStudio software

Syntax

```
IlmnStruct = ilmnbsread(File)
IlmnStruct = ilmnbsread(File, ...'Columns', ColumnsValue, ...)
IlmnStruct = ilmnbsread(File, ...'HeaderOnly', HeaderOnlyValue, ...)
IlmnStruct = ilmnbsread(File, ...'CleanColNames',
CleanColNamesValue, ...)
```

Input Arguments

File	String specifying a file name or a path and file name of a tab-delimited file or comma-separated expression data file exported from Illumina BeadStudio software. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.
ColumnsValue	Cell array that specifies the column names to read. Default is all column names.
HeaderOnlyValue	Controls the population of only the Header, ColumnNames, and TextColumnNames fields in *IlmnStruct*. Choices are true or false (default).
CleanColNamesValue	Controls the conversion of any ColumnNames containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are true or false (default).

Output Arguments

IlmnStruct	MATLAB structure containing data exported from Illumina BeadStudio software.

Description

IlmnStruct = ilmnbsread(*File*) reads *File*, a tab-delimited or comma-separated expression data file exported from the Illumina BeadStudio software, and creates *IlmnStruct*, a MATLAB structure containing the following fields.

Field	Description
Header	String containing a description of the data.
TargetID	Cell array containing unique identifiers for targets on an Illumina gene expression microarray.
ColumnNames	Cell array containing names of the columns that contain numeric data in the tab-delimited file exported from the Illumina BeadStudio software.
Data	Matrix containing numeric microarray data for each target on an Illumina gene expression microarray. **Note:** ColumnNames and Data have the same number of columns.
TextColumnNames	Cell array containing names of the columns that contain nonnumeric data in the tab-delimited file exported from the Illumina BeadStudio software. This field can be empty.
TextData	Cell array containing nonnumeric microarray data (such as annotations) for each target on an Illumina gene expression microarray. This field can be empty. **Note:** TextColumnNames and TextData have the same number of columns.

IlmnStruct = ilmnbsread(*File*, ...'*PropertyName*', *PropertyValue*, ...) calls ilmnbsread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

IlmnStruct = ilmnbsread(*File*, ...'Columns', *ColumnsValue*, ...) reads the data only from the columns specified by *ColumnsValue*, a cell array of column names. Default behavior is to read data from all columns.

IlmnStruct = ilmnbsread(*File*, ...'HeaderOnly', *HeaderOnlyValue*, ...) controls the population of only the Header, ColumnNames, and TextColumnNames fields in *IlmnStruct*. Choices are true or false (default).

IlmnStruct = ilmnbsread(*File*, ...'CleanColNames', *CleanColNamesValue*, ...) controls the conversion of any ColumnNames containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are true or false (default).

Tip Use the 'CleanColNames' property if you plan to use the ColumnNames field as variable names.

Examples

Note: The gene expression file, TumorAdjacent-probe-raw.txt used in the following example is not provided with the Bioinformatics Toolbox software.

Read the contents of a tab-delimited file exported from the Illumina BeadStudio software into a MATLAB structure.

```
ilmnStruct = ilmnbsread('TumorAdjacent-probe-raw.txt')

ilmnStruct =

            Header: [1x1 struct]
          TargetID: {22184x1 cell}
       ColumnNames: {1x37 cell}
              Data: [22184x37 double]
   TextColumnNames: {1x23 cell}
          TextData: {22184x23 cell}
```

See Also
affyread | agferead | celintensityread | galread | geoseriesread | geosoftread | gprread | ilmnbslookup | imageneread | magetfield | sptread

imageneread

Read microarray data from ImaGene Results file

Syntax

```
imagenedata = imageneread('File')
imagenedata = imageneread(..., 'CleanColNames',
CleanColNamesValue, ...)
```

Arguments

File	ImaGene® Results formatted file. Enter a file name or a path and file name.
CleanColNamesValue	Controls the conversion of any `ColumnNames` containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are `true` or `false` (default).

Description

imagenedata = imageneread('*File*') reads ImaGene results data from *File* and creates *imagenedata*, a MATLAB structure containing the following fields.

Field
HeaderAA
Data
Blocks
Rows
Columns
Fields
IDs

Field
ColumnNames
Indices
Shape

imagenedata = imageneread(*...*, '*PropertyName*', *PropertyValue*, *...*) calls imageneread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

imagenedata = imageneread(*...*, 'CleanColNames', *CleanColNamesValue*, *...*) controls the conversion of any ColumnNames containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are true or false (default).

The field Indices of the structure contains indices that you can use for plotting heat maps of the data with the function image or imagesc.

For more details on the ImaGene format and example data, see the ImaGene documentation.

Examples

In the following example, the file cy3.txt is not provided.

1 Read in a sample ImaGene Results file. Note that the example file, cy3.txt, is not provided with the Bioinformatics Toolbox software.

```
cy3Data = imageneread('cy3.txt');
```

2 Plot the signal mean.

```
maimage(cy3Data,'Signal Mean');
```

3 Read in a sample ImaGene Results file. Note that the example file, cy5.txt, is not provided with the Bioinformatics Toolbox software.

```
cy5Data = imageneread('cy5.txt');
```

4 Create a loglog plot of the signal median from two ImaGene Results files.

```
sigMedianCol = find(strcmp('Signal Median',cy3Data.ColumnNames));
cy3Median = cy3Data.Data(:,sigMedianCol);
cy5Median = cy5Data.Data(:,sigMedianCol);
maloglog(cy3Median,cy5Median,'title','Signal Median');
```

See Also
gprread | ilmnbsread | maboxplot | maimage | sptread

int2aa

Convert amino acid sequence from integer to letter representation

Syntax

SeqChar = int2aa(*SeqInt*)
SeqChar = int2aa(*SeqInt*, 'Case', *CaseValue*)

Input Arguments

SeqInt	Row vector of integers specifying an amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes. Integers are arbitrarily assigned to IUB/IUPAC letters.
CaseValue	String specifying the case of the returned string. Choices are 'upper' (default) or 'lower'.

Output Arguments

SeqChar	Amino acid sequence specified by a string of single-letter codes.

Description

SeqChar = int2aa(*SeqInt*) converts *SeqInt*, a row vector of integers specifying an amino acid sequence, to *SeqChar*, a string of single-letter codes specifying the same amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes.

SeqChar = int2aa(*SeqInt*, 'Case', *CaseValue*) specifies the case of the returned string. Choices are 'upper' (default) or 'lower'.

Mapping Amino Acid Integers to Letter Codes

Amino Acid	Integer	Code
Alanine	1	A
Arginine	2	R
Asparagine	3	N
Aspartic acid (Aspartate)	4	D
Cysteine	5	C
Glutamine	6	Q
Glutamic acid (Glutamate)	7	E
Glycine	8	G
Histidine	9	H
Isoleucine	10	I
Leucine	11	L
Lysine	12	K
Methionine	13	M
Phenylalanine	14	F
Proline	15	P
Serine	16	S
Threonine	17	T
Tryptophan	18	W
Tyrosine	19	Y
Valine	20	V
Asparagine or Aspartic acid (Aspartate)	21	B
Glutamine or Glutamic acid (Glutamate)	22	Z
Unknown amino acid (any amino acid)	23	X
Translation stop	24	*
Gap of indeterminate length	25	-
Unknown (any integer not in table)	0 or ≥ 26	?

Examples

Convert an amino acid sequence from integer to letter representation.

```
s = int2aa([13 1 17 11 1 21])

s =

MATLAB
```

See Also

aa2int | aminolookup | int2nt | isotopicdist | nt2int

int2nt

Convert nucleotide sequence from integer to letter representation

Syntax

```
SeqChar = int2nt(SeqInt)
SeqChar = int2nt(SeqInt, ...'Alphabet', AlphabetValue, ...)
SeqChar = int2nt(SeqInt, ...'Unknown', UnknownValue, ...)
SeqChar = int2nt(SeqInt, ...'Case', CaseValue, ...)
```

Input Arguments

SeqInt	Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes. Integers are arbitrarily assigned to IUB/IUPAC letters.
AlphabetValue	String specifying a nucleotide alphabet. Choices are: • 'DNA' (default) — Uses the symbols A, C, G, and T. • 'RNA' — Uses the symbols A, C, G, and U.
UnknownValue	Character to represent unknown nucleotides, that is 0 or integers ≥ 17. Choices are any character other than the nucleotide characters A, C, G, T, and U and the ambiguous nucleotide characters N, R, Y, K, M, S, W, B, D, H, and V. Default is *.
CaseValue	String specifying the case of the returned character string. Choices are 'upper' (default) or 'lower'.

Output Arguments

SeqChar	Nucleotide sequence specified by a character string of codes.

Description

SeqChar = int2nt(*SeqInt*) converts *SeqInt*, a row vector of integers specifying a nucleotide sequence, to *SeqChar*, a string of codes specifying the same nucleotide sequence. For valid codes, see the table Mapping Nucleotide Integers to Letter Codes.

Mapping Nucleotide Integers to Letter Codes

Nucleotide	Integer	Code
Adenosine	1	A
Cytidine	2	C
Guanine	3	G
Thymidine	4	T
Uridine (if 'Alphabet' set to 'RNA')	4	U
Purine (A or G)	5	R
Pyrimidine (T or C)	6	Y
Keto (G or T)	7	K
Amino (A or C)	8	M
Strong interaction (3 H bonds) (G or C)	9	S
Weak interaction (2 H bonds) (A or T)	10	W
Not A (C or G or T)	11	B
Not C (A or G or T)	12	D
Not G (A or C or T)	13	H
Not T or U (A or C or G)	14	V
Any nucleotide (A or C or G or T or U)	15	N
Gap of indeterminate length	16	-
Unknown (any integer not in table)	0 or $\geq$ 17	* (default)

SeqChar = int2nt(*SeqInt*, ...*PropertyName*', *PropertyValue*, ...) calls int2nt with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

SeqChar = int2nt(*SeqInt*, ...'Alphabet', *AlphabetValue*, ...) specifies a nucleotide alphabet. *AlphabetValue* can be 'DNA', which uses the symbols A, C, G, and T, or 'RNA', which uses the symbols A, C, G, and U. Default is 'DNA'.

SeqChar = int2nt(*SeqInt*, ...'Unknown', *UnknownValue*, ...) specifies the character to represent unknown nucleotides, that is 0 or integers $\geq$ 17. *UnknownValue* can be any character other than the nucleotide characters A, C, G, T, and U and the ambiguous nucleotide characters N, R, Y, K, M, S, W, B, D, H, and V. Default is *.

SeqChar = int2nt(*SeqInt*, ...'Case', *CaseValue*, ...) specifies the case of the returned character string. *CaseValue* can be 'upper' (default) or 'lower'.

Examples

- Convert a nucleotide sequence from integer to letter representation.

```
s = int2nt([1 2 4 3 2 4 1 3 2])

s =
ACTGCTAGC
```

- Convert a nucleotide sequence from integer to letter representation and define # as the symbol for unknown numbers 17 and greater.

```
si = [1 2 4 20 2 4 40 3 2];
s = int2nt(si, 'unknown', '#')

s =
ACT#CT#GC
```

See Also

aa2int | baselookup | int2aa | nt2int

isdag (biograph)

Test for cycles in biograph object

Syntax

isdag(*BGObj*)

Arguments

BGObj	Biograph object created by `biograph` (object constructor).

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

---

isdag(*BGObj*) returns logical 1 (`true`) if an N-by-N adjacency matrix extracted from a biograph object, *BGObj*, is a directed acyclic graph (DAG) and logical 0 (`false`) otherwise. In the N-by-N sparse matrix, all nonzero entries indicate the presence of an edge.

More About

- biograph object

References

[1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

```
biograph | allshortestpaths | isomorphism | maxflow | shortestpath |
traverse | graphisdag | conncomp | isspantree | minspantree | topoorder
```

isempty

Class: bioma.data.ExptData
Package: bioma.data

Determine whether ExptData object is empty

Syntax

TF = isempty(*EDObj*)

Description

TF = isempty(*EDObj*) returns logical 1 (true) if *EDObj* is an empty ExptData object. Otherwise, it returns logical 0 (false). An empty ExptData object contains no data elements.

Input Arguments

EDObj

Object of the bioma.data.ExptData class.

Default:

Examples

Construct an ExptData object, and then check to see if it is empty:

```
% Import bioma.data package to make constructor functions
% available
import bioma.data.*
% Create DataMatrix object from .txt file containing
% expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Determine if ExptData object is empty
```

```
isempty(EDObj)
```

See Also
`bioma.data.ExptData`

How To
·   "Representing Expression Data Values in ExptData Objects"

isempty

Class: bioma.data.MetaData
Package: bioma.data

Determine whether MetaData object is empty

Syntax

TF = isempty(*MDObj*)

Description

TF = isempty(*MDObj*) returns logical 1 (true) if *MDObj* is an empty MetaData object. Otherwise, it returns logical 0 (false). An empty MetaData object contains no variable names, values, or descriptions.

Input Arguments

MDObj

Object of the `bioma.data.MetaData` class.

Default:

Examples

Construct a MetaData object, and then check to see if it is empty:

```
% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Determine if MetaData object is empty
isempty(MDObj2)
```

See Also
`bioma.data.MetaData`

How To
- "Representing Sample and Feature Metadata in MetaData Objects"

isempty

Class: bioma.data.MIAME
Package: bioma.data

Determine whether MIAME object is empty

Syntax

TF = isempty(*MIAMEObj*)

Description

TF = isempty(*MIAMEObj*) returns logical 1 (true) if *MIAMEObj* is an empty MIAME object. Otherwise, it returns logical 0 (false). All properties are empty in an empty MIAME object.

Input Arguments

MIAMEObj

Object of the bioma.data.MIAME class.

Default:

Examples

Construct a MIAME object, and then check to see if it is empty:

```
% Create a MATLAB structure containing GEO Series data
geoStruct = getgeodata('GSE4616');
% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MIAME object
MIAMEObj = MIAME(geoStruct);
```

```
% Determine if MIAME object is empty
isempty(MIAMEObj)
```

See Also
`bioma.data.MIAME`

How To
· "Representing Experiment Information in a MIAME Object"

isequal (DataMatrix)

Test DataMatrix objects for equality

Syntax

TF = isequal(*DMObj1*, *DMObj2*)
TF = isequal(*DMObj1*, *DMObj2*, *DMObj3*, ...)

Input Arguments

DMObj1, *DMObj2*, *DMObj3*	DataMatrix objects, such as created by `DataMatrix` (object constructor).

Output Arguments

TF	Logical value indicating if inputs are numerically equal (have the same contents), have the same size (same `NRows` and `NCols` properties), and have the same `RowNames` and `ColNames` properties. `NaN`s are not considered equal to each other.

Description

TF = isequal(*DMObj1*, *DMObj2*) returns logical 1 (`true`) if the input DataMatrix objects, *DMObj1* and *DMObj2*, meet the following:

- Are numerically equal (have the same contents)
- Have the same size (same `NRows` and `NCols` properties)
- Have the same `RowNames` and `ColNames` properties

Otherwise, it returns logical 0 (`false`). *DMObj1* and *DMObj2* do not have to have the same `Name` property. `NaN`s are not considered equal to each other.

TF = isequal(*DMObj1, DMObj2, DMObj3, ...*) returns logical 1 (`true`) if all input DataMatrix objects, *DMObj1*, *DMObj2*, *DMObj3*, etc. meet the following:

- Are numerically equal (have the same contents)
- Have the same size (same `NRows` and `NCols` properties)
- Have the same `RowNames` and `ColNames` properties

Otherwise, it returns logical 0 (`false`). The input DataMatrix objects do not have to have the same `Name` property. `NaN`s are not considered equal to each other.

More About

- DataMatrix object

See Also
DataMatrix | isequaln

isequaln (DataMatrix)

Test DataMatrix objects for equality, treating NaNs as equal

Syntax

TF = isequaln(*DMObj1*, *DMObj2*)
TF = isequaln(*DMObj1*, *DMObj2*, *DMObj3*, ...)

Input Arguments

DMObj1, *DMObj2*, *DMObj3*	DataMatrix objects, such as created by DataMatrix (object constructor).

Output Arguments

TF	Logical value indicating if inputs are numerically equal (have the same contents), have the same size (same NRows and NCols properties), and have the same RowNames and ColNames properties. NaNs are considered equal to each other.

Description

TF = isequaln(*DMObj1*, *DMObj2*) returns logical 1 (true) if the input DataMatrix objects, *DMObj1* and *DMObj2*, meet the following:

- Are numerically equal (have the same contents)
- Have the same size (same NRows and NCols properties)
- Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). *DMObj1* and *DMObj2* do not need to have the same Name property. NaNs are considered equal to each other.

TF = isequaln(*DMObj1, DMObj2, DMObj3, ...*) returns logical 1 (true) if all input DataMatrix objects, *DMObj1*, *DMObj2*, *DMObj3*, etc. meet the following:

- Are numerically equal (have the same contents)
- Have the same size (same NRows and NCols properties)
- Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). The input DataMatrix objects do not need to have the same Name property. NaNs are considered equal to each other.

More About

- DataMatrix object

See Also
DataMatrix | isequal

isequalwithequalnans (DataMatrix)

Test DataMatrix objects for equality, treating NaNs as equal

Syntax

TF = isequalwithequalnans(*DMObj1*, *DMObj2*)
TF = isequalwithequalnans(*DMObj1*, *DMObj2*, *DMObj3*, ...)

Input Arguments

DMObj1, *DMObj2*, *DMObj3*	DataMatrix objects, such as created by DataMatrix (object constructor).

Output Arguments

TF	Logical value indicating if inputs are numerically equal (have the same contents), have the same size (same NRows and NCols properties), and have the same RowNames and ColNames properties. NaNs are considered equal to each other.

Description

TF = isequalwithequalnans(*DMObj1*, *DMObj2*) returns logical 1 (true) if the input DataMatrix objects, *DMObj1* and *DMObj2*, meet the following:

- Are numerically equal (have the same contents)
- Have the same size (same NRows and NCols properties)
- Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). *DMObj1* and *DMObj2* do not have to have the same Name property. NaNs are considered equal to each other.

TF = isequalwithequalnans(*DMObj1*, *DMObj2*, *DMObj3*, ...) returns logical 1 (true) if all input DataMatrix objects, *DMObj1*, *DMObj2*, *DMObj3*, etc. meet the following:

- Are numerically equal (have the same contents)
- Have the same size (same NRows and NCols properties)
- Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). The input DataMatrix objects do not have to have the same Name property. NaNs are considered equal to each other.

More About

- DataMatrix object

See Also
DataMatrix | isequal

isoelectric

Estimate isoelectric point for amino acid sequence

Syntax

```
pI = isoelectric(SeqAA)
[pI Charge] = isoelectric(SeqAA)
isoelectric(..., 'PropertyName', PropertyValue,...)
isoelectric(..., 'PKVals', PKValsValue)
isoelectric(..., 'Charge', ChargeValue)
isoelectric(..., 'Chart', ChartValue)
```

Arguments

SeqAA	Amino acid sequence. Enter a character string or a vector of integers from the table Mapping Amino Acid Letter Codes to Integers. Examples: `'ARN'` or `[1 2 3]`.
PKValsValue	String specifying a file name or path and file name of a PK file containing a table of pK values for amino acids, which `.isoelectric` uses to estimate the isoelectric point (*pI*) of an amino acid sequence. For an example of a PK file format, type `edit Emboss.pK` in the MATLAB command line.
ChargeValue	Property to select a specific pH for estimating charge. Enter a number between `0` and `14`. Default is `7.2`.
ChartValue	Controls the plotting a graph of charge versus pH. Enter `true` or `false`.

Description

pI = `isoelectric(`*SeqAA*`)` returns the estimated isoelectric point (*pI*) for an amino acid sequence using the following pK values:

```
N_term     8.6
```

```
K          10.8
R          12.5
H           6.5
D           3.9
E           4.1
C           8.5
Y          10.1
C_term      3.6
```

The isoelectric point is the pH at which the protein has a net charge of zero.

[*pI Charge*] = isoelectric(*SeqAA*) returns the estimated isoelectric point (*pI*) for an amino acid sequence and the estimated charge for a given pH (default is typical intracellular pH 7.2).

The estimates are skewed by the underlying assumptions that all amino acids are fully exposed to the solvent, that neighboring peptides have no influence on the pK of any given amino acid, and that the constitutive amino acids, as well as the N- and C-termini, are unmodified. Cysteine residues participating in disulfide bridges also affect the true pI and are not considered here. By default, isoelectric uses the EMBOSS amino acid pK table, or you can substitute other values using the property PKVals.

- If the sequence contains ambiguous amino acid characters (b z * –), isoelectric ignores the characters and displays a warning message.

  ```
  Warning: Symbols other than the standard 20 amino acids
  appear in the sequence.
  ```

- If the sequence contains undefined amino acid characters (i j o), isoelectric ignores the characters and displays a warning message.

  ```
  Warning: Sequence contains unknown characters. These will
  be ignored.
  ```

isoelectric(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

isoelectric(..., 'PKVals', *PKValsValue*) uses pK values stored in a *PKValValues*, a PK file, to estimate the isoelectric point (*pI*) of an amino acid sequence. For an example of a PK file format, type edit Emboss.pK in the MATLAB command line.

isoelectric(..., 'Charge', *ChargeValue*) returns the estimated charge of a sequence for a given pH (*ChargeValue*).

isoelectric(..., 'Chart', *ChartValue*) when *ChartValue* is `true`, returns a graph plotting the charge of the protein versus the pH of the solvent.

Examples

```
% Get a sequence from PDB.
pdbSeq = getpdb('1CIV', 'SequenceOnly', true)
% Estimate its isoelectric point.
isoelectric(pdbSeq)

% Plot the charge against the pH for a short polypeptide sequence.
isoelectric('PQGGGGWGQPHGGGWGQPHGGGGWGQGGSHSQG', 'CHART', true)

% Get the Rh blood group D antigen from NCBI and calculate
% its charge at pH 7.3 (typical blood pH).
gpSeq = getgenpept('AAB39602')
[pI Charge] = isoelectric(gpSeq, 'Charge', 7.38)
```

See Also

`aacount` | `molweight`

isomorphism (biograph)

Find isomorphism between two biograph objects

Syntax

```
[Isomorphic, Map] = isomorphism(BGObj1, BGObj2)
[Isomorphic, Map] = isomorphism(BGObj1,
BGObj2,'Directed', DirectedValue)
```

Arguments

BGObj1	Biograph object created by `biograph` (object constructor).
BGObj2	Biograph object created by `biograph` (object constructor).
DirectedValue	Property that indicates whether the graphs are directed or undirected. Enter `false` when both *BGObj1* and *BGObj2* produce undirected graphs. In this case, the upper triangles of the sparse matrices extracted from *BGObj1* and *BGObj2* are ignored. Default is `true`, meaning that both graphs are directed.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*Isomorphic*, *Map*] = isomorphism(*BGObj1*, *BGObj2*) returns logical 1 (`true`) in *Isomorphic* if two N-by-N adjacency matrices extracted from biograph objects *BGObj1* and *BGObj2* are isomorphic graphs, and logical 0 (`false`) otherwise. A graph isomorphism is a 1-to-1 mapping of the nodes in the graph from *BGObj1* and the nodes in the graph from *BGObj2* such that adjacencies are preserved. Return value *Isomorphic* is Boolean. When *Isomorphic* is `true`, *Map* is a row vector containing the node indices that map from *BGObj2* to *BGObj1*. When *Isomorphic* is `false`, the worst-case time complexity is `O(N!)`, where N is the number of nodes.

[*Isomorphic*, *Map*] = isomorphism(*BGObj1*,
BGObj2,'Directed', *DirectedValue*) indicates whether the graphs are directed or undirected. Set *DirectedValue* to `false` when both *BGObj1* and *BGObj2* produce undirected graphs. In this case, the upper triangles of the sparse matrices extracted from *BGObj1* and *BGObj2* are ignored. The default is `true`, meaning that both graphs are directed.

More About

- biograph object

References

[1] Fortin, S. (1996). The Graph Isomorphism Problem. Technical Report, 96-20, Dept. of Computer Science, University of Alberta, Edomonton, Alberta, Canada.

[2] McKay, B.D. (1981). Practical Graph Isomorphism. Congressus Numerantium *30*, 45-87.

[3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

biograph | allshortestpaths | isdag | maxflow | shortestpath | traverse | graphisomorphism | conncomp | isspantree | minspantree | topoorder

isotopicdist

Calculate high-resolution isotope mass distribution and density function

Syntax

```
[MD, Info, DF] = isotopicdist(SeqAA)
[MD, Info, DF] = isotopicdist(Compound)
[MD, Info, DF] = isotopicdist(Formula)
isotopicdist(..., 'NTerminal', NTerminalValue, ...)
isotopicdist(..., 'CTerminal', CTerminalValue, ...)
isotopicdist(..., 'Resolution', ResolutionValue, ...)
isotopicdist(..., 'FFTResolution', FFTResolutionValue, ...)
isotopicdist(..., 'FFTRange', FFTRangeValue, ...)
isotopicdist(..., 'FFTLocation', FFTLocationValue, ...)
isotopicdist(..., 'NoiseThreshold', NoiseThresholdValue, ...)
isotopicdist(..., 'ShowPlot', ShowPlotValue, ...)
```

Description

[*MD*, *Info*, *DF*] = isotopicdist(*SeqAA*) analyzes a peptide sequence and returns a matrix containing the expected mass distribution; a structure containing the monoisotopic mass, average mass, most abundant mass, nominal mass, and empirical formula; and a matrix containing the expected density function.

[*MD*, *Info*, *DF*] = isotopicdist(*Compound*) analyzes a compound specified by a numeric vector or matrix.

[*MD*, *Info*, *DF*] = isotopicdist(*Formula*) analyzes a compound specified by an empirical chemical formula represented by the structure *Formula*. The field names in *Formula* must be valid element symbols and are case sensitive. The respective values in *Formula* are the number of atoms for each element. *Formula* can also be an array of structures that specifies multiple formulas. The field names can be in any order within a structure. However, if there are multiple structures, the order must be the same in each.

isotopicdist(..., '*PropertyName*', *PropertyValue*, ...) calls isotopicdist with optional properties that use property name/property value pairs.

You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/ property value pairs are as follows:

isotopicdist(..., 'NTerminal', *NTerminalValue*, ...) modifies the N-terminal of the peptide.

isotopicdist(..., 'CTerminal', *CTerminalValue*, ...) modifies the C-terminal of the peptide.

isotopicdist(..., 'Resolution', *ResolutionValue*, ...) specifies the approximate resolution of the instrument, given as the Gaussian width (in daltons) at full width at half height (FWHH).

isotopicdist(..., 'FFTResolution', *FFTResolutionValue*, ...) specifies the number of data points per dalton, to compute the fast Fourier transform (FFT) algorithm.

isotopicdist(..., 'FFTRange', *FFTRangeValue*, ...) specifies the absolute range (window size) in daltons for the FFT algorithm and output density function.

isotopicdist(..., 'FFTLocation', *FFTLocationValue*, ...) specifies the location of the FFT range (window) defined by *FFTRangeValue*. It specifies this location by setting the location of the lower limit of the range, relative to the location of the monoisotopic peak, which is computed by isotopicdist.

isotopicdist(..., 'NoiseThreshold', *NoiseThresholdValue*, ...) removes points in the mass distribution that are smaller than 1/*NoiseThresholdValue* times the most abundant mass.

isotopicdist(..., 'ShowPlot', *ShowPlotValue*, ...) controls the display of a plot of the mass distribution.

Input Arguments

SeqAA

Peptide sequence specified by either a:

- String of single-letter codes
- Cell array of strings that specifies multiple peptide sequences

---

Tip You can use the `getgenpept` and `genpeptread` functions to retrieve peptide sequences from the GenPept database or a GenPept-formatted file. You can then use the `cleave` function to perform an insilico digestion on a peptide sequence. The `cleave` function creates a cell array of strings representing peptide fragments, which you can submit to the `isotopicdist` function.

Default:

`Compound`

Compound specified by either a:

- Numeric vector of form `[C H N O S]`, where `C`, `H`, `N`, `O`, and `S` are nonnegative numbers that represent the number of atoms of carbon, hydrogen, nitrogen, oxygen, and sulfur respectively in a compound.
- M-by-5 numeric matrix that specifies multiple compounds, with each row corresponding to a compound and each column corresponding to an atom.

Default:

`Formula`

Chemical formula specified by either a:

- Structure whose field names are valid element symbols and case sensitive. Their respective values are the number of atoms for each element.
- Array of structures that specifies multiple formulas.

Note: If *Formula* is a single structure, the order of the fields does not matter. If *Formula* is an array of structures, then the order of the fields must be the same in each structure.

Default:

`NTerminalValue`

Modification for the N-terminal of the peptide, specified by either:

- One of the strings `'none'`, `'amine'` (default), `'formyl'`, or `'acetyl'`

- Custom modification specified by an empirical formula, represented by a structure. The structure must have field names that are valid element symbols and case sensitive. Their respective values are the number of atoms for each element.

CTerminalValue

Modification for the C-terminal of the peptide, specified by either:

- One of the strings `'none'`, `'freeacid'` (default), or `'amide'`
- Custom modification specified by an empirical formula, represented by a structure. The structure must have field names that are valid element symbols and case sensitive. Their respective values are the number of atoms for each element.

ResolutionValue

Value in daltons specifying the approximate resolution of the instrument, given as the Gaussian width at full width half height (FWHH).

Default: 1/16 Da

FFTResolutionValue

Value specifying the number of data points per dalton, used to compute the FFT algorithm.

Default: 1000

FFTRangeValue

Value specifying the absolute range (window size) in daltons for the FFT algorithm and output density function. By default, this value is automatically estimated based on the weight of the molecule. The actual FFT range used internally by `isotopicdist` is further increased such that *FFTRangeValue * FFTResolutionValue* is a power of two.

Tip Increase the *FFTRangeValue* if the signal represented by the *DF* output appears to be truncated.

Tip Ultrahigh resolution allows you to resolve micropeaks that have the same nominal mass, but slightly different exact masses. To achieve ultrahigh resolution, increase

FFTResolutionValue and reduce *ResolutionValue*, but ensure that *FFTRangeValue* * *FFTResolutionValue* is within the available memory.

Default:

FFTLocationValue

Fraction that specifies the location of the FFT range (window) defined by *FFTRangeValue*. It specifies this location by setting the location of the lower limit of the FFT range, relative to the location of the monoisotopic peak, which is computed by `isotopicdist`. The location of the lower limit of the FFT range is set to the mass of the monoistopic peak - (*FFTLocationValue* * *FFTRangeValue*).

Tip You may need to shift the FFT range to the left in rare cases where a compound contains an element, such as Iron or Argon, whose most abundant isotope is not the lightest one.

Default: 1/16

NoiseThresholdValue

Value that removes points in the mass distribution that are smaller than 1/*NoiseThresholdValue* times the most abundant mass.

Default: 1e6

ShowPlotValue

Controls the display of a plot of the isotopic mass distribution. Choices are `true`, `false`, or *I*, which is an integer specifying a compound. If set to `true`, the first compound is plotted. Default is:

- `false` — When you specify return values.
- `true` — When you do not specify return values.

Default:

Output Arguments

MD

Mass distribution represented by a two-column matrix in which each row corresponds to an isotope. The first column lists the isotopic mass, and the second column lists the probability for that mass.

Info

Structure containing mass information for the peptide sequence or compound in the following fields:

- `NominalMass`
- `MonoisotopicMass`
- `ObservedAverageMass` — Estimated from the *DF* signal output, using instrument resolution specified by the `'Resolution'` property.
- `CalculatedAverageMass` — Calculated directly from the input formula, assuming perfect instrument resolution.
- `MostAbundantMass`
- `Formula` — Structure containing the number of atoms of each element.

DF

Density function represented by a two-column matrix in which each row corresponds to an m/z value. The first column lists the mass, and the second column lists the relative intensity of the signal at that mass.

Examples

Calculate and display the isotopic mass distribution of the peptide sequence MATLAP with an Acetyl N-terminal and an Amide C-terminal:

```
MD = isotopicdist('MATLAP','nterm','Acetyl','cterm','Amide', ...
                  'showplot',true)

MD =

  643.3363    0.6676
  644.3388    0.2306
```

```
       645.3378      0.0797
       646.3386      0.0181
       647.3396      0.0033
       648.3409      0.0005
       649.3423      0.0001
       650.3439      0.0000
       651.3455      0.0000
```
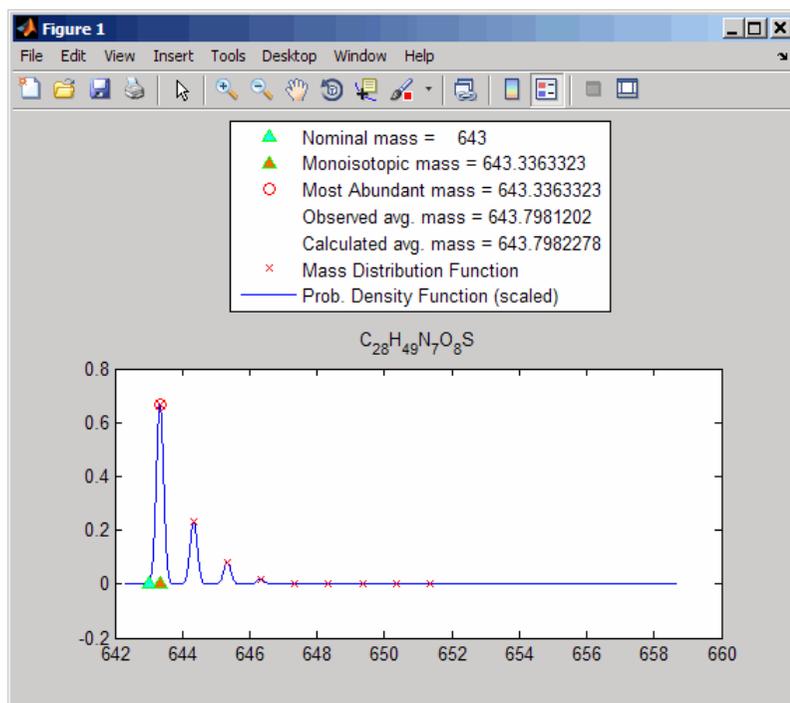

Calculate and display the isotopic mass distribution of Glutamine ($C_5H_{10}N_2O_3$):

```
MD = isotopicdist([5 10 2 3 0],'showplot',true)

MD =

       146.0691      0.9328
       147.0715      0.0595
       148.0733      0.0074
       149.0755      0.0004
       150.0774      0.0000
```
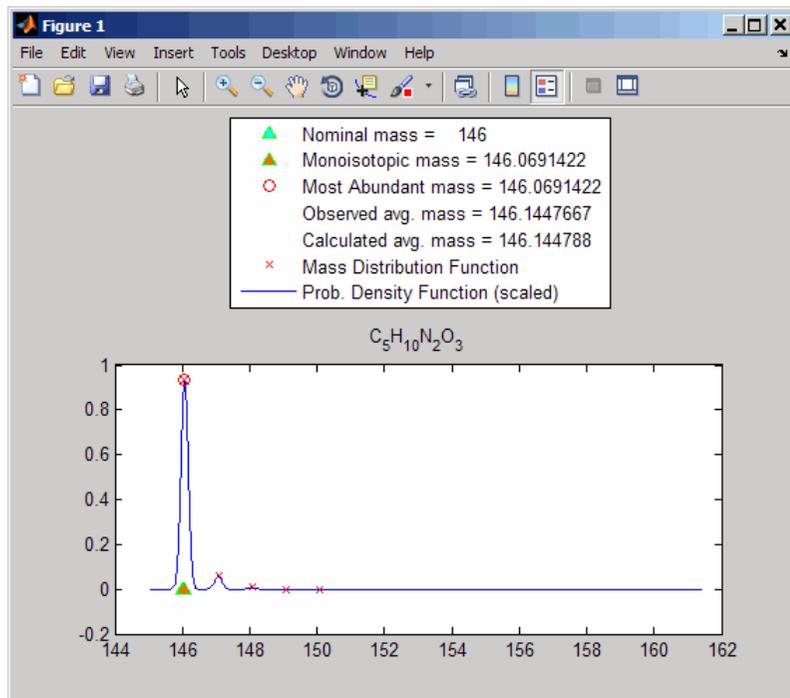
Display the isotopic mass distribution of the "averagine" model, whose molecular formula represents the statistical occurrences of amino acids from all known proteins:

```
isotopicdist([4.9384 7.7583 1.3577 1.4773 0.0417])
```
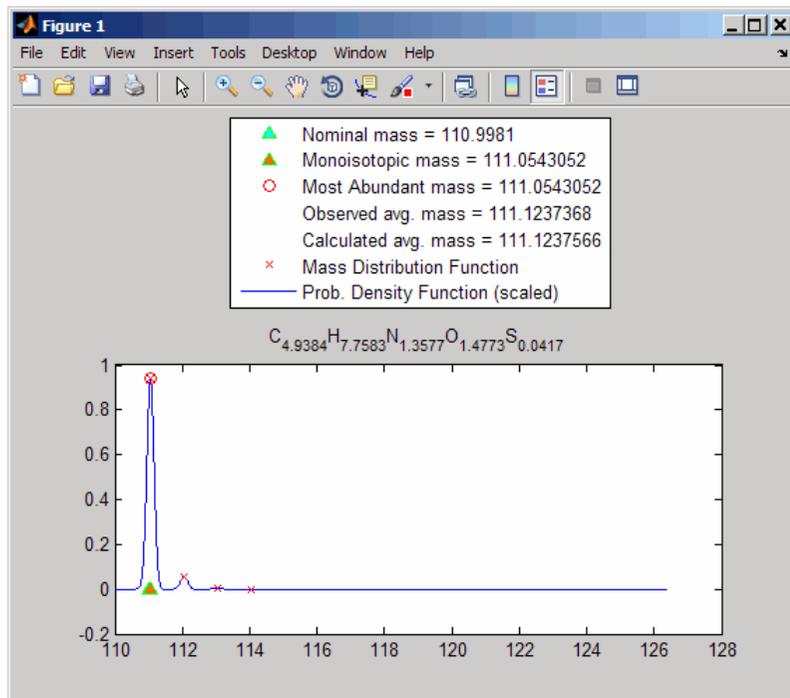
More About

Average Mass

Sum of the average atomic masses of the constituent elements in a molecule.

Monoisotopic Mass

Sum of the masses of the atoms in a molecule using the unbound, ground-state, rest mass of the principle (most abundant) isotope for each element instead of the isotopic average mass.

Most Abundant Mass

Mass of the molecule with the most-highly represented isotope distribution, based on the natural abundance of the isotopes.

Nominal Mass

Sum of the integer masses (ignoring the mass defect) of the most abundant isotope of each element in a molecule.

References

[1] Rockwood, A. L., Van Orden, S. L., and Smith, R. D. (1995). Rapid Calculation of Isotope Distributions. Anal. Chem. *67:15*, 2699–2704.

[2] Rockwood, A. L., Van Orden, S. L., and Smith, R. D. (1996). Ultrahigh Resolution Isotope Distribution Calculations. Rapid Commun. Mass Spectrum *10*, 54–59.

[3] Senko, M.W., Beu, S. C., and McLafferty, F. W. (1995). Automated assignment of charge states from resolved isotopic peaks for multiply charged ions. J. Am. Soc. Mass Spectrom. *6*, 52–56.

[4] Senko, M.W., Beu, S. C., and McLafferty, F. W. (1995). Determination of monoisotopic masses and ion populations for large biomolecules from resolved isotopic distributions. J. Am. Soc. Mass Spectrom. *6*, 229–233.

See Also
`cleave` | `getgenpept` | `genpeptread` | `int2aa` | `nt2aa` | `aminolookup` | `cleavelookup` | `molweight`

isspantree (biograph)

Determine if tree created from biograph object is spanning tree

Syntax

TF = isspantree(*BGObj*)

Arguments

BGObj	Biograph object created by `biograph` (object constructor).

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

TF = isspantree(*BGObj*) returns logical 1 (`true`) if the N-by-N adjacency matrix extracted from a biograph object, *BGObj*, is a spanning tree, and logical 0 (`false`) otherwise. A spanning tree must touch all the nodes and must be acyclic. The lower triangle of the N-by-N adjacency matrix represents an undirected graph, and all nonzero entries indicate the presence of an edge.

More About

- biograph object

References

[1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

biograph | allshortestpaths | isdag | maxflow | shortestpath | traverse | graphisspantree | conncomp | isomorphism | minspantree | topoorder

jcampread

Read JCAMP-DX-formatted files

Syntax

JCAMPStruct = jcampread(*File*)

Input Arguments

File	Either of the following: • String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a JCAMP-DX-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the current folder. • MATLAB character array that contains the text of a JCAMP-DX-formatted file.

Output Arguments

JCAMPStruct	MATLAB structure containing information from a JCAMP-DX-formatted file.

Description

JCAMP-DX is a file format for infrared, NMR, and mass spectrometry data from the Joint Committee on Atomic and Molecular Physical Data (JCAMP). `jcampread` supports reading data from files saved with Versions 4.24, 5, or 6 of the JCAMP-DX format. For more details, see:

`http://www.jcamp-dx.org/`

JCAMPStruct = jcampread(*File*) reads data from *File*, a JCAMP-DX-formatted file, and creates *JCAMPStruct*, a MATLAB structure containing the following fields.

Field
Title
DataType
DataClass
Origin
Owner
Blocks
Notes

The Blocks field of the structure is an array of structures corresponding to each set of data in the file. These structures have the following fields.

Field
XData
YData
XUnits
YUnits
Notes

Examples

1. Open a Web browser to

   `http://www.jcamp-dx.org/testdata.html`

2. Download the `testdata.zip` file to your MATLAB Current Folder.

3. Extract `isas_ms1.dx`, a JCAMP-DX-formatted file, from the `testdata.zip` file to your MATLAB Current Folder.

4. Read the data from the JCAMP-DX-formatted file, `isas_ms1.dx`, into the MATLAB software

   ```
   jcampStruct = jcampread('isas_ms1.dx')

   jcampStruct =
   ```

```
        Title: '2-Chlorphenol'
     DataType: 'MASS SPECTRUM'
    DataClass: 'PEAKTABLE'
       Origin: 'H. Mayer, ISAS Dortmund'
        Owner: 'COPYRIGHT (C) 1993 by ISAS Dortmund, FRG'
       Blocks: [1x1 struct]
        Notes: {8x2 cell}
```

5 Plot the mass spectrum.

```
data = jcampStruct.Blocks(1);
stem(data.XData,data.YData, '.', 'MarkerEdgeColor','w');
title(jcampStruct.Title);
xlabel(data.XUnits);
ylabel(data.YUnits);
```
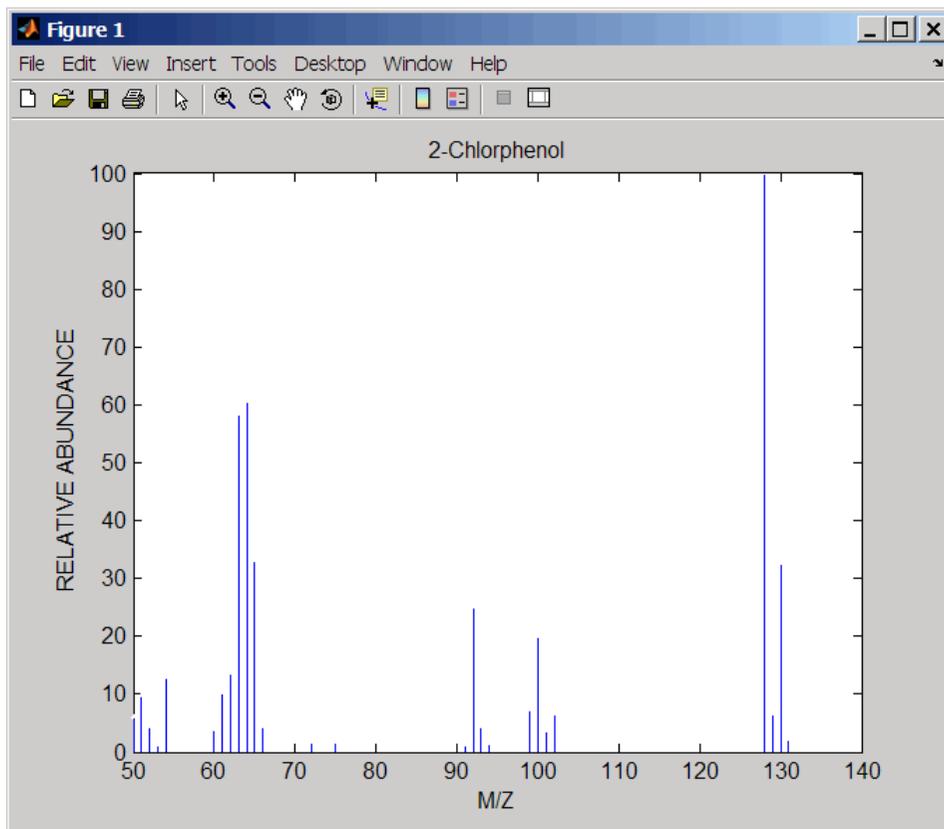
See Also
mslowess | mssgolay | msviewer | mzcdfread | mzxmlread | tgspcread

joinseq

Join two sequences to produce shortest supersequence

Syntax

SeqNT3 = joinseq(*SeqNT1*, *SeqNT2*)

Arguments

SeqNT1, *SeqNT2*	Nucleotide sequences.

Description

SeqNT3 = joinseq(*SeqNT1*, *SeqNT2*) creates a new sequence that is the shortest supersequence of *SeqNT1* and *SeqNT2*. If there is no overlap between the sequences, then *SeqNT2* is concatenated to the end of *SeqNT1*. If the length of the overlap is the same at both ends of the sequence, then the overlap at the end of *SeqNT1* and the start of *SeqNT2* is used to join the sequences.

If *SeqNT1* is a subsequence of *SeqNT2*, then *SeqNT2* is returned as the shortest supersequence and vice versa.

Examples

Join two sequences that contain an overlap.

```
seq1 = 'ACGTAAA';
seq2 = 'AAATGCA';
joined = joinseq(seq1,seq2)

joined =
    ACGTAAATGCA
```

See Also

cat | strcat | strfind

knnclassify

Classify data using nearest neighbor method

Compatibility

knnclassify will be removed in a future release. Instead use fitcknn to fit a knn classification model and classify data using the predict function of ClassificationKNN object.

Syntax

```
Class = knnclassify(Sample, Training, Group)
Class = knnclassify(Sample, Training, Group, k)
Class = knnclassify(Sample, Training, Group, k, distance)
Class = knnclassify(Sample, Training, Group, k, distance, rule)
```

Arguments

Sample	Matrix whose rows will be classified into groups. *Sample* must have the same number of columns as *Training*.
Training	Matrix used to group the rows in the matrix *Sample*. *Training* must have the same number of columns as *Sample*. Each row of *Training* belongs to the group whose value is the corresponding entry of *Group*.
Group	Vector whose distinct values define the grouping of the rows in *Training*.
k	The number of nearest neighbors used in the classification. Default is 1.
distance	String specifying the distance metric. Choices are: • `'euclidean'` — Euclidean distance (default) • `'cityblock'` — Sum of absolute differences

	• `'cosine'` — One minus the cosine of the included angle between points (treated as vectors)
	• `'correlation'` — One minus the sample correlation between points (treated as sequences of values)
	• `'hamming'` — Percentage of bits that differ (suitable only for binary data)
rule	String to specify the rule used to decide how to classify the sample. Choices are:
	• `'nearest'` — Majority rule with nearest point tie-break (default)
	• `'random'` — Majority rule with random point tie-break
	• `'consensus'` — Consensus rule

Description

Class = knnclassify(*Sample*, *Training*, *Group*) classifies the rows of the data matrix *Sample* into groups, based on the grouping of the rows of *Training*. *Sample* and *Training* must be matrices with the same number of columns. *Group* is a vector whose distinct values define the grouping of the rows in *Training*. Each row of *Training* belongs to the group whose value is the corresponding entry of *Group*. knnclassify assigns each row of Sample to the group for the closest row of *Training*. *Group* can be a numeric vector, a string array, or a cell array of strings. *Training* and *Group* must have the same number of rows. knnclassify treats NaNs or empty strings in *Group* as missing values, and ignores the corresponding rows of *Training*. *Class* indicates which group each row of *Sample* has been assigned to, and is of the same type as *Group*.

Class = knnclassify(*Sample*, *Training*, *Group*, *k*) enables you to specify k, the number of nearest neighbors used in the classification. Default is 1.

Class = knnclassify(*Sample*, *Training*, *Group*, *k*, *distance*) enables you to specify the distance metric. Choices for *distance* are:

- `'euclidean'` — Euclidean distance (default)
- `'cityblock'` — Sum of absolute differences
- `'cosine'` — One minus the cosine of the included angle between points (treated as vectors)

- `'correlation'` — One minus the sample correlation between points (treated as sequences of values)

- `'hamming'` — Percentage of bits that differ (suitable only for binary data)

Class = knnclassify(*Sample*, *Training*, *Group*, *k*, *distance*, *rule*) enables you to specify the rule used to decide how to classify the sample. Choices for *rule* are:

- `'nearest'` — Majority rule with nearest point tie-break (default)

- `'random'` — Majority rule with random point tie-break

- `'consensus'` — Consensus rule

The default behavior is to use majority rule. That is, a sample point is assigned to the class the majority of the k nearest neighbors are from. Use `'consensus'` to require a consensus, as opposed to majority rule. When using the `'consensus'` option, points where not all of the k nearest neighbors are from the same class are not assigned to one of the classes. Instead the output Class for these points is NaN for numerical groups, `''` for string named groups, or undefined for categorical groups. When classifying to more than two groups or when using an even value for k, it might be necessary to break a tie in the number of nearest neighbors. Options are `'random'`, which selects a random tiebreaker, and `'nearest'`, which uses the nearest neighbor among the tied groups to break the tie. The default behavior is majority rule, with nearest tie-break.

Examples

Classifying Rows

The following example classifies the rows of the matrix `sample`:

```
sample = [.9 .8;.1 .3;.2 .6]

sample =
    0.9000    0.8000
    0.1000    0.3000
    0.2000    0.6000

training=[0 0;.5 .5;1 1]

training =
```

```
         0         0
    0.5000    0.5000
    1.0000    1.0000

group = [1;2;3]

group =
     1
     2
     3

class = knnclassify(sample, training, group)

class =
     3
     1
     2
```

Row 1 of `sample` is closest to row 3 of `training`, so `class(1) = 3`. Row 2 of `sample` is closest to row 1 of `training`, so `class(2) = 1`. Row 3 of `sample` is closest to row 2 of `training`, so `class(3) = 2`.

Classifying Rows into One of Two Groups

The following example classifies each row of the data in `sample` into one of the two groups in `training`. The following commands create the matrix `training` and the grouping variable `group`, and plot the rows of `training` in two groups.

```
training = [mvnrnd([ 1  1],   eye(2), 100); ...
            mvnrnd([-1 -1], 2*eye(2), 100)];
group = [repmat(1,100,1); repmat(2,100,1)];
gscatter(training(:,1),training(:,2),group,'rb','+x');
legend('Training group 1', 'Training group 2');
hold on;
```
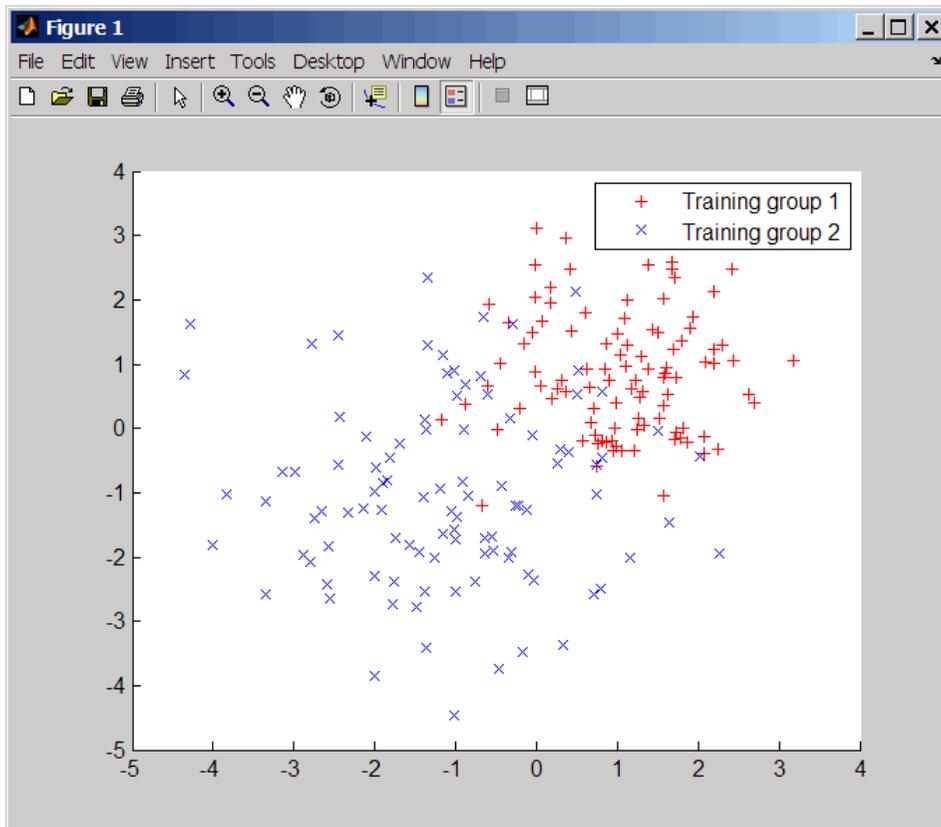
The following commands create the matrix `sample`, classify its rows into two groups, and plot the result.

```
sample = unifrnd(-5, 5, 100, 2);
% Classify the sample using the nearest neighbor classification
c = knnclassify(sample, training, group);
gscatter(sample(:,1),sample(:,2),c,'mc'); hold on;
legend('Training group 1','Training group 2', ...
       'Data in group 1','Data in group 2');
hold off;
```
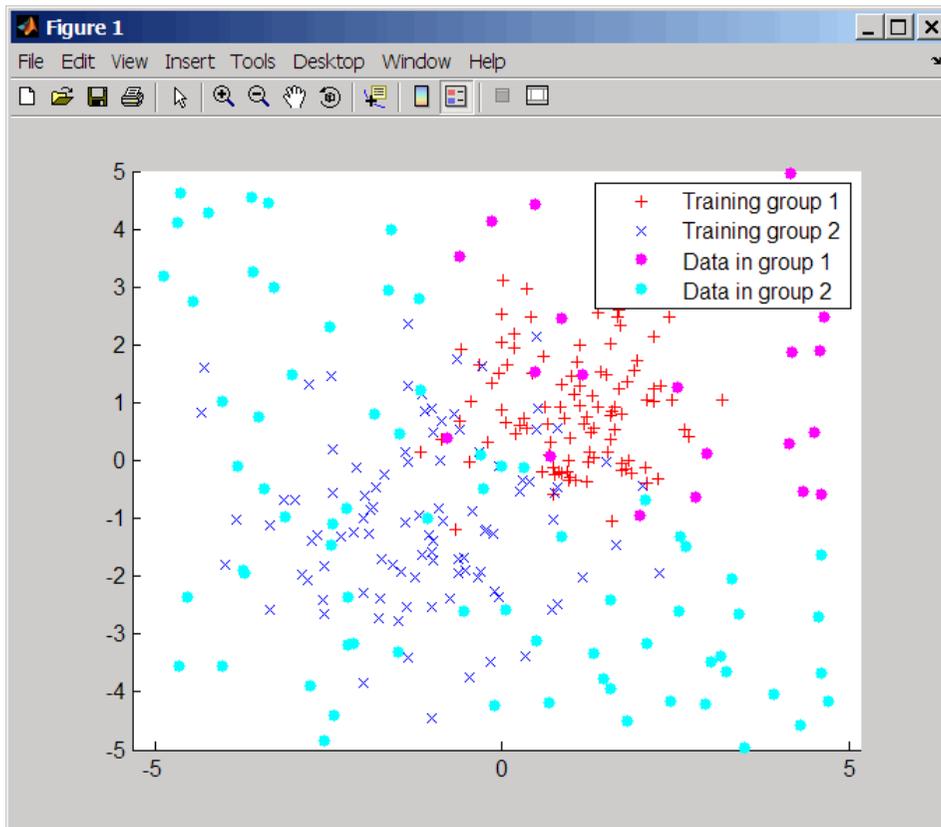
Classifying Rows Using the Three Nearest Neighbors

The following example uses the same data as in Classifying Rows into One of Two Groups, but classifies the rows of `sample` using three nearest neighbors instead of one.

```
gscatter(training(:,1),training(:,2),group,'rb','+x');
hold on;
c3 = knnclassify(sample, training, group, 3);
gscatter(sample(:,1),sample(:,2),c3,'mc','o');
legend('Training group 1','Training group 2','Data in group 1','Data in group 2');
```
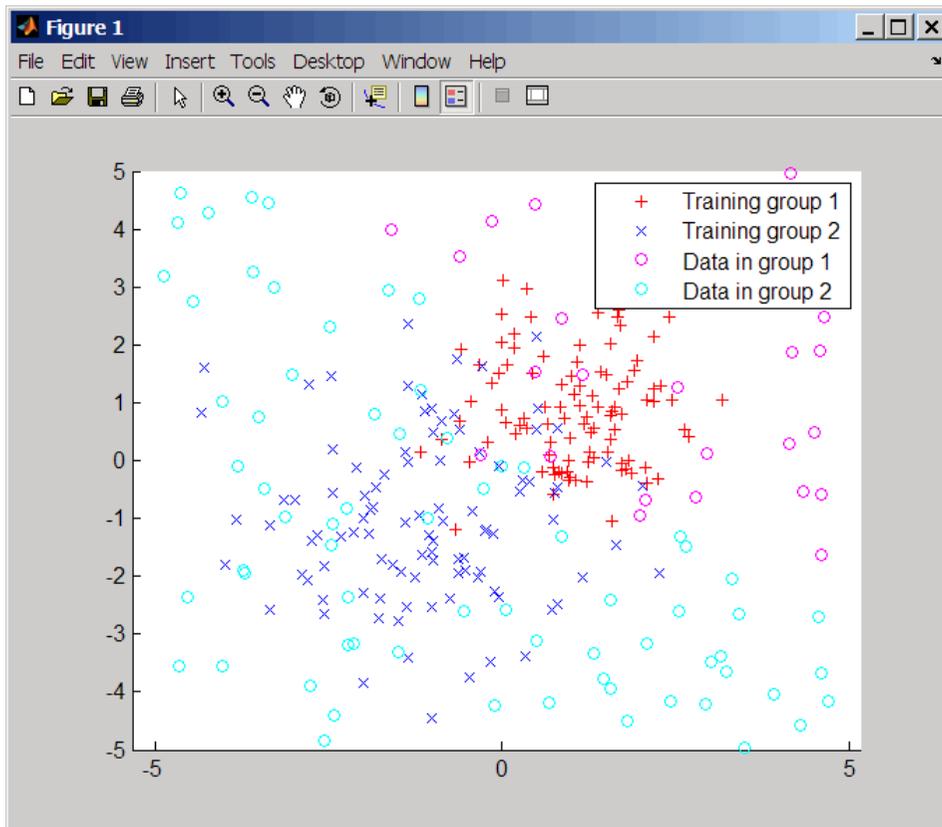
If you compare this plot with the one in Classifying Rows into One of Two Groups, you see that some of the data points are classified differently using three nearest neighbors.

References

[1] Mitchell, T. (1997). Machine Learning, (McGraw-Hill).

See Also
classperf | fitcknn | crossvalind | knnimpute | classify | svmclassify | svmtrain

knnimpute

Impute missing data using nearest-neighbor method

Syntax

```
knnimpute(Data)
knnimpute(Data, k)
knnimpute(..., 'Distance', DistanceValue, ...)
knnimpute(..., 'DistArgs', DistArgsValue, ...)
knnimpute(..., 'Weights', WeightsValues, ...)
knnimpute(..., 'Median', MedianValue, ...)
```

Arguments

Data	Matrix
k	The number of nearest neighbors used. The default is 1.

Description

knnimpute(*Data*) replaces NaNs in Data with the corresponding value from the nearest-neighbor column. The nearest-neighbor column is the closest column in Euclidean distance. If the corresponding value from the nearest-neighbor column is also NaN, the next nearest column is used.

knnimpute(*Data*, *k*) replaces NaNs in Data with a weighted mean of the k nearest-neighbor columns. The weights are inversely proportional to the distances from the neighboring columns.

knnimpute(..., '*PropertyName*', *PropertyValue*, ...) calls knnimpute with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

knnimpute(..., 'Distance', *DistanceValue*, ...) computes nearest-neighbor columns using the distance metric distfun. The choices for *DistanceValue* are:

'euclidean'	Euclidean distance (default).
'seuclidean'	Standardized Euclidean distance — each coordinate in the sum of squares is inversely weighted by the sample variance of that coordinate.
'cityblock'	City block distance.
'mahalanobis'	Mahalanobis distance.
'minkowski'	Minkowski distance with exponent 2.
'cosine'	One minus the cosine of the included angle.
'correlation'	One minus the sample correlation between observations, treated as sequences of values.
'hamming'	Hamming distance — the percentage of coordinates that differ.
'jaccard'	One minus the Jaccard coefficient — the percentage of nonzero coordinates that differ.
'chebychev'	Chebychev distance (maximum coordinate difference).
function handle	A handle to a distance function, specified using @, for example, @distfun.

See pdist for more details.

knnimpute(..., 'DistArgs', *DistArgsValue*, ...) passes arguments (*DistArgsValue*) to the function distfun. *DistArgsValue* can be a single value or a cell array of values.

knnimpute(..., 'Weights', *WeightsValues*, ...) lets you specify the weights used in the weighted mean calculation. w should be a vector of length k.

knnimpute(..., 'Median', *MedianValue*, ...) when *MedianValue* is true, uses the median of the k nearest neighbors instead of the weighted mean.

Examples

Example 1

```
A = [1 2 5;4 5 7;NaN -1 8;7 6 0]
```

```
A =

     1     2     5
     4     5     7
   NaN    -1     8
     7     6     0
```

Note that `A(3,1) = NaN`. Because column 2 is the closest column to column 1 in Euclidean distance, `knnimpute` imputes the `(3,1)` entry of column 1 to be the corresponding entry of column 2, which is `-1`.

```
knnimpute(A)

ans =

     1     2     5
     4     5     7
    -1    -1     8
     7     6     0
```

Example 2

The following example loads the data set `yeastdata` and imputes missing values in the array `yeastvalues`:

```
load yeastdata
% Remove data for empty spots
emptySpots = strcmp('EMPTY',genes);
yeastvalues(emptySpots,:) = [];
genes(emptySpots) = [];
% Impute missing values
imputedValues = knnimpute(yeastvalues);
```

References

[1] Speed, T. (2003). Statistical Analysis of Gene Expression Microarray Data (Chapman & Hall/CRC).

[2] Hastie, T., Tibshirani, R., Sherlock, G., Eisen, M., Brown, P., and Botstein, D. (1999). "Imputing missing data for gene expression arrays", Technical Report, Division of Biostatistics, Stanford University.

[3] Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. (2001). Missing value estimation methods for DNA microarrays. Bioinformatics *17(6)*, 520–525.

See Also

knnclassify | pdist | isnan | nanmean | nanmedian

ldivide (DataMatrix)

Left array divide DataMatrix objects

Syntax

```
DMObjNew = ldivide(DMObj1, DMObj2)
DMObjNew = DMObj1 .\ DMObj2
DMObjNew = ldivide(DMObj1, B)
DMObjNew = DMObj1 .\ B
DMObjNew = ldivide(B, DMObj1)
DMObjNew = B .\ DMObj1
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

DMObjNew	DataMatrix object created by left array division.

Description

DMObjNew = `ldivide`(*DMObj1*, *DMObj2*) or the equivalent *DMObjNew* = *DMObj1* .\ *DMObj2* performs an element-by-element left array division of the DataMatrix objects *DMObj1* and *DMObj2* and places the results in *DMObjNew*, another DataMatrix object. In other words, `ldivide` divides each element in *DMObj2* by the corresponding element in *DMObj1*. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = ldivide(*DMObj1*, *B*) or the equivalent *DMObjNew* = *DMObj1* .\ *B* performs an element-by-element left array division of the DataMatrix object *DMObj1* and *B*, a numeric or logical array, and places the results in *DMObjNew*, another DataMatrix object. In other words, ldivide divides each element in *B* by the corresponding element in *DMObj1*. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

DMObjNew = ldivide(*B*, *DMObj1*) or the equivalent *DMObjNew* = *B* .\ *DMObj1* performs an element-by-element left array division of *B*, a numeric or logical array, and the DataMatrix object *DMObj1*, and places the results in *DMObjNew*, another DataMatrix object. In other words, ldivide divides each element in *DMObj1* by the corresponding element in *B*. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

Note: Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls *DMObjNew* = ldivide(*X*, *Y*) for the syntax *DMObjNew* = *X* .\ *Y* when *X* or *Y* is a DataMatrix object.

More About

- DataMatrix object

See Also
DataMatrix | times | rdivide

le (DataMatrix)

Test DataMatrix objects for less than or equal to

Syntax

```
T = le(DMObj1, DMObj2)
T = DMObj1 <= DMObj2
T = le(DMObj1, B)
T = DMObj1 <= B
T = le(B, DMObj1)
T = B <= DMObj1
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

T	Logical matrix of the same size as *DMObj1* and *DMObj2* or *DMObj1* and *B*. It contains logical 1 (true) where elements in the first input are less than or equal to the corresponding element in the second input, and logical 0 (false) otherwise.

Description

T = `le(`*DMObj1*`, `*DMObj2*`)` or the equivalent *T* = *DMObj1* `<=` *DMObj2* compares each element in DataMatrix object *DMObj1* to the corresponding element in DataMatrix object *DMObj2*, and returns *T*, a logical matrix of the same size as *DMObj1* and *DMObj2*, containing logical 1 (true) where elements in *DMObj1* are less than or equal to the corresponding element in *DMObj2*, and logical 0 (false) otherwise. *DMObj1* and *DMObj2*

must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). *DMObj1* and *DMObj2* can have different Name properties.

T = le(*DMObj1*, *B*) or the equivalent *T* = *DMObj1* <= *B* compares each element in DataMatrix object *DMObj1* to the corresponding element in *B*, a numeric or logical array, and returns *T*, a logical matrix of the same size as *DMObj1* and *B*, containing logical 1 (true) where elements in *DMObj1* are less than or equal to the corresponding element in *B*, and logical 0 (false) otherwise. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

T = le(*B*, *DMObj1*) or the equivalent *T* = *B* <= *DMObj1* compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are less than or equal to the corresponding element in *DMObj1*, and logical 0 (false) otherwise. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls *T* = le(*X*, *Y*) for the syntax *T* = *X* <= *Y* when *X* or *Y* is a DataMatrix object.

More About

· DataMatrix object

See Also
DataMatrix | ge

localalign

Return local optimal and suboptimal alignments between two sequences

Syntax

AlignStruct = localalign(*Seq1*, *Seq2*)
AlignStruct = localalign(*Seq1*, *Seq2*, ...'NumAln', *NumAlnValue*, ...)
AlignStruct = localalign(*Seq1*, *Seq2*, ...'MinScore', *MinScoreValue*, ...)
AlignStruct = localalign(*Seq1*, *Seq2*, ...'Percent', *PercentValue*, ...)
AlignStruct = localalign(*Seq1*, *Seq2*, ...'DoAlignment', *DoAlignmentValue*, ...)
AlignStruct = localalign(*Seq1*, *Seq2*, ...'Alphabet', *AlphabetValue*, ...)
AlignStruct = localalign(*Seq1*, *Seq2*, ...'ScoringMatrix', *ScoringMatrixValue*, ...)
AlignStruct = localalign(*Seq1*, *Seq2*, ...'Scale', *ScaleValue*, ...)
AlignStruct = localalign(*Seq1*, *Seq2*, ...'GapOpen', *GapOpenValue*, ...)

Description

AlignStruct = localalign(*Seq1*, *Seq2*) returns information about the first optimal (highest scoring) local alignment between two sequences in a MATLAB structure.

AlignStruct = localalign(*Seq1*, *Seq2*, ...'*PropertyName*', *PropertyValue*, ...) calls localalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

AlignStruct = localalign(*Seq1*, *Seq2*, ...'NumAln', *NumAlnValue*, ...) returns information about one or more nonintersecting, local alignments (optimal and suboptimal). It limits the number of alignments to return by specifying the number of

local alignments to return. It returns the alignments in decreasing order according to their score.

AlignStruct = localalign(*Seq1*, *Seq2*, ...'MinScore', *MinScoreValue*, ...) returns information about nonintersecting, local alignments (optimal and suboptimal), whose score is greater than *MinScoreValue*.

AlignStruct = localalign(*Seq1*, *Seq2*, ...'Percent', *PercentValue*, ...) returns information about one or more nonintersecting local alignments (optimal and suboptimal), whose scores are within *PercentValue* percent of the highest score. It returns the alignments in decreasing order according to their score.

AlignStruct = localalign(*Seq1*, *Seq2*, ...'DoAlignment', *DoAlignmentValue*, ...) specifies whether to include the pairwise alignments in the Alignment field of the output structure. Choices are true (default) or false.

AlignStruct = localalign(*Seq1*, *Seq2*, ...'Alphabet', *AlphabetValue*, ...) specifies the type of sequences. Choices are 'AA' (default) or 'NT'.

AlignStruct = localalign(*Seq1*, *Seq2*, ...'ScoringMatrix', *ScoringMatrixValue*, ...) specifies the scoring matrix to use for the local alignment.

AlignStruct = localalign(*Seq1*, *Seq2*, ...'Scale', *ScaleValue*, ...) specifies a scale factor applied to the output scores, thereby controlling the units of the output scores. Choices are any positive value. Default is 1, which does not change the units of the output score.

AlignStruct = localalign(*Seq1*, *Seq2*, ...'GapOpen', *GapOpenValue*, ...) specifies the penalty for opening a gap in the alignment. Choices are any positive value. Default is 8.

Input Arguments

Seq1

First amino acid or nucleotide sequence specified by any of the following:

- Character string of letters representing amino acids or nucleotides, such as returned by int2aa or int2nt

- Vector of integers representing amino acids or nucleotides, such as returned by `aa2int` or `nt2int`
- MATLAB structure containing a `Sequence` field, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, `getgenbank`, `getgenpept`, `genpeptread`, `getpdb`, `pdbread`, or `sffread`

Tip For help with letter and integer representations of amino acids and nucleotides, see Amino Acid Lookup or Nucleotide Lookup.

Default:

Seq2

Second amino acid or nucleotide sequence, which `localalign` aligns with *Seq1*.

Default:

NumAlnValue

Positive scalar (< or = 2^12) specifying the number of alignments to return. `localalign` returns the top *NumAlnValue* local, nonintersecting alignments (optimal and suboptimal). If the number of optimal alignments is greater than *NumAlnValue*, then `localalign` returns the first *NumAlnValue* alignments based on their order in the trace back matrix.

Note: If you specify a *NumAlnValue*, you cannot specify a *MinScoreValue* or *PercentValue*.

Tip Use *NumAlnValue* to return multiple alignments when you are aligning low complexity sequences and must consider several local alignments.

Default: 1

MinScoreValue

Positive scalar specifying the minimum score of local, nonintersecting alignments (optimal and suboptimal) to return.

> **Note:** If you specify a *MinScoreValue*, you cannot specify a *NumAlnValue* or *PercentValue*.

> **Tip** Use *MinScoreValue* to return suboptimal alignments, for example when you are interested in accounting for sequencing errors or imperfect scoring matrices.

Default:

PercentValue

Positive scalar between 0 and 100 that limits the return of local, nonintersecting alignments (optimal and suboptimal) to those alignments with a score within *PercentValue* percent of the highest score. For example, if the highest score is 10.5 and you specify 5 for *PercentValue*, then localalign determines a minimum score of 10.5 − (10.5 * 0.05) = 9.975. It returns all alignments with a score of 9.975 or higher.

> **Note:** If you specify a *PercentValue*, you cannot specify a *NumAlnValue* or *MinScoreValue*.

> **Tip** Use *PercentValue* to return optimal and suboptimal alignments when you do not know how similar the two sequences are or how well they score against a given scoring matrix.

Default:

DoAlignmentValue

Controls the inclusion of the pairwise alignments in the Alignment field of the output structure. Choices are true (default) or false.

Default:

AlphabetValue

String specifying the type of sequences. Choices are 'AA' (default) or 'NT'.

Default:

ScoringMatrixValue

Either of the following:

- String specifying the scoring matrix to use for the local alignment. Choices for amino acid sequences are:

  - `'BLOSUM62'`
  - `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'`
  - `'BLOSUM100'`
  - `'PAM10'` increasing by 10 up to `'PAM500'`
  - `'DAYHOFF'`
  - `'GONNET'`

  Default is:

  - `'BLOSUM50'` — When *AlphabetValue* equals `'AA'`
  - `'NUC44'` — When *AlphabetValue* equals `'NT'`

  Note: The previous scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the `'Scale'` property to specify an additional scale factor to convert the output score from bits to another unit.

- Matrix representing the scoring matrix to use for the local alignment, such as returned by the `blosum`, `pam`, `dayhoff`, `gonnet`, or `nuc44` function.

  Note: If you use a scoring matrix that you created or was created by one of the previous functions, the matrix does not include a scale factor. The output score is returned in the same units as the scoring matrix. You can use the `'Scale'` property to specify a scale factor to convert the output score to another unit.

Note: If you need to compile `localalign` into a stand-alone application or software component using MATLAB Compiler™, use a matrix instead of a string for *ScoringMatrixValue*.

Default:

ScaleValue

Positive value that specifies a scale factor that is applied to the output scores, thereby controlling the units of the output scores.

For example, if the output score is initially determined in bits, and you enter `log(2)` for *ScaleValue*, then `localalign` returns *Score* in nats.

Default is `1`, which does not change the units of the output score.

Note: If the `'ScoringMatrix'` property also specifies a scale factor, then `localalign` uses it first to scale the output score. It then applies the scale factor specified by *ScaleValue* to rescale the output score.

Tip Before comparing alignment scores from multiple alignments, ensure that the scores are in the same units. Use the `'Scale'` property to control the units of the output scores.

Default:

GapOpenValue

Positive value specifying the penalty for opening a gap in the alignment.

Default: 8

Output Arguments

AlignStruct

MATLAB structure or array of structures containing information about the local optimal and suboptimal alignments between two sequences. Each structure represents an optimal or suboptimal alignment and contains the following fields.

Field	Description
Score	Score for the local optimal or suboptimal alignment.

Field	Description
Start	1-by-2 vector of indices indicating the starting point in each sequence for the alignment.
Stop	1-by-2 vector of indices indicating the stopping point in each sequence for the alignment.
Alignment	3-by-N character array showing the two sequences, *Seq1* and *Seq2*, in the first and third rows. It also shows symbols representing the optimal or suboptimal local alignment between the two sequences in the second row.

Examples

Limit the number of alignments to return between two sequences by specifying the number of alignments:

```
% Create variables containing two amino acid sequences.
Seq1 = 'VSPAGMASGYDPGKA';
Seq2 = 'IPGKATREYDVSPAG';

% Use the NumAln property to return information about the
% top three local alignments.
struct1 = localalign(Seq1, Seq2, 'numaln', 3)

struct1 =

        Score: [3x1 double]
        Start: [3x2 double]
         Stop: [3x2 double]
    Alignment: {3x1 cell}

% View the scores of the first and second alignments.
struct1.Score(1:2)

ans =

    11.0000
     9.6667

% View the first alignment.
struct1.Alignment{1}
```

```
ans =

VSPAG
|||||
VSPAG
```

Limit the number of alignments to return between two sequences by specifying a minimum score:

```
% Create variables containing two amino acid sequences.
Seq1 = 'VSPAGMASGYDPGKA';
Seq2 = 'IPGKATREYDVSPAG';

% Use the MinScore property to return information about
% only local alignments with a score greater than 8.
% Use the DoAlignment property to exclude the actual alignments.
struct2 = localalign(Seq1,Seq2,'minscore',8,'doalignment',false)

struct2 =

    Score: [2x1 double]
    Start: [2x2 double]
     Stop: [2x2 double]
```

Limit the number of alignments to return between two sequences by specifying a percentage from the maximum score:

```
% Create variables containing two amino acid sequences.
Seq1 = 'VSPAGMASGYDPGKA';
Seq2 = 'IPGKATREYDVSPAG';

% Use the Percent property to return information about only
% local alignments with a score within 15% of the maximum score.
struct3 = localalign(Seq1, Seq2, 'percent', 15)

struct3 =

        Score: [2x1 double]
        Start: [2x2 double]
         Stop: [2x2 double]
    Alignment: {2x1 cell}
```

Specify a scoring matrix and gap opening penalty when aligning two sequences:

```
% Create variables containing two nucleotide sequences.
Seq1 = 'CCAATCTACTACTGCTTGCAGTAC';
Seq2 = 'AGTCCGAGGGCTACTCTACTGAAC';

% Create a scoring matrix with a match score of 10 and a mismatch
% score of -9
sm = [10 -9 -9 -9;
      -9 10 -9 -9;
      -9 -9 10 -9;
      -9 -9 -9 10];

% Use the ScoringMatrix and GapOpen properties when returning
% information about the top three local alignments.
struct4 = localalign(Seq1, Seq2, 'alpha', 'nt', ...
        'scoringmatrix', sm, 'gapopen', 20, 'numaln', 3)

struct4 =

        Score: [3x1 double]
        Start: [3x2 double]
         Stop: [3x2 double]
    Alignment: {3x1 cell}
```

More About

Nonintersecting Alignments

Alignments having no matches or mismatches in common.

Optimal Alignment

An alignment with the highest score.

Suboptimal Alignment

An alignment with a score less than the highest score.

- "Retrieve Sequence Information from a Public Database"
- "View and Align Multiple Sequences"
- Amino Acid Lookup
- Nucleotide Lookup

- http://www.ncbi.nlm.nih.gov/
- http://www.expasy.ch/sprot/
- http://www.rcsb.org/pdb/home/home.do
- http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=show&f=main&m=main&s=main

References

[1] Barton, G. (1993). An efficient algorithm to locate all locally optimal alignments between two sequences allowing for gaps. CABIOS *9*, 729–734.

See Also

nwalign | swalign | showalignment | blosum | pam | dayhoff | gonnet | nuc44

lt (DataMatrix)

Test DataMatrix objects for less than

Syntax

```
T = lt(DMObj1, DMObj2)
T = DMObj1 < DMObj2
T = lt(DMObj1, B)
T = DMObj1 < B
T = lt(B, DMObj1)
T = B < DMObj1
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

T	Logical matrix of the same size as *DMObj1* and *DMObj2* or *DMObj1* and *B*. It contains logical 1 (true) where elements in the first input are less than the corresponding element in the second input, and logical 0 (false) otherwise.

Description

T = `lt`(*DMObj1*, *DMObj2*) or the equivalent *T* = *DMObj1* < *DMObj2* compares each element in DataMatrix object *DMObj1* to the corresponding element in DataMatrix object *DMObj2*, and returns *T*, a logical matrix of the same size as *DMObj1* and *DMObj2*, containing logical 1 (true) where elements in *DMObj1* are less than the corresponding element in *DMObj2*, and logical 0 (false) otherwise. *DMObj1* and *DMObj2* must have

the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). *DMObj1* and *DMObj2* can have different Name properties.

T = lt(*DMObj1*, *B*) or the equivalent *T* = *DMObj1* < *B* compares each element in DataMatrix object *DMObj1* to the corresponding element in *B*, a numeric or logical array, and returns *T*, a logical matrix of the same size as *DMObj1* and *B*, containing logical 1 (true) where elements in *DMObj1* are less than the corresponding element in *B*, and logical 0 (false) otherwise. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

T = lt(*B*, *DMObj1*) or the equivalent *T* = *B* < *DMObj1* compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are less than the corresponding element in *DMObj1*, and logical 0 (false) otherwise. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls *T* = lt(*X*, *Y*) for the syntax *T* = *X* < *Y* when *X* or *Y* is a DataMatrix object.

More About

- DataMatrix object

See Also
DataMatrix | gt

maboxplot

Create box plot for microarray data

Syntax

```
maboxplot(MAData)
maboxplot(MAData, ColumnName)
maboxplot(MAStruct, FieldName)
H = maboxplot(...)
[H, HLines] = maboxplot(...)
maboxplot(..., 'Title', TitleValue, ...)
maboxplot(..., 'Notch', NotchValue, ...)
maboxplot(..., 'Symbol', SymbolValue, ...)
maboxplot(..., 'Orientation', OrientationValue, ...)
maboxplot(..., 'WhiskerLength', WhiskerLengthValue, ...)
maboxplot(..., 'BoxPlot', BoxPlotValue, ...)
```

Arguments

MAData	DataMatrix object, numeric array, or a structure containing a field called `Data`. The values in the columns of *MAData* will be used to create box plots. If a DataMatrix object, the column names are used as labels in the box plot.
ColumnName	An array of column names corresponding to the data in *MAData* used as labels in the box plot.
MAStruct	A microarray data structure.
FieldName	A field within the microarray data structure, *MAStruct*. The values in the field *FieldName* will be used to create box plots.
TitleValue	String to use as the title for the plot. The default title is *FieldName*.
NotchValue	Logical specifying the type of boxes drawn. Choices are: • true — Notched boxes

	• false — Square boxes
	Default is false.
OrientationValue	String specifying the orientation of the box plot. Choices are:
	• 'Vertical'
	• 'Horizontal' (default)
WhiskerLengthValue	Value specifying the maximum length of the whiskers as a function of the interquartile range (IQR). The whisker extends to the most extreme data value within *WhiskerLengthValue**IQR of the box. Default = 1.5. If *WhiskerLengthValue* equals 0, then maboxplot displays all data values outside the box, using the plotting symbol Symbol.
BoxPlotValue	A cell array of property name/property value pairs to pass to the Statistics and Machine Learning Toolbox boxplot function, which creates the box plot. For valid pairs, see the boxplot function.

Description

maboxplot(*MAData*) displays a box plot of the values in the columns of *MAData*. *MAData* can be a DataMatrix object, numeric array, or a structure containing a field called Data, containing microarray data.

maboxplot(*MAData, ColumnName*) labels the box plot column names.

maboxplot(*MAStruct, FieldName*) displays a box plot of the values in the field FieldName in the microarray data structure *MAStruct*. If *MAStruct* is block based, maboxplot creates a box plot of the values in the field *FieldName* for each block.

Note: If you provide *MAStruct*, without providing *FieldName*, maboxplot uses the Signal element in the ColumnNames field of *MAStruct*, if Affymetrixdata, or the first element in the ColumnNames field of *MAStruct*, otherwise.

H = maboxplot(...) returns the handle of the box plot axes.

[*H*, *HLines*] = maboxplot(...) returns the handles of the lines used to separate the different blocks in the image.

maboxplot(..., '*PropertyName*', *PropertyValue*, ...) calls maboxplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

maboxplot(..., 'Title', *TitleValue*, ...) allows you to specify the title of the plot. The default *TitleValue* is FieldName.

maboxplot(..., 'Notch', *NotchValue*, ...) if *NotchValue* is true, draws notched boxes. The default is false to show square boxes.

maboxplot(..., 'Symbol', *SymbolValue*, ...) allows you to specify the symbol used for outlier values. The default Symbol is '+'.

maboxplot(..., 'Orientation', *OrientationValue*, ...) allows you to specify the orientation of the box plot. The choices are 'Vertical' and 'Horizontal'. The default is 'Vertical'.

maboxplot(..., 'WhiskerLength', *WhiskerLengthValue*, ...) allows you to specify the whisker length for the box plot. *WhiskerLengthValue* defines the maximum length of the whiskers as a function of the interquartile range (IQR) (default = 1.5). The whisker extends to the most extreme data value within WhiskerLength*IQR of the box. If *WhiskerLengthValue* equals 0, then maboxplot displays all data values outside the box, using the plotting symbol Symbol.

maboxplot(..., 'BoxPlot', *BoxPlotValue*, ...) allows you to specify arguments to pass to the boxplot function, which creates the box plot. *BoxPlotValue* is a cell array of property name/property value pairs. For valid pairs, see the boxplot function.

Examples

Display Box Plots for Microarray Data

This example shows how to display box plots for microarray data.

Load the MAT-file, provided with the Bioinformatics Toolbox™ software, that contains yeast data. This MAT-file includes three variables: yeastvalues, a matrix of gene

expression data, `genes` , a cell array of GenBank® accession numbers for labeling the rows in yeastvalues, and `times` , a vector of time values for labeling the columns in yeastvalues.

```
load yeastdata
```

Show the box plot of gene expression data.

```
maboxplot(yeastvalues,times);
xlabel('Sample Times');
```
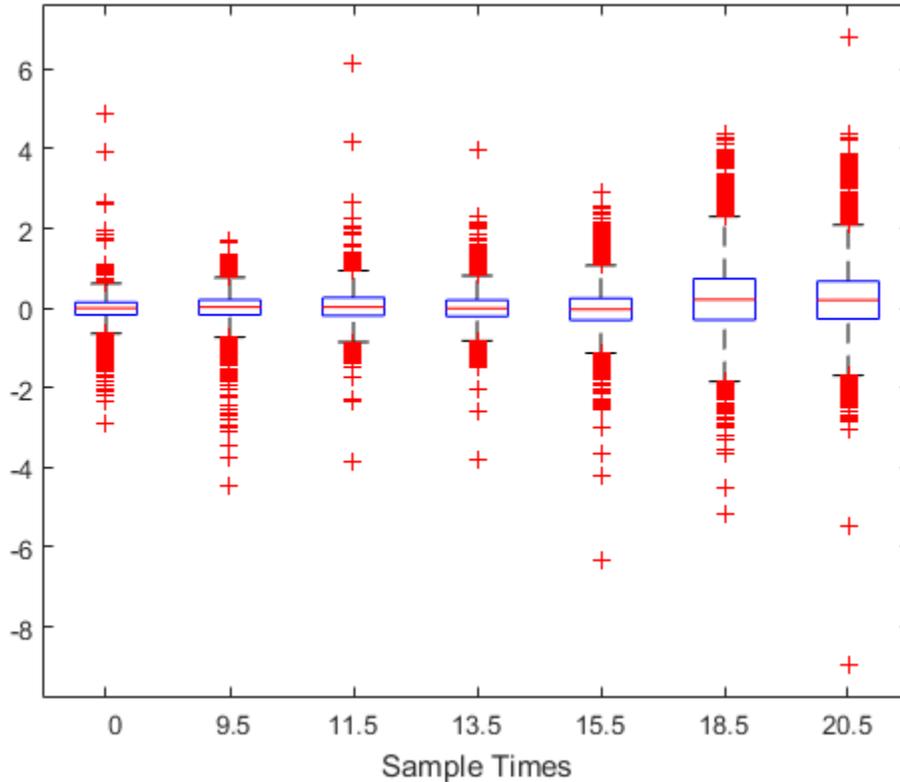

Use the `gprread` function to create a structure containing microarray data, and plot the data using name-value pair arguments of the `maboxplot` function.

```
madata = gprread('mouse_a1wt.gpr');
maboxplot(madata,'F635 Median - B635','TITLE', 'Cy5 Channel FG - BG');
```
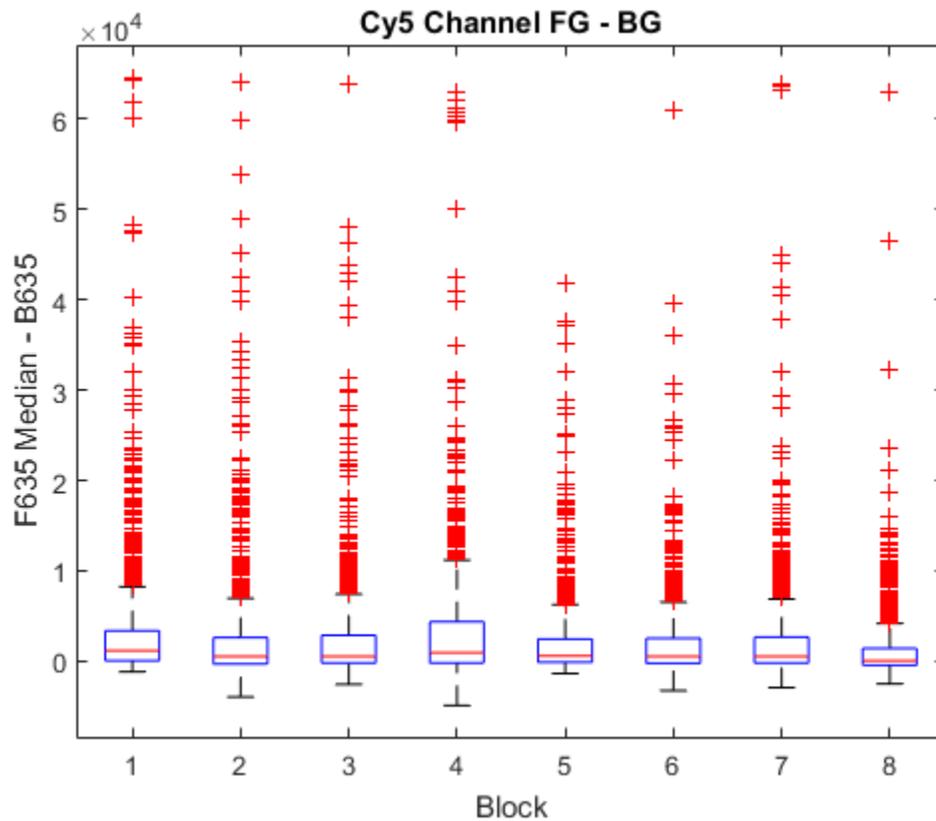

See Also

magetfield | maimage | mairplot | maloglog | malowess | manorm |
mavolcanoplot | boxplot

mafdr

Estimate false discovery rate (FDR) for multiple hypothesis testing

Syntax

FDR = mafdr(*PValues*)
[*FDR, Q*] = mafdr(*PValues*)
[*FDR, Q, Pi0*] = mafdr(*PValues*)
[*FDR, Q, Pi0, R2*] = mafdr(*PValues*)
FDR = mafdr(*PValues*, ...'BHFDR', *BHFDRValue*, ...)
... = mafdr(*PValues*, ...'Lambda', *LambdaValue*, ...)
... = mafdr(*PValues*, ...'Method', *MethodValue*, ...)
... = mafdr(*PValues*, ...'Showplot', *ShowplotValue*, ...)

Input Arguments

PValues	Either of the following: • Column vector of p-values for each feature (for example, gene) in a data set, such as returned by mattest. • DataMatrix object containing p-values for each feature (for example, gene) in a data set, such as returned by mattest.
BHFDRValue	Controls the use of the linear step-up (LSU) procedure originally introduced by Benjamini and Hochberg, 1995 (instead of the procedure introduced by Storey, 2002). Choices are true or false (default). **Note:** If you set *BHFDRValue* to true, then: • Lambda and Method properties are ignored. • There can be only one output argument, *FDR*.

LambdaValue	Specifies lambda, λ, the tuning parameter used to estimate the a priori probability that the null hypothesis, $\hat{\pi}_0(\lambda)$, is true. *LambdaValue* can be either:
	• A single value that is > 0 and < 1.
	• A vector of four or more values. Each value must be > 0 and < 1.
	Tip The series of values can be expressed by a colon operator with the form [*first:incr:last*], where *first* is the first value in the series, *incr* is the increment, and *last* is the last value in the series.
	Default *LambdaValue* is the series of values [0.01:0.01:0.95].
	Note: If you set *LambdaValue* to a single value, the Method property is ignored.
	If you set *LambdaValue* to a vector of values, mafdr chooses the optimal value using the method specified by the Method property.
MethodValue	String that specifies a method to choose lambda, λ, the tuning parameter, from *LambdaValue*, when it is a vector. Choices are:
	• 'bootstrap' (default)
	• 'polynomial'
	Note: *MethodValue* must be 'polynomial' to return the fourth output argument, *R2*.

ShowplotValue	Property to display two plots:
	• Plot of the estimated a priori probability that the null hypothesis, $\hat{\pi}_0(\lambda)$, is true versus the tuning parameter, lambda, λ, with a cubic polynomial fitting curve
	• Plot of q-values versus p-values
	Choices are `true` or `false` (default).
	Note: If you set the BHFDR property to `true`, only the second plot displays.

Output Arguments

FDR	One of the following:
	• Column vector of positive FDR (pFDR) values (if *PValues* is a column vector).
	• DataMatrix object containing positive FDR (pFDR) values and the same row names as *PValues* (if *PValues* is a DataMatrix object).
Q	Column vector of q-values, which are measures of hypothesis testing error for each observation in *PValues*.
Pi0	Estimated a priori probability that the null hypothesis, $\hat{\pi}_0$, is true.
R2	Square of the correlation coefficient.

Description

FDR = mafdr(*PValues*) estimates a positive FDR (pFDR) value for each value in *PValues*, a column vector or DataMatrix object containing p-values for each feature (for example, gene) in a data set, using the procedure introduced by Storey, 2002. *FDR* is a column vector or a DataMatrix object containing positive FDR (pFDR) values.

[*FDR, Q*] = mafdr(*PValues*) also returns a q-value for each p-value in *PValues*, using the procedure introduced by Storey, 2002. *Q* is a column vector containing measures of hypothesis testing error for each observation in *PValues*.

[*FDR, Q, PiO*] = mafdr(*PValues*) also returns *PiO*, the estimated a priori probability that the null hypothesis, $\hat{\pi}_0$, is true, using the procedure introduced by Storey, 2002.

[*FDR, Q, PiO, R2*] = mafdr(*PValues*) also returns *R2*, the square of the correlation coefficient, using the procedure introduced by Storey, 2002, and the polynomial method to choose the tuning parameter, lambda, λ.

... = mafdr(*PValues*, ...'*PropertyName*', *PropertyValue*, ...) calls mafdr with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

FDR = mafdr(*PValues*, ...'BHFDR', *BHFDRValue*, ...) controls the use of the linear step-up (LSU) procedure originally introduced by Benjamini and Hochberg, 1995 (instead of the procedure introduced by Storey, 2002), to estimate an FDR-adjusted p-value for each value in *PValues*. Choices are true or false (default).

Note: If you set *BHFDRValue* to true, then:

- Lambda and Method properties are ignored.

- There can be only one output argument, *FDR*.

... = mafdr(*PValues*, ...'Lambda', *LambdaValue*, ...) specifies lambda, λ, the tuning parameter used to estimate the a priori probability that the null hypothesis, $\hat{\pi}_0(\lambda)$, is true. *LambdaValue* can be either:

- A single value that is > 0 and < 1.
- A vector of four or more values. Each value must be > 0 and < 1.

> **Tip** The series of values can be expressed by a colon operator with the form [*first*:*incr*:*last*], where *first* is the first value in the series, *incr* is the increment, and *last* is the last value in the series.

Default *LambdaValue* is the series of values [0.01:0.01:0.95].

> **Note:** If you set *LambdaValue* to a single value, the Method property is ignored.

If you set *LambdaValue* to a vector of values, mafdr chooses the optimal value using the method specified by the Method property.

... = mafdr(*PValues*, ...'Method', *MethodValue*, ...) specifies a method to choose lambda, λ, the tuning parameter, from *LambdaValue*, when it is a vector. Choices are bootstrap (default) or polynomial.

> **Note:** *MethodValue* must be 'polynomial' to return the fourth output argument, *R2*.

... = mafdr(*PValues*, ...'Showplot', *ShowplotValue*, ...) controls the display of two plots:

- Plot of the estimated a priori probability that the null hypothesis, $\hat{\pi}_0(\lambda)$, is true versus the tuning parameter, lambda, λ, with a cubic polynomial fitting curve
- Plot of q-values versus p-values

Choices are true or false (default).

> **Note:** If you set the BHFDR property to true, only the second plot displays.

Examples

1   Load the MAT-file, included with the Bioinformatics Toolbox software, that contains Affymetrix data from a prostate cancer study, specifically probe intensity data from Affymetrix HG-U133A GeneChip arrays. The two variables in the MAT-file, `dependentData` and `independentData`, are two matrices of gene expression values from two experimental conditions.

```
load prostatecancerexpdata
```

2   Use the `mattest` function to calculate p-values for the gene expression values in the two matrices.

```
pvalues = mattest(dependentData, independentData, 'permute', true);
```

3 Use the `mafdr` function to calculate positive FDR values and q-values for the gene expression values in the two matrices and plot the data.

```
[fdr, q] = mafdr(pvalues, 'showplot', true);
```

The `prostatecancerexpdata.mat` file used in this example contains data from Best et al., 2005.

References

[1] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

[2] Storey, J.D. (2002). A direct approach to false discovery rates. Journal of the Royal Statistical Society *64(3)*, 479–498.

[3] Storey, J.D., and Tibshirani, R. (2003). Statistical significance for genomewide studies. Proc Nat Acad Sci *100(16)*, 9440–9445.

[4] Storey, J.D., Taylor, J.E., and Siegmund, D. (2004). Strong control conservative point estimation and simultaneous conservative consistency of false discovery rates: A unified approach. Journal of the Royal Statistical Society *66*, 187–205.

[5] Benjamini, Y., and Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. Journal of the Royal Statistical Society *57*, 289–300.

See Also

affygcrma | gcrma | affyrma | mairplot | maloglog | mapcaplot | mattest | mavolcanoplot | rmasummary

magetfield

Extract data from microarray structure

Syntax

```
magetfield(MAStruct, FieldName)
```

Arguments

MAStruct	Microarray structure.
FieldName	A column in *MAStruct*.

Description

magetfield(*MAStruct*, *FieldName*) extracts data for *FieldName*, a column in *MAStruct*, microarray structure.

The benefit of this function is to hide the details of extracting a column of data from a structure created with one of the microarray reader functions (gprread, agferead, sptread, imageneread).

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
cy5data = magetfield(maStruct,'F635 Median');
cy3data = magetfield(maStruct,'F532 Median');
mairplot(cy5data,cy3data,'title','R vs G IR plot');
```

See Also

agferead | gprread | ilmnbsread | imageneread | maboxplot | mairplot | maloglog | malowess | sptread

maimage

Spatial image for microarray data

Syntax

```
maimage(X, FieldName)
H = maimage(...)
[H, HLines] = maimage(...)
maimage(..., 'PropertyName', PropertyValue,...)
maimage(..., 'Title', TitleValue)
maimage(..., 'ColorBar', ColorBarValue)
maimage(..., 'HandleGraphicsPropertyName' PropertyValue)
```

Arguments

X	A microarray data structure.
FieldName	A field in the microarray data structure *X*.
TitleValue	A string to use as the title for the plot. The default title is `FieldName`.
ColorBarValue	Property to control displaying a color bar in the Figure window. Enter either `true` or `false`. The default value is `false`.

Description

`maimage(X, FieldName)` displays an image of field *FieldName* from microarray data structure *X*. Microarray data can be GenPix Results (GPR) format. After creating the image, click a data point to display the value and ID, if known.

`H = maimage(...)` returns the handle of the image.

`[H, HLines] = maimage(...)` returns the handles of the lines used to separate the different blocks in the image.

maimage(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

maimage(..., 'Title', *TitleValue*) allows you to specify the title of the plot. The default title is FieldName.

maimage(..., 'ColorBar', *ColorBarValue*), when *ColorBarValue* is true, a color bar is shown. If *ColorBarValue* is false, no color bar is shown. The default is for the color bar to be shown.

maimage(..., '*HandleGraphicsPropertyName*' *PropertyValue*) allows you to pass optional Handle Graphics® property name/value pairs to the function. For example, a name/value pair for color could be maimage(..., 'color' 'r').

Examples

Generate Spatial Image for Microarray Data

This example shows how to generate spatial images for microarray data.

Read in a sample GPR file.

```
madata = gprread('mouse_a1wt.gpr');
```

Plot the median foreground intensity for the 635 nm channel.

```
maimage(madata,'F635 Median')
```
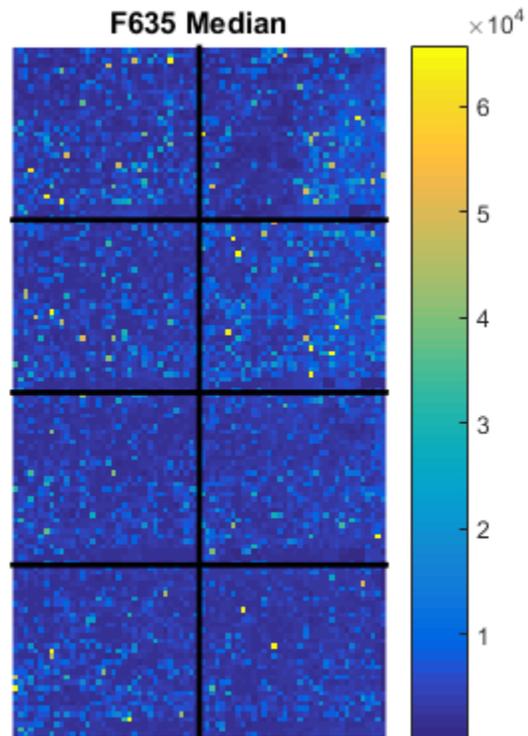
Alternatively, create a similar plot using more basic graphics commands.

```
F635Median = magetfield(madata,'F635 Median');
figure
imagesc(F635Median(madata.Indices));
```
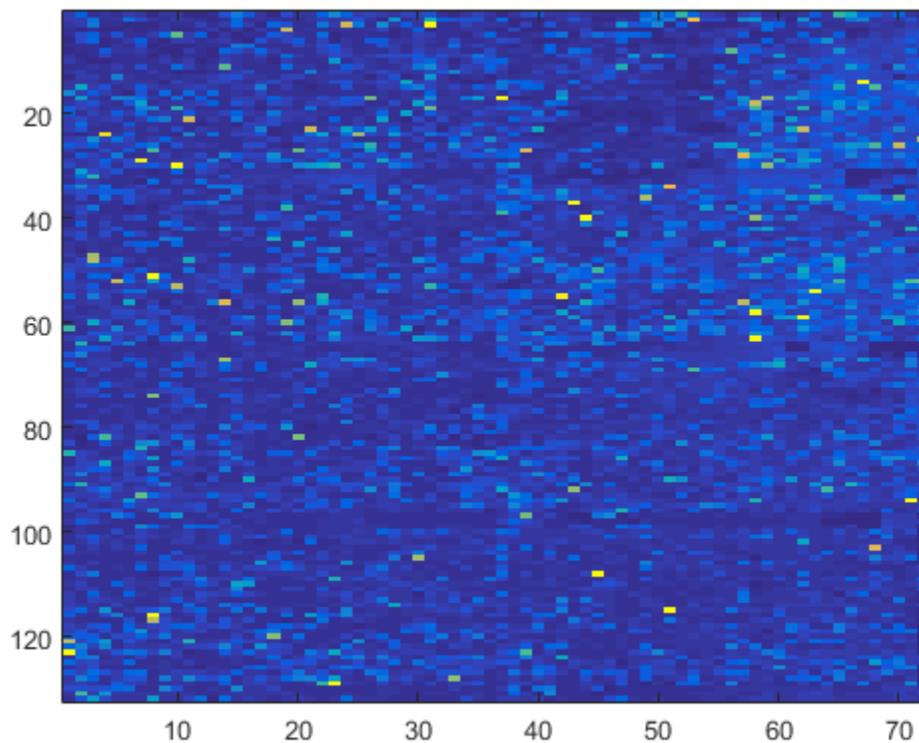
Change the colormap and add a color bar.

```
colormap bone
colorbar
```
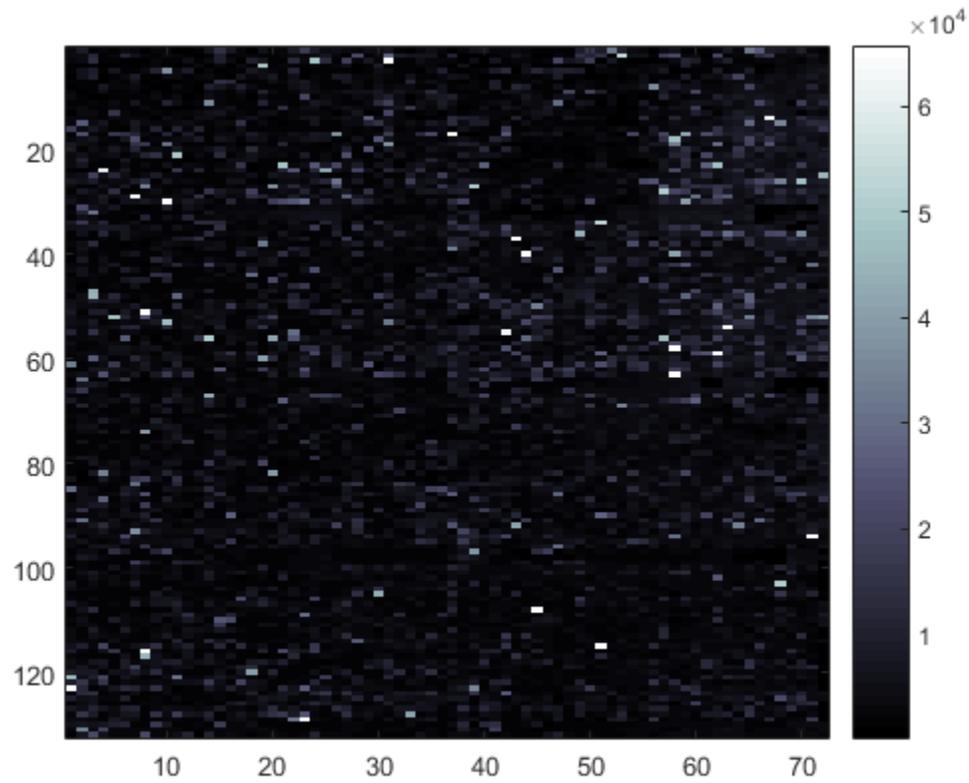
See Also
maboxplot | magetfield | mairplot | maloglog | malowess | imagesc

mainvarsetnorm

Perform rank invariant set normalization on gene expression values from two experimental conditions or phenotypes

Syntax

```
NormDataY = mainvarsetnorm(DataX, DataY)
NormDataY = mainvarsetnorm(..., 'Thresholds', ThresholdsValue, ...)
NormDataY = mainvarsetnorm(..., 'Exclude', ExcludeValue, ...)
NormDataY = mainvarsetnorm(..., 'Percentile', PercentileValue, ...)
NormDataY = mainvarsetnorm(..., 'Iterate', IterateValue, ...)
NormDataY = mainvarsetnorm(..., 'Method', MethodValue, ...)
NormDataY = mainvarsetnorm(..., 'Span', SpanValue, ...)
NormDataY = mainvarsetnorm(..., 'Showplot', ShowplotValue, ...)
```

Arguments

DataX	Vector of gene expression values from a single experimental condition or phenotype, where each row corresponds to a gene. These data points are used as the baseline.
DataY	Vector of gene expression values from a single experimental condition or phenotype, where each row corresponds to a gene. These data points will be normalized using the baseline.
ThresholdsValue	Vector that sets the thresholds for the lowest average rank and the highest average rank between the two data sets. The average rank for each data point is determined by first converting the values in *DataX* and *DataY* to ranks, then averaging the two ranks for each data point. Then, the threshold for each data point is determined by interpolating between the threshold for the lowest average rank and the threshold for the highest average rank.
	Note: These individual thresholds are used to determine the rank invariant set, which is a set of data points, each

	having a proportional rank difference (prd) smaller than its predetermined threshold. For more information on the rank invariant set, see "Description" on page 1-1082.
	ThresholdsValue is a 1-by-2 vector [*LT, HT*], where *LT* is the threshold for the lowest average rank and *HT* is threshold for the highest average rank. Select these two thresholds empirically to limit the spread of the invariant set, but allow enough data points to determine the normalization relationship. Values must be between 0 and 1. Default is [0.03, 0.07].
ExcludeValue	Property to filter the invariant set of data points, by excluding the data points whose average rank (between *DataX* and *DataY*) is in the highest *N* ranked averages or lowest *N* ranked averages.
PercentileValue	Property to stop the iteration process when the number of data points in the invariant set reaches *N* percent of the total number of input data points. Default is 1. **Note:** If you do not use this property, the iteration process continues until no more data points are eliminated.
IterateValue	Property to control the iteration process for determining the invariant set of data points. Enter true to repeat the process until either no more data points are eliminated, or a predetermined percentage of data points (*PercentileValue*) is reached. Enter false to perform only one iteration of the process. Default is true. **Tip** Select false for smaller data sets, typically less than 200 data points.
MethodValue	Property to select the smoothing method used to normalize the data. Enter 'lowess' or 'runmedian'. Default is 'lowess'.

SpanValue	Property to set the window size for the smoothing method. If *SpanValue* is less than 1, the window size is that percentage of the number of data points. If *SpanValue* is equal to or greater than 1, the window size is of size *SpanValue*. Default is `0.05`, which corresponds to a window size equal to 5% of the total number of data points in the invariant set.
ShowplotValue	Property to control the plotting of a pair of M-A scatter plots (before and after normalization). M is the ratio between *DataX* and *DataY*. A is the average of *DataX* and *DataY*. Enter `true` to create the pair of M-A scatter plots. Default is `false`.

Description

NormDataY = mainvarsetnorm(*DataX, DataY*) normalizes the values in *DataY*, a vector of gene expression values, to a reference vector, *DataX*, using the invariant set method. *NormDataY* is a vector of normalized gene expression values from *DataY*.

Specifically, mainvarsetnorm:

- Determines the proportional rank difference (*prd*) for each pair of ranks, *RankX* and *RankY*, from the two vectors of gene expression values, *DataX* and *DataY*.
  $prd = \mathrm{abs}(RankX - RankY)$

- Determines the invariant set of data points by selecting data points whose proportional rank differences (*prd*) are below *threshold*, which is a predetermined threshold for a given data point (defined by the *ThresholdsValue* property). It optionally repeats the process until either no more data points are eliminated, or a predetermined percentage of data points is reached.
  The invariant set is data points with a *prd* < *threshold*.

- Uses the invariant set of data points to calculate the lowess or running median smoothing curve, which is used to normalize the data in *DataY*.

Note: If *DataX* or *DataY* contains NaN values, then *NormDataY* will also contain NaN values at the corresponding positions.

Tip mainvarsetnorm is useful for correcting for dye bias in two-color microarray data.

NormDataY = mainvarsetnorm(..., '*PropertyName*', *PropertyValue*, ...) calls mainvarsetnorm with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

NormDataY = mainvarsetnorm(..., 'Thresholds', *ThresholdsValue*, ...) sets the thresholds for the lowest average rank and the highest average rank between the two data sets. The average rank for each data point is determined by first converting the values in *DataX* and *DataY* to ranks, then averaging the two ranks for each data point. Then, the threshold for each data point is determined by interpolating between the threshold for the lowest average rank and the threshold for the highest average rank.

Note: These individual thresholds are used to determine the rank invariant set, which is a set of data points, each having a proportional rank difference (prd) smaller than its predetermined threshold. For more information on the rank invariant set, see "Description" on page 1-1082.

ThresholdsValue is a 1-by-2 vector [*LT, HT*], where *LT* is the threshold for the lowest average rank and *HT* is threshold for the highest average rank. Select these two thresholds empirically to limit the spread of the invariant set, but allow enough data points to determine the normalization relationship. Values must be between 0 and 1. Default is [0.03, 0.07].

NormDataY = mainvarsetnorm(..., 'Exclude', *ExcludeValue*, ...) filters the invariant set of data points, by excluding the data points whose average rank (between *DataX* and *DataY*) is in the highest *N* ranked averages or lowest *N* ranked averages.

NormDataY = mainvarsetnorm(..., 'Percentile', *PercentileValue*, ...) stops the iteration process when the number of data points in the invariant set reaches *N* percent of the total number of input data points. Default is 1.

Note: If you do not use this property, the iteration process continues until no more data points are eliminated.

NormDataY = mainvarsetnorm(..., 'Iterate', *IterateValue*, ...) controls the iteration process for determining the invariant set of data points. When *IterateValue* is true, mainvarsetnorm repeats the process until either no more data points are eliminated, or a predetermined percentage of data points (*PercentileValue*) is reached. When *IterateValue* is false, performs only one iteration of the process. Default is true.

Tip Select false for smaller data sets, typically less than 200 data points.

NormDataY = mainvarsetnorm(..., 'Method', *MethodValue*, ...) selects the smoothing method for normalizing the data. When *MethodValue* is 'lowess', mainvarsetnorm uses the lowess method. When *MethodValue* is 'runmedian', mainvarsetnorm uses the running median method. Default is 'lowess'.

NormDataY = mainvarsetnorm(..., 'Span', *SpanValue*, ...) sets the window size for the smoothing method. If *SpanValue* is less than 1, the window size is that percentage of the number of data points. If *SpanValue* is equal to or greater than 1, the window size is of size *SpanValue*. Default is 0.05, which corresponds to a window size equal to 5% of the total number of data points in the invariant set.

NormDataY = mainvarsetnorm(..., 'Showplot', *ShowplotValue*, ...) determines whether to plot a pair of M-A scatter plots (before and after normalization). M is the ratio between *DataX* and *DataY*. A is the average of *DataX* and *DataY*. When *ShowplotValue* is true, mainvarsetnorm plots the M-A scatter plots. Default is false.

Examples

Normalize Microarray Data

This example illustrates how to correct for dye bias or scanning differences between two channels of data from a two-color microarray experiment.

Read microarray data from a sample GPR file.

```
maStruct = gprread('mouse_a1wt.gpr');
```

Extract gene expression values from two different experimental conditions.

```
cy5data = magetfield(maStruct, 'F635 Median');
cy3data = magetfield(maStruct, 'F532 Median');
```

Normalize `cy3data` using `cy5data` as reference and plot the results.

```
Normcy3data = mainvarsetnorm(cy5data, cy3data, 'showplot', true);
```


M-A plots
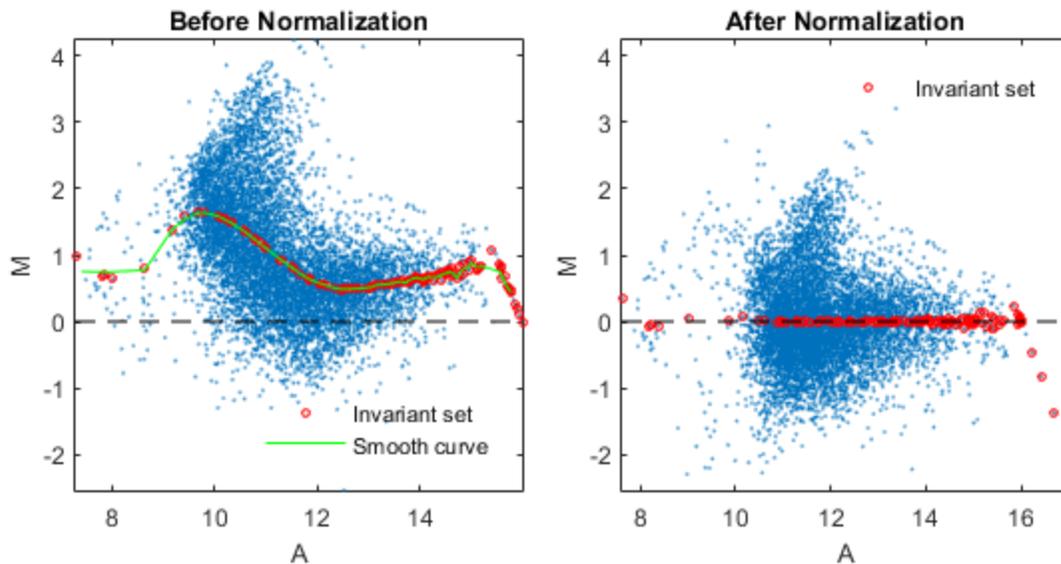
Under perfect experimental conditions, data points with equal expression values would fall along the M = 0 line, which represents a gene expression ratio of 1. However, dye bias caused the measured values in one channel to be higher than the other channel, as seen

in the Before normalization plot. Normalization corrected the variance, as seen in the After normalization plot.

References

[1] Tseng, G.C., Oh, Min-Kyu, Rohlin, L., Liao, J.C., and Wong, W.H. (2001) Issues in cDNA microarray analysis: quality filtering, channel normalization, models of variations and assessment of gene effects. Nucleic Acids Research. *29*, 2549-2557.

[2] Hoffmann, R., Seidl, T., and Dugas, M. (2002) Profound effect of normalization on detection of differentially expressed genes in oligonucleotide microarray data analysis. Genome Biology. *3(7)*: research 0033.1-0033.11.

See Also
affyinvarsetnorm | malowess | manorm | quantilenorm

mairplot

Create intensity versus ratio scatter plot of microarray data

Syntax

```
mairplot(DataX, DataY)
[Intensity, Ratio] = mairplot(DataX, DataY)
[Intensity, Ratio, H] = mairplot(DataX, DataY)
... = mairplot(..., 'Type', TypeValue, ...)
... = mairplot(..., 'LogTrans', LogTransValue, ...)
... = mairplot(..., 'FactorLines', FactorLinesValue, ...)
... = mairplot(..., 'Title', TitleValue, ...)
... = mairplot(..., 'Labels', LabelsValue, ...)
... = mairplot(..., 'Normalize', NormalizeValue, ...)
... = mairplot(..., 'LowessOptions', LowessOptionsValue, ...)
... = mairplot(..., 'Showplot', ShowplotValue, ...)
... = mairplot(..., 'PlotOnly', PlotOnlyValue, ...)
```

Input Arguments

DataX, *DataY*	DataMatrix object or vector of gene expression values where each row corresponds to a gene. For example, in a two-color microarray experiment, *DataX* could be cy3 intensity values and *DataY* could be cy5 intensity values.
TypeValue	String that specifies the plot type. Choices are `'IR'` (plots $\log_{10}$ of the product of the *DataX* and *DataY* intensities versus $\log_2$ of the intensity ratios) or `'MA'` (plots $(1/2)\log_2$ of the product of the *DataX* and *DataY* intensities versus $\log_2$ of the intensity ratios). Default is `'IR'`.
LogTransValue	Controls the conversion of data in *X* and *Y* from natural scale to $\log_2$ scale. Set *LogTransValue* to `false`, when the data is already $\log_2$ scale. Default is `true`, which assumes the data is natural scale.

FactorLinesValue	Adds lines to the plot showing a factor of *N* change. Default is 2, which corresponds to a level of 1 and -1 on a $\log_2$ scale.
	Tip You can also change the factor lines interactively, after creating the plot.
TitleValue	String that specifies a title for the plot.
LabelsValue	Cell array of labels for the data. If labels are defined, then clicking a point on the plot shows the label corresponding to that point.
NormalizeValue	Controls the display of lowess normalized ratio values. Enter `true` to display to lowess normalized ratio values. Default is `false`.
	Tip You can also normalize the data from the MAIR Plot window, after creating the plot.
LowessOptionsValue	Cell array of one, two, or three property name/value pairs in any order that affect the lowess normalization. Choices for property name/value pairs are: • `'Order'`, *OrderValue* • `'Robust'`, *RobustValue* • `'Span'`, *SpanValue* For more information on the preceding property name/value pairs, see `malowess`.
ShowplotValue	Controls the display of the scatter plot. Choices are `true` (default) or `false`.
PlotOnlyValue	Controls the display of the scatter plot without user interface components. Choices are `true` or `false` (default).
	Note: If you set the `'PlotOnly'` property to `true`, you can still display labels for data points by clicking a data point, and you can still adjust the horizontal fold change lines by click-dragging the lines.

Output Arguments

Intensity	DataMatrix object or vector containing intensity values for the microarray gene expression data, calculated as: • $\log_{10}$ of the product of the *DataX* and *DataY* intensities (when Type is 'IR') • $(1/2)\log_2$ of the product of the *DataX* and *DataY* intensities (when Type is 'MA') **Note:** If *DataX* or *DataY* is a DataMatrix object, then *Intensity* is also a DataMatrix object with the same properties.
Ratio	DataMatrix object or vector containing ratios of the microarray gene expression data, calculated as log2(*DataX.*/*DataY*). **Note:** If *DataX* or *DataY* is a DataMatrix object, then *Ratio* is also a DataMatrix object with the same properties.
H	Handle of the plot.

Description

mairplot(*DataX*, *DataY*) creates a scatter plot that plots $\log_{10}$ of the product of the *DataX* and *DataY* intensities versus $\log_2$ of the intensity ratios.

[*Intensity*, *Ratio*] = mairplot(*DataX*, *DataY*) returns the intensity and ratio values. If you set 'Normalize' to true, the returned ratio values are normalized.

[*Intensity*, *Ratio*, *H*] = mairplot(*DataX*, *DataY*) returns the handle of the plot.

... = mairplot(..., '*PropertyName*', *PropertyValue*, ...) calls mairplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single

quotation marks and is case insensitive. These property name/property value pairs are as follows:

`... = mairplot(..., 'Type', `*`TypeValue`*`, ...)` specifies the plot type. Choices are `'IR'` (plots $\log_{10}$ of the product of the *DataX* and *DataY* intensities versus $\log_2$ of the intensity ratios) or `'MA'` (plots $(1/2)\log_2$ of the product of the *DataX* and *DataY* intensities versus $\log_2$ of the intensity ratios). Default is `'IR'`.

`... = mairplot(..., 'LogTrans', `*`LogTransValue`*`, ...)` controls the conversion of data in *X* and *Y* from natural to $\log_2$ scale. Set *LogTransValue* to `false`, when the data is already $\log_2$ scale. Default is `true`, which assumes the data is natural scale.

`... = mairplot(..., 'FactorLines', `*`FactorLinesValue`*`, ...)` adds lines to the plot showing a factor of *N* change. Default is 2, which corresponds to a level of 1 and -1 on a $\log_2$ scale.

Tip You can also change the factor lines interactively, after creating the plot.

`... = mairplot(..., 'Title', `*`TitleValue`*`, ...)` specifies a title for the plot.

`... = mairplot(..., 'Labels', `*`LabelsValue`*`, ...)` specifies a cell array of labels for the data. If labels are defined, then clicking a point on the plot shows the label corresponding to that point.

`... = mairplot(..., 'Normalize', `*`NormalizeValue`*`, ...)` controls the display of lowess normalized ratio values. Enter `true` to display to lowess normalized ratio values. Default is `false`.

Tip You can also normalize the data from the MAIR Plot window, after creating the plot.

`... = mairplot(..., 'LowessOptions', `*`LowessOptionsValue`*`, ...)` lets you specify up to three property name/value pairs (in any order) that affect the lowess normalization. Choices for property name/value pairs are:

- `'Order', `*`OrderValue`*
- `'Robust', `*`RobustValue`*

- `'Span'`, *SpanValue*

For more information on the previous three property name/value pairs, see the `malowess` function.

`... = mairplot(..., 'Showplot', `*ShowplotValue*`, ...)` controls the display of the scatter plot. Choices are `true` (default) or `false`.

`... = mairplot(..., 'PlotOnly', `*PlotOnlyValue*`, ...)` controls the display of the scatter plot without user interface components. Choices are `true` or `false` (default).

Note: If you set the `'PlotOnly'` property to `true`, you can still display labels for data points by clicking a data point, and you can still adjust the horizontal fold change lines by click-dragging the lines.

Following is an IR plot of normalized data.
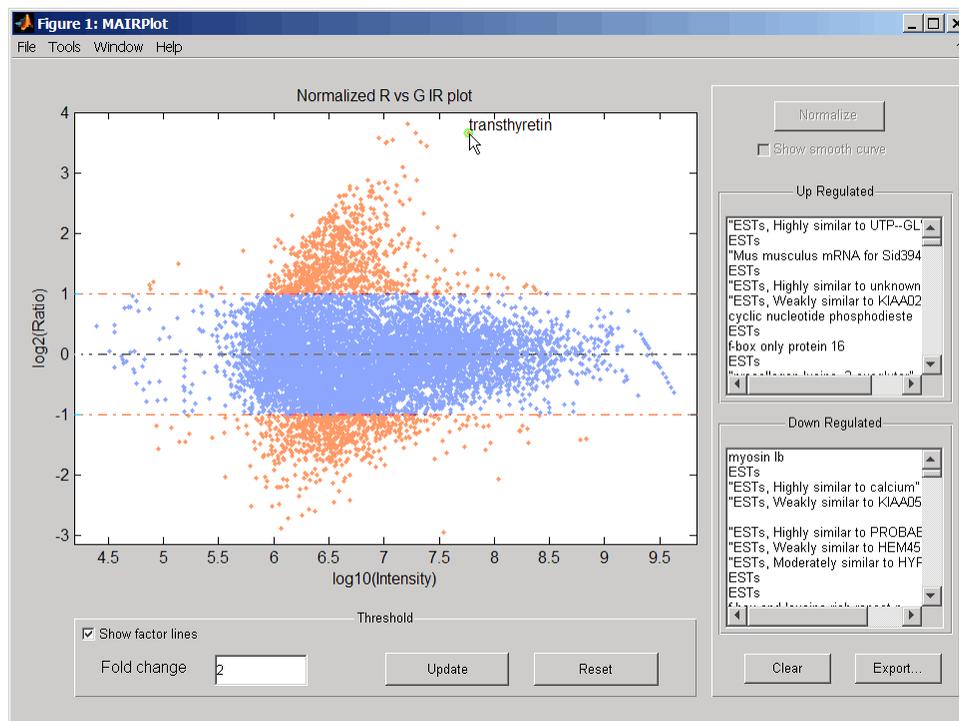
Following is an MA plot of unnormalized data.
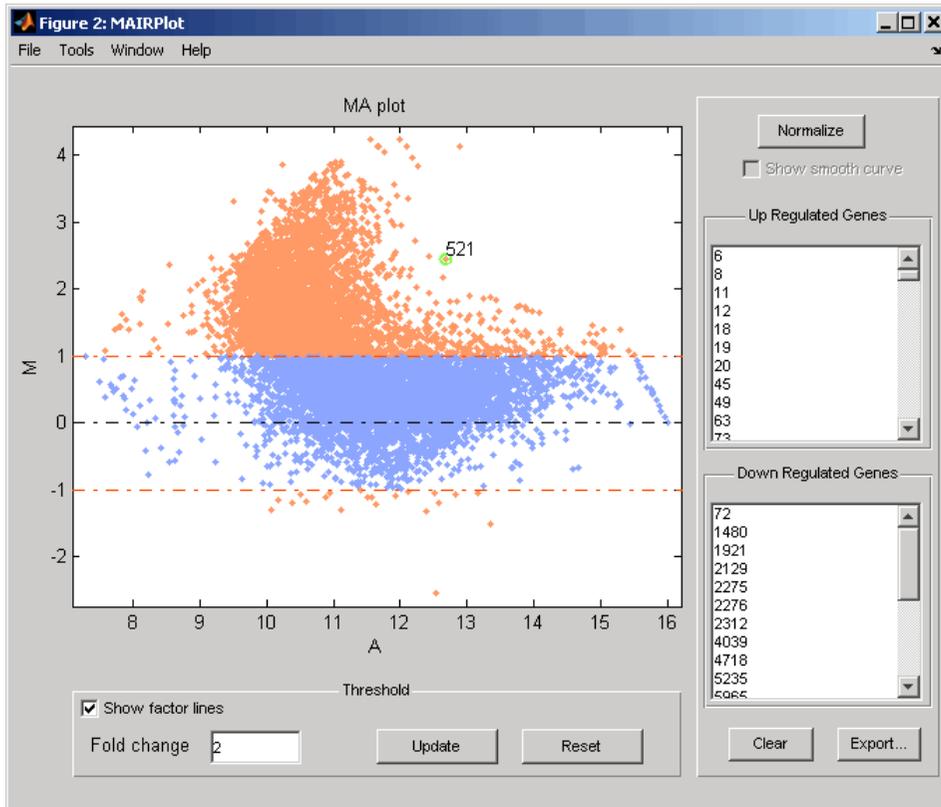
The intensity versus ratio scatter plot displays the following:

- $\log_{10}$ (Intensity) versus $\log_2$ (Ratio) scatter plot of genes.

- Two horizontal fold change lines at a fold change level of 2, which corresponds to a ratio of 1 and -1 on a $\log_2$ (Ratio) scale. (Lines will be at different fold change levels, if you used the `'FactorLines'` property.)

- Data points for genes that are considered differentially expressed (outside of the fold change lines) appear in orange.

After you display the intensity versus ratio scatter plot, you can interactively do the following:

- Adjust the horizontal fold change lines by click-dragging one line or entering a value in the **Fold Change** text box, then clicking **Update**.

- Display labels for data points by clicking a data point.

- Select a gene from the **Up Regulated** or **Down Regulated** list to highlight the corresponding data point in the plot. Press and hold **Ctrl** or **Shift** to select multiple genes.

- Zoom the plot by selecting **Tools > Zoom In** or **Tools > Zoom Out**.

- View lists of significantly up-regulated and down-regulated genes, and optionally, export the gene labels and indices to a structure in the MATLAB Workspace by clicking **Export**.

- Normalize the data by clicking the **Normalize** button, then selecting whether to show the normalized plot in a separate window. If you show the normalized plot in a separate window, the **Show smooth curve** check box becomes available in the original (unnormalized) plot.

Tip To select different lowess normalization options before normalizing, select **Tools > Set LOWESS Normalization Options**, then enter options in the Options for LOWESS dialog box.

Examples

1 Use the gprread function to create a structure containing microarray data.

```
maStruct = gprread('mouse_a1wt.gpr');
```

2 Use the magetfield function to extract the green (cy3) and red (cy5) signals from the structure.

```
cy5data = magetfield(maStruct,'F635 Median');
cy3data = magetfield(maStruct,'F532 Median');
```

3 Create an intensity versus ratio scatter plot of the cy3 and cy5 data. Normalize the data and add a title and labels:

```
mairplot(cy5data, cy3data, 'Normalize', true, ...
    'Title','Normalized R vs G IR plot', ...
    'Labels', maStruct.Names)
```
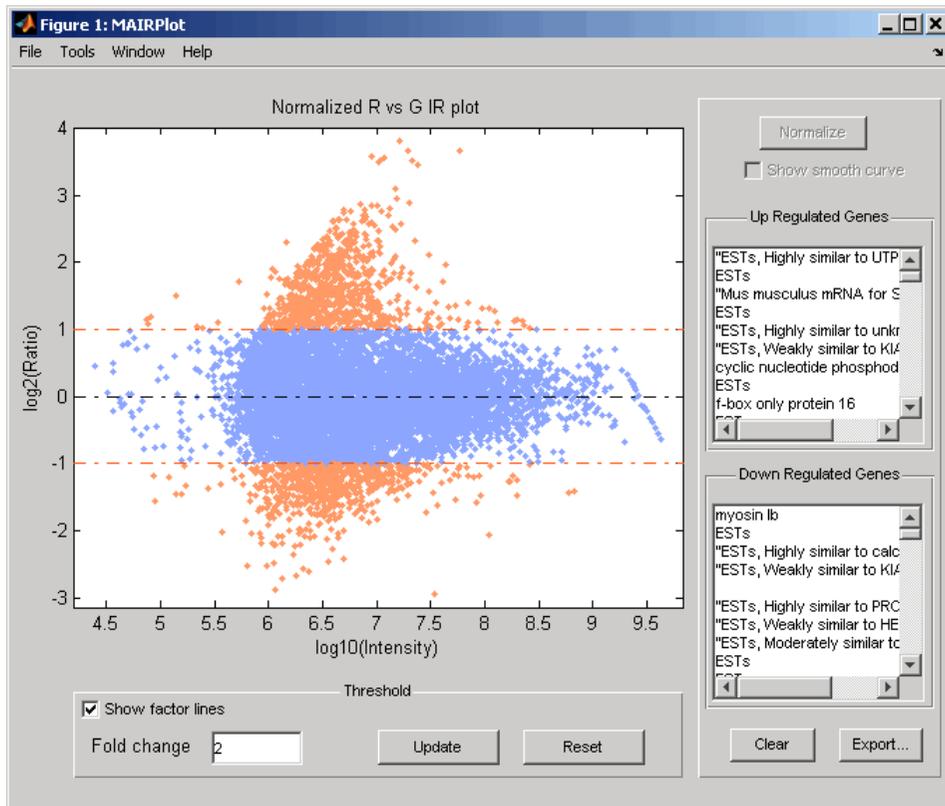
4   Return intensity values and ratios without displaying the plot.

```
[intensities, ratios] = mairplot(cy5data, cy3data, 'Showplot', false);
```

5   Create a normalized MA plot of the cy3 and cy5 data without the user interface components.

```
mairplot(cy5data, cy3data, 'Normalize', true, ...
    'Type','MA','PlotOnly',true)
```
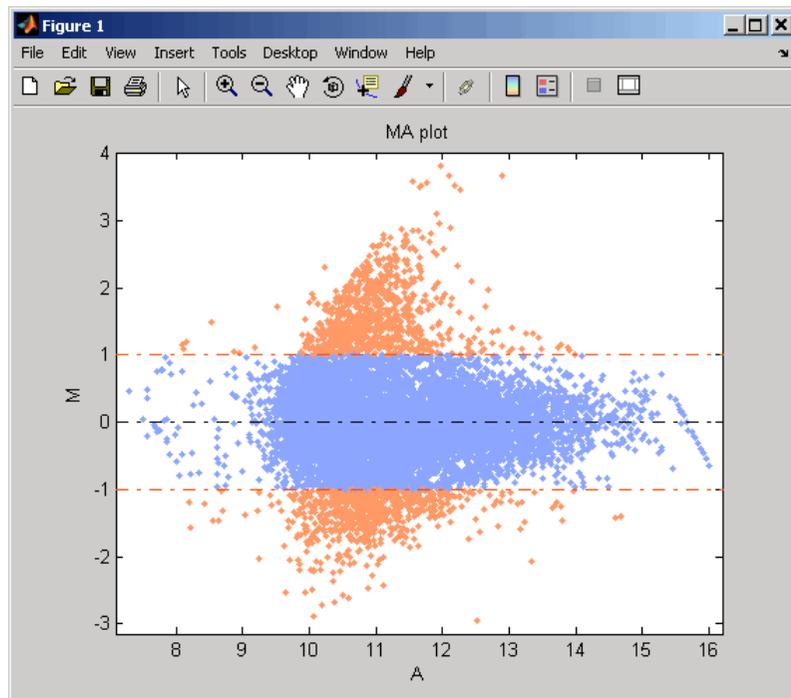
References

[1] Quackenbush, J. (2002). Microarray Data Normalization and Transformation. Nature Genetics *Suppl. 32*, 496–501.

[2] Dudoit, S., Yang, Y.H., Callow, M.J., and Speed, T.P. (2002). Statistical Methods for Identifying Differentially Expressed Genes in Replicated cDNA Microarray Experiments. Statistica Sinica *12*, 111–139.

See Also

maboxplot | magetfield | maimage | mainvarsetnorm | maloglog | malowess | manorm | mattest | mavolcanoplot

maloglog

Create loglog plot of microarray data

Syntax

```
maloglog(X, Y)
maloglog(X, Y, ...'FactorLines', N, ...)
maloglog(X, Y, ...'Title', TitleValue, ...)
maloglog(X, Y, ...'Labels', LabelsValues, ...)
maloglog(X, Y, ...'HandleGraphicsName', HGValue, ...)
H = maloglog(...)
```

Arguments

X, *Y*	DataMatrix object or numeric array of microarray expression values from a single experimental condition.
N	Property to add two lines to the plot showing a factor of *N* change.
TitleValue	A string to use as the title for the plot.
LabelsValue	A cell array of labels for the data in *X* and *Y*. If you specify *LabelsValue*, then clicking a data point in the plot shows the label corresponding to that point.

Description

maloglog(*X*, *Y*) creates a loglog scatter plot of *X* versus *Y*. *X* and *Y* are DataMatrix objects or numeric arrays of microarray expression values from two different experimental conditions.

maloglog(*X*, *Y*, ...'*PropertyName*', *PropertyValue*, ...) calls maloglog with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`maloglog(`*X*`, `*Y*`, ...'FactorLines', `*N*`, ...)` adds two lines to the plot showing a factor of *N* change.

`maloglog(`*X*`, `*Y*`, ...'Title', `*TitleValue*`, ...)` allows you to specify a title for the plot.

`maloglog(`*X*`, `*Y*`, ...'Labels', `*LabelsValues*`, ...)` allows you to specify a cell array of labels for the data. If *LabelsValues* is defined, then clicking a data point in the plot shows the label corresponding to that point.

`maloglog(`*X*`, `*Y*`, ...'HandleGraphicsName', `*HGValue*`, ...)` allows you to pass optional Handle Graphics property name/property value pairs to the function.

H = `maloglog(...)` returns the handle to the plot.

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
Red = magetfield(maStruct,'F635 Median');
Green = magetfield(maStruct,'F532 Median');
maloglog(Red,Green,'title','Red vs Green');
% Add factorlines and labels
figure
maloglog(Red,Green,'title','Red vs Green',...
                'FactorLines',2,'LABELS',maStruct.Names);
% Now create a normalized plot
figure
maloglog(manorm(Red),manorm(Green),'title',...
                'Normalized Red vs Green','FactorLines',2,...
                'LABELS',maStruct.Names);
```

See Also

`maboxplot` | `magetfield` | `mainvarsetnorm` | `maimage` | `mairplot` | `malowess` | `manorm` | `mattest` | `mavolcanoplot` | `loglog`

malowess

Smooth microarray data using Lowess method

Syntax

YSmooth = malowess(*X*, *Y*)
YSmooth = malowess(*X*, *Y*, ...'Order', *OrderValue*, ...)
YSmooth = malowess(*X*, *Y*, ...'Robust', *RobustValue*, ...)
YSmooth = malowess(*X*, *Y*, ...'Span', *SpanValue*, ...)

Arguments

X, *Y*	DataMatrix object or numeric vector containing scatter data.
OrderValue	Property to select the order of the algorithm. Enter either 1 (linear fit) or 2 (quadratic fit). The default order is 1.
RobustValue	Property to select a robust fit. Enter either true or false.
SpanValue	Property to specify the window size. The default value is 0.05 (5% of total points in *X*)

Description

YSmooth = malowess(*X*, *Y*) smooths scatter data in *X* and *Y* using the Lowess smoothing method. The default window size is 5% of the length of *X*. *YSmooth* is a numeric vector or, if *Y* is a DataMatrix object, also a DataMatrix object with the same properties as *Y*.

*YSmooth* = malowess(*X*, *Y*, ...'*PropertyName*', *PropertyValue*, ...) calls malowess with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

YSmooth = malowess(*X*, *Y*, ...'Order', *OrderValue*, ...) chooses the order of the algorithm. Note that the Curve Fitting Toolbox™ software refers to Lowess smoothing of order 2 as Loess smoothing.

YSmooth = malowess(*X*, *Y*, ...'Robust', *RobustValue*, ...) uses a robust fit when *RobustValue* is set to true. This option can take a long time to calculate.

YSmooth = malowess(*X*, *Y*, ...'Span', *SpanValue*, ...) modifies the window size for the smoothing function. If *SpanValue* is less than 1, the window size is taken to be a fraction of the number of points in the data. If *SpanValue* is greater than 1, the window is of size *SpanValue*.

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
cy5data = magetfield(maStruct, 'F635 Median');
cy3data = magetfield(maStruct, 'F532 Median');
[x,y] = mairplot(cy5data, cy3data);
drawnow
ysmooth = malowess(x,y);
hold on;
plot(x, ysmooth, 'rx')
ynorm = y - ysmooth;
```

See Also

affyinvarsetnorm | maboxplot | magetfield | maimage | mainvarsetnorm | mairplot | maloglog | manorm | quantilenorm | robustfit

manorm

Normalize microarray data

Syntax

```
XNorm = manorm(X)
XNorm = manorm(MAStruct, FieldName)
[XNorm, ColVal] = manorm(...)
manorm(..., 'Method', MethodValue, ...)
manorm(..., 'Extra_Args', Extra_ArgsValue, ...)
manorm(..., 'LogData', LogDataValue, ...)
manorm(..., 'Percentile', PercentileValue, ...)
manorm(..., 'Global', GlobalValue, ...)
manorm(..., 'StructureOutput', StructureOutputValue, ...)
manorm(..., 'NewColumnName', NewColumnNameValue, ...)
```

Arguments

X	Numeric array or DataMatrix object of microarray data.
MAStruct	Microarray structure.
FieldName	Field.

Description

XNorm = manorm(*X*) scales the values in each column of *X*, a numeric array or DataMatrix object of microarray data, by dividing by the mean column intensity. *XNorm* is a vector, matrix, or DataMatrix object of normalized microarray data.

XNorm = manorm(*MAStruct*, *FieldName*) scales the data in *MAStruct*, a microarray structure, for a field specified by *FieldName*, for each block or print-tip by dividing each block by the mean column intensity. The output is a matrix with each column corresponding to the normalized data for each block.

[*XNorm*, *ColVal*] = manorm(...) returns the values used to normalize the data.

manorm(..., '*PropertyName*', *PropertyValue*, ...) calls manorm with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

manorm(..., 'Method', *MethodValue*, ...) allows you to choose the method for scaling or centering the data. *MethodValue* can be 'Mean'(default), 'Median', 'STD' (standard deviation), 'MAD' (median absolute deviation), or a function handle. If you pass a function handle, then the function should ignore NaNs and must return a single value per column of the input data.

manorm(..., 'Extra_Args', *Extra_ArgsValue*, ...) allows you to pass extra arguments to the function *MethodValue*. *Extra_ArgsValue* must be a cell array.

manorm(..., 'LogData', *LogDataValue*, ...), when *LogDataValue* is true, works with log ratio data in which case the mean (or *MethodValue*) of each column is subtracted from the values in the columns, instead of dividing the column by the normalizing value.

manorm(..., 'Percentile', *PercentileValue*, ...) only uses the percentile (*PercentileValue*) of the data preventing large outliers from skewing the normalization. If *PercentileValue* is a vector containing two values, then the range from the *PercentileValue(1)* percentile to the *PercentileValue(2)* percentile is used. The default value is 100, that is to use all the data in the data set.

manorm(..., 'Global', *GlobalValue*, ...) when *GlobalValue* is true, normalizes the values in the data set by the global mean (or *MethodValue*) of the data, as opposed to normalizing each column or block of the data independently.

manorm(..., 'StructureOutput', *StructureOutputValue*, ...), when *StructureOutputValue* is true, the input data is a structure returns the input structure with an additional data field for the normalized data.

manorm(..., 'NewColumnName', *NewColumnNameValue*, ...), when using StructureOutput, allows you to specify the name of the column that is appended to the list of ColumnNames in the structure. The default behavior is to prefix 'Block Normalized' to the FieldName string.

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
```

```
% Extract some data of interest.
Red = magetfield(maStruct,'F635 Median');
Green = magetfield(maStruct,'F532 Median');
% Create a log-log plot.
maloglog(Red,Green,'factorlines',true)
% Center the data.
normRed = manorm(Red);
normGreen = manorm(Green);
% Create a log-log plot of the centered data.
figure
maloglog(normRed,normGreen,'title','Normalized','factorlines',true)

% Alternatively, you can work directly with the structure
normRedBs = manorm(maStruct,'F635 Median - B635');
normGreenBs = manorm(maStruct,'F532 Median - B532');
% Create a log-log plot of the centered data. This includes some
% zero values so turn off the warning.
figure
w = warning('off','Bioinfo:maloglog:ZeroValues');
warning('off','Bioinfo:maloglog:NegativeValues');
maloglog(normRedBs,normGreenBs,'title',...
                'Normalized Background-Subtracted Median Values',...
                'factorlines',true)
        warning(w);
```

See Also

affyinvarsetnorm | maboxplot | magetfield | mainvarsetnorm | mairplot | maloglog | malowess | quantilenorm | rmasummary

mapcaplot

Create Principal Component Analysis (PCA) plot of microarray data

Syntax

```
mapcaplot(Data)
mapcaplot(Data, Label)
```

Arguments

Data	DataMatrix object or numeric array containing microarray expression profile data. If a DataMatrix object, the row names are used as labels in the plot, unless you provide labels with the second input *Label*.
Label	Cell array of strings representing labels for the data points in the plot.

Description

mapcaplot(*Data*) creates 2-D scatter plots of principal components of *Data*, a DataMatrix object or numeric array containing microarray expression profile data.

mapcaplot(*Data*, *Label*) uses the elements of the cell array of strings *Label*, instead of the row numbers, to label the data points in the PCA plots.
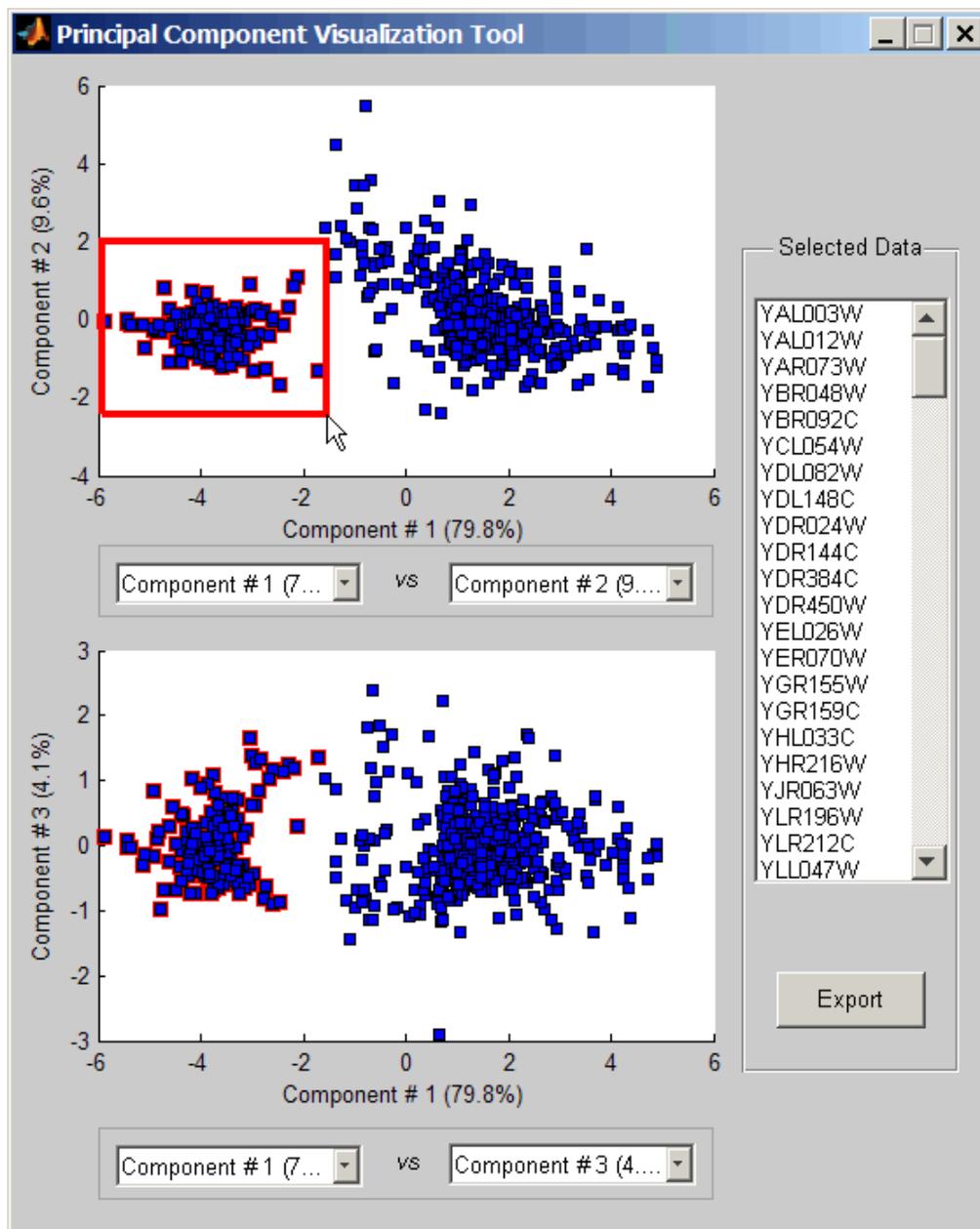
Once you plot the principal components, you can:

- Select principal components for the x and y axes from the drop-down list boxes below each scatter plot.
- Click a data point to display its label.
- Select a subset of data points by click-dragging a box around them. This will highlight the points in the selected region and the corresponding points in the other axes. The labels of the selected data points appear in the list box.
- Select a label in the list box to highlight the corresponding data point in the plot. Press and hold **Ctrl** or **Shift** to select multiple data points.
- Export the gene labels and indices to a structure in the MATLAB workspace by clicking **Export**.

Examples

Create Principal Component Analysis (PCA) Plot of Microarray Data

This example shows how to create a PCA plot of yeast microarray data.

This example uses data from an experiment (DeRisi et al., 1997) that used DNA microarrays to study temporal gene expression of almost all genes in Saccharomyces cerevisiae (yeast) during the metabolic shift from fermentation to respiration. Expression levels were measured at seven time points during the diauxic shift.

Load the MAT-file, provided with Bioinformatics Toolbox™, that contains filtered yeast microarray data.

```
load filteredyeastdata
```

This MAT-file includes three variables:

- yeastvalues - A matrix of gene expression data from Saccharomyces cerevisiae (yeast) during the metabolic shift from fermentation to respiration
- genes - A cell array of GenBank® accession numbers for labeling the rows in yeastvalues
- times - A vector of time values for labeling the columns in yeastvalues

Perform PCA on the expression data and plot the result.

```
mapcaplot(yeastvalues, genes)
```

[1] DeRisi, J.L., Iyer, V.R., and Brown, P.O. (1997). Exploring the metabolic and genetic control of gene expression on a genomic scale. Science 278, 680–686s.

References
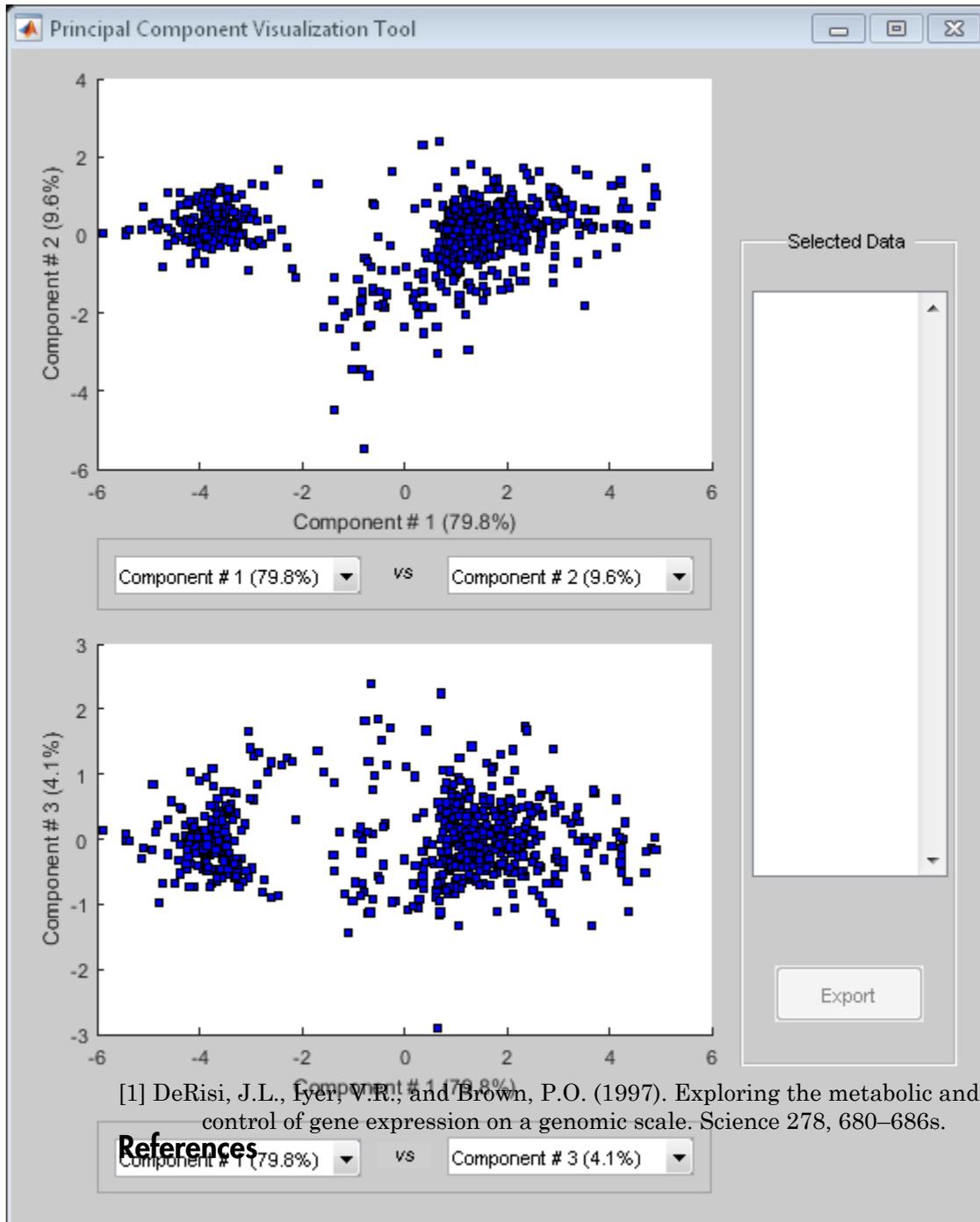
See Also
clustergram | mattest | mavolcanoplot | pca

mattest

Perform two-sample t-test to evaluate differential expression of genes from two experimental conditions or phenotypes

Syntax

```
PValues = mattest(DataX, DataY)
[PValues, TScores] = mattest(DataX, DataY)
[PValues, TScores, DFs] = mattest(DataX, DataY)
... = mattest(..., 'VarType', VarTypeValue, ...)
... = mattest(..., 'Permute', PermuteValue, ...)
... = mattest(..., 'Bootstrap', BootstrapValue, ...)
... = mattest(..., 'Showhist', ShowhistValue, ...)
... = mattest(..., 'Showplot', ShowplotValue, ...)
... = mattest(..., 'Labels', LabelsValue, ...)
```

Input Arguments

DataX, *DataY*	DataMatrix object or a matrix of gene expression values where each row corresponds to a gene and each column corresponds to a replicate. *DataX* and *DataY* must have the same number of rows and are assumed to be normally distributed in each class with equal variances. *DataX* contains data from one experimental condition and *DataY* contains data from a different experimental condition. For example, *DataX* could be expression values from cancer cells, and *DataY* could be expression values from normal cells.
VarTypeValue	String that specifies the variance type of the test. *VarTypeValue* can be `'equal'` or `'unequal'` (default). If set to `'equal'`, `mattest` performs the test assuming the two samples have equal variances. If set to `'unequal'`, `mattest` performs the test assuming the two samples have unknown and unequal variances.

PermuteValue	Controls whether permutation tests are run, and if so, how many. Choices are `true`, `false` (default), or any integer greater than 2. If set to `true`, the number of permutations is `1000`.
BootstrapValue	Controls whether bootstrap tests are run, and if so, how many. Choices are `true`, `false` (default), or any integer greater than 2. If set to `true`, the number of bootstrap tests is `1000`.
ShowhistValue	Controls the display of histograms of t-score distributions and p-value distributions. Choices are `true` or `false` (default).
ShowplotValue	Controls the display of a normal t-score quantile plot. Choices are `true` or `false` (default). In the t-score quantile plot, data points with t-scores > `(1 - 1/(2N))` or < `1/(2N)` display with red circles. `N` is the total number of genes.
LabelsValue	Cell array of labels (typically gene names or probe set IDs) for each row in *DataX* and *DataY*. The labels display if you click a data point in the t-score quantile plot.

Output Arguments

PValues	One of the following:
	• Column vector of p-values for each gene in *DataX* and *DataY* (if both inputs are matrices).
	• DataMatrix object with row names the same as the first input DataMatrix object and a column name of `p-values` (if at least one input is a DataMatrix object).
TScores	Column vector of t-scores for each gene in *DataX* and *DataY*.
DFs	Column vector containing the degree of freedom for each gene in *DataX* and *DataY*.

Description

PValues = `mattest`(*DataX, DataY*) performs an unpaired t-test for differential expression with a standard two-tailed and two-sample t-test on every gene in *DataX* and *DataY* and returns a p-value for each gene. *DataX* and *DataY* are either a DataMatrix

object or a matrix of gene expression values, in which each row corresponds to a gene, and each column corresponds to a replicate. *DataX* contains data from one experimental condition and *DataY* contains data from another experimental condition. *DataX* and *DataY* must have the same number of rows and are assumed to be normally distributed in each class. *PValues* is a column vector of p-values for each gene, or, if at least one of the inputs is a DataMatrix object, a DataMatrix object with row names the same as the first input DataMatrix object and a column name of `p-values`.

[*PValues, TScores*] = mattest(*DataX, DataY*) also returns a t-score for each gene in *DataX* and *DataY*. *TScores* is a column vector of t-scores for each gene.

[*PValues, TScores, DFs*] = mattest(*DataX, DataY*) also returns *DFs*, a column vector containing the degree of freedom for each gene across both data sets, *DataX* and *DataY*.

... = mattest(..., '*PropertyName*', *PropertyValue*, ...) calls mattest with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = mattest(..., 'VarType', *VarTypeValue*, ...) specifies the variance type of the test. *VarTypeValue* can be 'equal' or 'unequal' (default). If set to 'equal', mattest performs the test assuming the two samples have equal variances. If set to 'unequal', mattest performs the test assuming the two samples have unknown and unequal variances.

... = mattest(..., 'Permute', *PermuteValue*, ...) controls whether permutation tests are run, and if so, how many. *PermuteValue* can be true, false (default), or any integer greater than 2. If set to true, the number of permutations is 1000.

... = mattest(..., 'Bootstrap', *BootstrapValue*, ...) controls whether bootstrap tests are run, and if so, how many. *BootstrapValue* can be true, false (default), or any integer greater than 2. If set to true, the number of bootstrap tests is 1000.

... = mattest(..., 'Showhist', *ShowhistValue*, ...) controls the display of histograms of t-score distributions and p-value distributions. When *ShowhistValue* is true, mattest displays histograms. Default is false.
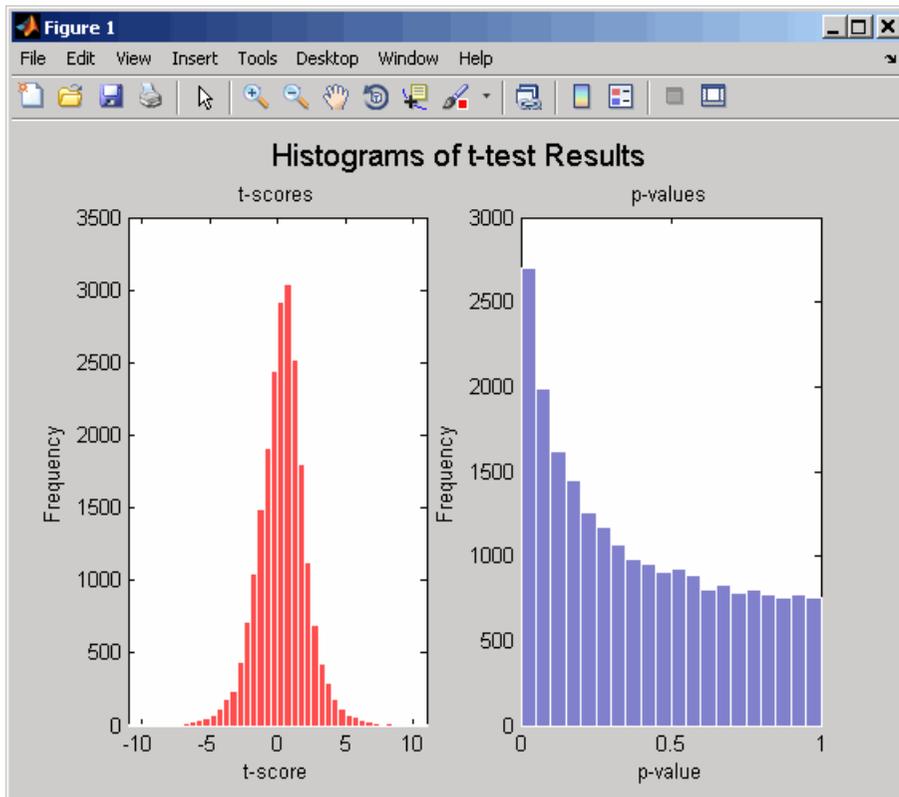
`... = mattest(..., 'Showplot', ShowplotValue, ...)` controls the display of a normal t-score quantile plot. When *ShowplotValue* is `true`, `mattest` displays a quantile-quantile plot. Default is `false`. In the t-score quantile plot, the black diagonal line represents the sample quantile being equal to the theoretical quantile. Data points of genes considered to be differentially expressed lie farther away from this line. Specifically, data points with t-scores > `(1 - 1/(2N))` or < `1/(2N)` display with red circles. `N` is the total number of genes.
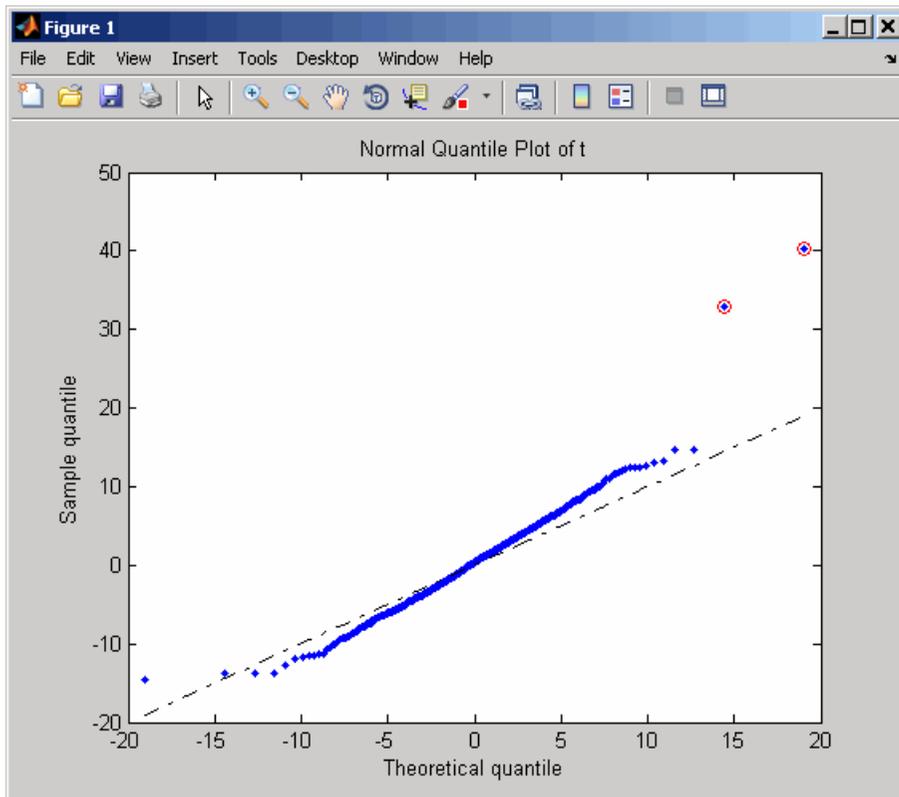
`... = mattest(..., 'Labels', `*`LabelsValue`*`, ...)` controls the display of labels when you click a data point in the t-score quantile plot. *LabelsValue* is a cell array of labels (typically gene names or probe set IDs) for each row in *DataX* and *DataY*.

Examples

1 Load the MAT-file, included with the Bioinformatics Toolbox software, that contains Affymetrix data from a prostate cancer study, specifically probe intensity data from Affymetrix HG-U133A GeneChip arrays. The two variables in the MAT-file, `dependentData` and `independentData`, are two matrices of gene expression values from two experimental conditions.

```
load prostatecancerexpdata
```

2   Calculate the p-values and t-scores for the gene expression values in the two matrices and display a normal t-score quantile plot.

```
[pvalues,tscores] = mattest(dependentData, independentData,...
                    'showplot',true);
```

3   Calculate the p-values and t-scores again using permutation tests (1000 permutations) and displaying histograms of t-score distributions and p-value distributions.

```
[pvalues,tscores] = mattest(dependentData,independentData,...
                    'permute',true,'showhist',true,...
                    'showplot',true);
```

4   Calculate the p-values and t-scores again using bootstrap tests (2000 tests) and displaying histograms of t-score distributions and p-value distributions.

```
[pvalues,tscores] = mattest(dependentData,independentData,...
                    'bootstrap',2000,'showhist',true,...
                    'showplot',true);
```
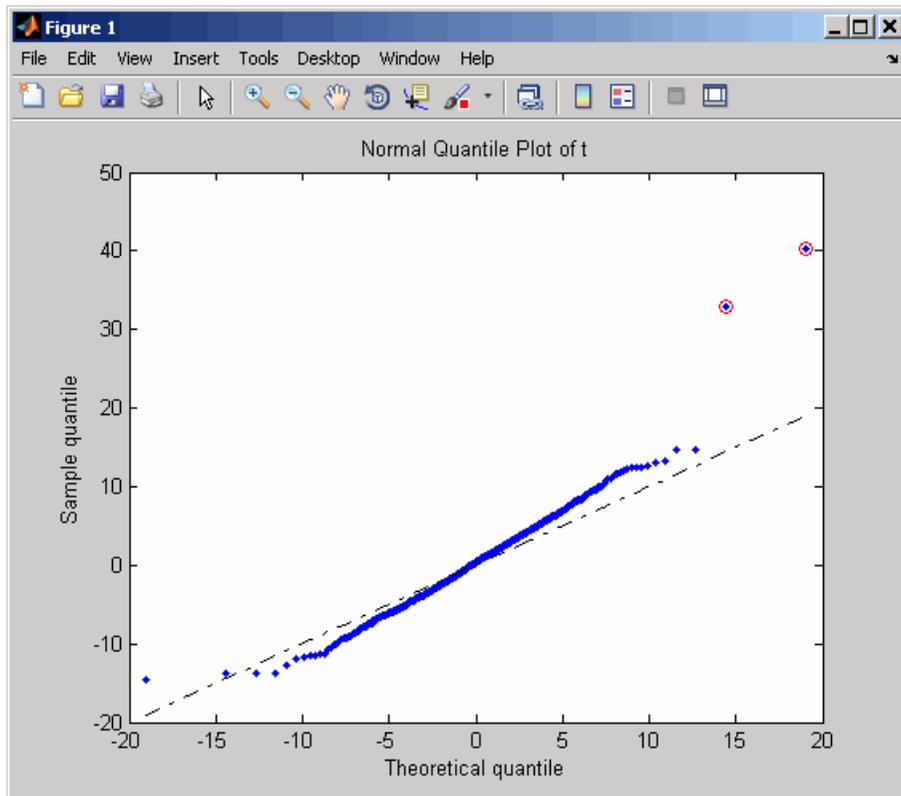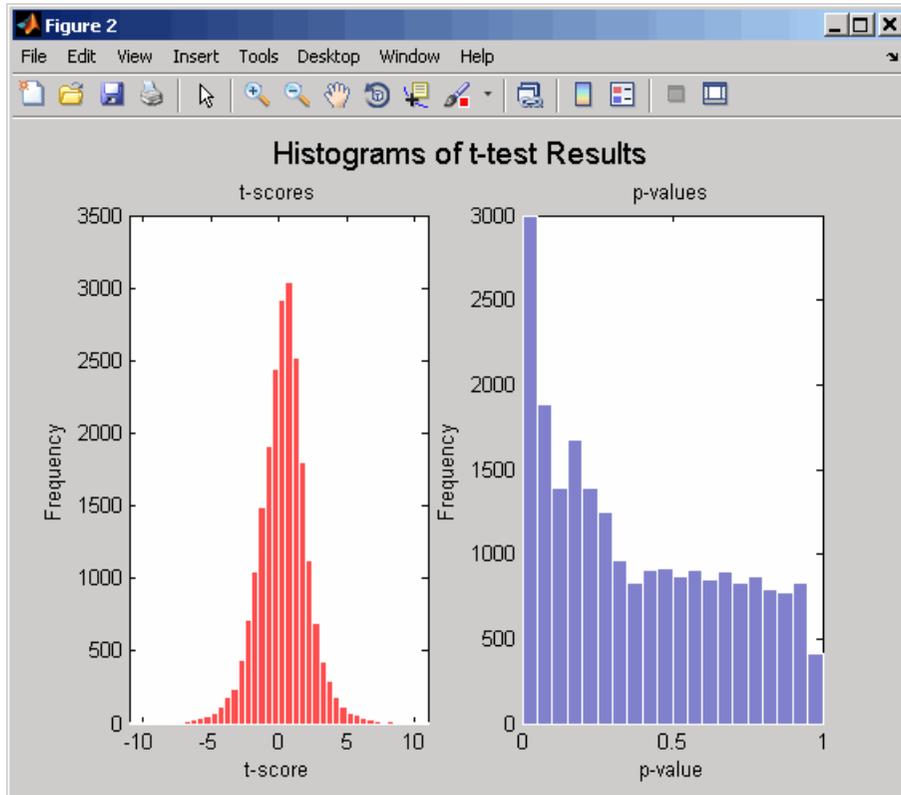
The `prostatecancerexpdata.mat` file used in this example contains data from Best et al., 2005.

References

[1] Review Literature: Huber, W., von Heydebreck, A., Sültmann, H., Poustka, A., and Vingron, M. (2002). Variance stabilization applied to microarray data calibration and to the quantification of differential expression. Bioinformatics *18 (Suppl. 1)*, S96–S104.

[2] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H.,

Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

See Also

affygcrma | affyrma | maboxplot | mafdr | mainvarsetnorm | mairplot | maloglog | malowess | manorm | mavolcanoplot | rmasummary

mavolcanoplot

Create significance versus gene expression ratio (fold change) scatter plot of microarray data

Syntax

```
mavolcanoplot(DataX, DataY, PValues)
SigStructure = mavolcanoplot(DataX, DataY, PValues)
... mavolcanoplot(..., 'Labels', LabelsValue, ...)
... mavolcanoplot(..., 'LogTrans', LogTransValue, ...)
... mavolcanoplot(..., 'PCutoff', PCutoffValue, ...)
... mavolcanoplot(..., 'Foldchange', FoldchangeValue, ...)
... mavolcanoplot(..., 'PlotOnly', PlotOnlyValue, ...)
```

Input Arguments

DataX, *DataY*	DataMatrix object, matrix, or vector of gene expression values from a single experimental condition. If a DataMatrix object or a matrix, each row is a gene, each column is a sample, and an average expression value is calculated for each gene.
	Note: If the values in *DataX* or *DataY* are natural scale, use the LogTrans property to convert them to log $_2$ scale.
PValues	Either of the following:
	• Column vector of p-values for each feature (for example, gene) in a data set, such as returned by mattest.
	• DataMatrix object containing p-values for each feature (for example, gene) in a data set, such as returned by mattest.
LabelsValue	Cell array of labels (typically gene names or probe set IDs) for the data. After creating the plot, you can click a data point to display the label associated with it. If you do not provide a *LabelsValue*, data points are labeled with row numbers from *DataX* and *DataY*.

LogTransValue	Property to control the conversion of data in *DataX* and *DataY* from natural scale to log $_2$ scale. Enter true to convert data to log $_2$ scale, or false. Default is false, which assumes data is already log $_2$ scale.
PCutoffValue	Lets you specify a cutoff p-value to define data points that are statistically significant. This value is displayed graphically as a horizontal line on the plot. Default is 0.05, which is equivalent to 1.3010 on the $-\log_{10}$ (p-value) scale. **Note:** You can also change the p-value cutoff interactively after creating the plot.
FoldchangeValue	Lets you specify a ratio fold change to define data points that are differentially expressed. Default is 2, which corresponds to a ratio of 1 and -1 on a $\log_2$ (ratio) scale. **Note:** You can also change the fold change interactively after creating the plot.
PlotOnlyValue	Controls the display of the volcano plot without user interface components. Choices are true or false (default). **Note:** If you set the 'PlotOnly' property to true, you can still display labels for data points by clicking a data point, and you can still adjust vertical fold change lines and the horizontal p-value cutoff line by click-dragging the lines.

Output Arguments

SigStructure	Structure containing information for genes that are considered to be both statistically significant (above the p-value cutoff) and significantly differentially expressed (outside of the fold change values). The fields are listed below.

Description

mavolcanoplot(*DataX, DataY, PValues*) creates a scatter plot of gene expression data, plotting significance versus fold change of gene expression ratios of two data sets, *DataX* and *DataY*. It plots significance as the $-\log_{10}$ (p-value) from the input, *PValues*. *DataX* and *DataY* can be vectors, matrices, or DataMatrix objects. *PValues* is a clumn vector or DataMatrix object.

SigStructure = mavolcanoplot(*DataX, DataY, PValues*) returns a structure containing information for genes that are considered to be both statistically significant (above the p-value cutoff) and significantly differentially expressed (outside of the fold change values). The fields within *SigStructure* are sorted by p-value and include:

- Name
- PCutoff
- FCThreshold
- GeneLabels
- PValues
- FoldChanges

Note: The fields PValues and FoldChanges will be either vectors or DataMatrix objects depending on the type of input *PValues*.

... mavolcanoplot(..., '*PropertyName*', *PropertyValue*, ...) defines optional properties that use property name/value pairs in any order. These property name/value pairs are as follows:

... mavolcanoplot(..., 'Labels', *LabelsValue*, ...) lets you provide a cell array of labels (typically gene names or probe set IDs) for the data. After creating the plot, you can click a data point to display the label associated with it. If you do not provide a *LabelsValue*, data points are labeled with row numbers from *DataX* and *DataY*.

... mavolcanoplot(..., 'LogTrans', *LogTransValue*, ...) controls the conversion of data from *DataX* and *DataY* to $\log_2$ scale. When *LogTransValue* is true, mavolcanoplot converts data from natural to $\log_2$ scale. Default is false, which assumes the data is already $\log_2$ scale.

... `mavolcanoplot(..., 'PCutoff', `*`PCutoffValue`*`, ...)` lets you specify a p-value cutoff to define data points that are statistically significant. This value displays graphically as a horizontal line on the plot. Default is `0.05`, which is equivalent to 1.3010 on the $-\log_{10}$ (p-value) scale.

Note: You can also change the p-value cutoff interactively after creating the plot.

... `mavolcanoplot(..., 'Foldchange', `*`FoldchangeValue`*`, ...)` lets you specify a ratio fold change to define data points that are differentially expressed. Fold changes display graphically as two vertical lines on the plot. Default is `2`, which corresponds to a ratio of 1 and -1 on a $\log_2$ (ratio) scale.

Note: You can also change the fold change interactively after creating the plot.

... `mavolcanoplot(..., 'PlotOnly', `*`PlotOnlyValue`*`, ...)` controls the display of the volcano plot without user interface components. Choices are `true` or `false` (default).

Note: If you set the `'PlotOnly'` property to `true`, you can still display labels for data points by clicking a data point, and you can still adjust vertical fold change lines and the horizontal p-value cutoff line by click-dragging the lines.
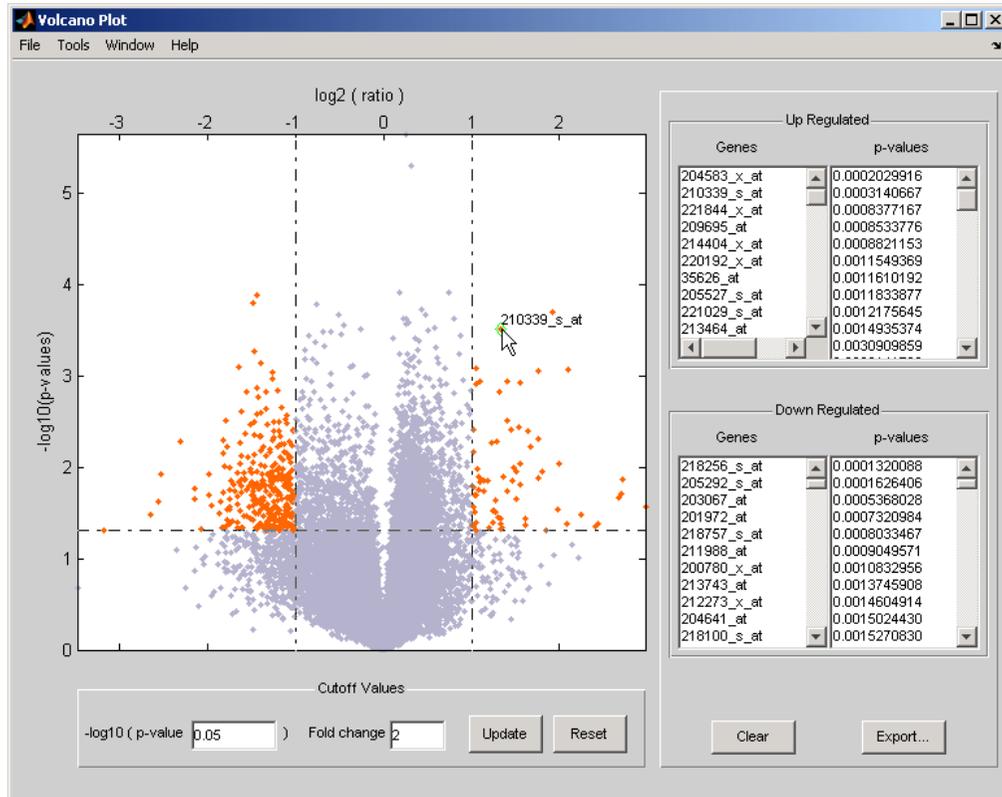
The volcano plot displays the following:

- $-\log_{10}$ (p-value) versus $\log_2$ (ratio) scatter plot of genes

- Two vertical fold change lines at a fold change level of 2, which corresponds to a ratio of 1 and −1 on a $\log_2$ (ratio) scale. (Lines will be at different fold change levels, if you used the `'Foldchange'` property.)

- One horizontal line at the 0.05 p-value level, which is equivalent to 1.3010 on the $-\log_{10}$ (p-value) scale. (The line will be at a different p-value level, if you used the `'PCutoff'` property.)

- Data points for genes that are considered both statistically significant (above the p-value line) and differentially expressed (outside of the fold changes lines) appear in orange.

After you display the volcano scatter plot, you can interactively:

- Adjust the vertical fold change lines by click-dragging one line or entering a value in the **Fold Change** text box.

- Adjust the horizontal p-value cutoff line by click-dragging or entering a value in the **p-value Cutoff** text box.

- Display labels for data points by clicking a data point.

- Select a gene from the **Up Regulated** or **Down Regulated** list to highlight the corresponding data point in the plot. Press and hold **Ctrl** or **Shift** to select multiple genes.

- Zoom the plot by selecting **Tools > Zoom In** or **Tools > Zoom Out**.

- View lists of significantly up-regulated and down-regulated genes and their associated p-values, and optionally, export the labels, p-values, and fold changes to a structure in the MATLAB Workspace by clicking **Export**.

Examples

1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains Affymetrix data variables, including `dependentData` and `independentData`, two matrices of gene expression values from two experimental conditions.

```
load prostatecancerexpdata
```

2 Use the `mattest` function to calculate p-values for the gene expression values in the two matrices.

```
pvalues = mattest(dependentData, independentData);
```

3 Using the two matrices, the `pvalues` calculated by `mattest`, and the `probesetIDs` column vector of labels provided, use `mavolcanoplot` to create a significance versus gene expression ratio scatter plot of the microarray data from the two experimental conditions.

```
mavolcanoplot(dependentData, independentData, pvalues,...
'Labels', probesetIDs)
```

4 View the volcano plot without the user interface components.

```
mavolcanoplot(dependentData, independentData, pvalues,...
'Labels', probesetIDs,'Plotonly', true)
```
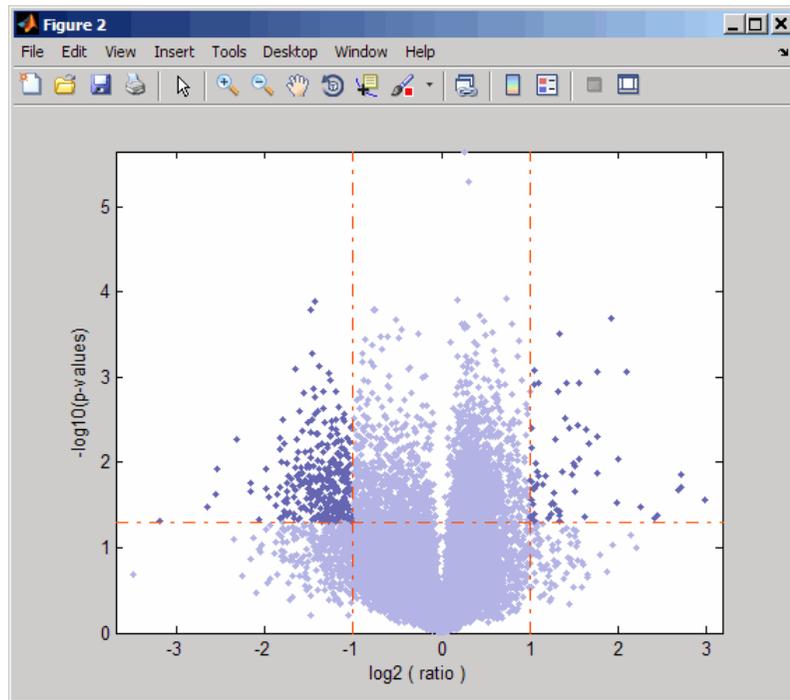
The `prostatecancerexpdata.mat` file used in the previous example contains data from Best et al., 2005.

References

[1] Cui, X., Churchill, G.A. (2003). Statistical tests for differential expression in cDNA microarray experiments. Genome Biology *4*, 210.

[2] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

See Also
maboxplot | maimage | mainvarsetnorm | mairplot | maloglog | malowess | manorm | mapcaplot | mattest

max (DataMatrix)

Return maximum values in DataMatrix object

Syntax

```
M = max(DMObj1)
[M, Indices] = max(DMObj1)
[M, Indices, Names] = max(DMObj1)
... = max(DMObj1, [], Dim)
MA = max(DMObj1, DMObj2)
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
	Note: *DMObj1* and *DMObj2* must be the same size, unless one is a scalar.
Dim	Scalar specifying the dimension of *DMObj* to return the maximum values. Choices are: • `1` — Default. Returns a row vector containing a maximum value for each column. • `2` — Returns a column vector containing a maximum value for each row.

Output Arguments

M	One of the following: • Scalar specifying the maximum value in *DMObj* when it contains vector of data

	• Row vector containing the maximum value for each column in *DMObj* (when *Dim* = 1) • Column vector containing the maximum value for each row in *DMObj* (when *Dim* = 2)
Indices	Either of the following: • Positive integer specifying the index of the maximum value in a DataMatrix object containing a vector of data • Vector containing the indices for the maximum value in each column (if *Dim* = 1) or row (if *Dim* = 2) in a DataMatrix object containing a matrix of data
Names	Vector of the row names (if *Dim* = 1) or column names (if *Dim* = 2) corresponding to the maximum value in each column or each row of a DataMatrix object.
MA	Numeric array created from the maximum elements in either of the following: • Two DataMatrix objects • A DataMatrix object and a numeric array

Description

M = max(*DMObj1*) returns the maximum value(s) in *DMObj1*, a DataMatrix object. If *DMObj1* contains a vector of data, *M* is a scalar. If *DMObj1* contains a matrix of data, *M* is a row vector containing a maximum value in each column.

[*M, Indices*] = max(*DMObj1*) returns *Indices*, the indices of the maximum value(s) in *DMObj1*, a DataMatrix object. If *DMObj1* contains a vector of data, *Indices* is a positive integer. If *DMObj1* contains a matrix of data, *Indices* is a vector containing the indices for the maximum value in each column (if *Dim* = 1) or row (if *Dim* = 2). If there are multiple maximum values in a column or row, the index for the first value is returned.

[*M, Indices, Names*] = max(*DMObj1*) returns *Names*, a vector of the row names (if *Dim* = 1) or column names (if *Dim* = 2) corresponding to the maximum value in each column or each row of*DMObj1*, a DataMatrix object. If there are multiple maximum values in a column or row, the row or column name for the first value is returned.

`... = max(`*DMObj1*`, [], `*Dim*`)` specifies which dimension to return the maximum values for, that is each column or each row in a DataMatrix object. If *Dim* = 1, returns *M*, a row vector containing the maximum value in each column. If *Dim* = 2, returns *M*, a column vector containing the maximum value in each row. Default *Dim* = 1.

MA = `max(`*DMObj1*`, `*DMObj2*`)` returns *MA*, a numeric array containing the larger of the two values from each position of *DMObj1* and *DMObj2*. *DMObj1* and *DMObj2* can both be DataMatrix objects, or one can be a DataMatrix object and the other a numeric array. They must be the same size, unless one is a scalar. *MA* has the same size (number of rows and columns) as the first nonscalar input.

More About

· DataMatrix object

See Also

`DataMatrix` | `sum` | `min`

maxflow (biograph)

Calculate maximum flow in biograph object

Syntax

```
[MaxFlow, FlowMatrix, Cut] = maxflow(BGObj, SNode, TNode)
[...] = maxflow(BGObj, SNode, TNode, ...'Capacity',
CapacityValue, ...)
[...] = maxflow(BGObj, SNode, TNode, ...'Method', MethodValue, ...)
```

Arguments

BGObj	Biograph object created by biograph (object constructor).
SNode	Node in a directed graph represented by an N-by-N adjacency matrix extracted from biograph object, *BGObj*.
TNode	Node in a directed graph represented by an N-by-N adjacency matrix extracted from biograph object, *BGObj*.
CapacityValue	Column vector that specifies custom capacities for the edges in the N-by-N adjacency matrix. It must have one entry for every nonzero value (edge) in the N-by-N adjacency matrix. The order of the custom capacities in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. By default, maxflow gets capacity information from the nonzero entries in the N-by-N adjacency matrix.
MethodValue	String that specifies the algorithm used to find the minimal spanning tree (MST). Choices are: • `'Edmonds'` — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the *labeling algorithm*. Time complexity is `O(N*E^2)`, where `N` and `E` are the number of nodes and edges respectively. • `'Goldberg'` — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as *preflow-*

	push. Time complexity is `O(N^2*sqrt(E))`, where `N` and `E` are the number of nodes and edges respectively.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*MaxFlow*, *FlowMatrix*, *Cut*] = maxflow(*BGObj*, *SNode*, *TNode*) calculates the maximum flow of a directed graph represented by an N-by-N adjacency matrix extracted from a biograph object, *BGObj*, from node *SNode* to node *TNode*. Nonzero entries in the matrix determine the capacity of the edges. Output *MaxFlow* is the maximum flow, and *FlowMatrix* is a sparse matrix with all the flow values for every edge. *FlowMatrix(X,Y)* is the flow from node *X* to node *Y*. Output *Cut* is a logical row vector indicating the nodes connected to *SNode* after calculating the minimum cut between *SNode* and *TNode*. If several solutions to the minimum cut problem exist, then *Cut* is a matrix.

Tip The algorithm that determines *Cut*, all minimum cuts, has a time complexity of `O(2^N)`, where *N* is the number of nodes. If this information is not needed, use the maxflow method without the third output.

[...] = maxflow(*BGObj*, *SNode*, *TNode*, ...'*PropertyName*', *PropertyValue*, ...) calls maxflow with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[...] = maxflow(*BGObj*, *SNode*, *TNode*, ...'Capacity', *CapacityValue*, ...) lets you specify custom capacities for the edges. *CapacityValue* is a column vector having one entry for every nonzero value (edge) in the N-by-N adjacency matrix. The order of the custom capacities in the vector must match the order of the nonzero values in the matrix when it is traversed column-wise. By default, graphmaxflow gets capacity information from the nonzero entries in the matrix.

[...] = maxflow(*BGObj*, *SNode*, *TNode*, ...'Method', *MethodValue*, ...) lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- `'Edmonds'` — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the *labeling algorithm*. Time complexity is `O(N*E^2)`, where `N` and `E` are the number of nodes and edges respectively.

- `'Goldberg'` — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as *preflow-push*. Time complexity is `O(N^2*sqrt(E))`, where `N` and `E` are the number of nodes and edges respectively.

More About

- biograph object

References

[1] Edmonds, J. and Karp, R.M. (1972). Theoretical improvements in the algorithmic efficiency for network flow problems. Journal of the ACM *19*, 248-264.

[2] Goldberg, A.V. (1985). A New Max-Flow Algorithm. MIT Technical Report MIT/LCS/TM-291, Laboratory for Computer Science, MIT.

[3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

biograph | allshortestpaths | isdag | isspantree | shortestpath | traverse | graphmaxflow | conncomp | isomorphism | minspantree | topoorder

mean (DataMatrix)

Return average or mean values in DataMatrix object

Syntax

M = mean(*DMObj*)
M = mean(*DMObj*, *Dim*)
M = mean(*DMObj*, *Dim*, *IgnoreNaN*)

Input Arguments

DMObj	DataMatrix object, such as created by `DataMatrix` (object constructor).
Dim	Scalar specifying the dimension of *DMObj* to calculate the means. Choices are: • `1` — Default. Returns mean values for elements in each column. • `2` — Returns mean values for elements in each row.
IgnoreNaN	Specifies if NaNs should be ignored. Choices are `true` (default) or `false`.

Output Arguments

M	Either of the following: • Row vector containing the mean values from elements in each column in *DMObj* (when *Dim* = 1) • Column vector containing the mean values from elements in each row in *DMObj* (when *Dim* = 2)

Description

M = mean(*DMObj*) returns the mean values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. *M* is a row vector containing the mean values for elements in each column in *DMObj*.

M = mean(*DMObj*, *Dim*) returns the mean values of the elements in the columns or rows of a DataMatrix object, as specified by *Dim*. If *Dim* = 1, returns *M*, a row vector containing the mean values for elements in each column in *DMObj*. If *Dim* = 2, returns *M*, a column vector containing the mean values for elements in each row in *DMObj*. Default *Dim* = 1.

M = mean(*DMObj*, *Dim*, *IgnoreNaN*) specifies if NaNs should be ignored. *IgnoreNaN* can be true (default) or false.

More About

· DataMatrix object

See Also

DataMatrix | median | sum | max | min

median (DataMatrix)

Return median values in DataMatrix object

Syntax

```
Med = median(DMObj)
Med = median(DMObj, Dim)
Med = median(DMObj, Dim, IgnoreNaN)
```

Input Arguments

DMObj	DataMatrix object, such as created by `DataMatrix` (object constructor).
Dim	Scalar specifying the dimension of *DMObj* to calculate the medians. Choices are: • `1` — Default. Returns median values for elements in each column. • `2` — Returns median values for elements in each row.
IgnoreNaN	Specifies if NaNs should be ignored. Choices are `true` (default) or `false`.

Output Arguments

Med	Either of the following: • Row vector containing the median values from elements in each column in *DMObj* (when *Dim* = 1) • Column vector containing the median values from elements in each row in *DMObj* (when *Dim* = 2)

Description

Med = median(*DMObj*) returns the median values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. *Med* is a row vector containing the median values for elements in each column in *DMObj*.

Med = median(*DMObj*, *Dim*) returns the median values of the elements in the columns or rows of a DataMatrix object, as specified by *Dim*. If *Dim* = 1, returns *Med*, a row vector containing the median values for elements in each column in *DMObj*. If *Dim* = 2, returns *Med*, a column vector containing the median values for elements in each row in *DMObj*. Default *Dim* = 1.

Med = median(*DMObj*, *Dim*, *IgnoreNaN*) specifies if NaNs should be ignored. *IgnoreNaN* can be true (default) or false.

More About

- DataMatrix object

See Also
DataMatrix | mean | min | max | sum

metafeatures

Attractor metagene algorithm for feature engineering using mutual information-based learning

Syntax

```
M = metafeatures(X)
[M,W] = metafeatures(X)
[M,W,GSorted] = metafeatures(X,G)
[M,W,GSorted,GSortedInd] = metafeatures( ___ )
[ ___ ] = metafeatures( ___ ,Name,Value)
[ ___ ] = metafeatures(T)
[ ___ ] = metafeatures(T,Name,Value)
```

Description

`M = metafeatures(X)` returns the weighted sums of features M in X using the attractor metagene algorithm described in [1].

M is a r-by-n matrix. r is the number of metafeatures identified during each repetition of the algorithm. The default number of repetitions is 1. By default, only unique metafeatures are returned in M. If multiple repetitions result in the same metafeature, then just one copy is returned in M. n is the number of samples (patients or time points).

X is a p-by-n numeric matrix. p is the number of variables, features, or genes. In other words, rows of X correspond to variables, such as measurements of gene expression for different genes. Columns correspond to different samples, such as patients or time points.

`[M,W] = metafeatures(X)` returns a p-by-r matrix W containing metafeatures weights. `M = W'*X`. p is the number of variables. r is the number of unique metafeatures or the number of times the algorithm is repeated (the default is 1).

`[M,W,GSorted] = metafeatures(X,G)` uses a p-by-1 cell array of strings G containing the variable names and returns a p-by-r cell array of variable names GSorted sorted by the decreasing weight.

The *i*th column of GSorted lists the feature (variable) names in order of their contributions to the *i*th metafeature.

`[M,W,GSorted,GSortedInd] = metafeatures( ___ )` returns the indices GSortedInd such that GSorted = G(GSortedInd).

`[ ___ ] = metafeatures( ___ ,Name,Value)` uses additional options specified by one or more Name,Value pair arguments.

`[ ___ ] = metafeatures(T)` uses a *p*-by-*n* table T. Gene names are the row names of the table. `M = W'*T{:,:}`.

`[ ___ ] = metafeatures(T,Name,Value)` uses additional options specified by one or more Name,Value pair arguments.

Note: It is possible that the number of metafeatures (*r*) returned in M can be fewer than the number of replicates (repetitions). Even though you may have set the number of replicates to a positive integer greater than 1, if each repetition returns the same metafeature, then *r* is 1, and M is 1-by-*n*. This is because, by default, the function returns only unique metafeatures. If you prefer to get all metafeatures, set `'ReturnUnique'` to `false`. A metafeature is considered unique if the Pearson correlation between it and all previously found metafeatures is less than the `'UniqueTolerance'` value (the default value is `0.98`).

Examples

Apply Attractor Metagene Algorithm to Gene Expression Data

Load the breast cancer gene expression data. The data was retrieved from the Cancer Genome Atlas (TCGA) on May 20, 2014 and contains gene expression data of 17814 genes for 590 different patients. The expression data is stored in the variable `geneExpression`. The gene names are stored in the variable `geneNames`.

```
load TCGA_Breast_Gene_Expression
```

The data has several NaN values.

```
sum(sum(isnan(geneExpression)))
```

```
ans =

        1695
```

Use the *k*-nearest neighbor imputation method to replace missing data with the corresponding value from an average of the *k* columns that are nearest.

```
geneExpression = knnimpute(geneExpression,3);
```

There are three common drivers of breast cancer: ERBB2, enstrogen, and progestrone. `metafeatures` allows you to seed the starting weights to focus on the genes of interest. In this case, set the weight for each of these genes to 1 in three different rows of `startValues`. Each row corresponds to initial values for a different replicate (repetition).

```
erbb         = find(strcmp('ERBB2',geneNames));
estrogen     = find(strcmp('ESR1',geneNames));
progestrone  = find(strcmp('PGR',geneNames));

startValues = zeros(size(geneExpression,1),3);
startValues(erbb,1)        = 1;
startValues(estrogen,2)    = 1;
startValues(progestrone,3) = 1;
```

Apply the attractor metagene algorithm to the imputed data.

```
[meta, weights, genes_sorted] = metafeatures(geneExpression,geneNames,'start',startValu
```

The variable `meta` has the value of three metagenes discovered for each sample. Plot these three metagenes to gain insight into the nature of gene regulation across different phenotypes of breast cancer.

```
plot3(meta(1,:),meta(2,:),meta(3,:),'o')
xlabel('ERBB2 metagene')
ylabel('Estrogen metagene')
zlabel('Progestrone metagene')
```
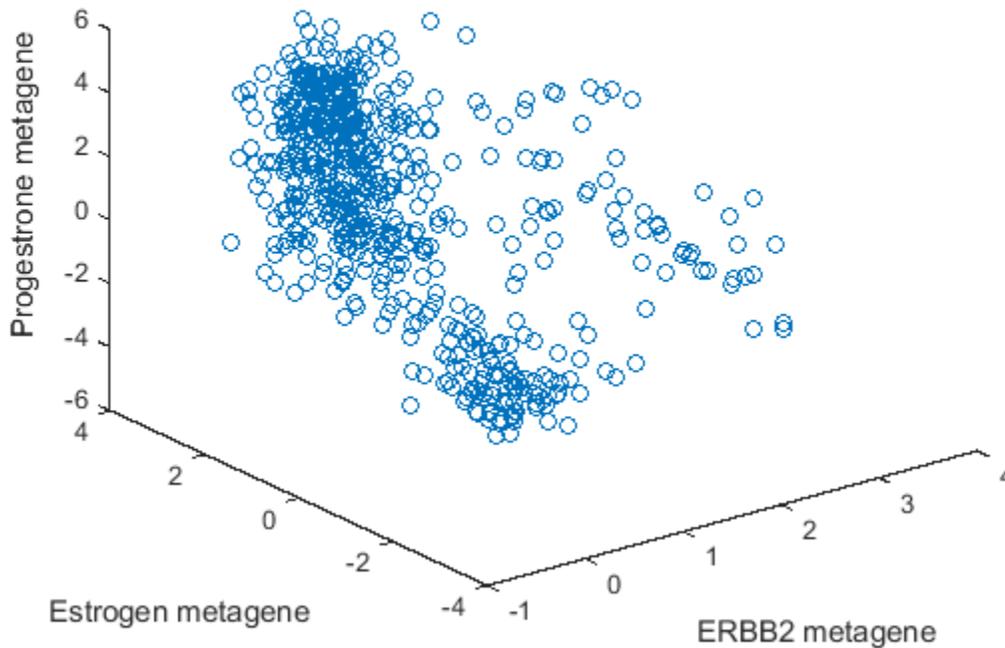
Based on the plot, observe the following.

- There is a group of points clustered together with low values for all three metagenes. Based on mRNA levels, these could be triple-negative or basal type breast cancer.

- There is a group of points that have high estrogen receptor metagene expression and span across both high and low progestrone metagene expression. There are no points with high progestrone metagene expression and low estrogen metagene expression. This is consistent with the observation that ER-/PR+ breast cancers are extremely rare [3].

- The remaining points are the ERBB2 positive cancers. They have less representation in this data set than the hormone-driven and triple negative cancers.

- "Identifying Biomolecular Subgroups Using Attractor Metagenes"

Input Arguments

X — Data
numeric matrix

Data, specified as a numeric matrix. Rows of X correspond to variables, such as measurements of gene expression. Columns correspond to different samples, such as patients or time points.

G — Variable names
cell array of strings

Variable names, specified as a cell array of strings.

T — Data
table

Data, specified as a table. The row names of the table correspond to the names of features or genes, and the columns represent different samples, such as patients or time points.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example:

'Alpha' — Tuning parameter for the number of metafeatures
5 (default) | positive scalar

Tuning parameter for the number of metafeatures, specified as the comma-separated pair consisting of 'Alpha' and a string or positive number. This parameter controls the nonlinearity of the function that calculates the weights as described in the "Attractor Metagene Algorithm" on page 1-1144. As alpha increases, the number of metafeatures tends to increase. This parameter is often the most important parameter to adjust in the analysis of a data set.

Example: 'Alpha',3

`'Start'` — Option for choosing initial weights
`'random'` (default) | `'robust'` | matrix

Option for choosing initial weights, specified as the comma-separated pair consisting of `'Alpha'` and a string or matrix. This table summarizes the available options.

Option	Description
`'random'`	Initialize the weights to a vector of positive weights chosen uniformly at random and scaled such that they sum to 1. Choose a different initial weight vector for each replicate. This option is the default.
`'robust'`	If X or T has n columns, run the algorithm n times. On the ith evaluation of the algorithm, the weights are initialized to all zeros with the exception of the ith weight, which is set to 1. This option is useful when you are attempting to find all metafeatures of a data set.
matrix	n-by-r matrix of initial weights. The algorithm runs r times. The weights in the ith run of the algorithm are initialized to the ith column of the matrix.

Example: `'Start','robust'`

`'Replicates'` — Number of times to repeat the algorithm
1 (default) | positive integer

Number of times to repeat the algorithm, specified as the comma-separated pair consisting of `'Replicates'` and a positive integer. This option is valid only with the `'random'` start option. The default is 1.

Example: `'Replicates',2`

`'ReturnUnique'` — Unique metafeatures flag
true (default) | false

Unique metafeatures flag, specified as the comma-separated pair consisting of `'ReturnUnique'` and `true` or `false`. If true, then only the unique metafeatures are returned. The default is `true`.

This option is useful when the algorithm is repeated multiple times. By setting this option to `true`, you choose to look at just the unique metafeatures since the same set of metafeatures can be discovered for different initializations.

A metafeature is considered unique if the Pearson correlation between it and all previously found metafeatures is less than the `'UniqueTolerance'` value (the default value is `0.98`).

To run the algorithm multiple times, set the `'Replicates'` name-value pair argument or the `'Start'` option to `'robust'` or a matrix with more than 1 row.

Example: `'ReturnUnique',false`

`'UniqueTolerance'` — Tolerance for metafeature uniqueness
0.98 (default) | real number between 0 and 1

Tolerance for metafeature uniqueness, specified as the comma-separated pair consisting of `'UniqueTolerance'` and a real number between 0 and 1.

A metafeature is considered unique if the Pearson correlation between it and all previously found metafeatures is less than the `'UniqueTolerance'` value.

Example: `'UniqueTolerance',0.90`

`'Options'` — Options for controlling the algorithm
`[]` (default) | structure

Options for controlling the algorithm, specified as the comma-separated pair consisting of `'Options'` and a structure. This table summarizes these options.

Option	Description
Display	Level of output display. Choices are `'off'` or `'iter'`. The default is `'off'`.
MaxIter	Maximum number of iterations allowed. The default is 100.
Tolerance	If M changes by less than the tolerance in an iteration, then the algorithm stops. The default is `1e-6`.
Streams	A `RandStream` object. If you do not specify any streams, metafeatures uses the default random stream.
UseParallel	Logical value indicating whether to perform calculations in parallel if a parallel pool and Parallel Computing Toolbox™ are available. For problems with large data sets relative to the available system memory, running in parallel can degrade performance. The default is `false`.

Example: `'Options',struct('Display','iter')`

Output Arguments

M — Metafeatures
numeric matrix

Metafeatures, returned as a numeric matrix. It is an *r*-by-*n* matrix containing the weighted sums of the features in X. *r* is the number of replicates performed by the algorithm. *n* is the number of different samples such as time points or patients.

Note: It is possible that the number of metafeatures (*r*) returned in M can be fewer than the number of replicates (repetitions). Even though you may have set the number of replicates to a positive integer greater than 1, if each repetition returns the same metafeature, then *r* is 1, and M is 1-by-*n*. This is because, by default, the function returns only unique metafeatures. If you prefer to get all metafeatures, set `'ReturnUnique'` to `false`. A metafeature is considered unique if the Pearson correlation between it and all previously found metafeatures is less than the `'UniqueTolerance'` value (the default value is `0.98`).

W — Metafeatures weights
numeric matrix

Metafeatures weights, returned as a numeric matrix. It is a *p*-by-*r* matrix. *p* is the number of variables. *r* is the number of replicates performed by the algorithm.

GSorted — Sorted variable names
cell array of strings

Sorted variable names, returned as a cell array of strings. It is a *p*-by-*r* cell array. The names are sorted by decreasing weight. The *i*th column of the GSorted lists the variable names in order of their contributions to *i*th metafeature.

If GSorted is requested without G or if `T.Properties.RowNames` is empty, then the algorithm names each variable (feature) as `Var`*i*, which corresponds to the *i*th row of X.

GSortedInd — Index to GSorted
matrix

Index to GSorted, returned as a matrix of indices. It is a *p*-by-*r* matrix. The indices satisfy `GSorted = G(GSortedInd)` or `GSorted = T.Properties.RowNames(GSortedInd)`.

More About

Attractor Metagene Algorithm

The attractor metagene algorithm [1] is an iterative algorithm that converges to metagenes with important features. A metagene is defined as any weighted sum of gene expression using a nonlinear distance metric. The distance metric is a nonlinear variant of mutual information using binning and splines as described in [2]. In fact, the use of mutual information as a distance metric is one of major benefits of this algorithm since mutual information is a robust information theoretic approach to determine the statistical dependence between variables. Therefore, it is useful for analyzing relationships among gene expression. Another advantage is that the results of the algorithm tend to be more clearly linked with a phenotype defined by gene expression.

The algorithm is initialized by either random or user-specified weights and proceeds in these steps.

1 The estimate of a metagene during the ith iteration of the algorithm is $M_i = W_i * G$, where W_i is a vector of weights of size 1-by-p (number of genes), and G is the gene expression matrix of size p-by-n (number of samples).

2 Update the weights by $W_{j,i+1} = J(M_i, G_j)$, where $W_{j,i+1}$ is the jth element of W_{i+1}, G_j is the jth row of G, and J is a similarity metric, which is defined as follows.

- If the Pearson correlation between M_i and G_j is greater than 0, then

    $J(M_i, G_j) = I(M_i, G_j)^\alpha$, where $I(M_i, G_j)$ is the measure of mutual information between two genes with minimum value 0 and maximum value 1, and α is any nonnegative number.

- If the correlation is less than or equal to 0, then $J(M_i, G_j) = 0$.

The algorithm iterates until the change in W_i between iterations is less than the defined tolerance, that is, $\|W_i - W_{i-1}\| < tolerance$ or the maximum number of iterations is reached.

The Role of α

In the similarity metric of the algorithm, the parameter α controls the degree of nonlinearity. As α increases, the number of metagenes tends to increase. If α is

sufficiently large, then each gene approximately becomes an attractor metagene. If α is zero, then all weights remain equal to each other. Therefore, there is only one attractor metagene representing the average of all genes.

Therefore, adjusting α for the data set under consideration is a key step in fine tuning the algorithm. In the case of [1], using the TCGA data from several types of cancer to identify attractor metagenes, α value of 5 resulted in between 50 and 150 attractor metagenes discovered from the data.

References

[1] Cheng, W-Y., Ou Yang, T-H., and Anastassiou, D. (2013). Biomolecular events in cancer revealed by attractor metagenes. PLoS Computational Biology 9(2): e1002920.

[2] Daub, C., Steuer, R., Selbig, J., and Kloska, S. (2004). Estimating mutual information using B-spline functions – an improved similarity measure for analysing gene expression data. BMC Bioinformatics 5, 118.

[3] Hefti, M.M., Hu, R., Knoblauch, N.W., Collins, L.C., Haibe-Kains, B., Tamimi, R.M., and Beck, A.H. (2013). Estrogen receptor negative/progesterone receptor positive breast cancer is not a reproducible subtype. Breast Cancer Research. 15:R68.

See Also
`randfeatures` | `rankfeatures` | `relieff` | `sequentialfs`

microplateplot

Display visualization of microtiter plate

Syntax

```
microplateplot(Data)
Handle = microplateplot(...)
microplateplot(Data, ...'RowLabels', RowLabelsValue, ...)
microplateplot(Data, ...'ColumnLabels', ColumnLabelsValue, ...)
microplateplot(Data, ...'TextLabels', TextLabelsValue, ...)
microplateplot(Data, ...'TextFontSize', TextFontSizeValue, ...)
microplateplot(Data, ...'MissingValueColor',
MissingValueColorValue, ...)
microplateplot(Data, ...'ToolTipFormat', ToolTipFormatValue, ...)
```

Description

microplateplot(*Data*) displays an image of a microtiter plate with each well colored according to intensity values, such as from a plate reader.

Handle = microplateplot(...) returns the handle to the axes of the plot.

microplateplot(..., '*PropertyName*', *PropertyValue*, ...) calls microplateplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

microplateplot(*Data*, ...'RowLabels', *RowLabelsValue*, ...) lets you specify labels for the rows of data.

microplateplot(*Data*, ...'ColumnLabels', *ColumnLabelsValue*, ...) lets you specify labels for the columns of data.

microplateplot(*Data*, ...'TextLabels', *TextLabelsValue*, ...) lets you specify text to overlay of the wells in the image.

`microplateplot(`*Data*`, ...'TextFontSize', `*TextFontSizeValue*`, ...)` lets you specify the font size of the text you specify with the `'TextLabels'` property.

`microplateplot(`*Data*`, ...'MissingValueColor', `*MissingValueColorValue*`, ...)` lets you specify the color of wells with missing values (NaN values).

`microplateplot(`*Data*`, ...'ToolTipFormat', `*ToolTipFormatValue*`, ...)` lets you specify the format of the text used in the well tooltips. The well tooltips display the actual value from the input matrix when you click a well. *ToolTipFormatValue* is a format string, such as used by the `sprintf` function. Default is `'Value: %.3f'`, which specifies including three digits to the right of the decimal in fixed-point notation.

Input Arguments

Data

DataMatrix object or matrix containing intensity values, such as from a plate reader.

Tip For help importing data from a spreadsheet or data file into a MATLAB matrix, see "Ways to Import Text Files".

Note: The `microplateplot` function converts any nonnumeric symbols or characters in the matrix to NaN values.

Default:

RowLabelsValue

Cell array of strings that specifies labels for the rows of data. Default is the first *N* letters of the alphabet, where *N* is the number of rows in *Data*. If there are more than 26 rows in *Data*, then the default is `AA, AB, ..., ZZ`. If *Data* is a DataMatrix object, then the default is the row labels of *Data*.

Default:

ColumnLabelsValue

Cell array of strings that specifies labels for the columns of data. Default is `1, 2, ...,` *M* , where *M* is the number of columns in *Data*. If *Data* is a DataMatrix object, then the default is the column labels of *Data*.

Default:

TextLabelsValue

Cell array of strings the same size as *Data* that specifies text to overlay on the wells of the image.

Default:

TextFontSizeValue

Positive integer specifying the font size of the text you specify with the `'TextLabels'` property. Default font size is determined automatically based on the size of the Figure window.

Default:

MissingValueColorValue

Three-element numeric vector of RGB values that specifies the color of wells with missing values (NaN values). Default is `[0, 0, 0]`, which defines black.

Default:

ToolTipFormatValue

Format string, such as used by the `sprintf` function, that specifies the format of the text used in the well tooltips. The well tooltips display the actual value from the input matrix when you click a well.

Default: `'Value: %.3f'`, which specifies including three digits to the right of the decimal in fixed-point notation.

Output Arguments

Handle

Handle to the axes of the plot.

Tip Use the *Handle* output with the `set` function and the `'YDir'` or `'XDir'` property to reverse the order of the A through H labels or 1 through 12 labels respectively. Note that in the microplate plot, the default order for the A through H labels, or `'YDir'` property, is `'reverse'` (top to bottom), and the default order for the 1 through 12 labels, or `'XDir'` property, is `'normal'` (left to right). For more information on the `'XDir'` and `'YDir'` properties, see Axes Properties.

Examples

Creating a Plot of a Microplate, Changing the Colormap, Viewing Well Values, and Adding Text Labels

1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains two variables: `assaydata`, an 8-by-12 matrix of data values from a microtiter plate, and `whiteToRed`, a 64-by-3 matrix that defines a colormap.

```
load microPlateAssay
```

2 Create a visualization of the data from the microtiter plate.

```
microplateplot(assaydata)
```

3 Change the visualization to use a white-to-red colormap, and then view a tooltip displaying the value of well D6 by clicking the well.

```
colormap(whiteToRed)
```
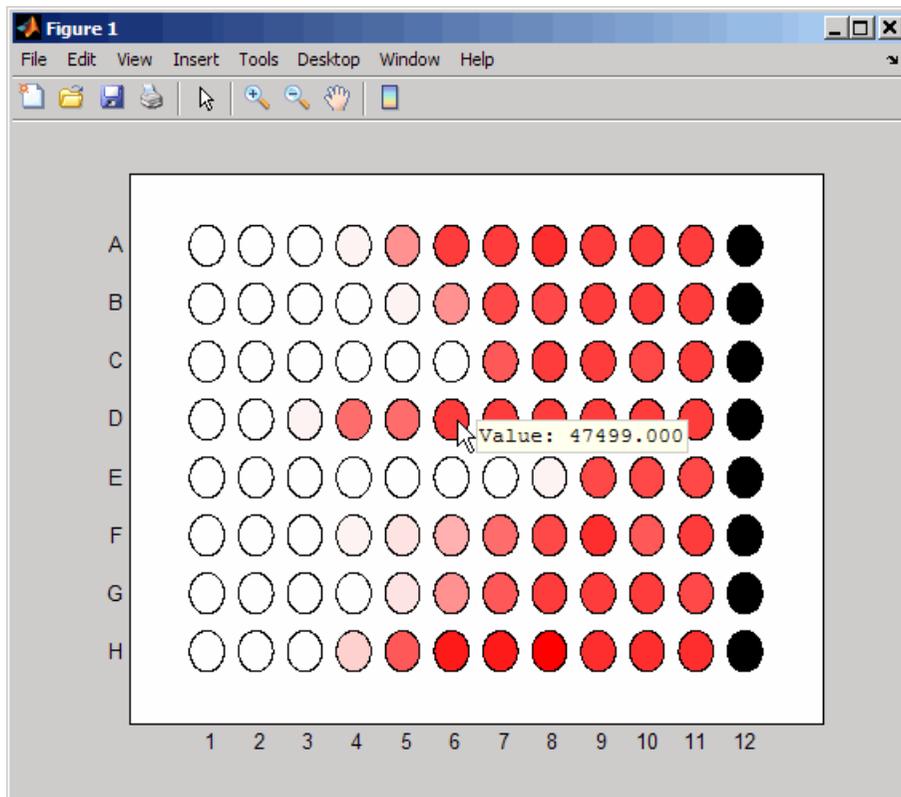
Notice that all wells in column 12 are black, indicating missing data.

4 Overlay an X on well E8.

    a Create an empty cell array.

```
mask = cell(8,12);
```

    b Add the string 'X' to the cell in the fifth row and eighth column of the array.

```
mask{5,8} = 'X';
```

    c Pass the cell array to the microplateplot function using the 'TextLabels' property.

```
microplateplot(assaydata,'TEXTLABELS',mask);
```
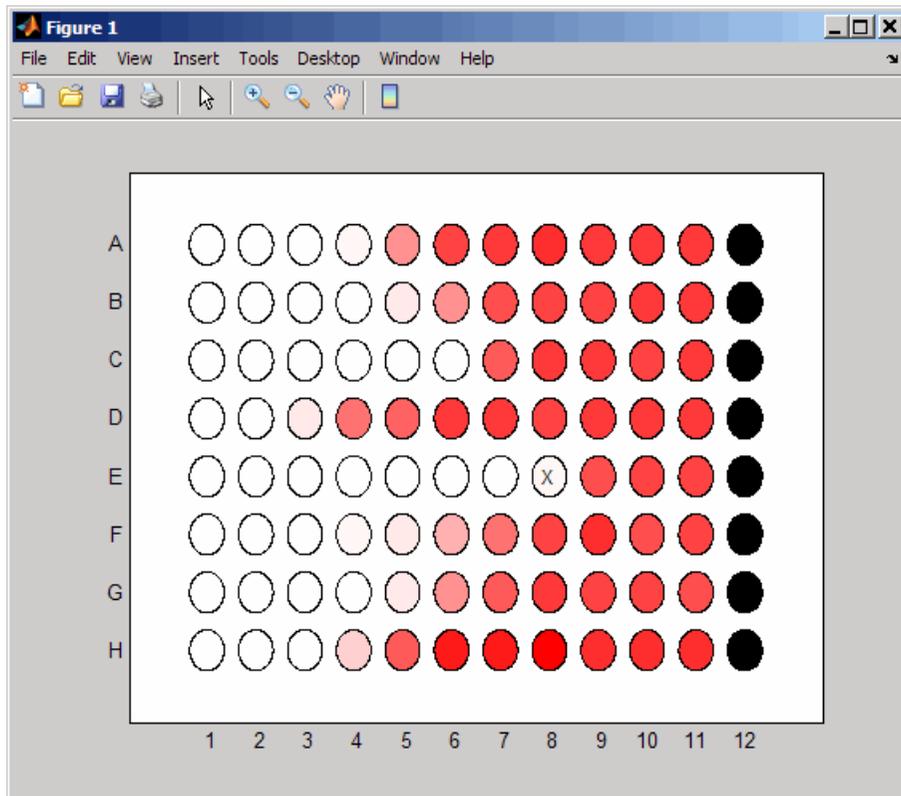
Changing the Order of Row Labels in the Plot

1   If you have not already done so, create a plot of a microplate by completing steps 1
    through 3 in Creating a Plot of a Microplate, Changing the Colormap, Viewing Well
    Values, and Adding Text Labels.

2   Return a handle to the axes of the plot, and then reverse the order of the row letter
    labels.

```
h = microplateplot(assaydata);
set(h,'YDir','normal')
```
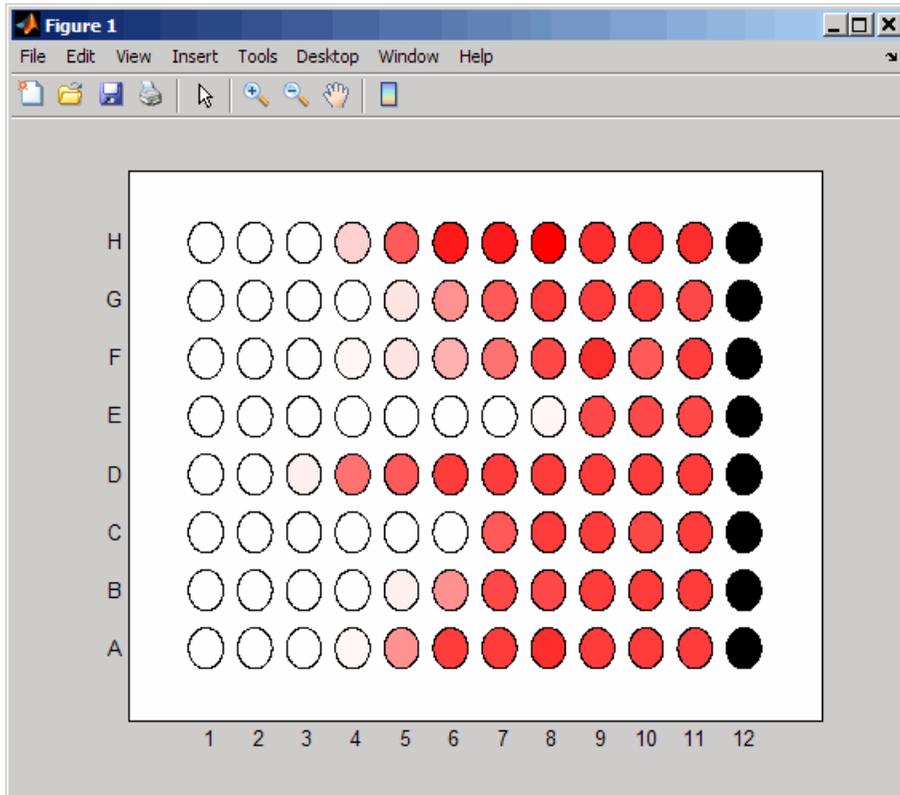
Adding a Title and Axis Labels to the Plot

For information on adding a title and *x*-axis and *y*-axis labels to your plot, see "Add Text to Graph Interactively".

Printing and Exporting the Plot

For information on printing or exporting your plot, see "Printing and Saving"

More About

- "Ways to Import Text Files"
- "Add Text to Graph Interactively"

- "Printing and Saving"

See Also
imagesc | sprintf | set

min (DataMatrix)

Return minimum values in DataMatrix object

Syntax

```
M = min(DMObj1)
[M, Indices] = min(DMObj1)
[M, Indices, Names] = min(DMObj1)
... = min(DMObj1, [], Dim)
MA = min(DMObj1, DMObj2)
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
	Note: *DMObj1* and *DMObj2* must be the same size, unless one is a scalar.
Dim	Scalar specifying the dimension of *DMObj* to return the minimum values. Choices are:
	• 1 — Default. Returns a row vector containing a minimum value for each column.
	• 2 — Returns a column vector containing a minimum value for each row.

Output Arguments

M	One of the following:
	• Scalar specifying the minimum value in *DMObj* when it contains vector of data
	• Row vector containing the minimum value for each column in *DMObj* (when *Dim* = 1)

	• Column vector containing the minimum value for each row in *DMObj* (when *Dim* = 2)
Indices	Either of the following: • Positive integer specifying the index of the minimum value in a DataMatrix object containing a vector of data • Vector containing the indices for the minimum value in each column (if *Dim* = 1) or row (if *Dim* = 2) in a DataMatrix object containing a matrix of data
Names	Vector of the row names (if *Dim* = 1) or column names (if *Dim* = 2) corresponding to the minimum value in each column or each row of a DataMatrix object.
MA	Numeric array created from the minimum elements in either of the following: • Two DataMatrix objects • A DataMatrix object and a numeric array

Description

M = min(*DMObj1*) returns the minimum value(s) in *DMObj1*, a DataMatrix object. If *DMObj1* contains a vector of data, *M* is a scalar. If *DMObj1* contains a matrix of data, *M* is a row vector containing a minimum value in each column.

[*M*, *Indices*] = min(*DMObj1*) returns *Indices*, the indices of the minimum value(s) in *DMObj1*, a DataMatrix object. If *DMObj1* contains a vector of data, *Indices* is a positive integer. If *DMObj1* contains a matrix of data, *Indices* is a vector containing the indices for the minimum value in each column (if *Dim* = 1) or row (if *Dim* = 2). If there are multiple minimum values in a column or row, the index for the first value is returned.

[*M*, *Indices*, *Names*] = min(*DMObj1*) returns *Names*, a vector of the row names (if *Dim* = 1) or column names (if *Dim* = 2) corresponding to the minimum value in each column or each row of*DMObj1*, a DataMatrix object. If there is more than one minimum value in a column or row, the row or column name for the first value is returned.

... = min(*DMObj1*, [], *Dim*) specifies which dimension to return the minimum values for, that is each column or each row in a DataMatrix object. If *Dim* = 1, returns

M, a row vector containing the minimum value in each column. If *Dim* = 2, returns *M*, a column vector containing the minimum value in each row. Default *Dim* = 1.

MA = min(*DMObj1, DMObj2*) returns *MA*, a numeric array containing the smaller of the two values from each position of *DMObj1* and *DMObj2*. *DMObj1* and *DMObj2* can both be DataMatrix objects, or one can be a DataMatrix object and the other a numeric array. They must be the same size, unless one is a scalar. *MA* has the same size (number of rows and columns) as the first nonscalar input.

More About

·   DataMatrix object

See Also
DataMatrix | sum | max

minspantree (biograph)

Find minimal spanning tree in biograph object

Syntax

```
[Tree, pred] = minspantree(BGObj)
[Tree, pred] = minspantree(BGObj, R)
[Tree, pred] = minspantree(..., 'Method', MethodValue, ...)
[Tree, pred] = minspantree(..., 'Weights', WeightsValue, ...)
```

Arguments

BGObj	Biograph object created by `biograph` (object constructor).
R	Scalar between 1 and the number of nodes.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*Tree*, *pred*] = minspantree(*BGObj*) finds an acyclic subset of edges that connects all the nodes in the undirected graph represented by an N-by-N adjacency matrix extracted from a biograph object, *BGObj*, and for which the total weight is minimized. Weights of the edges are all nonzero entries in the lower triangle of the N-by-N sparse matrix. Output *Tree* is a spanning tree represented by a sparse matrix. Output *pred* is a vector containing the predecessor nodes of the minimal spanning tree (MST), with the root node indicated by 0. The root node defaults to the first node in the largest connected component. This computation requires an extra call to the `graphconncomp` function.

[*Tree*, *pred*] = minspantree(*BGObj*, *R*) sets the root of the minimal spanning tree to node R.

[*Tree*, *pred*] = minspantree(..., '*PropertyName*', *PropertyValue*, ...) calls minspantree with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[*Tree*, *pred*] = minspantree(..., 'Method', *MethodValue*, ...) lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- 'Kruskal' — Grows the minimal spanning tree (MST) one edge at a time by finding an edge that connects two trees in a spreading forest of growing MSTs. Time complexity is O(E+X*log(N)), where X is the number of edges no longer than the longest edge in the MST, and N and E are the number of nodes and edges respectively.

- 'Prim' — Default algorithm. Grows the minimal spanning tree (MST) one edge at a time by adding a minimal edge that connects a node in the growing MST with any other node. Time complexity is O(E*log(N)), where N and E are the number of nodes and edges respectively.

Note: When the graph is unconnected, Prim's algorithm returns only the tree that contains R, while Kruskal's algorithm returns an MST for every component.

[*Tree*, *pred*] = minspantree(..., 'Weights', *WeightsValue*, ...) lets you specify custom weights for the edges. *WeightsValue* is a column vector having one entry for every nonzero value (edge) in the N-by-N sparse matrix. The order of the custom weights in the vector must match the order of the nonzero values in the N-by-N sparse matrix when it is traversed column-wise. By default, minspantree gets weight information from the nonzero entries in the N-by-N sparse matrix.

More About

- biograph object

References

[1] Kruskal, J.B. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society *7*, 48-50.

[2] Prim, R. (1957). Shortest Connection Networks and Some Generalizations. Bell System Technical Journal *36*, 1389-1401.

[3] Siek, J.G. Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

biograph | allshortestpaths | isdag | isspantree | shortestpath | traverse | graphminspantree | conncomp | isomorphism | maxflow | topoorder

minus (DataMatrix)

Subtract DataMatrix objects

Syntax

```
DMObjNew = minus(DMObj1, DMObj2)
DMObjNew = DMObj1 - DMObj2
DMObjNew = minus(DMObj1, B)
DMObjNew = DMObj1 - B
DMObjNew = minus(B, DMObj1)
DMObjNew = B - DMObj1
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

DMObjNew	DataMatrix object created by subtraction.

Description

DMObjNew = minus(*DMObj1*, *DMObj2*) or the equivalent *DMObjNew* = *DMObj1* - *DMObj2* performs an element-by-element subtraction of the DataMatrix object *DMObj2* from the DataMatrix object *DMObj1* and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = minus(*DMObj1*, *B*) or the equivalent *DMObjNew* = *DMObj1* - *B* performs an element-by-element subtraction of *B*, a numeric or logical array, from the DataMatrix object *DMObj1*, and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

DMObjNew = minus(*B*, *DMObj1*) or the equivalent *DMObjNew* = *B* - *DMObj1* performs an element-by-element subtraction of the DataMatrix object *DMObj1* from *B*, a numeric or logical array, and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

Note: Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls *DMObjNew* = minus(*X*, *Y*) for the syntax *DMObjNew* = *X* - *Y* when *X* or *Y* is a DataMatrix object.

More About

·   DataMatrix object

See Also
DataMatrix | plus

molweight

Calculate molecular weight of amino acid sequence

Syntax

```
molweight(SeqAA)
```

Arguments

SeqAA	Amino acid sequence. Enter a character string or a vector of integers from the tableAmino Acid Lookup. Examples: `'ARN'`, `[1 2 3]`. You can also enter a structure with the field `Sequence`.

Description

molweight(*SeqAA*) calculates the molecular weight for the amino acid sequence *SeqAA*.

Examples

1   Retrieve an amino acid sequence from the NCBI GenPept database.

```
rhodopsin = getgenpept('NP_000530');
```

2   Calculate the molecular weight of the sequence.

```
rhodopsinMW = molweight(rhodopsin)

rhodopsinMW =

  3.8892e+004
```

See Also
aacount | atomiccomp | isoelectric | isotopicdist | proteinplot

molviewer

Display and manipulate 3-D molecule structure

Syntax

```
molviewer
molviewer(File)
molviewer(pdbID)
molviewer(pdbStruct)
FigureHandle = molviewer(...)
```

Input Arguments

File	String specifying one of the following:
	• File name of a file on the MATLAB search path or in the MATLAB Current Folder
	• Path and file name
	• URL pointing to a file (URL must begin with a protocol such as http://, ftp://, or file://)
	The referenced file is a molecule model file, such as a Protein Data Bank (PDB)-formatted file (ASCII text file). Valid file types include:
	• PDB
	• MOL (MDL)
	• SDF
	• XYZ
	• SMOL
	• JVXL
	• CIF/mmCIF
pdbID	String specifying a unique identifier for a protein structure record in the PDB database.

	Note: Each structure in the PDB database is represented by a four-character alphanumeric identifier. For example, `4hhb` is the identifier for hemoglobin.
pdbStruct	A structure containing a field for each PDB record, such as returned by the `getpdb` or `pdbread` function.

Output Arguments

FigureHandle	Figure handle to the Molecule Viewer.

Description

`molviewer` opens the Molecule Viewer app. You can display 3-D molecular structures by selecting **File > Open**, **File > Load PDB ID**, or **File > Open URL**.

`molviewer(`*File*`)` reads the data in a molecule model file, *File*, and opens the Molecule Viewer app displaying the 3-D molecular structure for viewing and manipulation.

`molviewer(`*pdbID*`)` retrieves the structural data of a protein, *pdbID*, from the PDB database and opens the Molecule Viewer app displaying the 3-D molecular structure for viewing and manipulation.

`molviewer(`*pdbStruct*`)` reads the data from *pdbStruct*, a structure containing a field for each PDB record, and opens the Molecule Viewer app displaying a 3-D molecular structure for viewing and manipulation.

FigureHandle = `molviewer(...)` returns the figure handle to the Molecule Viewer window.

Tip You can pass the *FigureHandle* to the `evalrasmolscript` function, which sends RasMol script commands to the Molecule Viewer window.

Tip If you receive any errors related to memory or Java heap space, try increasing your Java heap space as described at:

```
http://www.mathworks.com/support/solutions/data/1-18I2C.html
```
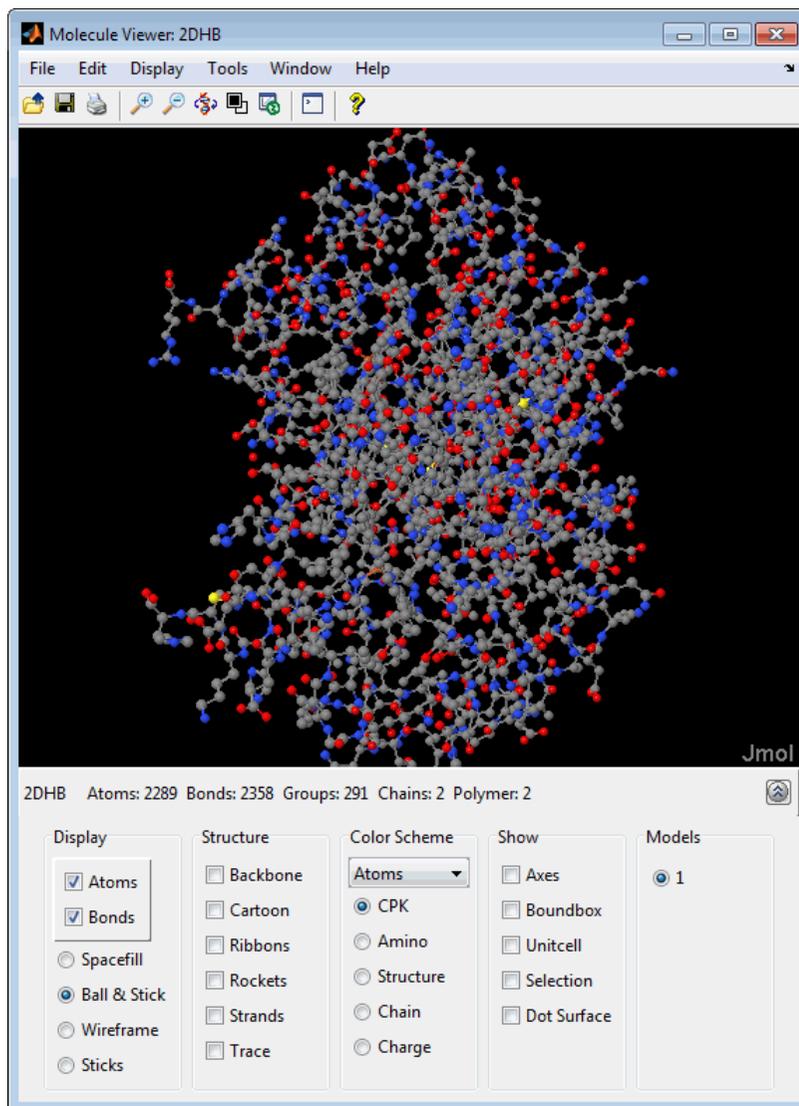
After displaying the 3-D molecule structure, you can:

- Hover the mouse over a subcomponent of the molecule to display an identification label for it.

- Spin and rotate the molecule at different angles by click-dragging it.

- Spin the molecule in the *x-z* plane by clicking .

- Spin the molecule in the *x-y* plane by pressing and holding the **Shift** key, then click-dragging left and right.

- Zoom in a stepless fashion by pressing and holding the **Shift** key, then click-dragging up and down.

- Zoom in a stepwise fashion by clicking the figure, then turning the mouse scroll wheel, or by clicking the following buttons:

   or

- Move the molecule by pressing and holding **Ctrl** + **Alt**, then click-dragging.

- Change the background color between black and white by clicking .

- Reset the molecule position by clicking .

- Show or hide the Control Panel by clicking .

- Manipulate and annotate the 3-D structure by selecting options in the Control Panel or, for a complete list of options, by right-clicking the Molecule Viewer window to select commands:
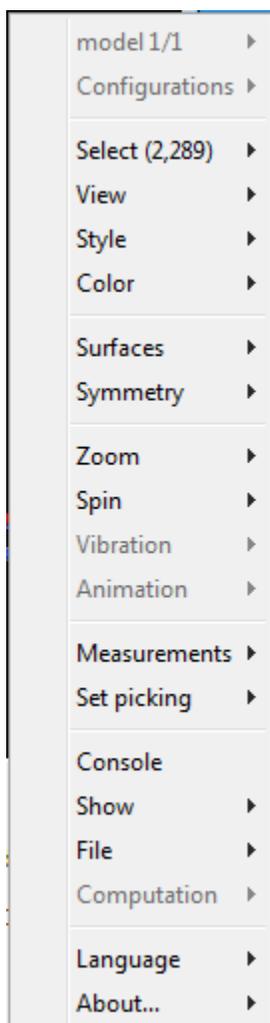
- Display the Jmol Script Console by clicking .
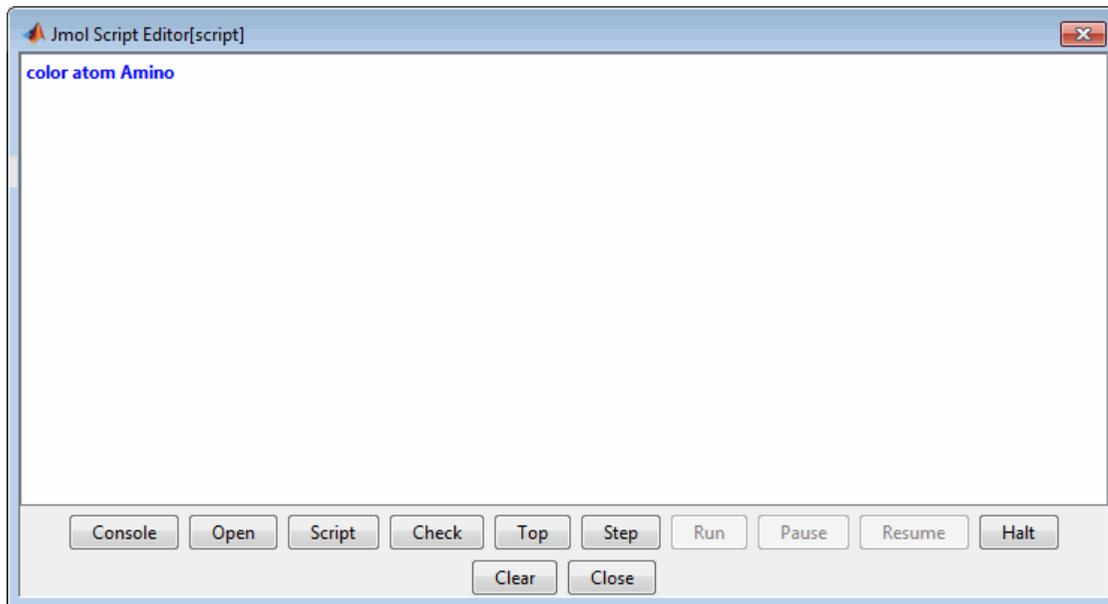
Examples

View the acetylsalicylic acid (aspirin) molecule, whose structural information is contained in the Elsevier MDL molecule file `aspirin.mol`.

```
molviewer('aspirin.mol')
```

View the H5N1 influenza virus hemagglutinin molecule, whose structural information is located at www.rcsb.org/pdb/files/2FK0.pdb.gz.

```
molviewer('http://www.rcsb.org/pdb/files/2FK0.pdb.gz')
```

View the molecule with a PDB identifier of 2DHB.

```
molviewer('2DHB')
```

View the molecule with a PDB identifier of 4hhb, and create a figure handle for the Molecule Viewer.

```
FH = molviewer('4hhb')
```

Use the `getpdb` function to retrieve protein structure data from the PDB database and create a MATLAB structure. Then view the protein molecule.

```
pdbstruct = getpdb('1vqx')
molviewer(pdbstruct)
```

See Also

evalrasmolscript | getpdb | pdbread | pdbsuperpose | pdbtransform | pdbwrite

msalign

Align peaks in signal to reference peaks

Syntax

```
IntensitiesOut = msalign(X, Intensities, RefX)
... = msalign(..., 'Rescaling', RescalingValue, ...)
... = msalign(..., 'Weights', WeightsValue, ...)
... = msalign(..., 'MaxShift', MaxShiftValue, ...)
... = msalign(..., 'WidthOfPulses', WidthOfPulsesValue, ...)
... = msalign(..., 'WindowSizeRatio', WindowSizeRatioValue, ...)
... = msalign(..., 'Iterations', IterationsValue, ...)
... = msalign(..., 'GridSteps', GridStepsValue, ...)
... = msalign(..., 'SearchSpace', SearchSpaceValue, ...)
... = msalign(..., 'ShowPlot', ShowPlotValue, ...)
[IntensitiesOut, RefXOut] = msalign(..., 'Group', GroupValue, ...)
```

Input Arguments

X	Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix *Intensities*. The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.
Intensities	Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector *X*.
RefX	Vector of separation-unit values of known reference masses in a sample signal.

	Tip For reference peaks, select compounds that are not expected to have significant shifts among the different signals. For example, in mass spectrometry, select compounds that do not undergo structural transformation, such as phosphorylation. Doing so increases the accuracy of your alignment and lets you detect compounds that exhibit structural transformations among the sample signal.
RescalingValue	Controls the rescaling of *X*. Choices are `true` (default) or `false`. When `false`, the output signal is aligned only to the reference peaks by using constant shifts. By default, `msalign` estimates a rescaling factor, unless *RefX* contains only one reference peak.
WeightsValue	Vector of positive values, with the same number of elements as *RefX*. The default vector is `ones(size(`*RefX*`))`.
MaxShiftValue	Two-element vector, in which the first element is negative and the second element is positive, that specifies the lower and upper limits of a range, in separation units, relative to each peak. No peak shifts beyond these limits. Default is `[-100 100]`.
WidthOfPulsesValue	Positive value that specifies the width, in separation units, for all the Gaussian pulses used to build the correlating synthetic signal. The point of the peak where the Gaussian pulse reaches `60.65`% of its maximum is set to the width specified by *WidthOfPulsesValue*. Default is `10`.
WindowSizeRatioValue	Positive value that specifies a scaling factor that determines the size of the window around every alignment peak. The synthetic signal is compared to the input signal only within these regions, which saves computation time. The size of the window is given in separation-units by *WidthOfPulsesValue* * *WindowSizeRatioValue*. Default is `2.5`, which means at the limits of the window, the Gaussian pulses have a value of `4.39`% of their maximum.

IterationsValue	Positive integer that specifies the number of refining iterations. At every iteration, the search grid is scaled down to improve the estimates. Default is 5.
GridStepsValue	Positive integer that specifies the number of steps for the search grid. At every iteration, the search area is divided by *GridStepsValue*^2. Default is 20.
SearchSpaceValue	String that specifies the type of search space. Choices are: • 'regular' — Default. Evenly spaced lattice. • 'latin' — Random Latin hypercube with *GridStepsValue*^2 samples.
ShowPlotValue	Controls the display of a plot of an original and aligned signal over the reference masses specified by *RefX*. Choices are true, false, or *I*, an integer specifying the index of a signal in *Intensities*. If you set to true, the first signal in *Intensities* is plotted. Default is: • false — When return values are specified. • true — When return values are not specified.
GroupValue	Controls the creation of *RefXOut*, a new vector of separation-unit values to be used as reference masses for aligning the peaks. This vector is created by adjusting the values in *RefX*, based on the sample data from multiple signals in *Intensities*, such that the overall shifting and scaling of the peaks is minimized. Choices are true or false (default). --- **Tip** Set *GroupValue* to true only if *Intensities* contains data for a large number of signals, and you are not confident of the separation-unit values used for your reference peaks in *RefX*. Leave *GroupValue* set to false if you are confident of the separation-unit values used for your reference peaks in *RefX*.

Output Arguments

IntensitiesOut	Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The intensity values represent a shifting and scaling of the data.
RefXOut	Vector of separation-unit values of reference masses, calculated from *RefX* and the sample data from multiple signals in *Intensities*, when you set *GroupValue* to true.

Description

Tip Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

IntensitiesOut = msalign(*X, Intensities, RefX*) aligns the peaks in raw, noisy signal data, represented by *Intensities* and *X*, to reference peaks, provided by *RefX*. First, it creates a synthetic signal from the reference peaks using Gaussian pulses centered at the separation-unit values specified by *RefX*. Then, it shifts and scales the separation-unit scale to find the maximum alignment between the input signals and the synthetic signal. (It uses an iterative multiresolution grid search until it finds the best scale and shift factors for each signal.) Once the new separation-unit scale is determined, the corrected signals are created by resampling their intensities at the original separation-unit values, creating *IntensitiesOut*, a vector or matrix of corrected intensity values. The resampling method preserves the shape of the peaks.

Tip The msalign function works best with three to five reference peaks that you know will appear in the signal. If you use a single reference peak (internal standard), there is a possibility of aligning sample peaks to the incorrect reference peaks as msalign both scales and shifts the *X* vector. If using a single reference peak, you might need to only shift the *X* vector. To do this, use *IntensitiesOut* = interp1(*X, Intensities, X-*

(*ReferencePeak*-*ExperimentalPeak*)). For more information, see Aligning a Mass Spectrum with One Reference Peak.

... = msalign(..., '*PropertyName*', *PropertyValue*, ...) calls msalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = msalign(..., 'Rescaling', *RescalingValue*, ...) controls the rescaling of *X*. Choices are true (default) or false. When false, the output signal is aligned only to the reference peaks by using constant shifts. By default, msalign estimates a rescaling factor, unless *RefX* contains only one reference peak.

... = msalign(..., 'Weights', *WeightsValue*, ...) specifies the relative weight for each mass in *RefX*, the vector of reference separation-unit values. *WeightsValue* is a vector of positive values, with the same number of elements as *RefX*. The default vector is ones(size(*RefX*)), which means each reference peak is weighted equally, so that more intense reference peaks have a greater effect in the alignment algorithm. If you have a less intense reference peak, you can increase its weight to emphasize it more in the alignment algorithm.

... = msalign(..., 'MaxShift', *MaxShiftValue*, ...) specifies the lower and upper limits of the range, in separation units, relative to each peak. No peak shifts beyond these limits. *MaxShiftValue* is a two-element vector, in which the first element is negative and the second element is positive. Default is [-100 100].

Note: Use these values to tune the robustness of the algorithm. Ideally, you should keep the range within the maximum expected shift. If you try to correct larger shifts by increasing the limits, you increase the possibility of picking incorrect peaks to align to the reference masses.

... = msalign(..., 'WidthOfPulses', *WidthOfPulsesValue*, ...) specifies the width, in separation units, for all the Gaussian pulses used to build the correlating synthetic signal. The point of the peak where the Gaussian pulse reaches 60.65% of its maximum is set to the width you specify with *WidthOfPulsesValue*. Choices are any positive value. Default is 10. *WidthOfPulsesValue* may also be a function

handle. The function is evaluated at the respective separation-unit values and returns a variable width for the pulses. Its evaluation should give reasonable values from 0 to `max(abs(Range))`; otherwise, the function returns an error.

Note: Tuning the spread of the Gaussian pulses controls a tradeoff between robustness (wider pulses) and precision (narrower pulses). However, the spread of the pulses is unrelated to the shape of the observed peaks in the signal. The purpose of the pulse spread is to drive the optimization algorithm.

... = msalign(..., 'WindowSizeRatio', *WindowSizeRatioValue*, ...) specifies a scaling factor that determines the size of the window around every alignment peak. The synthetic signal is compared to the sample signal only within these regions, which saves computation time. The size of the window is given in separation units by *WidthOfPulsesValue* * *WindowSizeRatioValue*. Choices are any positive value. Default is 2.5, which means at the limits of the window, the Gaussian pulses have a value of 4.39% of their maximum.

... = msalign(..., 'Iterations', *IterationsValue*, ...) specifies the number of refining iterations. At every iteration, the search grid is scaled down to improve the estimates. Choices are any positive integer. Default is 5.

... = msalign(..., 'GridSteps', *GridStepsValue*, ...) specifies the number of steps for the search grid. At every iteration, the search area is divided by *GridStepsValue*^2. Choices are any positive integer. Default is 20.

... = msalign(..., 'SearchSpace', *SearchSpaceValue*, ...) specifies the type of search space. Choices are:

- 'regular' — Default. Evenly spaced lattice.
- 'latin' — Random Latin hypercube with *GridStepsValue*^2 samples.

... = msalign(..., 'ShowPlot', *ShowPlotValue*, ...) controls the display of a plot of an original and aligned signal over the reference masses specified by *RefX*. Choices are true, false, or *I*, an integer specifying the index of a signal in *Intensities*. If set to true, the first signal in *Intensities* is plotted. Default is:

- false — When return values are specified.
- true — When return values are not specified.

[*IntensitiesOut, RefXOut*] = msalign(..., 'Group', *GroupValue*, ...) controls the creation of *RefXOut*, a new vector of separation-unit values to use as reference masses for aligning the peaks. This vector is created by adjusting the values in *RefX*, based on the sample data from multiple signals in *Intensities*, such that the overall shifting and scaling of the peaks is minimized. Choices are true or false (default).

Tip Set *GroupValue* to true only if *Intensities* contains data for a large number of signals, and you are not confident of the separation-unit values used for your reference peaks in *RefX*. Leave *GroupValue* set to false if you are confident of the separation-unit values used for your reference peaks in *RefX*.

Examples

Aligning a Mass Spectrum with Three or More Reference Peaks

1   Load a MAT-file, included with the Bioinformatics Toolbox software, that contains sample data, reference masses, and parameter data for synthetic peak width.

```
load sample_lo_res
R = [3991.4 4598 7964 9160];
W = [60 100 60 100];
```

2   Display a color image of the mass spectra before alignment.

```
msheatmap(MZ_lo_res,Y_lo_res,'markers',R,'range',[3000 10000])
title('before alignment')
```
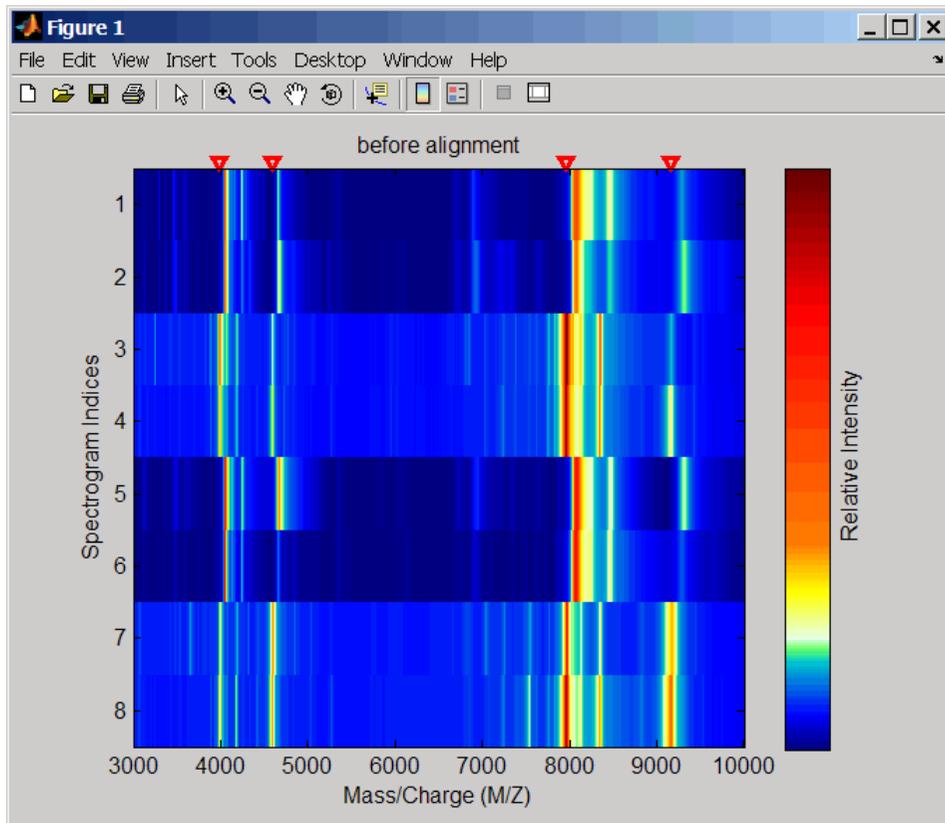
3 Align spectra with reference masses and display a color image of mass spectra after alignment.

```
YA = msalign(MZ_lo_res,Y_lo_res,R,'weights',W);
msheatmap(MZ_lo_res,YA,'markers',R,'range',[3000 10000])
title('after alignment')
```
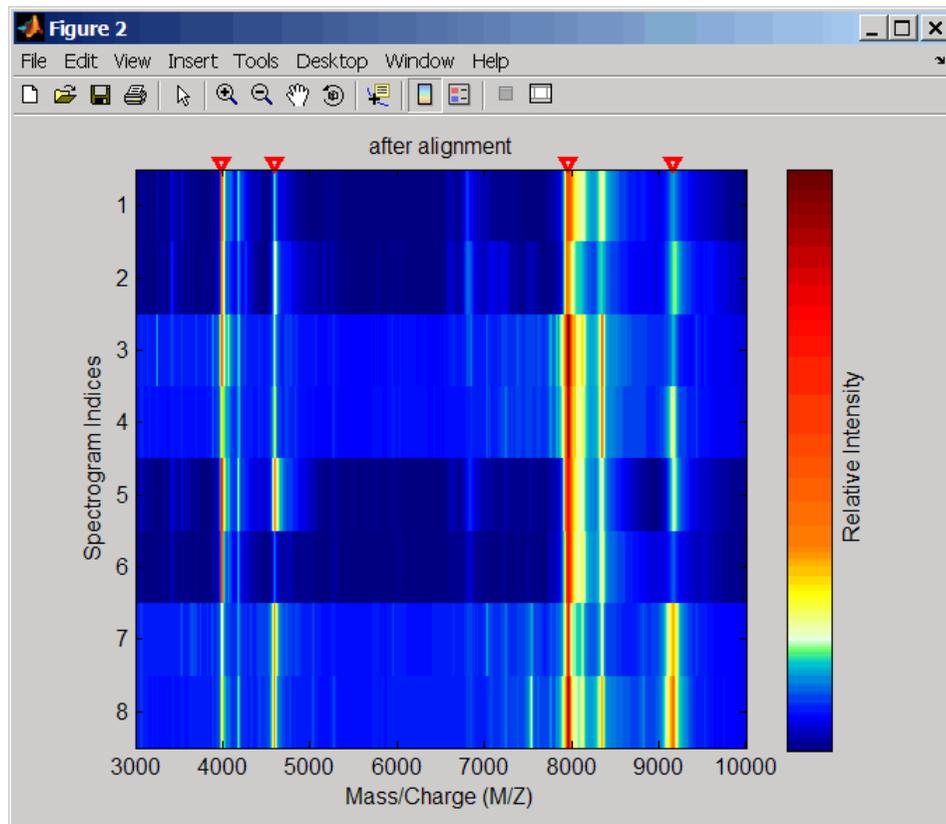
Aligning a Mass Spectrum with One Reference Peak

It is not recommended to use the `msalign` function if you have only one reference peak. Instead, use the following procedure, which shifts the *X* input vector, but does not scale it.

1   Load sample data and view the first sample spectrum.

```
load sample_lo_res
MZ = MZ_lo_res;
Y = Y_lo_res(:,1);
msviewer(MZ, Y)
```
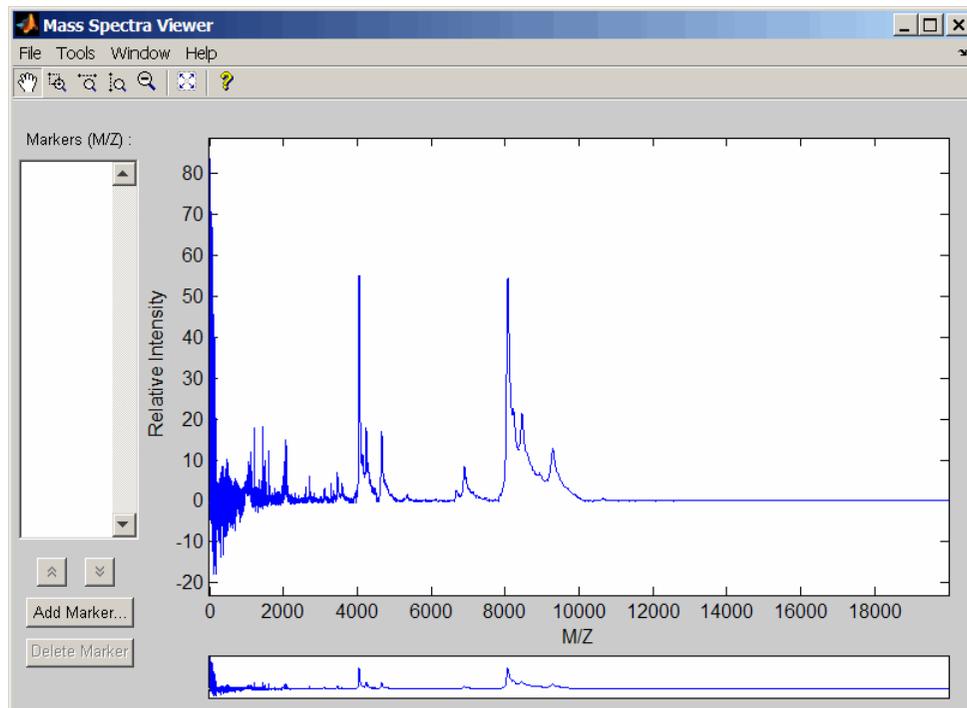
2 Use the tall peak around 4000 m/z as the reference peak. To determine the reference peak's m/z value, click [Q], and then click-drag to zoom in on the peak. Right-click in the center of the peak, and then click **Add Marker** to label the peak with its m/z value.
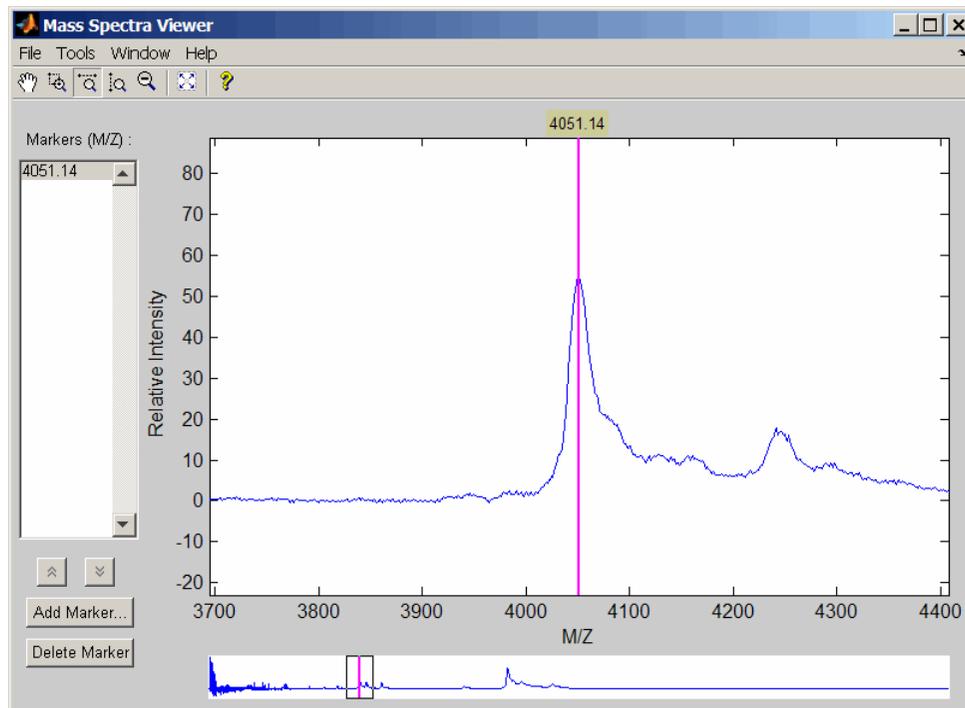
3   Shift a spectrum by the difference between RP, the known reference mass of 4000 m/z, and SP, the experimental mass of 4051.14 m/z.

```
RP = 4000;
SP = 4051.14;
YOut = interp1(MZ, Y, MZ-(RP-SP));
```

4   Plot the original spectrum in red and the shifted spectrum in blue and zoom in on the reference peak.

```
plot(MZ,Y,'r',MZ,YOut,'b:')
xlabel('Mass/Charge (M/Z)')
ylabel('Relative Intensity')
legend('Y','YOut')
axis([3600 4800 -2 60])
```
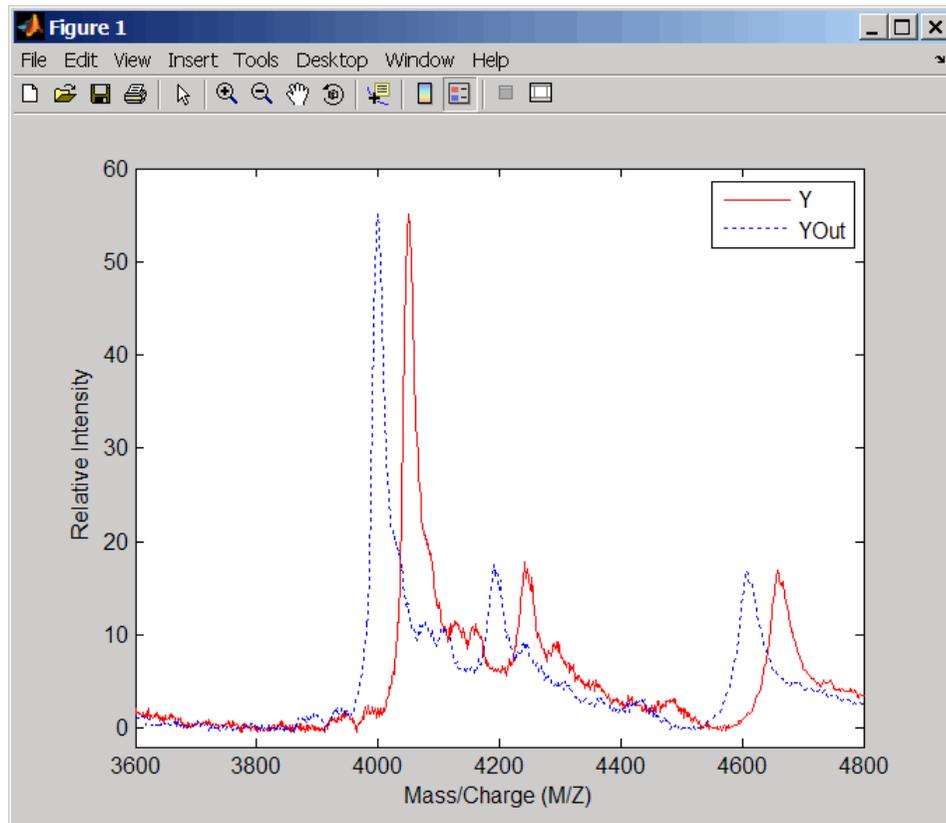
More About

- Preprocessing Raw Mass Spectrometry Data

References

[1] Monchamp, P., Andrade-Cetto, L., Zhang, J.Y., and Henson, R. (2007) Signal Processing Methods for Mass Spectrometry. In Systems Bioinformatics: An Engineering Case-Based Approach, G. Alterovitz and M.F. Ramoni, eds. (Artech House Publishers).

See Also
msbackadj | msheatmap | mspalign | mspeaks | msresample | msviewer

msbackadj

Correct baseline of signal with peaks

Syntax

Yout = msbackadj(*X*, *Intensities*)
Yout = msbackadj(*X*, *Intensities*, ...'WindowSize',
WindowSizeValue, ...)
Yout = msbackadj(*X*, *Intensities*, ...'StepSize', *StepSizeValue*, ...)
Yout = msbackadj(*X*, *Intensities*, ...'RegressionMethod',
RegressionMethodValue, ...)
Yout = msbackadj(*X*, *Intensities*, ...'EstimationMethod',
EstimationMethodValue, ...)
Yout = msbackadj(*X*, *Intensities*, ...'SmoothMethod',
SmoothMethodValue, ...)
Yout = msbackadj(*X*, *Intensities*, ...'QuantileValue',
QuantileValueValue, ...)
Yout = msbackadj(*X*, *Intensities*, ...'PreserveHeights',
PreserveHeightsValue, ...)
Yout = msbackadj(*X*, *Intensities*, ...'ShowPlot', *ShowPlotValue*, ...)

Arguments

X	Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix *Intensities*. The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.
Intensities	Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector *X*.

Description

Tip Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

Yout = msbackadj(*X*, *Intensities*) adjusts the variable baseline of a raw signal with peaks by following steps:

1 Estimates the baseline within multiple shifted windows of width 200 separation units

2 Regresses the varying baseline to the window points using a spline approximation

3 Adjusts the baseline of the peak signals supplied by *Intensities*

Yout = msbackadj(*X*, *Intensities*, ...'*PropertyName*', *PropertyValue*, ...) calls msbackadj with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Yout = msbackadj(*X*, *Intensities*, ...'WindowSize', *WindowSizeValue*, ...) specifies the width for the shifting window. *WindowSizeValue* can also be a function handle. The function is evaluated at the respective *X* values and returns a variable width for the windows. This option is useful for cases where the resolution of the signal is dissimilar at different regions. The default value is 200 (baseline point estimated for windows with a width of 200 separation units).

Note: The result of this algorithm depends on carefully choosing the window size and the step size. Consider the width of your peaks in the signal and the presence of possible drifts. If you have wider peaks toward the end of the signal, you may want to use variable parameters.

Yout = msbackadj(*X*, *Intensities*, ...'StepSize', *StepSizeValue*, ...) specifies the steps for the shifting window. The default value is 200 separation units (baseline point is estimated for windows placed every 200 separation units).

StepSizeValue can also be a function handle. The function is evaluated at the respective separation-unit values and returns the distance between adjacent windows.

Yout = msbackadj(*X*, *Intensities*, ...'RegressionMethod', *RegressionMethodValue*, ...) specifies the method to regress the window estimated points to a soft curve. Enter 'pchip' (shape-preserving piecewise cubic interpolation), 'linear' (linear interpolation), or 'spline' (spline interpolation). The default value is 'pchip'.

Yout = msbackadj(*X*, *Intensities*, ...'EstimationMethod', *EstimationMethodValue*, ...) specifies the method for finding the likely baseline value in every window. Enter 'quantile' (quantile value is set to 10%) or 'em' (assumes a doubly stochastic model). With em, every sample is the independent and identically distributed (i.i.d.) draw of any of two normal distributed classes (background or peaks). Because the class label is hidden, the distributions are estimated with an Expectation-Maximization algorithm. The ultimate baseline value is the mean of the background class.

Yout = msbackadj(*X*, *Intensities*, ...'SmoothMethod', *SmoothMethodValue*, ...) specifies the method for smoothing the curve of estimated points and eliminating the effects of possible outliers. Enter 'none', 'lowess' (linear fit), 'loess' (quadratic fit), 'rlowess' (robust linear), or 'rloess' (robust quadratic fit). Default is 'none'.

Yout = msbackadj(*X*, *Intensities*, ...'QuantileValue', *QuantileValueValue*, ...) specifies the quantile value. The default value is 0.10.

Yout = msbackadj(*X*, *Intensities*, ...'PreserveHeights', *PreserveHeightsValue*, ...), when *PreserveHeightsValue* is true, sets the baseline subtraction mode to preserve the height of the tallest peak in the signal. The default value is false and peak heights are not preserved.

Yout = msbackadj(*X*, *Intensities*, ...'ShowPlot', *ShowPlotValue*, ...) plots the baseline-estimated points, the regressed baseline, and the original signal. When you call msbackadj without output arguments, the signal is plotted unless *ShowPlotValue* is false. When *ShowPlotValue* is true, only the first signal in *Intensities* is plotted. *ShowPlotValue* can also contain an index to one of the signals in *Intensities*.

Examples

1 Load a MAT-file, included with the Bioinformatics Toolbox software, that contains some sample data.

```
load sample_lo_res
```

2 Adjust the baseline for a group of spectra and show only the third spectrum and its estimated background.

```
YB = msbackadj(MZ_lo_res,Y_lo_res,'SHOWPLOT',3);
```
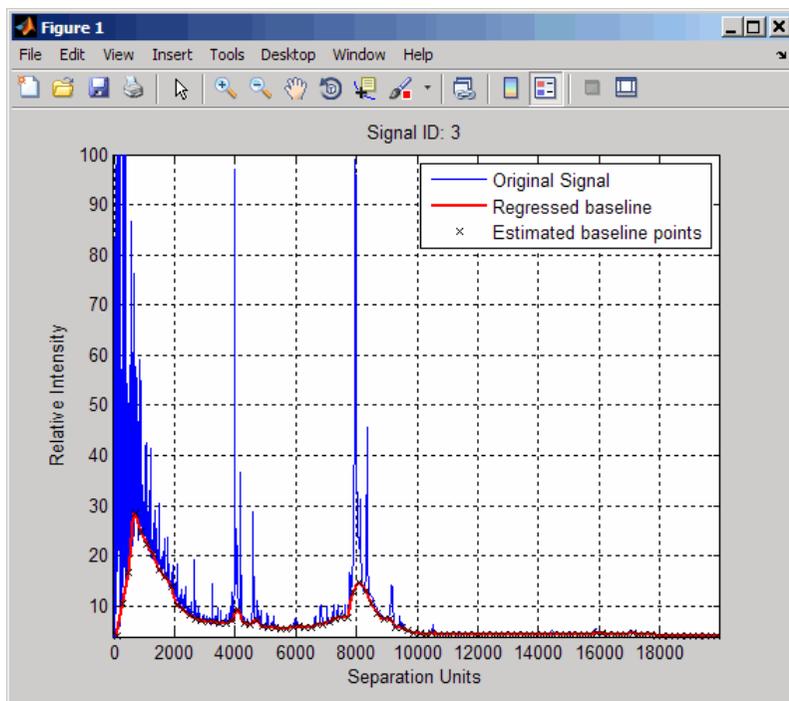

3 Plot the estimated baseline for the fourth spectrum in `Y_lo_res` using an anonymous function to describe an m/z dependent parameter.

```
wf = @(mz) 200 + .001 .* mz;
msbackadj(MZ_lo_res,Y_lo_res(:,4),'STEPSIZE',wf);
```
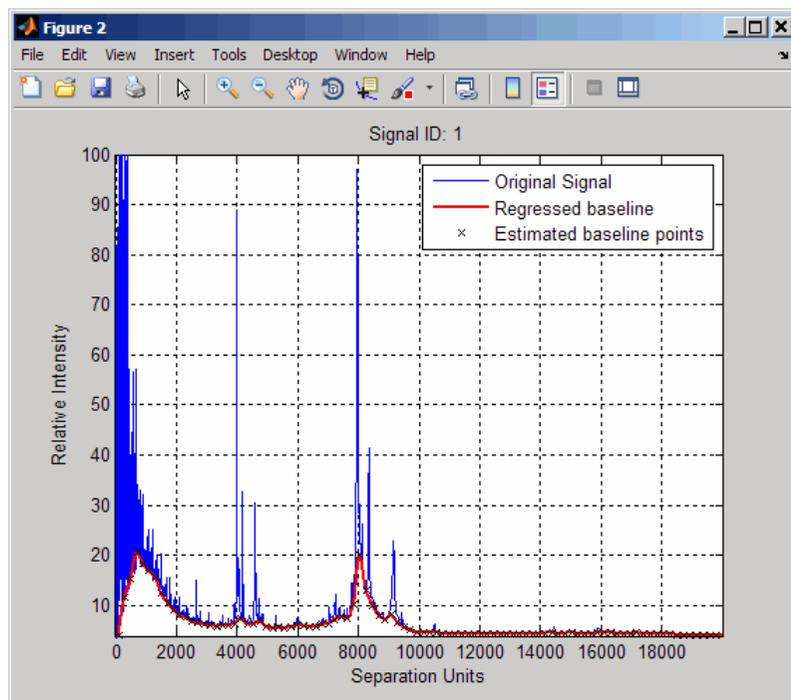
See Also

msalign | msheatmap | mslowess | msnorm | mspeaks | msresample | mssgolay | msviewer

msdotplot

Plot set of peak lists from LC/MS or GC/MS data set

Syntax

```
msdotplot(Peaklist, Times)
msdotplot(FigHandle, Peaklist, Times)
msdotplot(..., 'Quantile', QuantileValue)
PlotHandle = msdotplot(...)
```

Input Arguments

Peaklist	Cell array of peak lists, where each element is a two-column matrix with m/z values in the first column and ion intensity values in the second column. Each element corresponds to a spectrum or retention time.
	Tip You can use the mzxml2peaks function to create the *Peaklist* cell array.
Times	Vector of retention times associated with an LC/MS or GC/MS data set. The number of elements in *Times* equals the number of elements in the cell array *Peaklist*.
	Tip You can use the mzxml2peaks function to create the *Times* vector.
FigHandle	Handle to an open Figure window such as one created by the msheatmap function.
QuantileValue	Value that specifies a percentage. When peaks are ranked by intensity, only those that rank above this percentage are plotted. Choices are any value ≥ 0 and ≤ 1. Default is 0. For example, setting *QuantileValue* = 0 plots all peaks, and setting *QuantileValue* = 0.8 plots only the 20% most intense peaks.

Output Arguments

PlotHandle	Handle to the line series object (figure plot).

Description

msdotplot(*Peaklist, Times*) plots a set of peak lists from a liquid chromatography/ mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set represented by *Peaklist*, a cell array of peak lists, where each element is a two-column matrix with m/z values in the first column and ion intensity values in the second column, and *Times*, a vector of retention times associated with the spectra. *Peaklist* and *Times* have the same number of elements. The data is plotted into any existing figure generated by the msheatmap function; otherwise, the data is plotted into a new Figure window.

msdotplot(*FigHandle, Peaklist, Times*) plots the set of peak lists into the axes contained in an open Figure window with the handle *FigHandle*.

Tip This syntax is useful to overlay a dot plot on top of a heat map of mass spectrometry data created with the msheatmap function.

---

msdotplot(..., 'Quantile', *QuantileValue*) plots only the most intense peaks, specifically those in the percentage above the specified *QuantileValue*. Choices are any value ≥ 0 and ≤ 1. Default is 0. For example, setting *QuantileValue* = 0 plots all peaks, and setting *QuantileValue* = 0.8 plots only the 20% most intense peaks.

PlotHandle = msdotplot(...) returns a handle to the line series object (figure plot). You can use this handle as input to the get function to display a list of the plot's properties. You can use this handle as input to the set function to change the plot's properties, including showing and hiding points.

Examples

1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains LC/MS data variables, including peaks and ret_time. peaks is a cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity

values, and each element corresponds to a spectrum or retention time. `ret_time` is a column vector of retention times associated with the LC/MS data set.

```
load lcmsdata
```

2 Create a dot plot with only the 5% most intense peaks.

```
msdotplot(ms_peaks,ret_time,'Quantile',0.95)
```
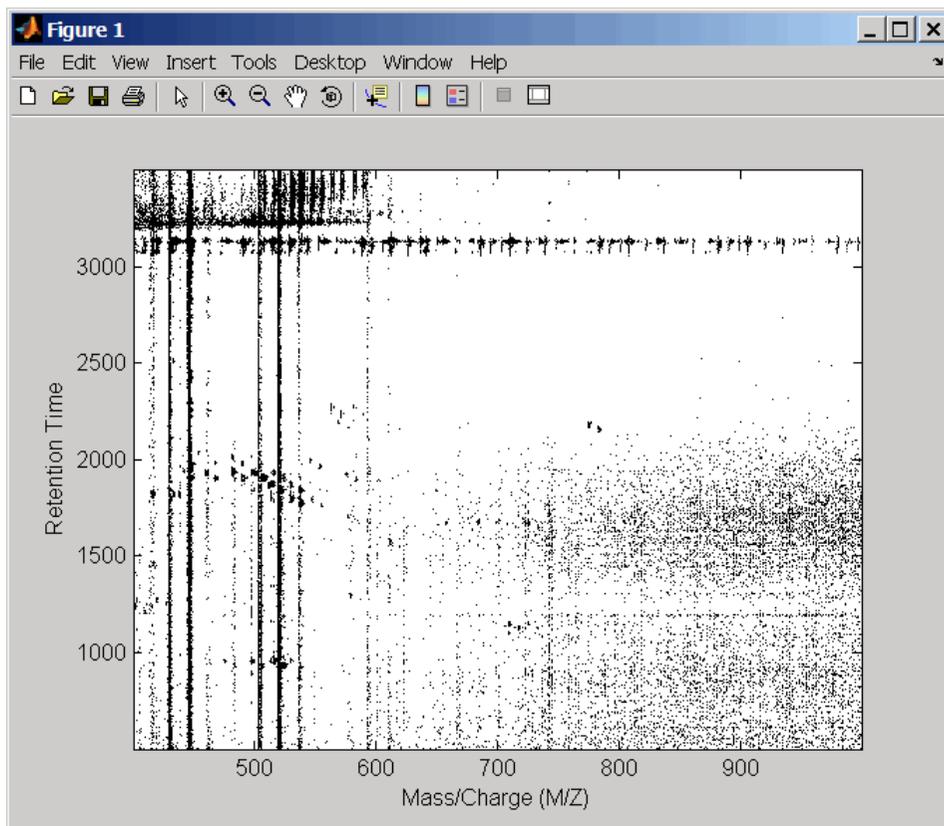

3 Resample the data, then create a heat map of the LC/MS data.

```
[MZ,Y] = msppresample(ms_peaks,5000);
msheatmap(MZ,ret_time,log(Y))
```
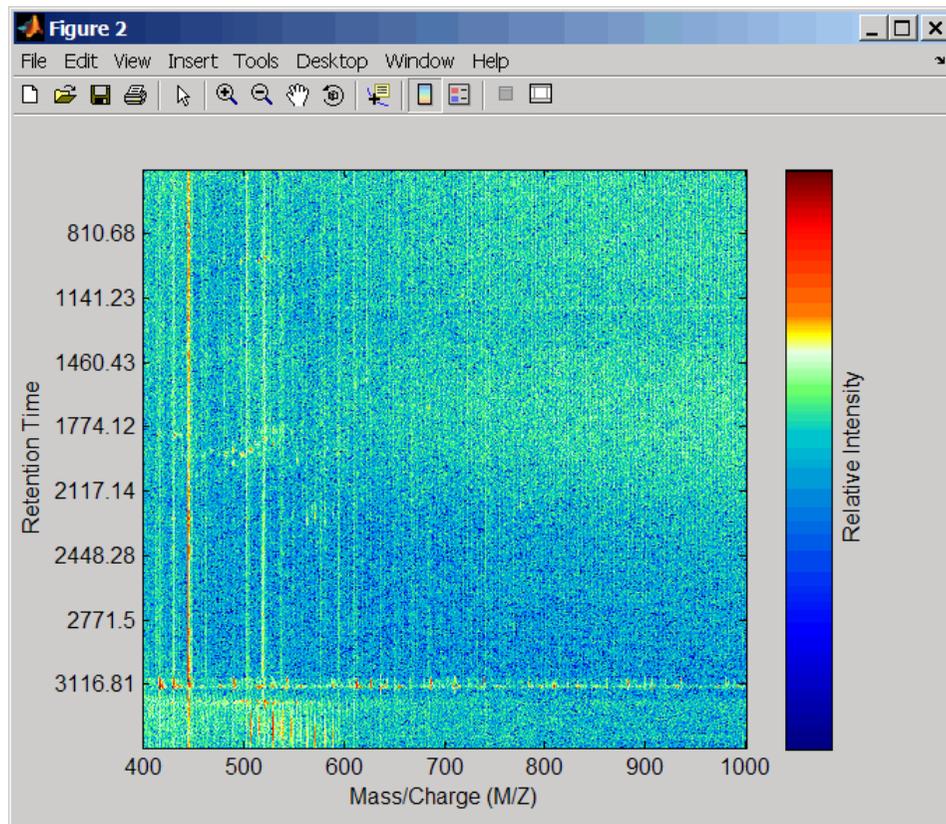
4 Overlay the dot plot on the heat map, and then zoom in to see the detail.

```
msdotplot(ms_peaks,ret_time)
axis([480 532 375 485])
```
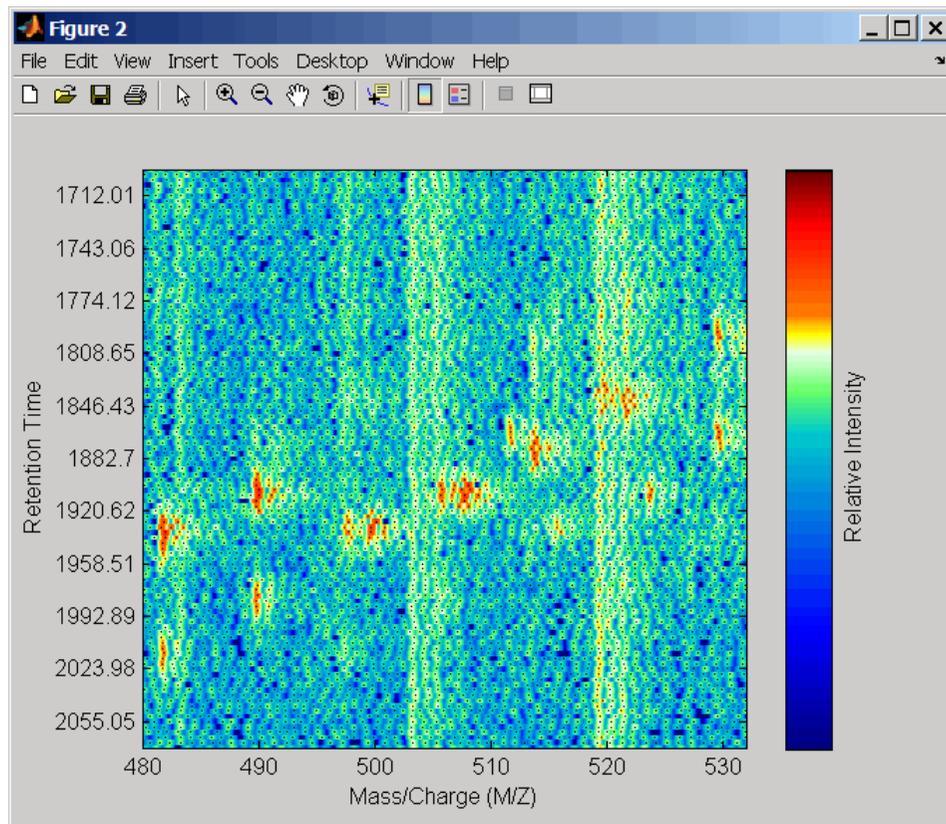
See Also
msheatmap | mspalign | mspeaks | msppresample | mzcdf2peaks | mzcdfread | mzxml2peaks | mzxmlread

msheatmap

Create pseudocolor image of set of mass spectra

Syntax

```
msheatmap(MZ, Intensities)
msheatmap(MZ, Times, Intensities)
msheatmap(..., 'Midpoint', MidpointValue, ...)
msheatmap(..., 'Range', RangeValue, ...)
msheatmap(..., 'Markers', MarkersValue, ...)
msheatmap(..., 'SpecIdx', SpecIdxValue, ...)
msheatmap(..., 'Group', GroupValue, ...)
msheatmap(..., 'Resolution', ResolutionValue, ...)
```

Arguments

MZ	Column vector of common mass/charge (m/z) values for a set of spectra. The number of elements in the vector equals the number of rows in the matrix *Intensities*.
	Note: You can use the msppresample function to create the *MZ* vector.
Times	Column vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. The number of elements in the vector equals the number of columns in the matrix *Intensities*. The retention times are used to label the *y*-axis of the heat map.
	Tip You can use the mzxml2peaks function to create the *Times* vector.
Intensities	Matrix of intensity values for a set of mass spectra that share the same m/z range. Each row corresponds to an m/z value, and

	each column corresponds to a spectrum or retention time. The number of rows equals the number of elements in vector *MZ*. The number of columns equals the number of elements in vector *Times*.
	Note: You can use the `msppresample` function to create the *Intensities* matrix.
MidpointValue	Value specifying a quantile of the ion intensity values to fall below the midpoint of the colormap, meaning they do not represent peaks. `msheatmap` uses a custom colormap where cool colors represent nonpeak regions, white represents the midpoint, and warm colors represent peaks. Choices are any value ≥ `0` and ≤ `1`. Default is: • `0.99` — For LC/MS or GC/MS data or when input *T* is provided. This means that 1% of the pixels are warm colors and represent peaks. • `0.95` — For non-LC/MS or non-GC/MS data or when input *T* is not provided. This means that 5% of the pixels are warm colors and represent peaks. **Tip** You can also change the midpoint interactively after creating the heat map by right-clicking the color bar, selecting **Interactive Colormap Shift**, and then click-dragging the cursor vertically on the color bar. This technique is useful when comparing multiple heat maps.
RangeValue	1-by-2 vector specifying the m/z range for the *x*-axis of the heat map. *RangeValue* must be within `[min(MZ) max(MZ)]`. Default is the full range `[min(MZ) max(MZ)]`.
MarkersValue	Vector of m/z values to mark on the top horizontal axis of the heat map. Default is `[]`.

SpecIdxValue	Either of the following:
	• Vector of values with the same number of elements as columns (spectra) in the matrix *Intensities*.
	• Cell array of strings with the same number of elements as columns (spectra) in the matrix *Intensities*.
	Each value or string specifies a label for the corresponding spectrum. These values or strings are used to label the *y*-axis of the heat map.
	Note: If input *Times* is provided, it is assumed that *Intensities* contains LC/MS or GC/MS data, and *SpecIdxValue* is ignored.
GroupValue	Either of the following:
	• Vector of values with the same number of elements as rows in the matrix *Intensities*
	• Cell array of strings with the same number of elements as rows (spectra) in the matrix *Intensities*
	Each value or string specifies a group to which the corresponding spectrum belongs. The spectra are sorted and combined into groups along the *y*-axis in the heat map.
	Note: If input *Times* is provided, it is assumed that *Intensities* contains LC/MS or GC/MS data, and *GroupValue* is ignored.
ResolutionValue	Value specifying the horizontal resolution of the heat map image. Increase this value to enhance details. Decrease this value to reduce memory usage. Default is:
	• 0.5 — When *MZ* contains > 2,500 elements.
	• 0.05 — When *MZ* contains <= 2,500 elements.

Description

msheatmap(*MZ*, *Intensities*) displays a pseudocolor heat map image of the intensities for the spectra in matrix *Intensities*.

msheatmap(*MZ*, *Times*, *Intensities*) displays a pseudocolor heat map image of the intensities for the spectra in matrix *Intensities*, using the retention times in vector *Times* to label the *y*-axis.

msheatmap(..., '*PropertyName*', *PropertyValue*, ...) calls msheatmap with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

msheatmap(..., 'Midpoint', *MidpointValue*, ...) specifies a quantile of the ion intensity values to fall below the midpoint of the colormap, meaning they do not represent peaks. msheatmap uses a custom colormap where cool colors represent nonpeak regions, white represents the midpoint, and warm colors represent peaks. Choices are any value between 0 and 1. Default is:

- 0.99 — For LC/MS or GC/MS data or when input *T* is provided. This means that 1% of the pixels are warm colors and represent peaks.

- 0.95 — For non-LC/MS or non-GC/MS data or when input *T* is not provided. This means that 5% of the pixels are warm colors and represent peaks.

Tip You can also change the midpoint interactively after creating the heat map by right-clicking the color bar, selecting **Interactive Colormap Shift**, then click-dragging the cursor vertically on the color bar. This technique is useful when comparing multiple heat maps.

---

msheatmap(..., 'Range', *RangeValue*, ...) specifies the m/z range for the *x*-axis of the heat map. *RangeValue* is a 1-by-2 vector that must be within [min(*MZ*) max(*MZ*)]. Default is the full range [min(*MZ*) max(*MZ*)].

msheatmap(..., 'Markers', *MarkersValue*, ...) places markers along the top horizontal axis of the heat map for the m/z values specified in the vector *MarkersValue*. Default is [].

msheatmap(..., 'SpecIdx', *SpecIdxValue*, ...) labels the spectra along the *y*-axis in the heat map. The labels are specified by *SpecIdxValue*, a vector of values or cell array of strings. The number of values or strings is the same as the number of columns (spectra) in the matrix *Intensities*. Each value or string specifies a label for the corresponding spectrum.

msheatmap(..., 'Group', *GroupValue*, ...) sorts and combines spectra into groups along the *y*-axis in the heat map. The groups are specified by *GroupValue*, a vector of values or cell array of strings. The number of values or strings is the same as the number of rows in the matrix *Intensities*. Each value or string specifies a group to which the corresponding spectrum belongs. Default is [1:numSpectra].

msheatmap(..., 'Resolution', *ResolutionValue*, ...) specifies the horizontal resolution of the heat map image. Increase this value to enhance details. Decrease this value to reduce memory usage. Default is:

- 0.5 — When *MZ* contains > 2,500 elements.
- 0.05 — When *MZ* contains <= 2,500 elements.

Examples

SELDI-TOF Data

1   Load SELDI-TOF sample data.

    load sample_lo_res

2   Create a vector of four m/z values to mark along the top horizontal axis of the heat map.

    M = [3991.4 4598 7964 9160];

3   Display the heat map with m/z markers and a limited m/z range.

    msheatmap(MZ_lo_res,Y_lo_res,'markers',M,'range',[3000 10000])
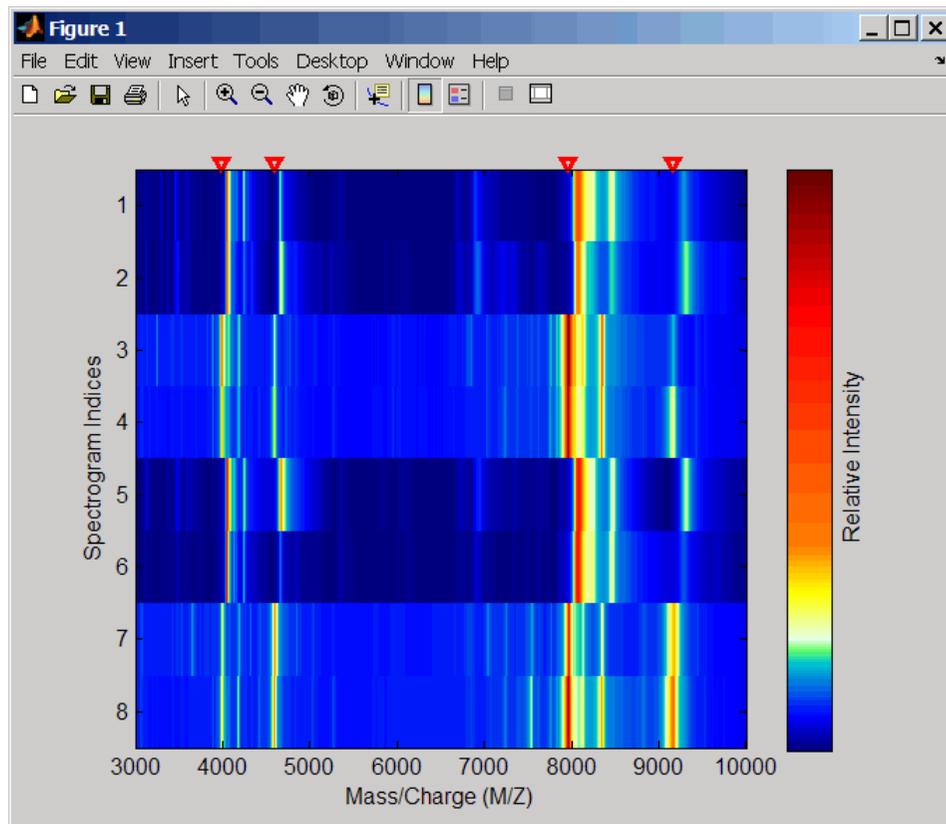
4 Display the heat map again grouping each spectrum into one of two groups.

```
TwoGroups = [1 1 2 2 1 1 2 2];
msheatmap(MZ_lo_res,Y_lo_res,'markers',M,'group',TwoGroups)
```
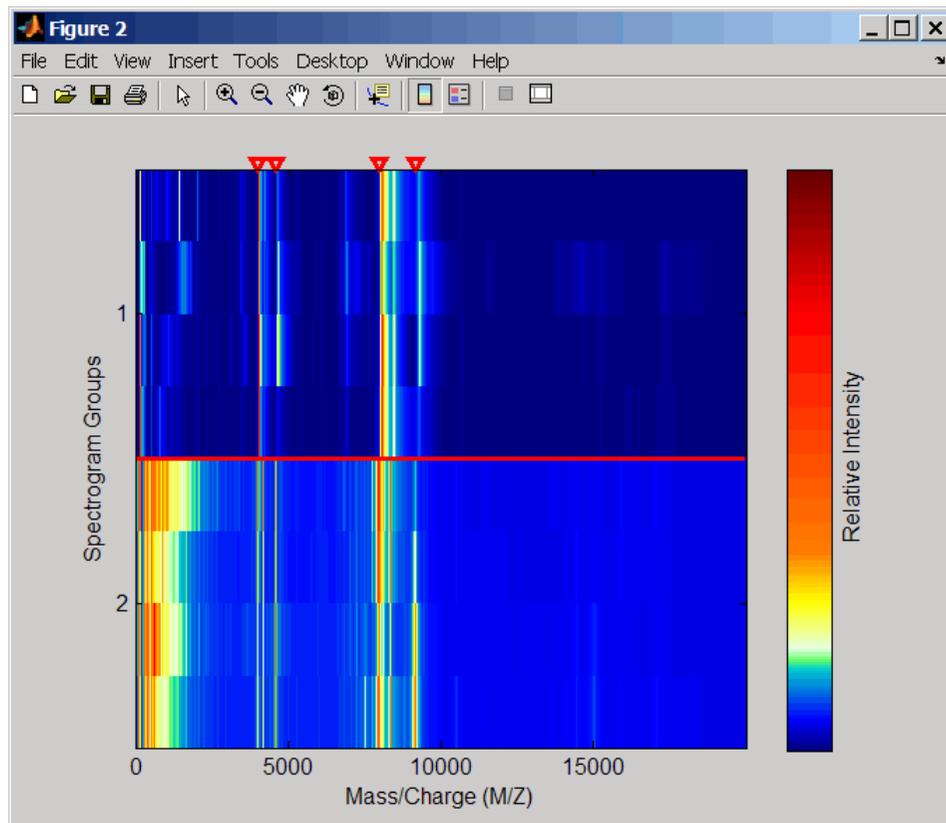
Liquid Chromatography/Mass Spectrometry (LC/MS) Data

1 Load LC/MS sample data.

```
load lcmsdata
```

2 Resample the peak lists to create a vector of m/z values and a matrix of intensity values.

```
[MZ, Intensities] = msppresample(ms_peaks, 5000);
```

3 Display the heat map showing mass spectra at different retention times.

```
msheatmap(MZ, ret_time, log(Intensities))
```
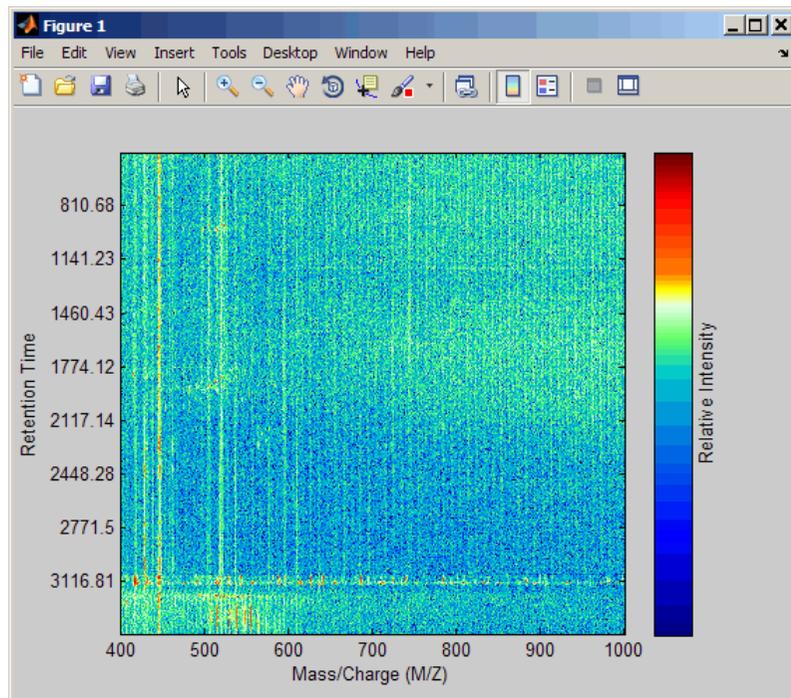
See Also
msalign | msbackadj | msdotplot | mslowess | msnorm | mspalign | msresample | mssgolay | msviewer

mslowess

Smooth signal with peaks using nonparametric method

Syntax

```
Yout = mslowess(X, Intensities)
mslowess(..., 'Order', OrderValue, ...)
mslowess(..., 'Span', SpanValue, ...)
mslowess(..., 'Kernel', KernelValue, ...)
mslowess(..., 'RobustIterations', RobustIterationsValue, ...)
mslowess(..., 'ShowPlot', ShowPlotValue, ...)
```

Arguments

X	Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix *Intensities*. The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.
Intensities	Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector *X*.

Description

Tip Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

Yout = mslowess(*X*, *Intensities*) smooths raw noisy signal data, *Intensities*, using a locally weighted linear regression (Lowess) method with a default span of 10 samples.

Note: mslowess assumes the input vector, *X*, may not have uniformly spaced separation units. Therefore, the sliding window for smoothing is centered using the closest samples in terms of the *X* value and not in terms of the *X* index.

Note: When the input vector, *X*, does not have repeated values or NaN values, the algorithm is approximately twice as fast.

---

mslowess(*X*, *Intensities*, ...'*PropertyName*', *PropertyValue*, ...) calls mslowess with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

mslowess(..., 'Order', *OrderValue*, ...) specifies the order (*OrderValue*) of the Lowess smoother. Enter 1 (linear polynomial fit or Lowess), 2 (quadratic polynomial fit or Loess), or 0 (equivalent to a weighted local mean estimator and presumably faster because only a mean computation is performed instead of a least-squares regression). The default value is 1.

Note: Curve Fitting Toolbox software also refers to Lowess smoothing of order 2 as Loess smoothing.

---

mslowess(..., 'Span', *SpanValue*, ...) specifies the window size for the smoothing kernel. If *SpanValue* is greater than 1, the window is equal to *SpanValue* number of samples independent of the separation-unit vector, *X*. The default value is 10 samples. Higher values will smooth the signal more at the expense of computation time. If *SpanValue* is less than 1, the window size is taken to be a fraction of the number of points in the data. For example, when *SpanValue* is 0.005, the window size is equal to 0.50% of the number of points in *X*.

mslowess(..., 'Kernel', *KernelValue*, ...) selects the function specified by *KernelValue* for weighting the observed intensities. Samples close to the separation-

unit location being smoothed have the most weight in determining the estimate. *KernelValue* can be any of the following strings:

- `'tricubic'` (default) — `(1 - (dist/dmax).^3).^3`
- `'gaussian'` — `exp(-(2*dist/dmax).^2)`
- `'linear'` — `1-dist/dmax`

`mslowess(..., 'RobustIterations', ` *RobustIterationsValue*`, ...)` specifies the number of iterations (*RobustValue*) for a robust fit. If *RobustIterationsValue* is `0` (default), no robust fit is performed. For robust smoothing, small residual values at every span are outweighed to improve the new estimate. `1` or `2` robust iterations are usually adequate, while larger values might be computationally expensive.

Note: For an *X* vector that has uniformly spaced separation units, a nonrobust smoothing with *OrderValue* equal to `0` is equivalent to filtering the signal with the kernel vector.

`mslowess(..., 'ShowPlot', ` *ShowPlotValue*`, ...)` plots the smoothed signal over the original signal. When you call `mslowess` without output arguments, the signals are plotted unless *ShowPlotValue* is `false`. When *ShowPlotValue* is `true`, only the first signal in *Intensities* is plotted. *ShowPlotValue* can also contain an index to one of the signals in *Intensities*.

Examples

1 Load a MAT-file, included with the Bioinformatics Toolbox software, that contains some sample data.

   ```
   load sample_lo_res
   ```

2 Smooth the spectra and draw a figure of the first spectrum with original and smoothed signals.

   ```
   YS = mslowess(MZ_lo_res,Y_lo_res,'Showplot',true);
   ```
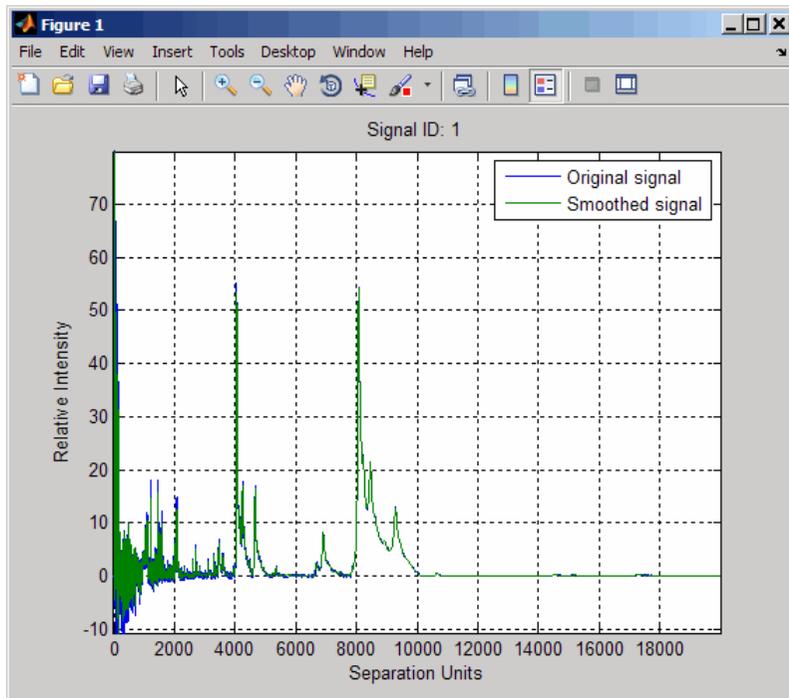
3 Zoom in on a region of the figure to see the difference in the original and smoothed signals.

```
axis([7350 7550 0.1 1.0])
```
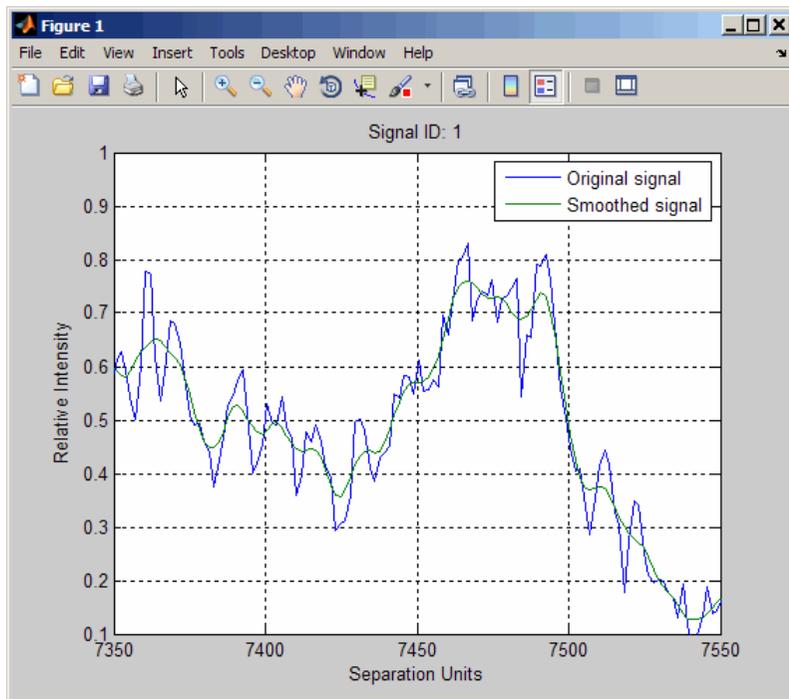
More About

- Preprocessing Raw Mass Spectrometry Data

See Also

msalign | msbackadj | msheatmap | msnorm | mspeaks | msresample | mssgolay | msviewer

msnorm

Normalize set of signals with peaks

Syntax

```
Yout = msnorm(X, Intensities)
[Yout, NormParameters] = msnorm(...)
msnorm(X, NewY, NormParameters)
msnorm(..., 'Quantile', QuantileValue, ...)
msnorm(..., 'Limits', LimitsValue, ...)
msnorm(..., 'Consensus', ConsensusValue, ...)
msnorm(..., 'Method', MethodValue, ...)
msnorm(..., 'Max', MaxValue, ...)
```

Arguments

X	Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix *Intensities*. The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.
Intensities	Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector *X*.

Description

Tip Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

Yout = msnorm(*X*, *Intensities*) normalizes a group of signals with peaks by standardizing the area under the curve (AUC) to the group median.

[*Yout*, *NormParameters*] = msnorm(...) returns a structure containing the parameters to normalize another group of signals.

msnorm(*X*, *NewY*, *NormParameters*) uses the parameter information from a previous normalization specified by *NormParameters* to normalize a new set of signals specified by *NewY* using the same parameters to select the separation-unit positions and output scale from the previous normalization. *NormParameters* is a structure created by msnorm. If a consensus proportion, *ConsensusValue*, was given in the previous normalization, no new separation-unit positions are selected, and normalization is performed using the same separation-unit positions.

msnorm(..., '*PropertyName*', *PropertyValue*, ...) calls msnorm with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

msnorm(..., 'Quantile', *QuantileValue*, ...) specifies a 1-by-2 vector with the quantile limits for reducing the set of separation-unit values in *X*. For example, when *QuantileValue* is [0.9 1], only the largest 10% of intensities in each signal are used to compute the AUC. When *QuantileValue* is a scalar, the scalar value represents the lower quantile limit and the upper quantile limit is set to 1. The default value is [0 1] (use the whole area under the curve, AUC).

msnorm(..., 'Limits', *LimitsValue*, ...) specifies a 1-by-2 vector with a separation-unit range for picking normalization points. This parameter is useful to eliminate low-mass noise from the AUC calculation, for example the matrix noise that appears in the low-mass region of SELDI mass spectrometers. Default is [0, max(*X*)].

msnorm(..., 'Consensus', *ConsensusValue*, ...) sets a consensus rule. To be included in the AUC, a separation-unit position must have an intensity within the quantile limits of at least part (specified by *ConsensusValue*) of the signals in *Intensities*. The same separation-unit positions are used to normalize all the signals. Enter a scalar from 0 to 1.

Tip Use the 'Consensus' property to eliminate low-intensity peaks and noise from the normalization.

---

msnorm(..., 'Method', *MethodValue*, ...) selects a method for normalizing the AUC of every signal. Enter either 'Median' (default) or 'Mean'.

msnorm(..., 'Max', *MaxValue*, ...), after individually normalizing each signal, scales each signal to an overall maximum intensity specified by *MaxValue*. *MaxValue* is a scalar. If omitted, no postscaling is performed. If *QuantileValue* is [1 1], then a single point (peak height of the tallest peak) is normalized to *MaxValue*.

Examples

AUC Normalization

This example shows how to normalize the area under the curve of every mass spectrum from the mass spec data.

Load a MAT-file, included with the Bioinformatics Toolbox™ software, that contains sample mass spec data, including MZ_lo_res, a vector of m/z values, and Y_lo_res, a matrix of intensity values.

```
load sample_lo_res
```

Create a subset (four signals) of the data.

```
MZ = MZ_lo_res;
Y = Y_lo_res(:,[1 2 5 6]);
```

Plot the four spectra.

```
plot(MZ, Y)
axis([-1000 20000 -20 105])
xlabel('Mass-charge Ratio')
ylabel('Relative Ion Intensities')
title('Original Spectra')
```
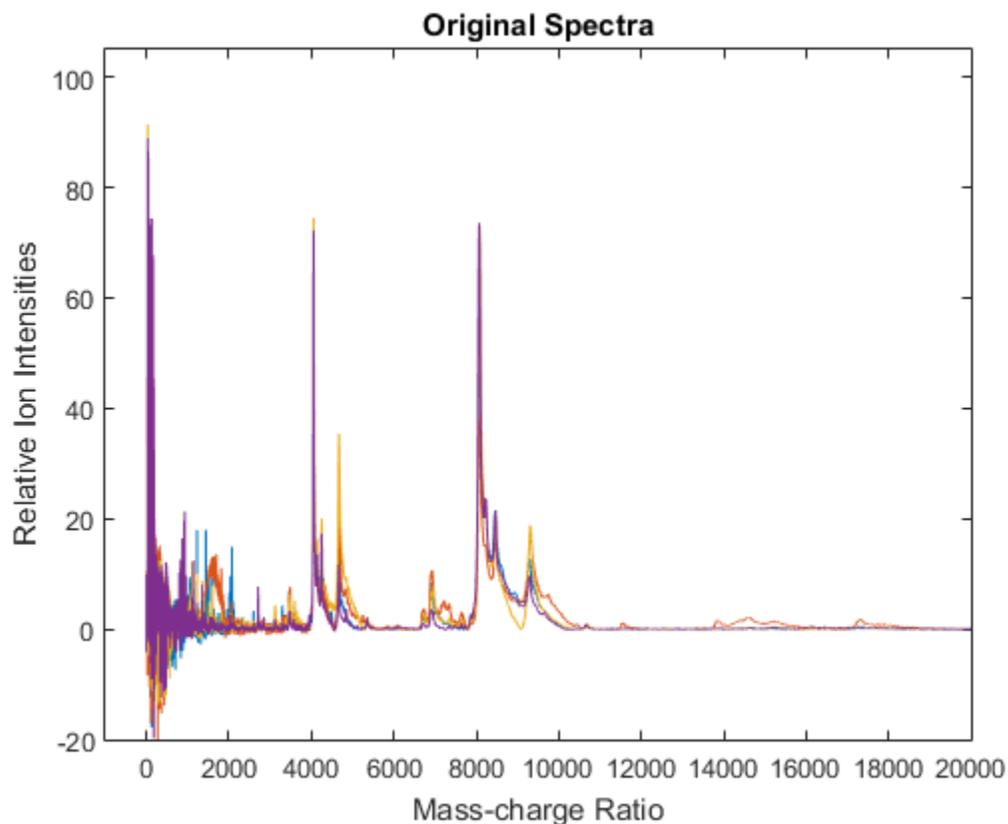
Normalize the area under the curve (AUC) of every spectrum to the median, eliminating low-mass (m/z < 1,000) noise, and post-rescaling such that the maximum intensity is 100. Plot the four spectra.

```
Y1 = msnorm(MZ,Y,'Limits',[1000 inf],'Max',100);
plot(MZ, Y1)
axis([-1000 20000 -20 105])
xlabel('Mass-charge Ratio')
ylabel('Relative Ion Intensities')
title('AUC Normalized Spectra')
```

AUC Normalized Spectra
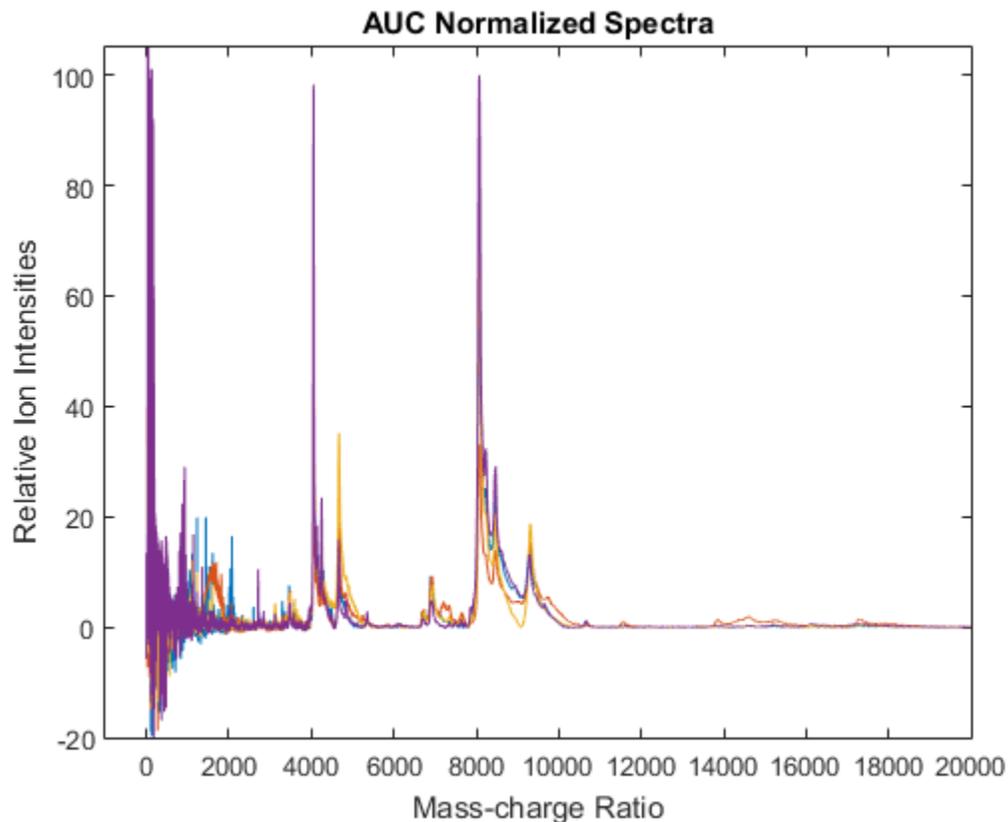
Maximum Intensity Normalization

This example shows how to normalize the ion intensity of every spectrum from the mass spec data.

Load a MAT-file, included with the Bioinformatics Toolbox™ software, that contains sample mass spec data, including MZ_lo_res, a vector of m/z values, and Y_lo_res, a matrix of intensity values.

```
load sample_lo_res
```

Create a subset (four signals) of the data.

```
MZ = MZ_lo_res;
```

```
Y = Y_lo_res(:,[1 2 5 6]);
```

Normalize the ion intensity of every spectrum to the maximum intensity of the single highest peak from any of the spectra in the range above 1000 m/z. Plot the four spectra.

```
Y2 = msnorm(MZ,Y,'QUANTILE', [1 1],'LIMITS',[1000 inf]);
plot(MZ, Y2)
axis([-1000 20000 -20 105])
xlabel('Mass-charge Ratio')
ylabel('Relative Ion Intensities')
title('Maximum-Intensity Normalized Spectra')
```
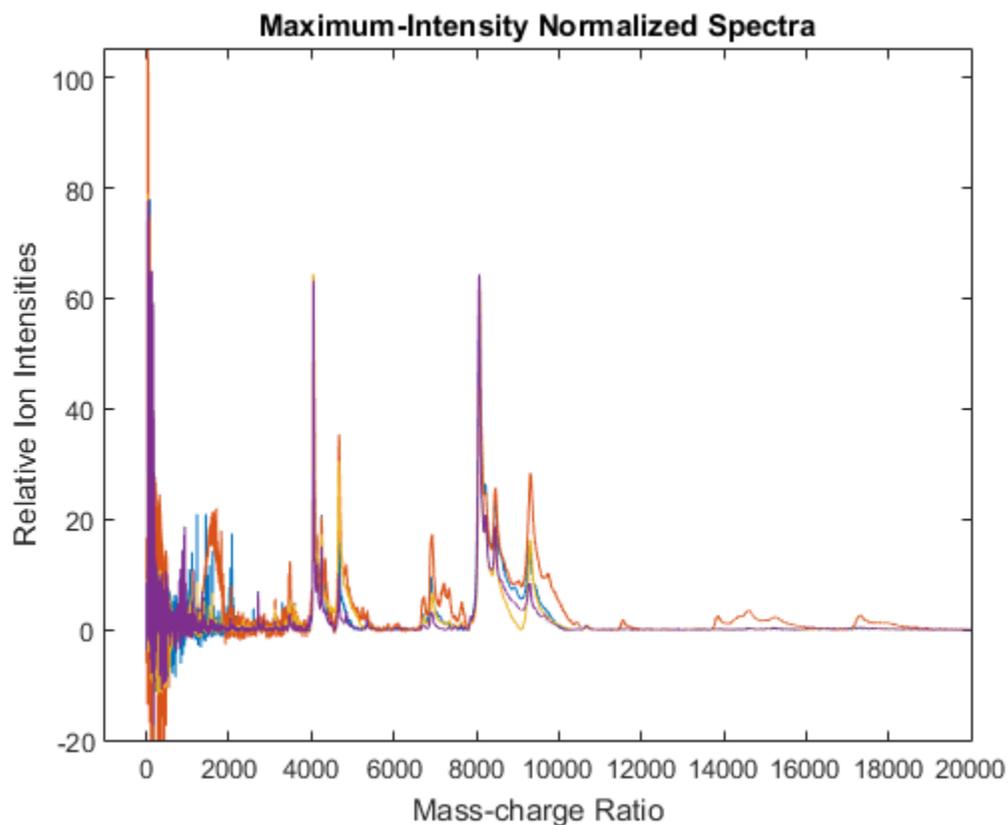

Quantile Normalization

This example shows how to perform quantile normalization for mass spec data.

Load a MAT-file, included with the Bioinformatics Toolbox™ software, that contains sample mass spec data, including MZ_lo_res, a vector of m/z values, and Y_lo_res, a matrix of intensity values.

```
load sample_lo_res
```

Create a subset (four signals) of the data.

```
MZ = MZ_lo_res;
Y = Y_lo_res(:,[1 2 5 6]);
```

Normalize using the data in the m/z regions where the intensities are within the fourth quartile in at least 90% of the spectrograms. Note that you can use the normalization parameters in the second output to normalize another set of data in the same m/z regions. Plot the four spectra.

```
[Y3,S] = msnorm(MZ,Y,'Quantile',[0.75 1],'Consensus',0.9);
area(MZ,S.Xh.*1000,'LineStyle','None','FaceColor',[.8 .8 .8])
hold on
plot(MZ, Y3)
hold off
axis([-1000 20000 -20 105])
xlabel('Mass-charge Ratio')
ylabel('Relative Ion Intensities')
title('Fourth-quartile Normalized Spectra')
```

Fourth-quartile Normalized Spectra
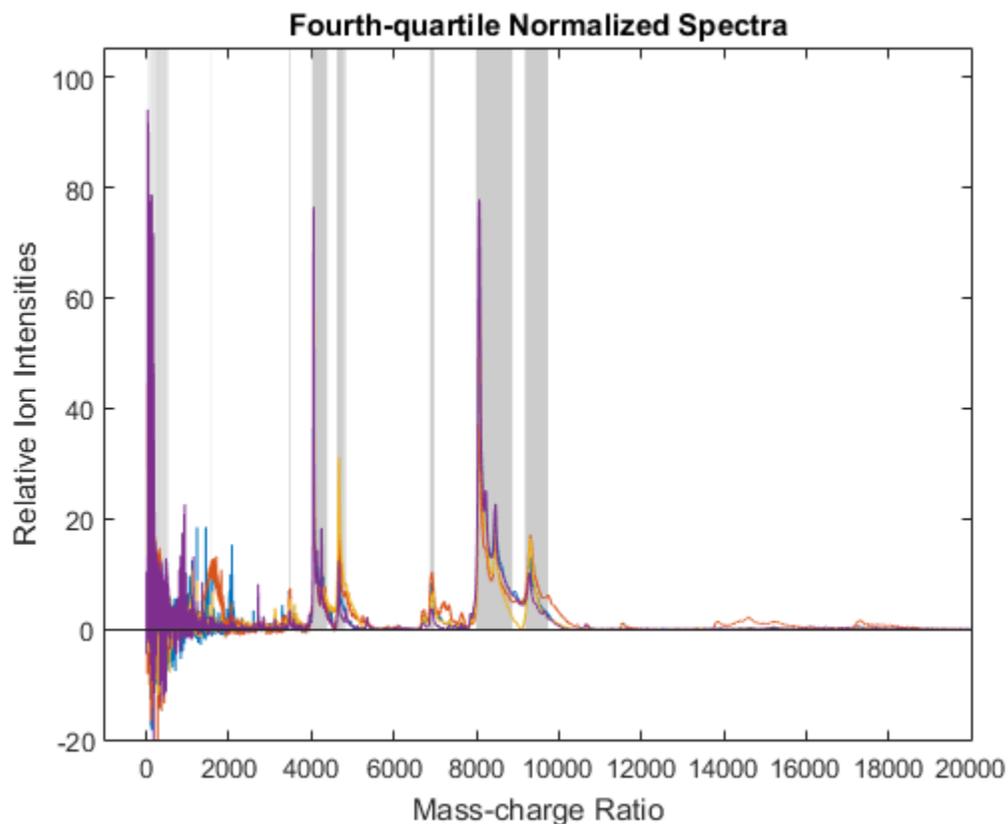
Use the normalization parameters in the second output of the previous step to normalize a different subset of data (four signals) using the data in the same m/z regions as the previous data set. Plot the four spectra.

```
Y4 = msnorm(MZ,Y_lo_res(:,[3 4 7 8]),S);

area(MZ,S.Xh.*1000,'LineStyle','None','FaceColor',[.8 .8 .8])
hold on
plot(MZ, Y4)
hold off
axis([-1000 20000 -20 105])
xlabel('Mass-charge Ratio')
ylabel('Relative Ion Intensities')
title('Fourth-quartile Normalized Spectra')
```

Fourth-quartile Normalized Spectra
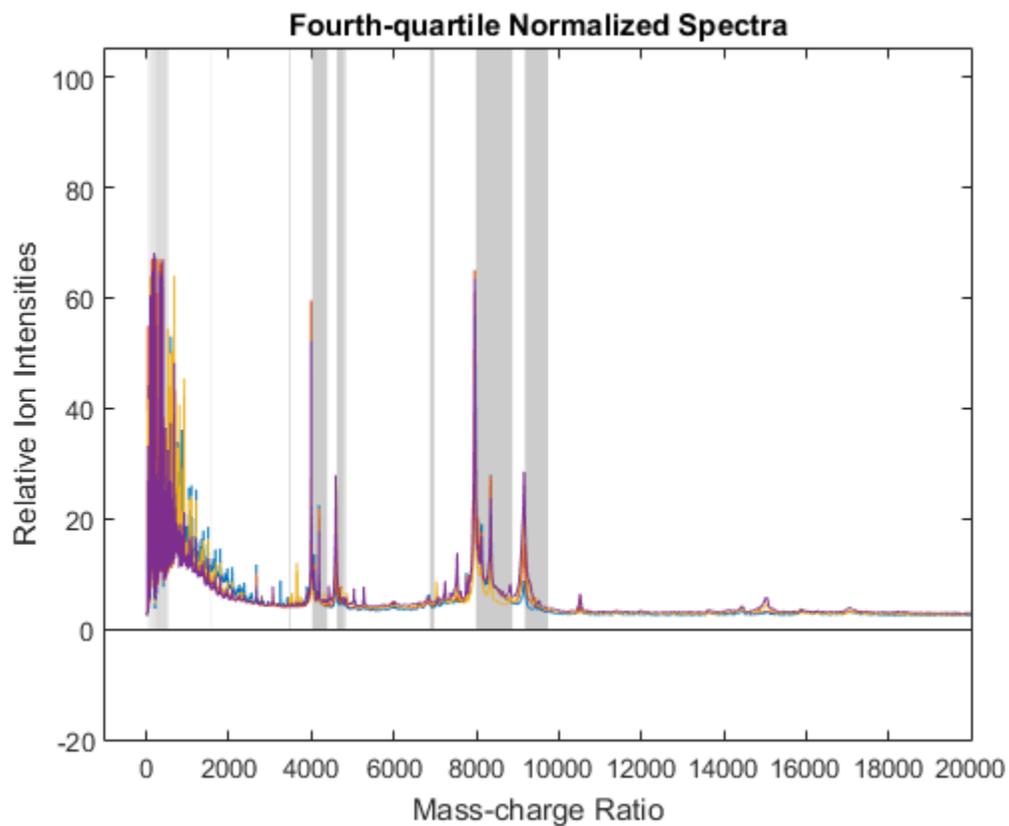
More About

· Preprocessing Raw Mass Spectrometry Data

See Also

`msalign` | `msbackadj` | `msheatmap` | `mslowess` | `msresample` | `mssgolay` | `msviewer`

mspalign

Align mass spectra from multiple peak lists from LC/MS or GC/MS data set

Syntax

```
[CMZ, AlignedPeaks] = mspalign(Peaklist)
[CMZ, AlignedPeaks] = mspalign(Peaklist, ...'Quantile',
QuantileValue, ...)
[CMZ, AlignedPeaks] = mspalign(Peaklist, ...'EstimationMethod',
EstimationMethodValue, ...)
[CMZ, AlignedPeaks] = mspalign(Peaklist, ...'CorrectionMethod',
CorrectionMethodValue, ...)
[CMZ, AlignedPeaks] = mspalign(Peaklist, ...'ShowEstimation',
ShowEstimationValue, ...)
```

Input Arguments

Peaklist	Cell array of peak lists from a liquid chromatography/ mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. Each element in the cell array is a two-column matrix with m/z values in the first column and ion intensity values in the second column. Each element corresponds to a spectrum or retention time. **Note:** You can use the mzxml2peaks function or the mspeaks function to create the *Peaklist* cell array.
QuantileValue	Value that determines which peaks are selected by the estimation method to create *CMZ*, the vector of common m/z values. Choices are any value ≥ 0 and ≤ 1. Default is 0.95.
EstimationMethodValue	String specifying the method to estimate *CMZ*, the vector of common mass/charge (m/z) values. Choices are:

	• `histogram` — Default method. Peak locations are clustered using a kernel density estimation approach. The peak ion intensity is used as a weighting factor. The center of all the clusters conform to the *CMZ* vector.
	• `regression` — Takes a sample of the distances between observed significant peaks and regresses the inter-peak distance to create the *CMZ* vector with similar inter-element distances.
CorrectionMethodValue	String specifying the method to align each peak list to the *CMZ* vector. Choices are:
	• `nearest-neighbor` — Default method. For each common peak in the *CMZ* vector, its counterpart in each peak list is the peak that is closest to the common peak's m/z value.
	• `shortest-path` — For each common peak in the *CMZ* vector, its counterpart in each peak list is selected using the shortest path algorithm.
ShowEstimationValue	Controls the display of an assessment plot relative to the estimation method and the vector of common mass/charge (m/z) values. Choices are `true` or `false`. Default is either:
	• `false` — When return values are specified.
	• `true` — When return values are not specified.

Output Arguments

CMZ	Vector of common mass/charge (m/z) values estimated by the `mspalign` function.
AlignedPeaks	Cell array of peak lists, with the same form as *Peaklist*, but with corrected m/z values in the first column of each matrix.

Description

[*CMZ*, *AlignedPeaks*] = mspalign(*Peaklist*) aligns mass spectra from multiple peak lists (centroided data), by first estimating *CMZ*, a vector of common mass/charge (m/z) values estimated by considering the peaks in all spectra in *Peaklist*, a cell array of peak lists, where each element corresponds to a spectrum or retention time. It then aligns the peaks in each spectrum to the values in *CMZ*, creating *AlignedPeaks*, a cell array of aligned peak lists.

[*CMZ*, *AlignedPeaks*] = mspalign(*Peaklist*, ...'*PropertyName*', *PropertyValue*, ...) calls mspalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

[*CMZ*, *AlignedPeaks*] = mspalign(*Peaklist*, ...'Quantile', *QuantileValue*, ...) determines which peaks are selected by the estimation method to create *CMZ*, the vector of common m/z values. Choices are a scalar between 0 and 1. Default is 0.95.

[*CMZ*, *AlignedPeaks*] = mspalign(*Peaklist*, ...'EstimationMethod', *EstimationMethodValue*, ...) specifies the method used to estimate *CMZ*, the vector of common mass/charge (m/z) values. Choices are:

- histogram — Default method. Peak locations are clustered using a kernel density estimation approach. The peak ion intensity is used as a weighting factor. The center of all the clusters conform to the *CMZ* vector.

- regression — Takes a sample of the distances between observed significant peaks and regresses the inter-peak distance to create the *CMZ* vector with similar inter-element distances.

[*CMZ*, *AlignedPeaks*] = mspalign(*Peaklist*, ...'CorrectionMethod', *CorrectionMethodValue*, ...) specifies the method used to align each peak list to the *CMZ* vector. Choices are:

- nearest-neighbor — Default method. For each common peak in the *CMZ* vector, its counterpart in each peak list is the peak that is closest to the common peak's m/z value.

- shortest-path — For each common peak in the *CMZ* vector, its counterpart in each peak list is selected using the shortest path algorithm.

[*CMZ*, *AlignedPeaks*] = mspalign(*Peaklist*, ...'ShowEstimation', *ShowEstimationValue*, ...) controls the display of an assessment plot relative to the estimation method and the estimated vector of common mass/charge (m/z) values. Choices are true or false. Default is either:

- false — When return values are specified.
- true — When return values are not specified.

Examples

1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains liquid chromatography/mass spectrometry (LC/MS) data variables, including peaks and ret_time. peaks is a cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time. ret_time is a column vector of retention times associated with the LC/MS data set.

```
load lcmsdata
```

2 Resample the unaligned data, display it in a heat map, and then overlay a dot plot.

```
[MZ,Y] = msppresample(ms_peaks,5000);
msheatmap(MZ,ret_time,log(Y))
```
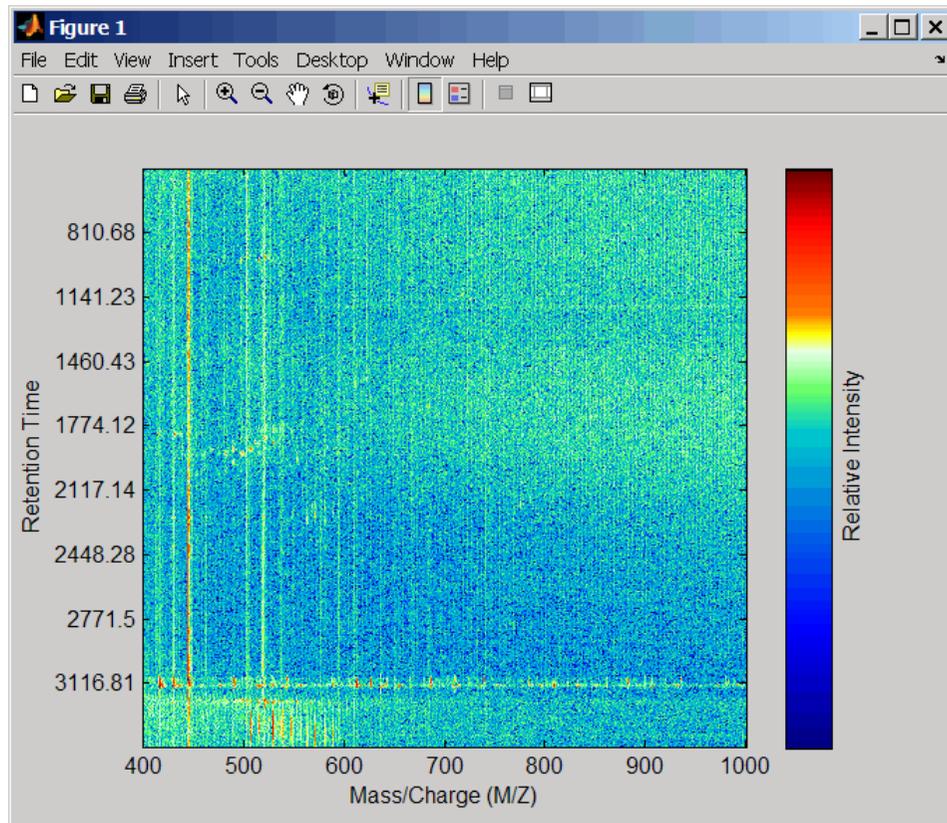
```
msdotplot(ms_peaks,ret_time)
```

3

Click the Zoom In ![zoom icon] button, and then click the dot plot two or three times to zoom in and see how the dots representing peaks overlay the heat map image.
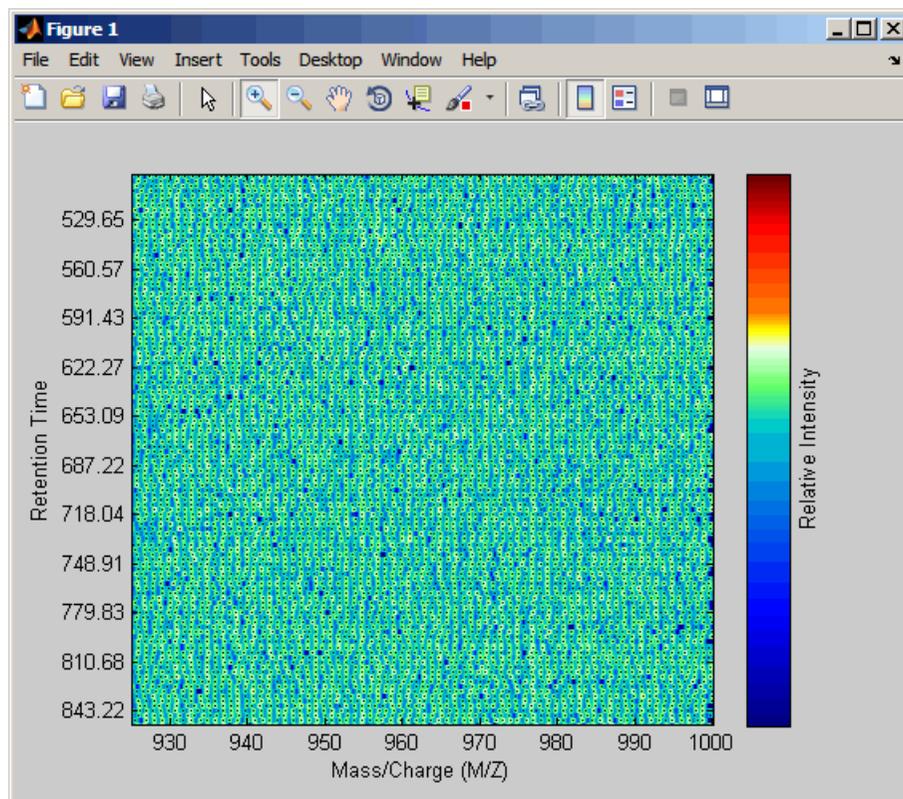
4. Align the peak lists from the mass spectra using the default estimation and correction methods.

```
[CMZ, aligned_peaks] = mspalign(ms_peaks);
```

5. Resample the unaligned data, display it in a heat map, and then overlay a dot plot.

```
[MZ2,Y2] = msppresample(aligned_peaks,5000);
msheatmap(MZ2,ret_time,log(Y2))
```
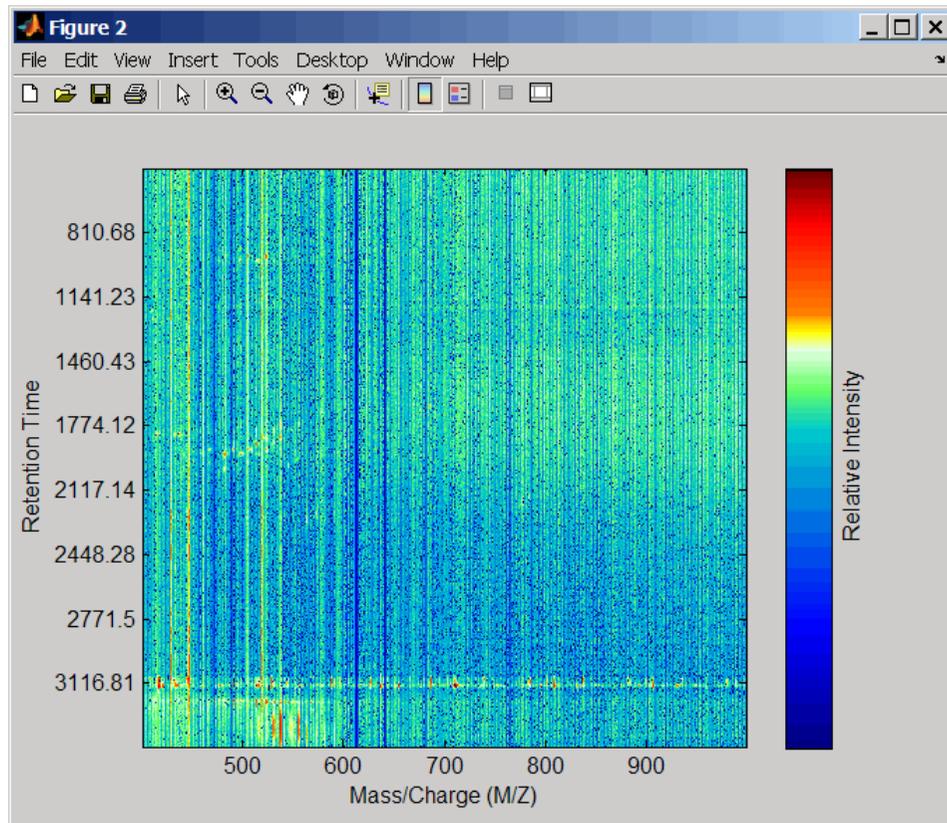
```
msdotplot(aligned_peaks,ret_time)
```

6 Link the axes of the two heat plots and zoom in to observe the detail to compare the unaligned and aligned LC/MS data sets.

```
linkaxes(findobj(0,'Tag','MSHeatMap'))
axis([480 532 375 485])
```
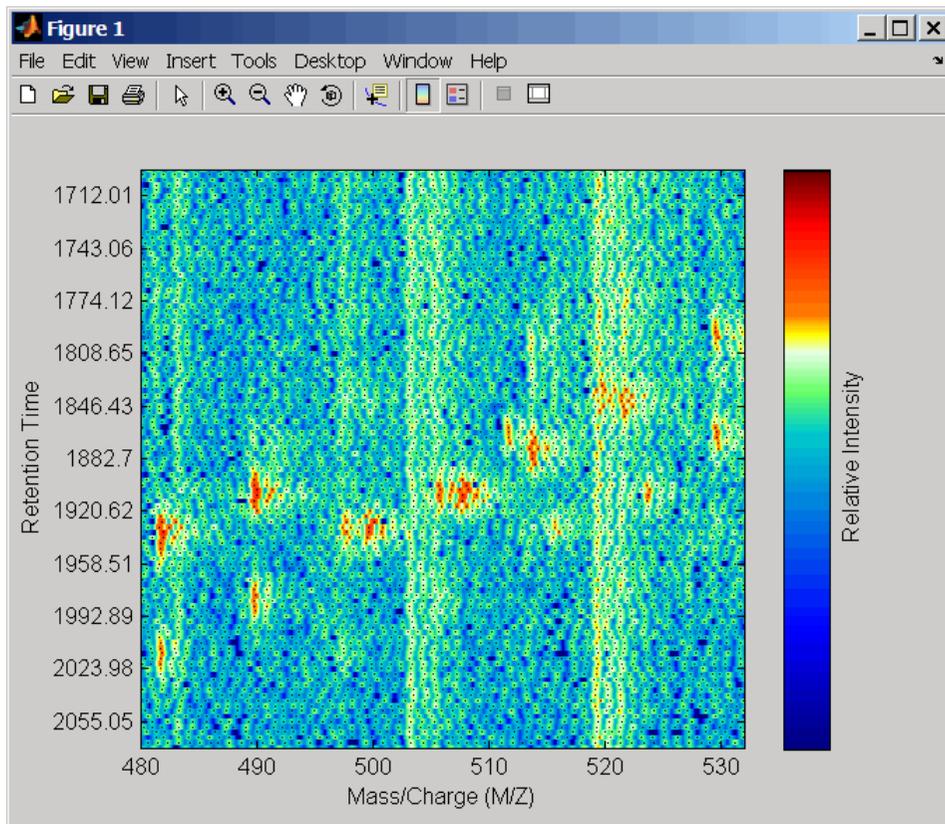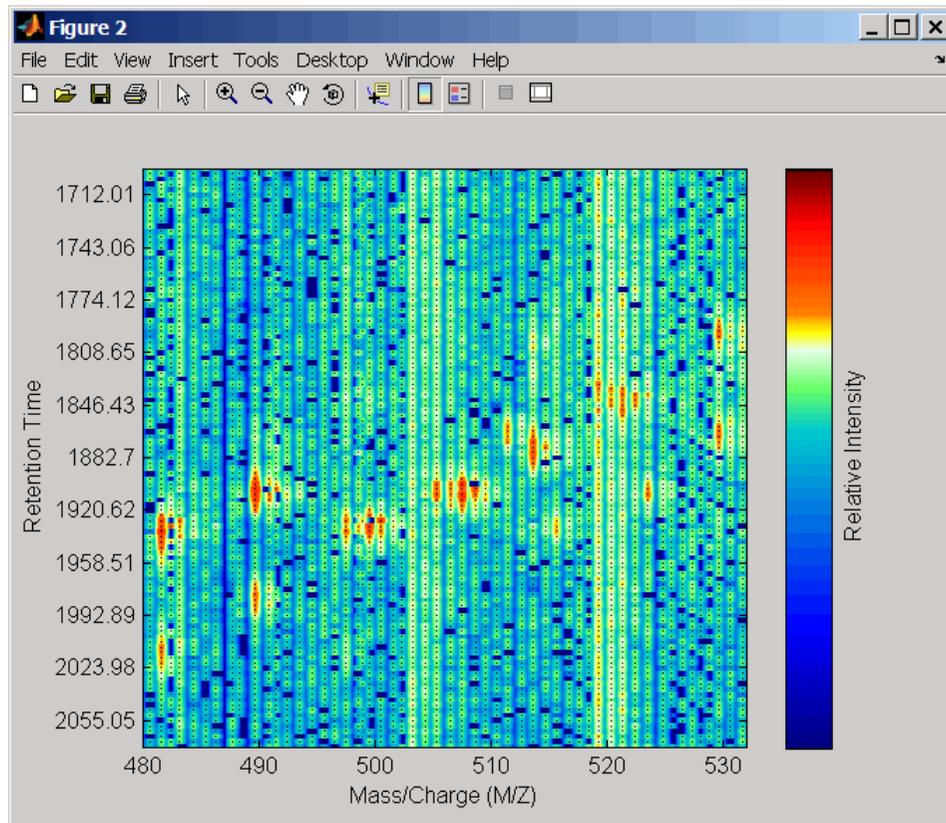
References

[1] Jeffries, N. (2005) Algorithms for alignment of mass spectrometry proteomic data. Bioinfomatics *21:14*, 3066–3073.

[2] Purvine, S., Kolker, N., and Kolker, E. (2004) Spectral Quality Assessment for High-Throughput Tandem Mass Spectrometry Proteomics. OMICS: A Journal of Integrative Biology *8:3*, 255–265.

See Also

msalign | msdotplot | msheatmap | mspeaks | msppresample | mzcdf2peaks | mzxml2peaks

mspeaks

Convert raw peak data to peak list (centroided data)

Syntax

```
Peaklist = mspeaks(X, Intensities)
[Peaklist, PFWHH] = mspeaks(X, Intensities)
[Peaklist, PFWHH, PExt] = mspeaks(X, Intensities)
mspeaks(X, Intensities, ...'Base', BaseValue, ...)
mspeaks(X, Intensities, ...'Levels', LevelsValue, ...)
mspeaks(X, Intensities, ...'NoiseEstimator',
NoiseEstimatorValue, ...)
mspeaks(X, Intensities, ...'Multiplier', MultiplierValue, ...)
mspeaks(X, Intensities, ...'Denoising', DenoisingValue, ...)
mspeaks(X, Intensities, ...'PeakLocation', PeakLocationValue, ...)
mspeaks(X, Intensities, ...'FWHHFilter', FWHHFilterValue, ...)
mspeaks(X, Intensities, ...'OverSegmentationFilter',
OverSegmentationFilterValue, ...)
mspeaks(X, Intensities, ...'HeightFilter', HeightFilterValue, ...)
mspeaks(X, Intensities, ...'ShowPlot', ShowPlotValue, ...)
mspeaks(X, Intensities, ...'Style', StyleValue, ...)
```

Description

Peaklist = mspeaks(*X*, *Intensities*) finds relevant peaks in raw, noisy peak signal data, and creates *Peaklist*, a two-column matrix, containing the separation-axis value and intensity for each peak. *X* is a vector of separation-unit values for a set of signals with peaks. *Intensities* is a matrix of intensity values for a set of peaks that share the same separation-unit range.

[*Peaklist*, *PFWHH*] = mspeaks(*X*, *Intensities*) returns *PFWHH*, a two-column matrix indicating the left and right locations of the full width at half height (FWHH) markers for each peak. For any peak not resolved at FWHH, mspeaks returns the peak shape extents instead. When *Intensities* includes multiple signals, then *PFWHH* is a cell array of matrices.

[*Peaklist*, *PFWHH*, *PExt*] = mspeaks(*X*, *Intensities*) returns *PExt*, a two-column matrix indicating the left and right locations of the peak shape extents determined after wavelet denoising. When *Intensities* includes multiple signals, then *PExt* is a cell array of matrices.

mspeaks(*X*, *Intensities*, ...'*PropertyName*', *PropertyValue*, ...) calls mspeaks with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

mspeaks(*X*, *Intensities*, ...'Base', *BaseValue*, ...) specifies the wavelet base.

mspeaks(*X*, *Intensities*, ...'Levels', *LevelsValue*, ...) specifies the number of levels for the wavelet decomposition.

mspeaks(*X*, *Intensities*, ...'NoiseEstimator', *NoiseEstimatorValue*, ...) specifies the method to estimate the threshold, T, to filter out noisy components in the first high-band decomposition (y_h).

mspeaks(*X*, *Intensities*, ...'Multiplier', *MultiplierValue*, ...) specifies the threshold multiplier constant.

mspeaks(*X*, *Intensities*, ...'Denoising', *DenoisingValue*, ...) controls the use of wavelet denoising to smooth the signal. Choices are true (default) or false.

mspeaks(*X*, *Intensities*, ...'PeakLocation', *PeakLocationValue*, ...) specifies the proportion of the peak height to use to select the points used to compute the centroid separation-axis value of the respective peak. *PeakLocationValue* must be a value ≥ 0 and ≤ 1. Default is 1.0.

mspeaks(*X*, *Intensities*, ...'FWHHFilter', *FWHHFilterValue*, ...) specifies the minimum full width at half height (FWHH), in separation units, for reported peaks. Peaks with FWHH below this value are excluded from the output list *Peaklist*.

mspeaks(*X*, *Intensities*, ...'OverSegmentationFilter', *OverSegmentationFilterValue*, ...) specifies the minimum distance, in separation units, between neighboring peaks. When the signal is not smoothed appropriately, multiple maxima can appear to represent the same peak. Increase this filter value to join oversegmented peaks into a single peak.

mspeaks(*X*, *Intensities*, ...'HeightFilter', *HeightFilterValue*, ...)
specifies the minimum height for reported peaks. Peaks with heights below this value are
excluded from the output list *Peaklist*.

mspeaks(*X*, *Intensities*, ...'ShowPlot', *ShowPlotValue*, ...) controls the
display of a plot of the original and the smoothed signal, with the peaks included in the
output matrix *Peaklist* marked.

mspeaks(*X*, *Intensities*, ...'Style', *StyleValue*, ...) specifies the style for
marking the peaks in the plot.

mspeaks finds peaks in data from any separation technique that produces signal
data, such as spectroscopy, nuclear magnetic resonance (NMR), electrophoresis,
chromatography, or mass spectrometry.

Input Arguments

X

Vector of separation-unit values for a set of signals with peaks. The number of elements
in the vector equals the number of rows in the matrix *Intensities*. The separation unit
can quantify wavelength, frequency, distance, time, or m/z depending on the instrument
that generates the signal data.

Default:

Intensities

Matrix of intensity values for a set of peaks that share the same separation-unit range.
Each row corresponds to a separation-unit value, and each column corresponds to either
a set of signals with peaks or a retention time. The number of rows equals the number of
elements in vector *X*.

Default:

BaseValue

Integer from 2 to 20 that specifies the wavelet base.

Default: 4

LevelsValue

Integer from 1 to 12 that specifies the number of levels for the wavelet decomposition.

Default: 10

NoiseEstimatorValue

String or scalar that specifies the method to estimate the threshold, T, to filter out noisy components in the first high-band decomposition (y_h). Choices are:

- mad — Default. Median absolute deviation, which calculates $T =$ $\mathrm{sqrt}(2*\mathrm{log}(n))*\mathrm{mad}(y\_h)$ / 0.6745, where n = the number of rows in the *Intensities* matrix.

- std — Standard deviation, which calculates $T = \mathrm{std}(y\_h)$.

- A positive real value.

Default:

MultiplierValue

Positive real value that specifies the threshold multiplier constant.

Default: 1.0

DenoisingValue

Controls the use of wavelet denoising to smooth the signal. Choices are true (default) or false.

Tip If your data was previously smoothed, for example, with the mslowess or mssgolay function, you do not need to use wavelet denoising. Set this property to false.

Default:

PeakLocationValue

Value that specifies the proportion of the peak height to use to select the points to compute the centroid separation-axis value of the respective peak. The value must be ≥ 0 and ≤ 1.

Note: When *PeakLocationValue* = 1.0, the peak location is at the maximum of the peak. When *PeakLocationValue* = 0, mspeaks computes the peak location with all the points from the closest minimum to the left of the peak to the closest minimum to the right of the peak.

Default: 1.0

FWHHFilterValue

Positive real value that specifies the minimum full width at half height (FWHH), in separation units, for reported peaks. Peaks with FWHH below this value are excluded from the output list *Peaklist*.

Default: 0

OverSegmentationFilterValue

Positive real value that specifies the minimum distance, in separation units, between neighboring peaks. When the signal is not smoothed appropriately, multiple maxima can appear to represent the same peak. Increase this filter value to join oversegmented peaks into a single peak.

Default: 0

HeightFilterValue

Positive real value that specifies the minimum height for reported peaks.

Default: 0

ShowPlotValue

Controls the display of a plot of the original signal and the smoothed signal, with the peaks included in the output matrix *Peaklist* marked. Choices are true, false, or *I*, an integer specifying the index of a spectrum in *Intensities*. If set to true, the first spectrum in *Intensities* is plotted. Default is:

- false — When you specify return values.
- true — When you do not specify return values.

Default:

StyleValue

String specifying the style for marking the peaks in the plot. Choices are:

- `'peak'` (default) — Places a marker at the peak crest.
- `'exttriangle'` — Draws a triangle using the peak crest and the extents.
- `'fwhhtriangle'` — Draws a triangle using the peak crest and the FWHH points.
- `'extline'` — Places a marker at the peak crest and vertical lines at the extents.
- `'fwhhline'` — Places a marker at the peak crest and a horizontal line at FWHH.

Default:

Output Arguments

Peaklist

Two-column matrix where each row corresponds to a peak. The first column contains separation-unit values (indicating the location of peaks along the separation axis). The second column contains intensity values. When *Intensities* includes multiple signals, then *Peaklist* is a cell array of matrices, each containing a peak list.

PFWHH

Two-column matrix indicating the left and right locations of the full width at half height (FWHH) markers for each peak. For any peak not resolved at FWHH, mspeaks returns the peak shape extents instead. When *Intensities* includes multiple signals, then *PFWHH* is a cell array of matrices.

PExt

Two-column matrix indicating the left and right locations of the peak shape extents determined after wavelet denoising. When *Intensities* includes multiple signals, then *PExt* is a cell array of matrices.

Examples

1   Load a MAT-file, included with the Bioinformatics Toolbox software, that contains two mass spectrometry data variables, MZ_lo_res and Y_lo_res. MZ_lo_res is a

vector of m/z values for a set of spectra. `Y_lo_res` is a matrix of intensity values for a set of mass spectra that share the same m/z range.

```
load sample_lo_res
```

2 Adjust the baseline of the eight spectra stored in `Y_lo_res`.

```
YB = msbackadj(MZ_lo_res,Y_lo_res);
```

3 Convert the raw mass spectrometry data to a peak list by finding the relevant peaks in each spectrum.

```
P = mspeaks(MZ_lo_res,YB);
```

4 Plot the third spectrum in `YB`, the matrix of baseline-corrected intensity values, with the detected peaks marked.

```
P = mspeaks(MZ_lo_res,YB,'SHOWPLOT',3);
```
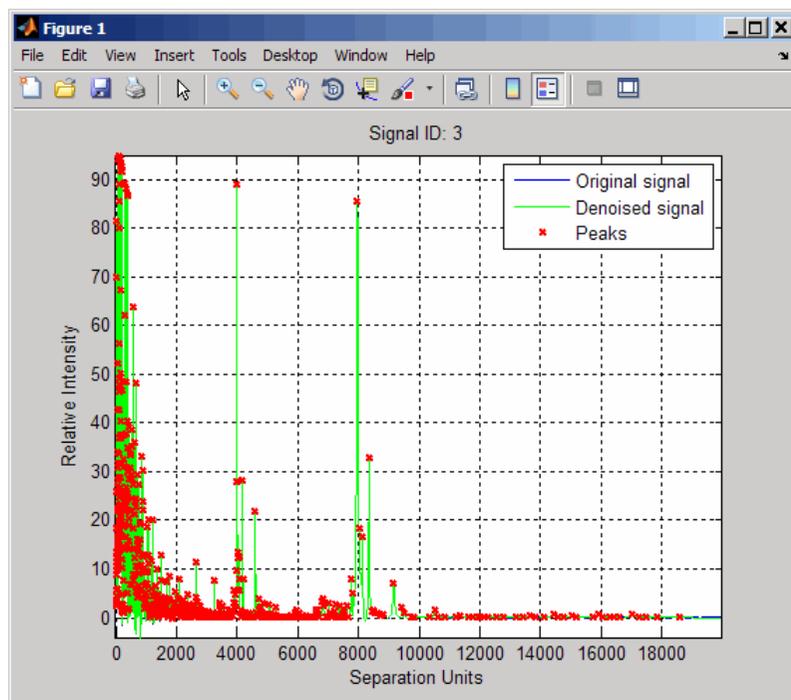

5 Smooth the signal using the `mslowess` function. Then convert the smoothed data to a peak list by finding relevant peaks and plot the third spectrum.

```
YS = mslowess(MZ_lo_res,YB,'SHOWPLOT',3);
```
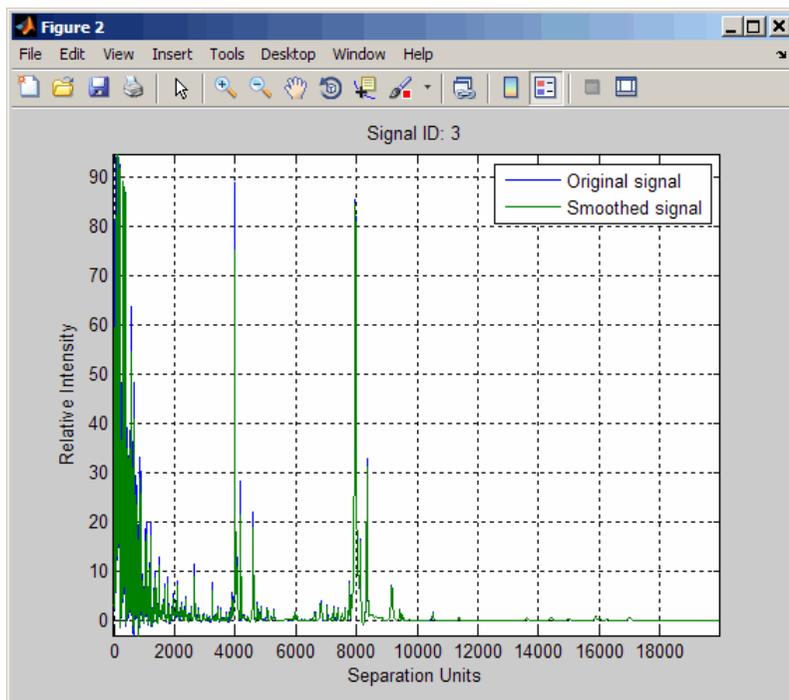


```
P = mspeaks(MZ_lo_res,YS,'DENOISING',false,'SHOWPLOT',3);
```
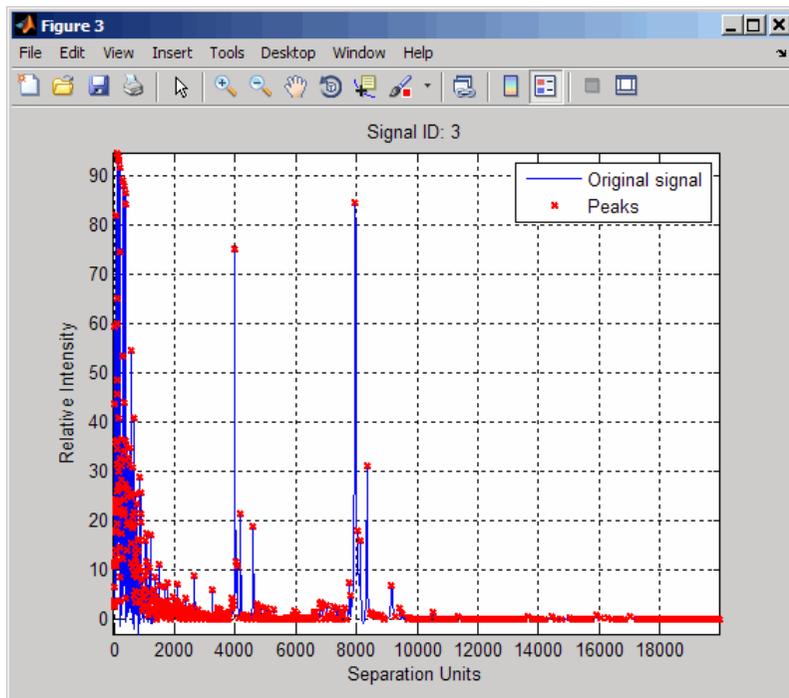
6   Use the `cellfun` function to remove all peaks with m/z values less than 2000 from the eight peaks listed in output `P`. Then plot the peaks of the third spectrum (in red) over its smoothed signal (in blue).

```
Q = cellfun(@(p) p(p(:,1)>2000,:),P,'UniformOutput',false);
figure
plot(MZ_lo_res,YS(:,3),'b',Q{3}(:,1),Q{3}(:,2),'rx')
xlabel('Mass/Charge (M/Z)')
ylabel('Relative Intensity')
axis([0 20000 -5 95])
```
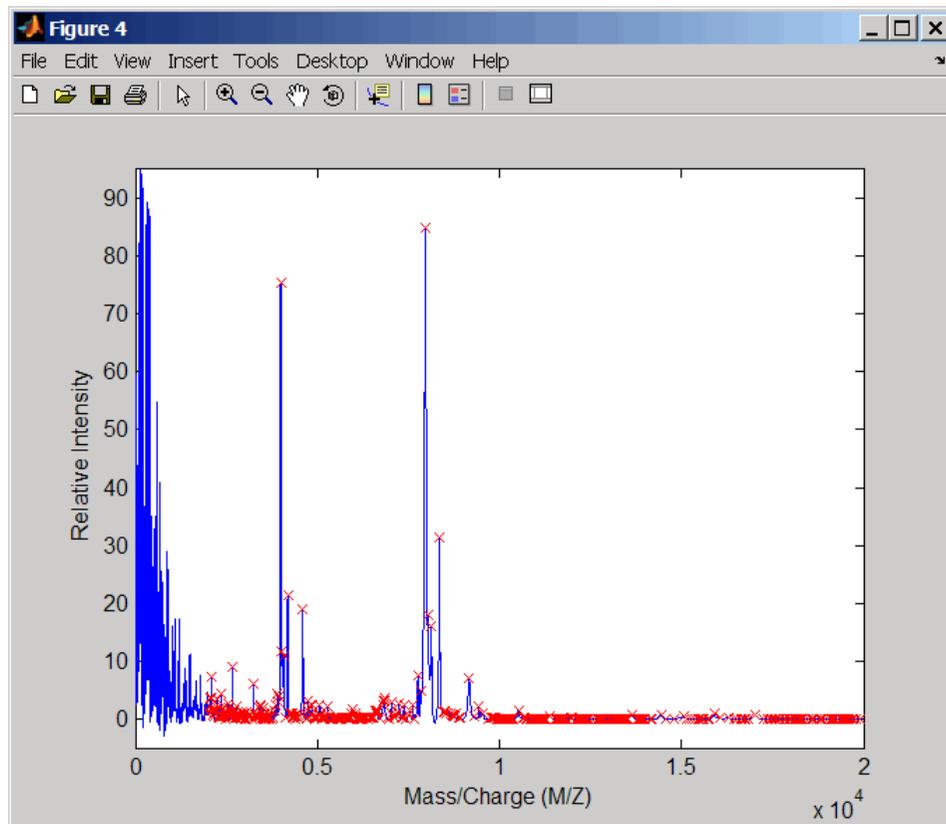
More About

Algorithms

mspeaks converts raw peak data to a peak list (centroided data) by:

1 Smoothing the signal using undecimated wavelet transform with Daubechies coefficients

2 Assigning peak locations

3 Estimating noise

4 Eliminating peaks that do not satisfy specified criteria

References

[1] Morris, J.S., Coombes, K.R., Koomen, J., Baggerly, K.A., and Kobayash, R. (2005) Feature extraction and quantification for mass spectrometry in biomedical applications using the mean spectrum. Bioinfomatics *21:9*, 1764–1775.

[2] Yasui, Y., Pepe, M., Thompson, M.L., Adam, B.L., Wright, G.L., Qu, Y., Potter, J.D., Winget, M., Thornquist, M., and Feng, Z. (2003) A data-analytic strategy for protein biomarker discovery: profiling of high-dimensional proteomic data for cancer detection. Biostatistics *4:3*, 449–463.

[3] Donoho, D.L., and Johnstone, I.M. (1995) Adapting to unknown smoothness via wavelet shrinkage. J. Am. Statist. Asso. *90*, 1200–1224.

[4] Strang, G., and Nguyen, T. (1996) Wavelets and Filter Banks (Wellesley: Cambridge Press).

[5] Coombes, K.R., Tsavachidis, S., Morris, J.S., Baggerly, K.A., Hung, M.C., and Kuerer, H.M. (2005) Improved peak detection and quantification of mass spectrometry data acquired from surface-enhanced laser desorption and ionization by denoising spectra with the undecimated discrete wavelet transform. Proteomics *5(16)*, 4107–4117.

See Also

```
msbackadj | msdotplot | mslowess | mspalign | msppresample | mssgolay |
cellfun
```

msppresample

Resample signal with peaks while preserving peaks

Syntax

```
[X, Intensities] = msppresample(Peaklist, N)
msppresample(Peaklist, N, ...'Range', RangeValue, ...)
msppresample(Peaklist, N, ...'FWHH', FWHHValue, ...)
msppresample(Peaklist, N, ...'ShowPlot', ShowPlotValue, ...)
```

Input Arguments

Peaklist	Either of the following: • Two-column matrix, where the first column contains separation-unit values and the second column contains intensity values. The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data. • Cell array of peak lists, where each element is a two-column matrix of separation-unit values and intensity values, and each element corresponds to a signal or retention time. **Tip** You can use the `mzxml2peaks` function or the `mspeaks` function to create the *Peaklist* matrix or cell array.
N	Integer specifying the number of equally spaced points (separation-unit values) in the resampled signal.
RangeValue	1-by-2 vector specifying the minimum and maximum separation-unit values for the output matrix *Intensities*. *RangeValue* must be within [min(*inputSU*) max(*inputSU*)], where *inputSU* is the concatenated separation-unit values from the input *Peaklist*. Default is the full range [min(*inputSU*) max(*inputSU*)].
FWHHValue	Value that specifies the full width at half height (FWHH) in separation units. The FWHH is used to convert each peak to a

	Gaussian shaped curve. Default is median(diff(*inputSU*))/2, where *inputSU* is the concatenated separation-unit values from the input *Peaklist*. The default is a rough approximation of resolution observed in the input data, *Peaklist*.
	Tip To ensure that the resolution of the peaks is preserved, set *FWHHValue* to half the distance between the two peaks of interest that are closest to each other.
ShowPlotValue	Controls the display of a plot of an original and resampled signal. Choices are true, false, or *I*, an integer specifying the index of a signal in *Intensities*. If you set to true, the first signal in *Intensities* is plotted. Default is: • false — When return values are specified. • true — When return values are not specified.

Output Arguments

| *X* | Vector of equally spaced, common separation-unit values for a set of signals with peaks. The number of elements in the vector equals *N*, or the number of rows in matrix *Intensities*. |
| *Intensities* | Matrix of reconstructed intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals *N*, or the number of elements in vector *X*. |

Description

Tip Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

[*X*, *Intensities*] = msppresample(*Peaklist*, *N*) resamples *Peaklist*, a peak list, by converting centroided peaks to a semicontinuous, raw signal that preserves peak information. The resampled signal has *N* equally spaced points. Output *X* is a vector of *N* elements specifying the equally spaced, common separation-unit values for the set of signals with peaks. Output *Intensities* is a matrix of reconstructed intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals *N*.

msppresample uses a Gaussian kernel to reconstruct the signal. The intensity at any given separation-unit value is taken from the maximum intensity of any contributing (overlapping) peaks.

Tip msppresample is useful to prepare a set of signals for imaging functions such as msheatmap and preprocessing functions such as msbackadj and msnorm.

---

msppresample(*Peaklist*, *N*, ... 'PropertyName', *PropertyValue*, ...) calls msppresample with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

msppresample(*Peaklist*, *N*, ...'Range', *RangeValue*, ...) specifies a separation-unit range for the output matrix *Intensities* using the minimum and maximum separation values specified in the 1-by-2 vector *RangeValue*. *RangeValue* must be within [min(*inputSU*) max(*inputSU*)], where *inputSU* is the concatenated separation-unit values from the input *Peaklist*. Default is the full range [min(*inputSU*) max(*inputSU*)]

msppresample(*Peaklist*, *N*, ...'FWHH', *FWHHValue*, ...) sets the full width at half height (FWHH) in separation units. The FWHH is used to convert each peak to a Gaussian shaped curve. Default is median(diff(*inputSU*))/2, where *inputSU* is the concatenated separation-unit values from the input *Peaklist*. The default is a rough approximation of resolution observed in the input data, *Peaklist*.

Tip To ensure that the resolution of the peaks is preserved, set *FWHHValue* to half the distance between the two peaks of interest that are closest to each other.

---

msppresample(*Peaklist*, *N*, ...'ShowPlot', *ShowPlotValue*, ...) controls the display of a plot of an original and resampled signal. Choices are true, false, or *I*, an integer specifying the index of a signal in *Intensities*. If you set to true, the first signal in *Intensities* is plotted. Default is:

- false — When return values are specified.
- true — When return values are not specified.

Examples

1 Load a MAT-file, included with the Bioinformatics Toolbox software, that contains liquid chromatography/mass spectrometry (LC/MS) data variables. It includes peaks, a cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time.

```
load lcmsdata
```

2 Resample the data, specifying 5000 m/z values in the resampled signal. Then create a heat map of the LC/MS data.

```
[MZ,Y] = msppresample(ms_peaks,5000);
msheatmap(MZ,ret_time,log(Y))
```
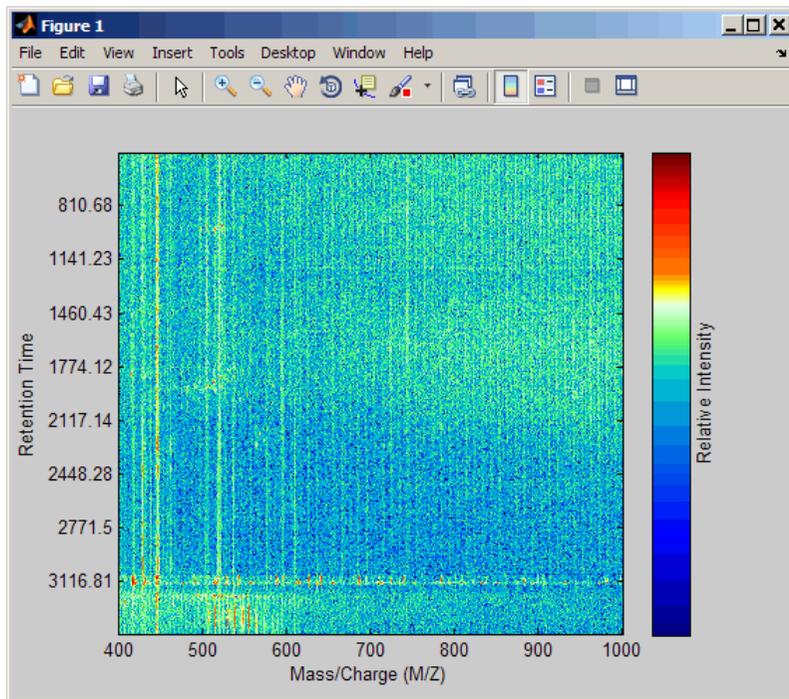
3 Plot the reconstructed profile spectra between two retention times.

```
figure
t1 = 3370;
t2 = 3390;
h = find(ret_time>t1 & ret_time<t2);
[MZ,Y] = msppresample(ms_peaks(h),10000);
plot3(repmat(MZ,1,numel(h)),repmat(ret_time(h)',10000,1),Y)
xlabel('Mass/Charge (M/Z)')
ylabel('Retention Time')
zlabel('Relative Intensity')
```
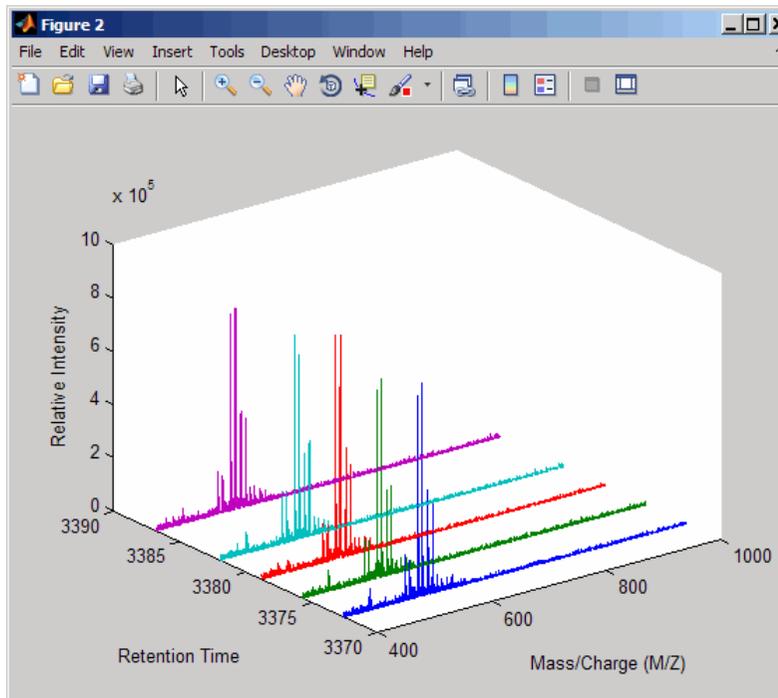
4 Resample the data to plot the Total Ion Chromatogram (TIC).

```
figure
[MZ,Y] = msppresample(ms_peaks,5000);
plot(ret_time,sum(Y))
title('Total Ion Chromatogram (TIC)')
xlabel('Retention Time')
ylabel('Relative Intensity')
```
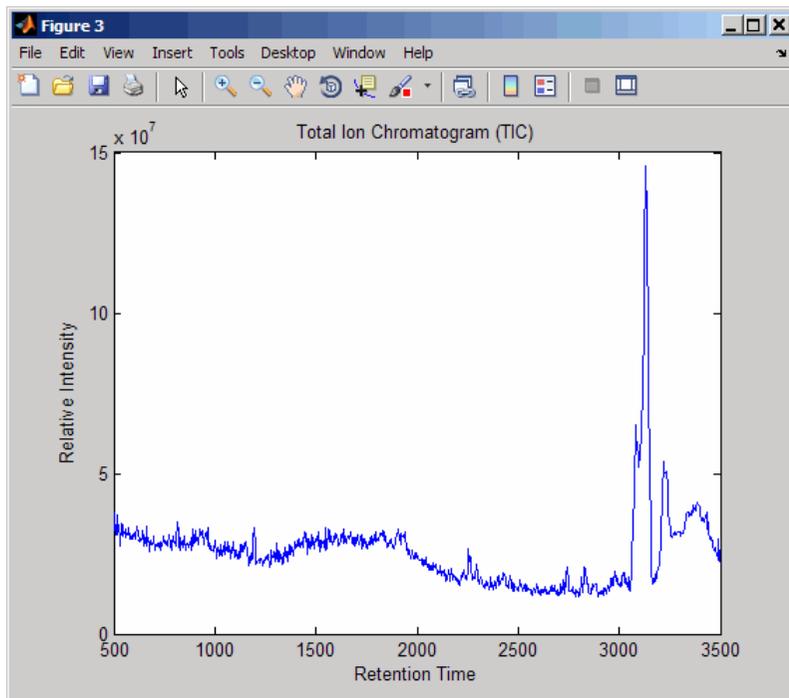
5   Resample the data to plot the Extracted Ion Chromatogram (XIC) in the 450 to 500 m/z range.

```
figure
[MZ,Y] = msppresample(ms_peaks,5000,'Range',[450 500]);
plot(ret_time,sum(Y))
title('Extracted Ion Chromatogram (XIC) from 450 to 500 M/Z')
xlabel('Retention Time')
ylabel('Relative Intensity')
```
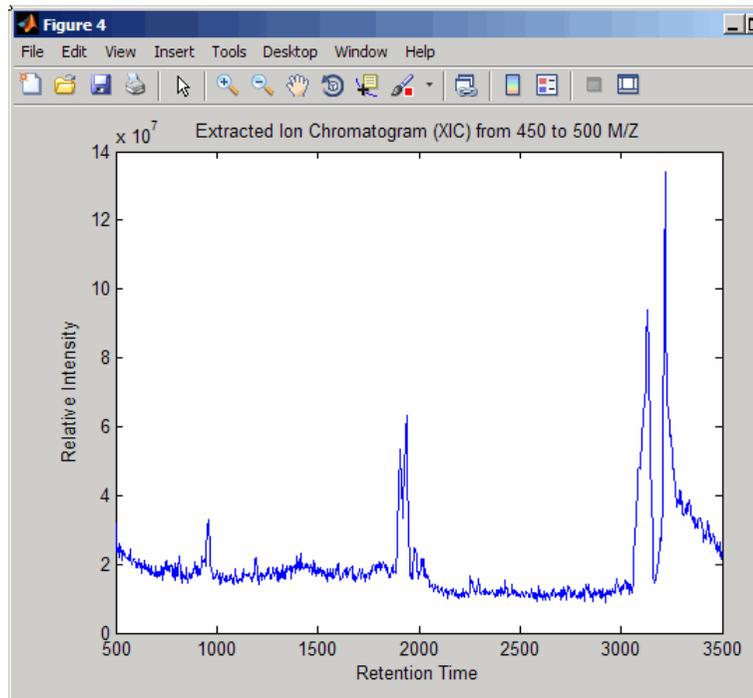
More About

- Differential Analysis of Complex Protein and Metabolite Mixtures Using Liquid Chromatography/Mass Spectrometry (LC/MS)

See Also

`msdotplot` | `mspalign` | `mspeaks` | `msresample` | `mzcdf2peaks` | `mzcdfread` | `mzxml2peaks` | `mzxmlread`

msresample

Resample signal with peaks

Syntax

```
[Xout, Intensitiesout] = msresample(X, Intensities, N)
msresample(..., 'Uniform', UniformValue, ...)
msresample(..., 'Range', RangeValue, ...)
msresample(..., 'RangeWarnOff', RangeWarnOffValue, ...)
msresample(..., 'Missing', MissingValue, ...)
msresample(..., 'Window', WindowValue, ...)
msresample(..., 'Cutoff', CutoffValue, ...)
msresample(..., 'ShowPlot', ShowPlotValue, ...)
```

Arguments

X	Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix *Intensities*. The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.
Intensities	Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector *X*.
N	Positive integer specifying the total number of samples.

Description

Tip Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

[*Xout*, *Intensitiesout*] = msresample(*X*, *Intensities*, *N*) resamples raw noisy signal data, *Intensities*. The output signal has *N* samples with a spacing that increases linearly within the range [min(*X*) max(*X*)]. *X* can be a linear or a quadratic function of its index. When you set input arguments such that down-sampling takes place, msresample applies a lowpass filter before resampling to minimize aliasing.

For the antialias filter, msresample uses a linear-phase FIR filter with a least-squares error minimization. The cutoff frequency is set by the largest down-sampling ratio when comparing the same regions in the *X* and *Xout* vectors.

Tip msresample is particularly useful when you have signals with different separation-unit vectors and you want to match the scales.

---

msresample(..., '*PropertyName*', *PropertyValue*, ...) calls msresample with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

msresample(..., 'Uniform', *UniformValue*, ...), when *UniformValue* is true, it forces the vector *X* to be uniformly spaced. The default value is false.

msresample(..., 'Range', *RangeValue*, ...) specifies a 1-by-2 vector with the separation-unit range for the output signal, *Intensitiesout*. *RangeValue* must be within [min(*X*) max(*X*)]. Default value is the full range [min(*X*) max(*X*)]. When *RangeValue* values exceed the values in *X*, msresample extrapolates the signal with zeros and returns a warning message.

msresample(..., 'RangeWarnOff', *RangeWarnOffValue*, ...) controls the return of a warning message when *RangeValue* values exceed the values in *X*. *RangeWarnOffValue* can be true or false (default).

msresample(..., 'Missing', *MissingValue*, ...), when *MissingValue* is true, analyzes the input vector, *X*, for dropped samples. The default value is false. If the down-sample factor is large, checking for dropped samples might not be worth the extra computing time. Dropped samples can only be recovered if the original separation-unit values follow a linear or a quadratic function of the *X* vector index.

msresample(..., 'Window', *WindowValue*, ...) specifies the window used when calculating parameters for the lowpass filter. Enter 'Flattop', 'Blackman', 'Hamming', or 'Hanning'. The default value is 'Flattop'.

msresample(..., 'Cutoff', *CutoffValue*, ...) specifies the cutoff frequency. Enter a scalar value from 0 to 1 (Nyquist frequency or half the sampling frequency). By default, msresample estimates the cutoff value by inspecting the separation-unit vectors, *X* and *XOut*. However, the cutoff frequency might be underestimated if *X* has anomalies.

msresample(..., 'ShowPlot', *ShowPlotValue*, ...) plots the original and the resampled signal. When msresample is called without output arguments, the signals are plotted unless *ShowPlotValue* is false. When *ShowPlotValue* is true, only the first signal in *Intensities* is plotted. *ShowPlotValue* can also contain an index to one of the signals in *Intensities*.

Tip LC/MS data analysis requires extended amounts of memory from the operating system.

- If you receive errors related to memory, try the following:

    - Increase the virtual memory (swap space) for your operating system (with a recommended initial size of 3,069 and a maximum size of 16,368) as described in "Memory Usage".
    - Set the 3 GB switch (32-bit Windows XP only) as described in "Memory Usage".

- If you receive errors related to Java heap space, increase your Java heap space:
    - If you have MATLAB version 7.10 (R2010a) or later, see "Java Heap Memory Preferences"
    - If you have MATLAB version 7.9 (R2009b) or earlier, see http://www.mathworks.com/support/solutions/data/1-18I2C.html

---

Examples

Resample Mass Spectrometry Data

This example shows how to resample mass spec data.

Load a MAT-file, included with Bioinformatics Toolbox™, that contains mass spectrometry data, and then extract m/z and intensity value vectors.

```
load sample_hi_res;
mz = MZ_hi_res;
y = Y_hi_res;
```

Plot the original data.

```
plot(mz, y, '.')
```
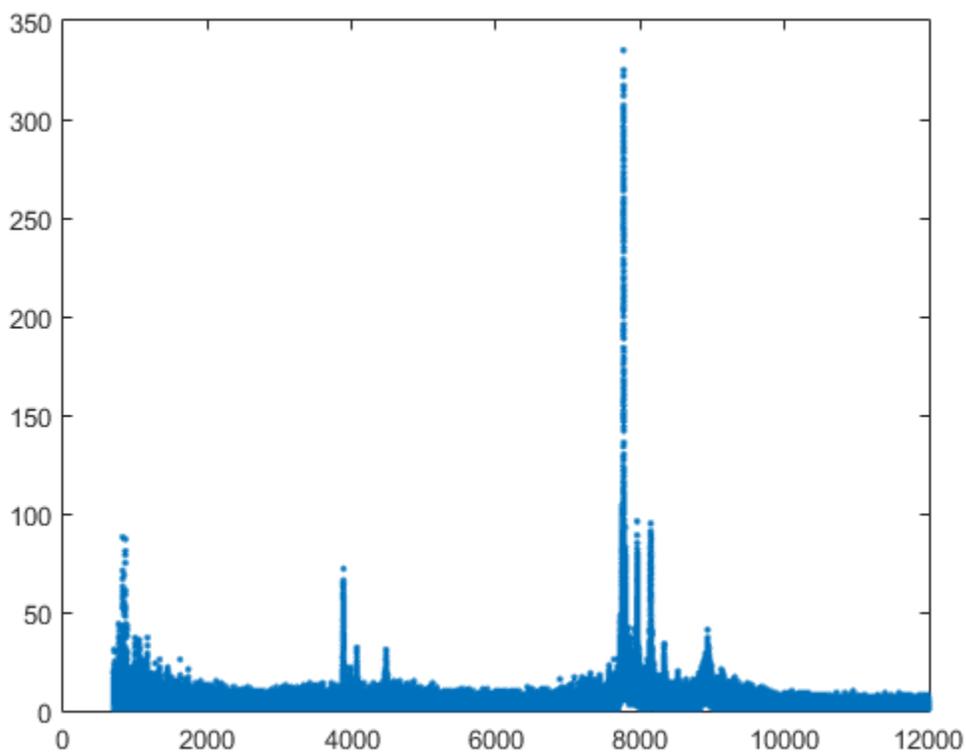

Resample the spectrogram to have 10000 samples between 2000 and maximum m/z value in the data set, and show both the resampled and original data.

```
[mz1,y1] = msresample(mz, y, 10000, 'range',[2000 max(mz)],'SHOWPLOT',true);
```
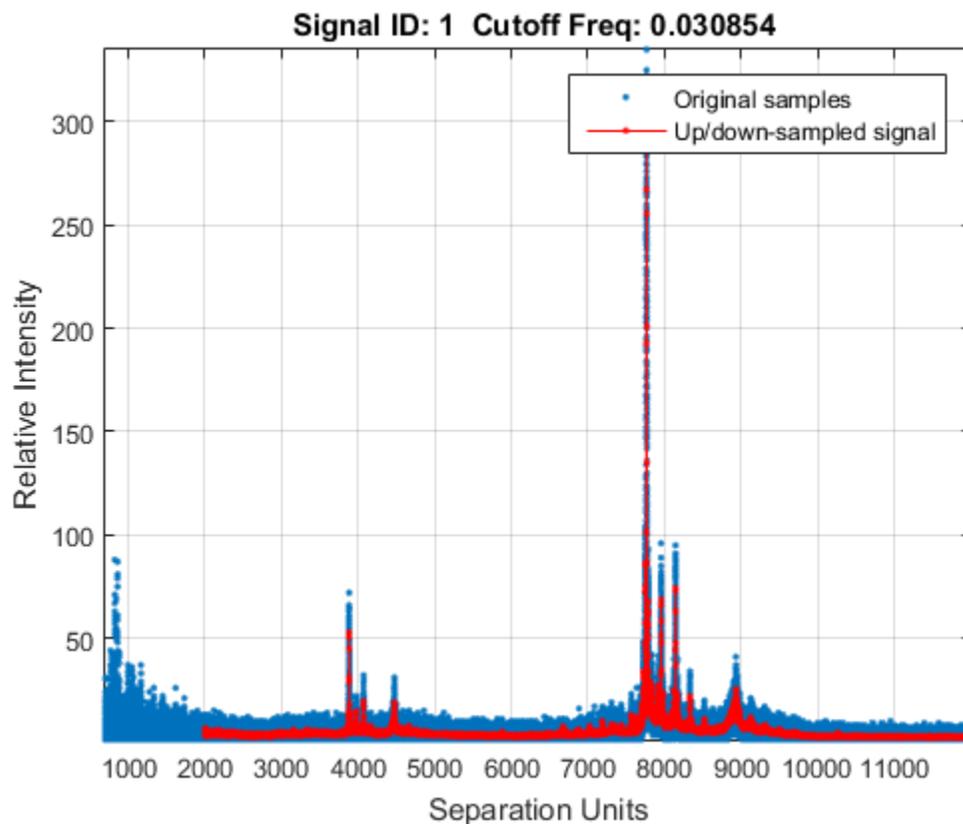
Signal ID: 1 Cutoff Freq: 0.030854

More About

- Preprocessing Raw Mass Spectrometry Data

See Also

msalign | msbackadj | msheatmap | mslowess | msnorm | msppresample | mssgolay | msviewer

mssgolay

Smooth signal with peaks using least-squares polynomial

Syntax

```
Yout = mssgolay(X, Intensities)
mssgolay(X, Intensities, ...'Span', SpanValue, ...)
mssgolay(X, Intensities, ...'Degree', DegreeValue, ...)
mssgolay(X, Intensities, ...'ShowPlot', ShowPlotValue, ...)
```

Arguments

X	Vector of separation-unit values for a set of signals with peaks. The number of elements in the vector equals the number of rows in the matrix *Intensities*. The separation unit can quantify wavelength, frequency, distance, time, or m/z depending on the instrument that generates the signal data.
Intensities	Matrix of intensity values for a set of peaks that share the same separation-unit range. Each row corresponds to a separation-unit value, and each column corresponds to either a set of signals with peaks or a retention time. The number of rows equals the number of elements in vector *X*.

Description

Tip Use the following syntaxes with data from any separation technique that produces signal data, such as spectroscopy, NMR, electrophoresis, chromatography, or mass spectrometry.

Yout = mssgolay(*X*, *Intensities*) smooths raw noisy signal data, *Intensities*, using a least-squares digital polynomial filter (Savitzky and Golay filters). The default span or frame is 15 samples.

mssgolay(*X*, *Intensities*, ...'*PropertyName*', *PropertyValue*, ...) calls mssgolay with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

mssgolay(*X*, *Intensities*, ...'Span', *SpanValue*, ...) modifies the frame size for the smoothing function. If *SpanValue* is greater than 1, the window is the size of *SpanValue* in samples independent of the *X* vector. Higher values smooth the signal more with an increase in computation time. If *SpanValue* is less than 1, the window size is a fraction of the number of points in the input data, *X*. For example, if *SpanValue* is 0.05, the window size is equal to 5% of the number of points in *X*.

Note: The original algorithm by Savitzky and Golay assumes the input vector, *X*, has uniformly spaced separation units, while mssgolay also allows one that is not uniformly spaced. Therefore, the sliding frame for smoothing is centered using the closest samples in terms of the *X* value and not in terms of the *X* index.

When the input vector, *X*, does not have repeated values or NaN values, the algorithm is approximately twice as fast.

When the input vector, *X*, is evenly spaced, the least-squares fitting is performed once so that the signal is filtered with the same coefficients, and the speed of the algorithm increases considerably.

If the input vector, *X*, is evenly spaced and *SpanValue* is even, span is incremented by 1 to include both edge samples in the frame.

---

mssgolay(*X*, *Intensities*, ...'Degree', *DegreeValue*, ...) specifies the degree of the polynomial (*DegreeValue*) fitted to the points in the moving frame. The default value is 2. *DegreeValue* must be smaller than *SpanValue*.

mssgolay(*X*, *Intensities*, ...'ShowPlot', *ShowPlotValue*, ...) plots smoothed signals over the original. When mssgolay is called without output arguments, the signals are plotted unless *ShowPlotValue* is false. When *ShowPlotValue* is true, only the first signal in *Intensities* is plotted. *ShowPlotValue* can also contain an index to one of the signals in *Intensities*.

Examples

Smooth Mass Spectrometry Data

This example shows how to smooth mass spectrometry data using least-squares polynomial approach.

Load a MAT-file, included with Bioinformatics Toolbox™, that contains mass spectrometry data including MZ_lo_res , a vector of m/z values for a set of spectra, and Y_lo_res , a matrix of intensity values for a set of mass spectra that share the same m/z charge.

```
load sample_lo_res
```

Apply least-squares polynomial smoothing to the data.

```
YS = mssgolay(MZ_lo_res, Y_lo_res);
```

Plot the third sample/spectrogram in Y_lo_res , and its smoothed signal.

```
mssgolay(MZ_lo_res,Y_lo_res,'SHOWPLOT',3);
```
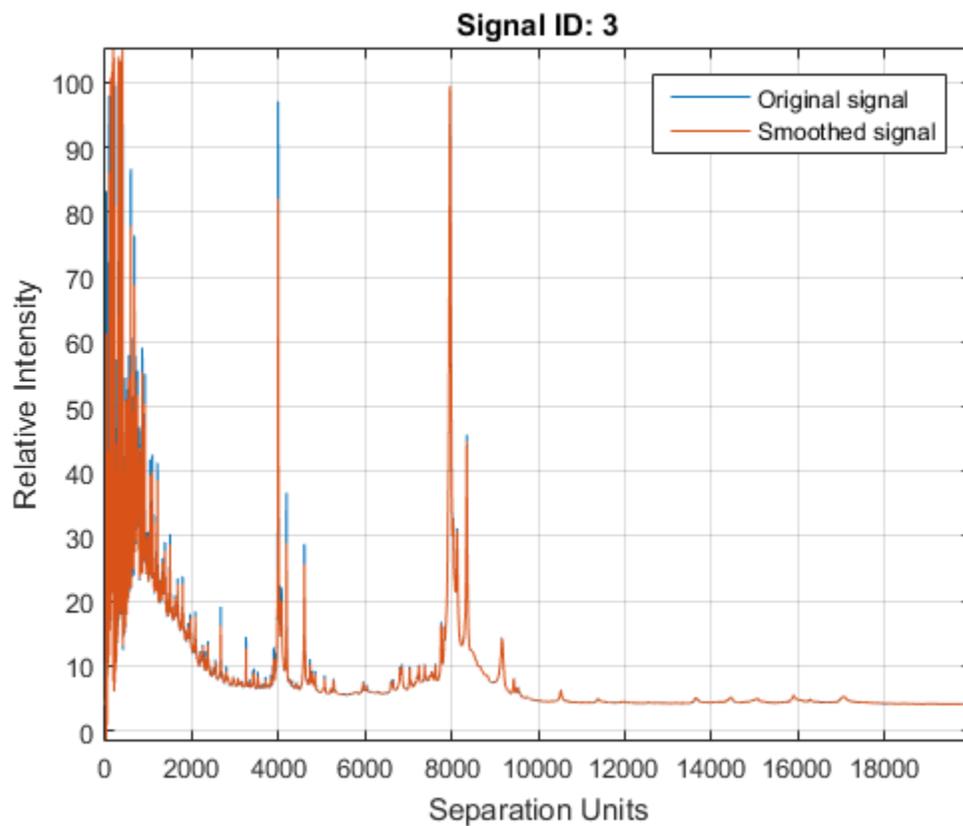
More About

- Preprocessing Raw Mass Spectrometry Data

See Also

msalign | msbackadj | msheatmap | mslowess | msnorm | mspeaks | msresample | msviewer

msviewer

Explore mass spectrum or set of mass spectra

Syntax

```
msviewer(MZ , Intensities)
msviewer(..., 'Markers', MarkersValue)
msviewer(..., 'Group', GroupValue)
```

Arguments

MZ	Column vector of common mass/charge (m/z) values for a set of spectra. The number of elements in the vector equals the number of rows in the matrix *Intensities*.
Intensities	Matrix of intensity values for a set of mass spectra that share the same m/z range. Each row corresponds to an m/z value, and each column corresponds to a spectrum or retention time. The number of rows equals the number of elements in vector *MZ*.
GroupValue	Either of the following: • Vector of values with the same number of elements as rows in the matrix *Intensities* • Cell array of strings with the same number of elements as rows (spectra) in the matrix *Intensities* Each value or string specifies a group to which the corresponding spectrum belongs. Spectra from the same group are plotted with the same color. Default is [1:numSpectra].

Description

msviewer(*MZ* , *Intensities*) displays the MS Viewer, which lets you view and explore a mass spectrum defined by *MZ* and *Intensities*.

msviewer(..., 'Markers', *MarkersValue*) specifies a list of marker positions from the mass/charge vector, *MZ*, for exploration and easy navigation. Enter a column vector with *MZ* values.

msviewer(..., 'Group', *GroupValue*) specifies a group to which the spectra belong. The groups are specified by *GroupValue*, a vector of values or cell array of strings. The number of values or strings is the same as the number of rows in the matrix *Intensities*. Each value or string specifies a group to which the corresponding spectrum belongs. Spectra from the same group are plotted with the same color. Default is [1:numSpectra].

The MS Viewer includes the following features:

- Plot mass spectra. The spectra are plotted with different colors according to their group labels.
- An overview displays a full spectrum, and a box indicates the region that is currently displayed in the main window.
- Five different zoom in options, one zoom out option, and a reset view option resize the spectrum.
- Add/focus/move/delete marker operations
- Import/Export markers from/to MATLAB workspace
- Print and preview the spectra plot
- Print the spectra plot to a MATLAB Figure window

MSViewer has five components:

- Menu bar: **File**, **Tools**, **Window**, and **Help**
- Toolbar: Move marker, Zoom XY, Zoom X, Zoom Y, Zoom out, Reset view, and Help
- Main window: display the spectra
- Overview window: display the overview of a full spectrum (the average of all spectra in display)
- Marker control panel: a list of markers, Add Marker, Delete Marker, up and down buttons

Examples

Plot Mass Spectra Data

This example shows how to plot mass spectra data.

Load and plot a sample mass spectra data.

```
load sample_lo_res
msviewer(MZ_lo_res, Y_lo_res)
```
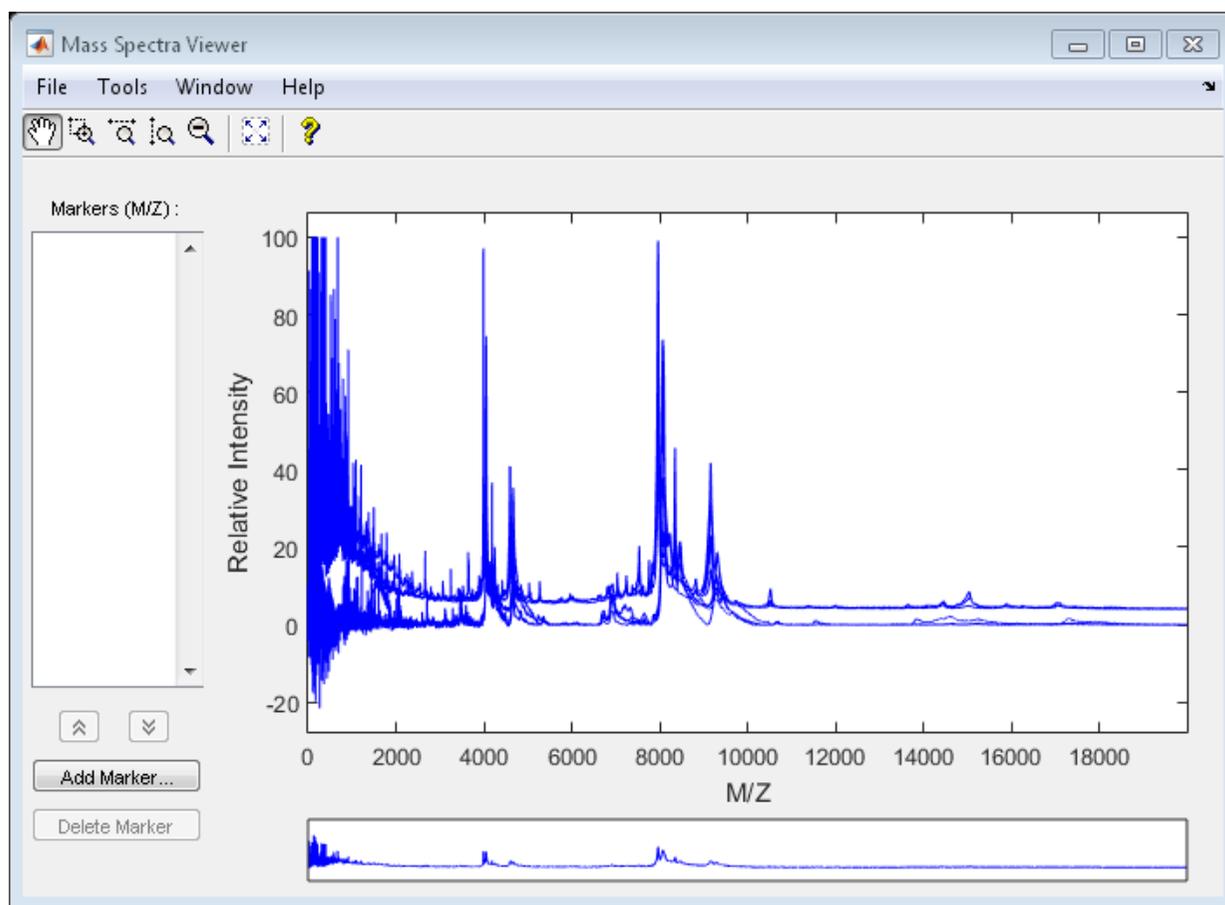

Add a marker by pointing to a mass peak, right-clicking, and then clicking **Add Marker**.

The **File** menu has the following options.

- **Import Markers from Workspace** - Opens the Import Markers From MATLAB® Workspace dialog. The dialog displays a list of double Mx1 or 1xM variables. If the selected variable is out of range, the viewer displays an error message.
- **Export Markers to Workspace** - Opens the Export Markers to MATLAB® Workspace dialog. Enter a variable name for the markers. All markers are saved. If thre is no marker available, this menu item is disabled.
- **Print to Figure** - Prints the spectra plot in the main display to a MATLAB® figure window.

The **Tools** menu has the following options.

- **Add Marker** - Opens the Add Marker dialog where you can enter an m/z marker.
- **Delete Marker** - Removes the currently selected m/z marker from the **Markers** (m/z) list.
- **Next Marker** or **Previous Marker** - Moves the selection up and down the **Markers** list.
- **Zoom XY**, **Zoom X**, **Zoom Y**, or **Zoom Out** - Changes the cursor from an arror to a crosshair. Left-click and drag a rectangle box over an area and then release it. The display zooms the area covered by the box.

From the range window at the bottom, move the view box to a new location.

More About

- `msalign`
- `msbackadj`
- `mslowess`
- `msnorm`
- `msresample`
- `mssgolay`

See Also

`msheatmap`

multialign

Align multiple sequences using progressive method

Syntax

```
SeqsMultiAligned = multialign(Seqs)
SeqsMultiAligned = multialign(Seqs, Tree)
multialign(..., 'PropertyName', PropertyValue,...)
multialign(..., 'Weights', WeightsValue)
multialign(..., 'ScoringMatrix', ScoringMatrixValue)
multialign(..., 'SMInterp', SMInterpValue)
multialign(..., 'GapOpen', GapOpenValue)
multialign(..., 'ExtendGap', ExtendGapValue)
multialign(..., 'DelayCutoff', DelayCutoffValue)
multialign(..., 'UseParallel', UseParallelValue)
multialign(..., 'Verbose', VerboseValue)
multialign(..., 'ExistingGapAdjust', ExistingGapAdjustValue)
multialign(..., 'TerminalGapAdjust', TerminalGapAdjustValue)
```

Input Arguments

Seqs	Vector of structures with the fields `'Sequence'` for the residues and `'Header'` or `'Name'` for the labels.
	Seqs can also be a cell array of strings or a char array.
Tree	Phylogenetic tree calculated with the `seqlinkage` or `seqneighjoin` function.
WeightsValue	Property to select the sequence weighting method. Enter `'THG'` (default) or `'equal'`.
ScoringMatrixValue	Either of the following:
	• String specifying the scoring matrix to use for the alignment. Choices for amino acid sequences are:

- `'BLOSUM62'`
- `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'`
- `'BLOSUM100'`
- `'PAM10'` increasing by 10 up to `'PAM500'`
- `'DAYHOFF'`
- `'GONNET'`

Default is:

- `'BLOSUM80'` to `'BLOSUM30'` series — When *AlphabetValue* equals `'AA'`
- `'NUC44'` — When *AlphabetValue* equals `'NT'`

Note: The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the `'Scale'` property to specify an additional scale factor to convert the output score from bits to another unit.

- Matrix representing the scoring matrix to use for the alignment. It can be a matrix, such as returned by the `blosum`, `pam`, `dayhoff`, `gonnet`, or `nuc44` function. It can also be an *M*-by-*M* matrix or *M*-by-*M*-by-*N* array of matrices with *N* user-defined scoring matrices.

Note: If you use a scoring matrix that you created or was created by one of the above functions, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix. When passing your own series of scoring matrices, ensure they share the same scale.

	Note: If you need to compile `multialign` into a stand-alone application or software component using MATLAB Compiler, use a matrix instead of a string for *ScoringMatrixValue*.
SMInterpValue	Property to specify whether linear interpolation of the scoring matrices is on or off. When `false`, the scoring matrix is assigned to a fixed range depending on the distances between the two profiles (or sequences) being aligned. Default is `true`.
GapOpenValue	Scalar or a function specified using @. If you enter a function, `multialign` passes four values to the function: the average score for two matched residues (`sm`), the average score for two mismatched residues (`sx`), and, the length of both profiles or sequences (`len1`, `len2`). Default is `@(sm,sx,len1,len2) 5*sm`.
ExtendGapValue	Scalar or a function specified using @. If you enter a function, `multiialign` passes four values to the function: the average score for two matched residues (`sm`), the average score for two mismatched residues (`sx`), and the length of both profiles or sequences (`len1`, `len2`). Default is `@(sm,sx,len1,len2) sm/4`.
DelayCutoffValue	Property to specify the threshold delay of divergent sequences. Default is unity where sequences with the closest sequence farther than the median distance are delayed.

UseParallelValue	Controls the computation of the pairwise alignments using `parfor`-loops. When `true`, and Parallel Computing Toolbox is installed and a `parpool` is open, computation occurs in parallel. If there are no open `parpool`, but automatic creation is enabled in the Parallel Preferences, the default pool will be automatically open and computation occurs in parallel. If Parallel Computing Toolbox is installed, but there are no open `parpool` and automatic creation is disabled, then computation uses `parfor`-loops in serial mode. If Parallel Computing Toolbox is not installed, then computation uses `parfor`-loops in serial mode. Default is `false`, which uses for-loops in serial mode.
VerboseValue	Property to control displaying the sequences with sequence information. Default is `false`.
ExistingGapAdjustValue	Property to control automatic adjustment based on existing gaps. Default is `true`.
TerminalGapAdjustValue	Property to adjust the penalty for opening a gap at the ends of the sequence. Default is `false`.

Output Arguments

SeqsMultiAligned	Vector of structures (same as *Seqs*) but with the field `'Sequence'` updated with the alignment. When *Seqs* is a cell or char array, *SeqsMultiAligned* is a char array with the output alignment following the same order as the input.

Description

SeqsMultiAligned = `multialign`(*Seqs*) performs a progressive multiple alignment for a set of sequences (*Seqs*). Pairwise distances between sequences are computed after pairwise alignment with the Gonnet scoring matrix and then by counting the proportion

of sites at which each pair of sequences are different (ignoring gaps). The guide tree is calculated by the neighbor-joining method assuming equal variance and independence of evolutionary distance estimates.

SeqsMultiAligned = multialign(*Seqs*, *Tree*) uses a tree (*Tree*) as a guide for the progressive alignment. The sequences (*Seqs*) should have the same order as the leaves in the tree (*Tree*) or use a field ('Header' or 'Name') to identify the sequences.

multialign(..., '*PropertyName*', *PropertyValue,...*) enters optional arguments as property name/property value pairs. Specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

multialign(..., 'Weights', *WeightsValue*) selects the sequence weighting method. Weights emphasize highly divergent sequences by scaling the scoring matrix and gap penalties. Closer sequences receive smaller weights.

Values of the property Weights are:

- 'THG' (default) — Thompson-Higgins-Gibson method using the phylogenetic tree branch distances weighted by their thickness.
- 'equal' — Assigns the same weight to every sequence.

multialign(..., 'ScoringMatrix', *ScoringMatrixValue*) selects the scoring matrix (*ScoringMatrixValue*) for the progressive alignment. Match and mismatch scores are interpolated from the series of scoring matrices by considering the distances between the two profiles or sequences being aligned. The first matrix corresponds to the smallest distance, and the last matrix to the largest distance. Intermediate distances are calculated using linear interpolation.

multialign(..., 'SMInterp', *SMInterpValue*), when *SMInterpValue* is false, turns off the linear interpolation of the scoring matrices. Instead, each supplied scoring matrix is assigned to a fixed range depending on the distances between the two profiles or sequences being aligned.

multialign(..., 'GapOpen', *GapOpenValue*) specifies the initial penalty for opening a gap.

multialign(..., 'ExtendGap', *ExtendGapValue*) specifies the initial penalty for extending a gap.

`multialign(..., 'DelayCutoff', `*`DelayCutoffValue`*`)` specifies a threshold to delay the alignment of divergent sequences whose closest neighbor is farther than

`(`*`DelayCutoffValue`*`) * (median patristic distance between sequences)`

`multialign(..., 'UseParallel', `*`UseParallelValue`*`)` specifies whether to use `parfor`-loops when computing the pairwise alignments. When `true`, and Parallel Computing Toolbox is installed and a `parpool` is open, computation occurs in parallel. If there are no open `parpool`, but automatic creation is enabled in the Parallel Preferences, the default pool will be automatically open and computation occurs in parallel. If Parallel Computing Toolbox is installed, but there are no open `parpool` and automatic creation is disabled, then computation uses `parfor`-loops in serial mode. If Parallel Computing Toolbox is not installed, then computation uses `parfor`-loops in serial mode. Default is `false`, which uses for-loops in serial mode.

`multialign(..., 'Verbose', `*`VerboseValue`*`)`, when *`VerboseValue`* is `true`, turns on verbosity.

The remaining input optional arguments are analogous to the function `profalign` and are used through every step of the progressive alignment of profiles.

`multialign(..., 'ExistingGapAdjust', `*`ExistingGapAdjustValue`*`)`, when *`ExistingGapAdjustValue`* is `false`, turns off the automatic adjustment based on existing gaps of the position-specific penalties for opening a gap.

When *`ExistingGapAdjustValue`* is `true`, for every profile position, `profalign` proportionally lowers the penalty for opening a gap toward the penalty of extending a gap based on the proportion of gaps found in the contiguous symbols and on the weight of the input profile.

`multialign(..., 'TerminalGapAdjust', `*`TerminalGapAdjustValue`*`)`, when *`TerminalGapAdjustValue`* is `true`, adjusts the penalty for opening a gap at the ends of the sequence to be equal to the penalty for extending a gap.

Examples

Align multiple sequences

This example shows how to align multiple protein sequences.

Use the `fastaread` function to read p53samples.txt, a FASTA-formatted file included with Bioinformatics Toolbox™, which contains p53 protein sequences of seven species.

```
p53 = fastaread('p53samples.txt')


p53 =

7x1 struct array with fields:

    Header
    Sequence
```

Compute the pairwise distances between each pair of sequences using the 'GONNET' scoring matrix.

```
dist = seqpdist(p53,'ScoringMatrix','GONNET');
```

Build a phylogenetic tree using an unweighted average distance (UPGMA) method. This tree will be used as a guiding tree in the next step of progressive alignment.

```
tree = seqlinkage(dist,'average',p53)

    Phylogenetic tree object with 7 leaves (6 branches)
```

Perform progressive alignment using the PAM family scoring matrices.

```
ma = multialign(p53,tree,'ScoringMatrix',...
                {'pam150','pam200','pam250'})
showalignment(ma)


ma =

7x1 struct array with fields:

    Header
    Sequence
```
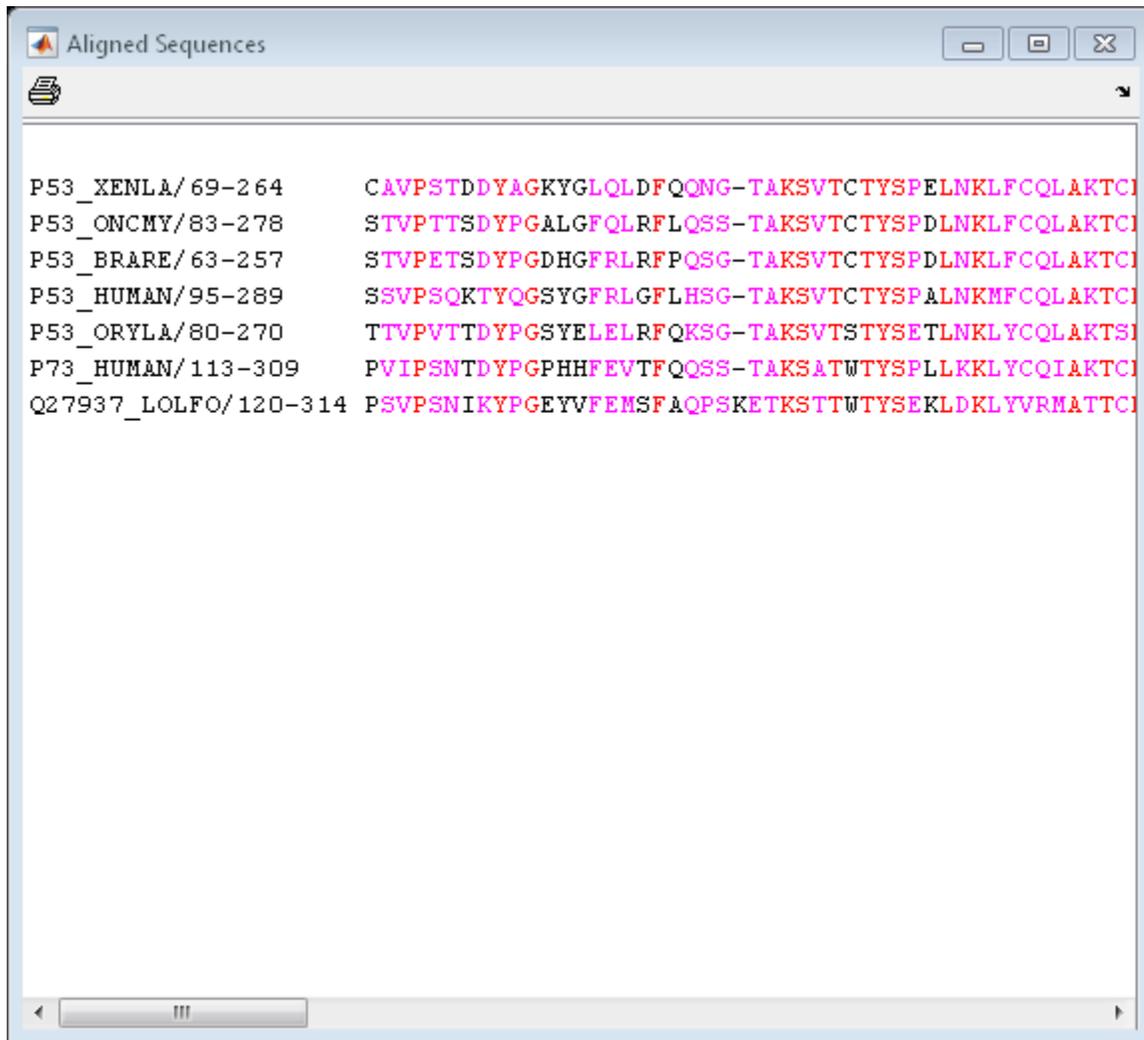
Align Nucleotide Sequences

1   Enter an array of sequences.

```
seqs = {'CACGTAACATCTC','ACGACGTAACATCTTCT','AAACGTAACATCTCGC'};
```

2   Promote terminations with gaps in the alignment.

```
multialign(seqs,'terminalGapAdjust',true)

ans =
--CACGTAACATCTC--
ACGACGTAACATCTTCT
-AAACGTAACATCTCGC
```

3 Compare the alignment without termination gap adjustment.

```
multialign(seqs)

ans =
CA--CGTAACATCT--C
ACGACGTAACATCTTCT
AA-ACGTAACATCTCGC
```

See Also

align2cigar | hmmprofalign | multialignread | multialignwrite | nwalign | profalign | seqprofile | seqconsensus | seqneighjoin | showalignment

multialignread

Read multiple sequence alignment file

Syntax

```
S = multialignread(File)
[Headers, Sequences] = multialignread(File)
... = multialignread(File, 'IgnoreGaps', IgnoreGapsValue)
```

Input Arguments

File	Multiple sequence alignment file specified by one of the following: • File name or path and file name • URL pointing to a file • MATLAB character array that contains the text of a multiple sequence alignment file You can read common multiple sequence alignment file types, such as ClustalW (`.aln`), GCG (`.msf`), and PHYLIP.
IgnoreGapsValue	Controls removing gap symbols, such as `'-'` or `'.'`, from the sequences. Choices are `true` or `false` (default).

Output Arguments

S	MATLAB structure array containing the following fields: • `Header` — Header information from the file. • `Sequence` — Amino acid or nucleotide sequences.
Headers	Cell array containing the header information from the file.
Sequences	Cell array containing the amino acid or nucleotide sequences.

Description

S = multialignread(*File*) reads a multiple sequence alignment file. The file contains multiple sequence lines that start with a sequence header followed by an optional number (not used by multialignread) and a section of the sequence. The multiple sequences are broken into blocks with the same number of blocks for every sequence. To view an example multiple sequence alignment file, type open aagag.aln at the MATLAB command line.

The output, *S*, is a structure array where *S*.Header contains the header information and *S*.Sequence contains the amino acid or nucleotide sequences.

[*Headers*, *Sequences*] = multialignread(*File*) reads the file into separate variables, *Headers* and *Sequences*, which are cell arrays containing header information and amino acid or nucleotide sequences, respectively.

... = multialignread(*File*, 'IgnoreGaps', *IgnoreGapsValue*) controls the removal of any gap symbol, such as '-' or '.', from the sequences. Choices are true or false (default).

Examples

Read a multiple sequence alignment of the gag polyprotein for several HIV strains.

```
gagaa = multialignread('aagag.aln')

gagaa =

1x16 struct array with fields:
    Header
    Sequence
```

See Also
fastaread | gethmmalignment | multialign | seqalignviewer | multialignwrite | seqconsensus | seqdisp | seqprofile

multialignwrite

Write multiple alignment to file

Syntax

```
multialignwrite(File, Alignment)
multialignwrite(..., 'Format', FormatValue, ...)
multialignwrite(..., 'Header', HeaderValue, ...)
multialignwrite(..., 'WriteCount', WriteCountValue, ...)
```

Description

`multialignwrite(File, Alignment)` writes the contents of an alignment to a ClustalW ALN-formatted (default) or MSF-formatted file.

`multialignwrite(..., 'PropertyName', PropertyValue, ...)` calls `multialignwrite` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each `PropertyName` in single quotation marks. Each `PropertyName` is case insensitive. These property name/property value pairs are as follows:

`multialignwrite(..., 'Format', FormatValue, ...)` specifies the format of the file. `FormatValue` can be `'ALN'` (default) or `'MSF'`.

`multialignwrite(..., 'Header', HeaderValue, ...)` specifies the first line of the file. The default for `HeaderValue` is `'MATLAB multiple sequence alignment'`.

`multialignwrite(..., 'WriteCount', WriteCountValue, ...)` specifies whether to add the residue counts to the end of each line. `WriteCountValue` can be `true` (default) or `false`.

Input Arguments

Alignment

An alignment, such as returned by the `multialign` function, represented by either a:

- Vector of structures, each containing the fields `Header` and `Sequence`
- Character array

Default:

File

String specifying either a file name or a path and file name for saving the data. If you specify only a file name, the file is saved to the MATLAB Current Folder browser.

Tip If you use an `.msf` extension when supplying a file name for *File*, the data is written to an MSF-formatted file. Otherwise, the data is written to a ClustalW ALN-formatted file.

Below the columns of the ClustalW ALN-formatted file, symbols can appear that denote:

- `*` — Residues or nucleotides in the column are identical in all sequences in the alignment.
- `:` — Conserved substitutions exist in the column for all sequences in the alignment.
- `.` — Semiconserved substitutions exist in the column for all sequences in the alignment.

For more information on these symbols and the groups of residues considered conserved and semiconserved, see section 12 in "Changes since version 1.6" at http://web.mit.edu/seven/src/clustalw-1.82/README.

Default:

FormatValue

String that specifies the format of *File*. Choices are `'ALN'` (default) or `'MSF'`.

Tip You can also write to an MSF-formatted file by using an `.msf` extension when supplying a file name for *File*.

Default:

HeaderValue

String that specifies the first line of the file.

Tip Use the `'Header'` property if your file header must be a specific format for a third-party software application.

Default: `'MATLAB multiple sequence alignment'`

WriteCountValue

Specifies whether to add the residue counts to the end of each line. Choices are `true` (default) or `false`.

Default:

Examples

1 Use the `fastaread` function to read `p53samples.txt`, a FASTA-formatted file included with the Bioinformatics Toolbox software, which contains seven cellular tumor antigen p53 sequences.

```
p53 = fastaread('p53samples.txt')

p53 =

7x1 struct array with fields:
    Header
    Sequence
```

2 Use the `multialign` function to align the seven cellular tumor antigen p53 sequences.

```
ma = multialign(p53,'verbose',true);
```

3 Write the alignment to a file named `p53.aln`.

```
multialignwrite('p53.aln',ma)
```

See Also

fastaread | fastawrite | gethmmalignment | multialign | multialignread | seqalignviewer | phytreewrite | seqconsensus | seqdisp | seqprofile

mzcdf2peaks

Convert mzCDF structure to peak list

Syntax

[*Peaklist*, *Times*] = mzcdf2peaks(*mzCDFStruct*)

Input Arguments

mzCDFStruct	MATLAB structure containing information from a netCDF file, such as one created by the mzcdfread function. Its fields correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with the _attributes string. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are mass_values and intensity_values fields.

Output Arguments

Peaklist	Either of the following: • Two-column matrix, where the first column contains mass/charge (m/z) values and the second column contains ion intensity values. • Cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time.
Times	Scalar of vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. If *Times* is a vector, the number of elements equals the number of peak lists contained in *Peaklist*.

Description

[*Peaklist*, *Times*] = mzcdf2peaks(*mzCDFStruct*) extracts peak information from *mzCDFStruct*, a MATLAB structure containing information from a netCDF file, such as one created by the mzcdfread function, and creates *Peaklist*, a single matrix or a cell array of matrices containing mass/charge (m/z) values and ion intensity values, and *Times*, a scalar or vector of retention times associated with a liquid chromatography/ mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set.

mzCDFStruct contains fields that correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with the _attributes string. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are mass_values and intensity_values fields.

Examples

In the following example, the file results.cdf is not provided.

1   Use the mzcdfread function to read a netCDF file into the MATLAB software as a structure. Then extract the peak information from the structure.

```
mzcdf_struct = mzcdfread('results.cdf');
[peaks,time] = mzcdf2peaks(mzcdf_struct)

peaks =

    [7008x2 single]
    [7008x2 single]
    [7008x2 single]
    [7008x2 single]

time =

     8.3430
    12.6130
    16.8830
    21.1530
```

2   Create a color map containing a color for each peak list (retention time).

```
colors = hsv(numel(peaks));
```

3 Create a 3-D figure of the peaks and add labels to it.

```
figure
hold on

for i = 1:numel(peaks)
    t = repmat(time(i),size(peaks{i},1),1);
    plot3(t,peaks{i}(:,1),peaks{i}(:,2),'color',colors(i,:))
end

view(70,60)
xlabel('Time')
ylabel(mzcdf_struct.mass_axis_label)
zlabel(mzcdf_struct.intensity_axis_label)
```
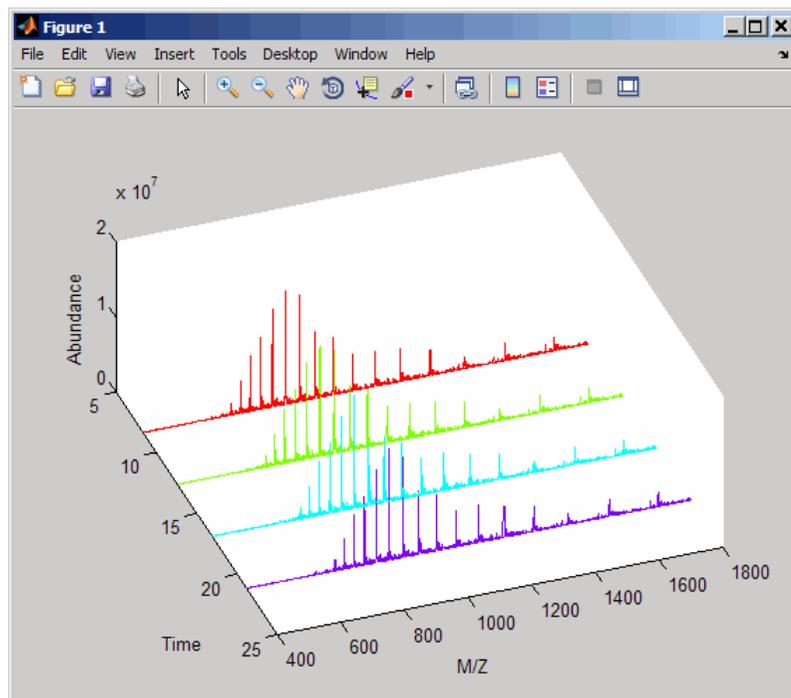
See Also
msdotplot | mspalign | msppresample | mzcdfread

mzcdfinfo

Return information about netCDF file containing mass spectrometry data

Syntax

InfoStruct = mzcdfinfo(*File*)

Input Arguments

File	String containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data and conforms to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications.
	If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

Output Arguments

InfoStruct	MATLAB structure containing information from a netCDF file. It includes the fields in the following table.

Description

InfoStruct = mzcdfinfo(*File*) returns a MATLAB structure, *InfoStruct*, containing summary information about a netCDF file, *File*.

File is a string containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data. The file must conform to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications.

InfoStruct includes the following fields.

Field	Description
`Filename`	Name of the netCDF file.
`FileTimeStamp`	Date time stamp of the netCDF file.
`FileSize`	Size of the file in bytes.
`NumberOfScans`	Number of scans in the file.
`StartTime`	Run start time.
`EndTime`	Run end time.
`TimeUnits`	Units for time.
`GlobalMassMin`	Minimum m/z value in all scans.
`GlobalMassMax`	Maximum m/z value in all scans.
`GlobalIntensityMin`	Minimum intensity value in all scans.
`GlobalIntensityMax`	Maximum intensity value in all scans.
`ExperimentType`	Indicates if data is raw or centroided.

Note: If any of the associated attributes are not in the netCDF file (because they are optional in the specifications), the value for that field will be set to `N/A` or `NaN`.

Examples

In the following example, the file `results.cdf` is not provided.

Return a MATLAB structure containing summary information about a netCDF file.

```
info = mzcdfinfo('results.cdf')

info =

            Filename: 'results.cdf'
       FileTimeStamp: '19930703134354-700'
            FileSize: 339892
       NumberOfScans: 4
           StartTime: 8.3430
             EndTime: 21.1530
           TimeUnits: 'N/A'
       GlobalMassMin: 399.9990
```

```
          GlobalMassMax: 1.8000e+003
     GlobalIntensityMin: NaN
     GlobalIntensityMax: NaN
         ExperimentType: 'Continuum Mass Spectrum'
```

See Also

mzcdfread

mzcdfread

Read mass spectrometry data from netCDF file

Syntax

```
mzCDFStruct = mzcdfread(File)
mzCDFStruct = mzcdfread(File, ...'TimeRange', TimeRangeValue, ...)
mzCDFStruct = mzcdfread(File, ...'ScanIndices',
ScanIndicesValue, ...)
mzCDFStruct = mzcdfread(File, ...'Verbose', VerboseValue, ...)
```

Input Arguments

File	String containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data and conforms to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications.
	If you specify only a file name, that file must be on the MATLAB search path or in the current folder.
TimeRangeValue	Two-element numeric array [Start End] that specifies the time range in *File* for which to read spectra. Default is to read spectra from all times [0 Inf].
	Tip Time units are indicated in the netCDF global attributes. For summary information about the time ranges in a netCDF file, use the mzcdfinfo function.
	Note: If you specify a *TimeRangeValue*, you cannot specify a *ScanIndicesValue*.
ScanIndicesValue	Positive integer, vector of integers, or a two-element numeric array [Start_Ind End_Ind] that specifies a scan, multiple scans, or a range of scans in *File* to read. Start_Ind and

	End_Ind are each positive integers indicating a scan index number. Start_Ind must be less than End_Ind. Default is to read all scans.
	Tip For information about the scan indices in a netCDF file, check the NumberOfScans field in the structure returned by the mzcdfinfo function.
	Note: If you specify a *ScanIndicesValue*, you cannot specify *TimeRangeValue*.
VerboseValue	Controls the display of the progress of the reading of *File*. Choices are true (default) or false.

Output Arguments

mzCDFStruct	MATLAB structure containing mass spectrometry information from a netCDF file. Its fields correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with the _attributes string. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are mass_values and intensity_values fields.

Description

mzCDFStruct = mzcdfread(*File*) reads a netCDF file, *File*, and then creates a MATLAB structure, *mzCDFStruct*.

File is a string containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data. The file must conform to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications.

mzCDFStruct contains fields that correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with the _attributes

string. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are `mass_values` and `intensity_values` fields.

Tip LC/MS data analysis requires extended amounts of memory from the operating system.

- If you receive errors related to memory, try the following:

  - Increase the virtual memory (swap space) for your operating system (with a recommended initial size of 3,069 and a maximum size of 16,368) as described in "Memory Usage".
  - Set the 3 GB switch (32-bit Windows XP only) as described in "Memory Usage".

- If you receive errors related to Java heap space, increase your Java heap space:
  - If you have MATLAB version 7.10 (R2010a) or later, see "Java Heap Memory Preferences"
  - If you have MATLAB version 7.9 (R2009b) or earlier, see http://www.mathworks.com/support/solutions/data/1-18I2C.html

---

mzCDFStruct = mzcdfread(*File*, ...'*PropertyName*', *PropertyValue*, ...) calls `mzcdfread` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

mzCDFStruct = mzcdfread(*File*, ...'TimeRange', *TimeRangeValue*, ...) specifies the range of time in *File* to read. *TimeRangeValue* is a two-element numeric array [Start End]. Default is to read spectra from all times [0 Inf].

Tip Time units are indicated in the netCDF global attributes. For summary information about the time ranges in a netCDF file, use the `mzcdfinfo` function.

Note: If you specify a *TimeRangeValue*, you cannot specify *ScanIndicesValue*.

---

mzCDFStruct = mzcdfread(*File*, ...'ScanIndices',
ScanIndicesValue, ...) specifies a scan, multiple scans, or range of scans in *File*
to read. *ScanIndicesValue* is a positive integer, vector of integers, or a two-element
numeric array [Start_Ind End_Ind]. Start_Ind and End_Ind are each positive
integers indicating a scan index number. Start_Ind must be less than End_Ind.
Default is to read all scans.

Tip For information about the scan indices in a netCDF file, check the NumberOfScans
field in the structure returned by the mzcdfinfo function.

Note: If you specify a *ScanIndicesValue*, you cannot specify a *TimeRangeValue*.

---

mzCDFStruct = mzcdfread(*File*, ...'Verbose', *VerboseValue*, ...) controls
the progress display when reading *File*. Choices are true (default) or false.

Examples

In the following example, the file results.cdf is not provided.

1 Read a netCDF file into the MATLAB software as a structure.

```
out = mzcdfread('results.cdf');
```

2 View the second scan in the netCDF file by creating separate variables containing
the intensity and m/z values, and then plotting these values. Add a title and *x*- and *y*-
axis labels using fields in the output structure.

```
idx1 = out.scan_index(2)+1;
idx2 = out.scan_index(3);
y = out.intensity_values(idx1:idx2);
z = out.mass_values(idx1:idx2);
stem(z,y,'marker','none')

title(sprintf('Time: %f',out.scan_acquisition_time(2)))
xlabel(out.mass_axis_units)
ylabel(out.intensity_axis_units)
```
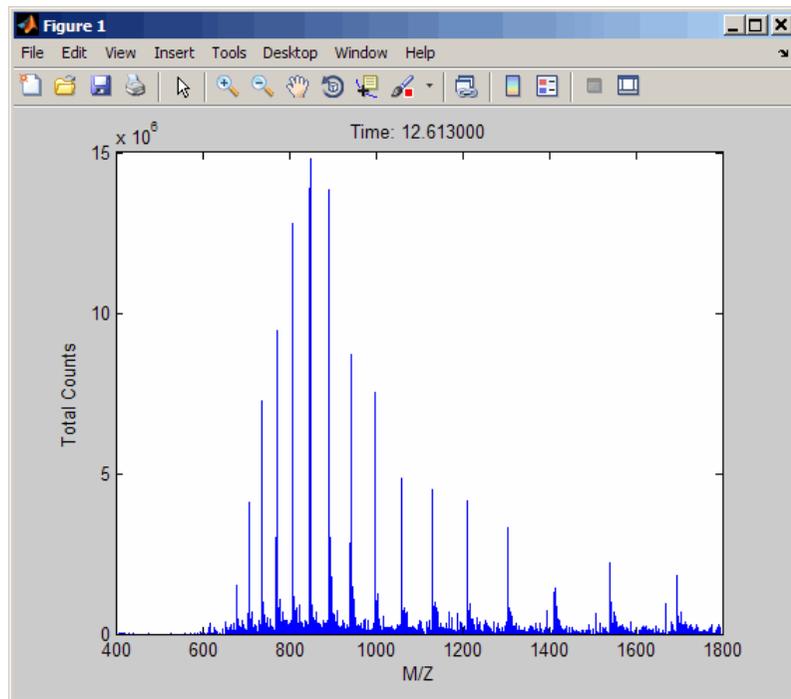
See Also

jcampread | mzcdf2peaks | mzcdfinfo | mzxmlread | tgspcread

mzxml2peaks

Convert mzXML structure to peak list

Syntax

```
[Peaklist, Times] = mzxml2peaks(mzXMLStruct)
[Peaklist, Times] = mzxml2peaks(mzXMLStruct, 'Levels', LevelsValue)
```

Input Arguments

mzXMLStruct	MATLAB structure containing information from an mzXML file, such as one created by the mzxmlread function. It includes the fields shown in the table below.
LevelsValue	Positive integer or vector of integers that specifies the level(s) of spectra in *mzXMLStruct* to convert, assuming the spectra are from tandem MS data sets. Default is 1, which converts only the first-level spectra, that is, spectra containing precursor ions. Setting *LevelsValue* to 2 converts only the second-level spectra, which are the fragment spectra (created from a precursor ion).

Output Arguments

Peaklist	Either of the following: • Two-column matrix, where the first column contains mass/charge (m/z) values and the second column contains ion intensity values. • Cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time.
Times	Vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. The number of elements in *Times* equals the number of elements in *Peaklist*.

Description

[*Peaklist*, *Times*] = mzxml2peaks(*mzXMLStruct*) extracts peak information from *mzXMLStruct*, a MATLAB structure containing information from an mzXML file, such as one created by the mzxmlread function, and creates *Peaklist*, a cell array of matrices containing mass/charge (m/z) values and ion intensity values, and *Times*, a vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. *mzXMLStruct* includes the following fields:

Field	Description
scan	Structure array containing the data pertaining to each individual scan, such as mass spectrometry level, total ion current, polarity, precursor mass (when it applies), and the spectrum data.
index	Structure containing indices to the positions of scan elements in the XML document.
mzXML	Structure containing: • Information in the root element of the mzXML schema, such as instrument details, experiment details, and preprocessing method • URLs pointing to schemas for the individual scans • Indexing approach • Digital signature calculated for the current instance of the document

[*Peaklist*, *Times*] = mzxml2peaks(*mzXMLStruct*, 'Levels', *LevelsValue*) specifies the level(s) of the spectra in *mzXMLStruct* to convert, assuming the spectra are from tandem MS data sets. Default is 1, which converts only the first-level spectra, that is, spectra containing precursor ions. Setting *LevelsValue* to 2 converts only the second-level spectra, which are the fragment spectra (created from a precursor ion).

Examples

Note: In the following example, the file results.mzxml is not provided. Sample mzXML files can be found at:

• Peptide Atlas Repository at the Institute for Systems Biology (ISB)

- The Sashimi Project

---

1 Use the `mzxmlread` function to read an mzXML file into the MATLAB software as structure. Then extract the peak information of only the first-level ions from the structure.

```
mzxml_struct = mzxmlread('results.mzxml');
[peaks,time] = mzxml2peaks(mzxml_struct);
```

2 Create a dot plot of the LC/MS data.

```
msdotplot(peaks,time)
```

See Also
msdotplot | mspalign | msppresample | mzxmlread

mzxmlinfo

Return information about mzXML file

Syntax

InfoStruct = mzxmlinfo(*File*)
InfoStruct = mzxmlinfo(*File*, 'NumOfLevels', *NumOfLevelsValue*)

Input Arguments

File	String containing a file name, or a path and file name, of an mzXML file that conforms to the mzXML 2.1 specification or earlier specifications.
	If you specify only a file name, that file must be on the MATLAB search path or in the current folder.
NumOfLevelsValue	Controls the return of NumOfLevels, an additional field in *InfoStruct*, that contains the number of mass spectrometry (MS) levels of spectra in *File*. Choices are true or false (default).

Output Arguments

| *InfoStruct* | MATLAB structure containing information from an mzXML file. It includes the fields shown in the table below. |

Description

InfoStruct = mzxmlinfo(*File*) returns a MATLAB structure, *InfoStruct*, containing summary information about an mzXML file, *File*.

File is a string containing a file name, or a path and file name, of an mzXML file. The file must conform to the mzXML 2.1 specification or earlier specifications. You can view the mzXML 2.1 specification at:

http://sashimi.sourceforge.net/schema_revision/mzXML_2.1/Doc/mzXML_2.1_tutorial.pdf

InfoStruct includes the following fields.

Field	Description
Filename	Name of the mzXML file.
FileModDate	Modification date of the file.
FileSize	Size of the file in bytes.
NumberOfScans	Number of scans in the file.*****
StartTime	Run start time.*****
EndTime	Run end time.*****
DataProcessingIntensityCutoff	Minimum mass/charge (m/z) intensity value.*****
DataProcessingCentroided	Indicates if data is centroided.*****
DataProcessingDeisotoped	Indicates if data is deisotoped.*****
DataProcessing ChargeDeconvoluted	Indicates if data is deconvoluted.*****
DataProcessingSpotIntegration	For LC/MALDI experiments, indicates if peaks eluting over multiple spots have been integrated into a single spot.*****

* — These fields contain N/A if the mzXML file does not include the associated attributes. The associated attributes are optional in the mzXML file, per the mzXML 2.1 specification.

InfoStruct = mzxmlinfo(*File*, 'NumOfLevels', *NumOfLevelsValue*) controls the return of NumOfLevels, an additional field in *mzXMLInfo*, that contains the number of mass spectrometry levels of spectra in *File*. Choices are true or false (default).

Examples

Note: In the following example, the file results.mzxml is not provided. Sample mzXML files can be found at:

- Peptide Atlas Repository at the Institute for Systems Biology (ISB)

- The Sashimi Project

Return a MATLAB structure containing summary information about an mzXML file.

```
info = mzxmlinfo('results.mzxml');

info =

                             Filename: 'results.mzxml'
                          FileModDate: '07-May-2008 13:39:12'
                             FileSize: 10607
                        NumberOfScans: 2
                            StartTime: 'PT0.00683333S'
                              EndTime: 'PT200.036S'
        DataProcessingIntensityCutoff: 'N/A'
             DataProcessingCentroided: 'false'
             DataProcessingDeisotoped: 'N/A'
      DataProcessingChargeDeconvoluted: 'N/A'
        DataProcessingSpotIntegration: 'N/A'
```

Return a MATLAB structure containing summary information, including the number of mass spectrometry levels, about an mzXML file.

```
info = mzxmlinfo('results.mzxml','numoflevels',true);

info =

                             Filename: 'results.mzxml'
                          FileModDate: '07-May-2008 13:39:12'
                             FileSize: 10607
                        NumberOfScans: 2
                            StartTime: 'PT0.00683333S'
                              EndTime: 'PT200.036S'
        DataProcessingIntensityCutoff: 'N/A'
             DataProcessingCentroided: 'false'
             DataProcessingDeisotoped: 'N/A'
      DataProcessingChargeDeconvoluted: 'N/A'
        DataProcessingSpotIntegration: 'N/A'
                      NumberOfMSLevels: 2
```

See Also
mzxmlread

mzxmlread

Read data from mzXML file

Syntax

```
mzXMLStruct = mzxmlread(myFile)
mzXMLStruct = mzxmlread(myFile,Name,Value)
```

Description

`mzXMLStruct = mzxmlread(myFile)` returns a MATLAB structure, mzXMLStruct, from an mzXML file, myFile.

`mzXMLStruct = mzxmlread(myFile,Name,Value)` reads an mzXML file, myFile, and then returns a MATLAB structure, mzXMLStruct, using additional options specified by one or more Name,Value pair arguments.

Examples

Create a MATLAB Structure from an mzXML File

In this example, the file `results_1.mzxml` is not provided. You can find sample mzXML files at:

- The Sashimi Project
- Peptide Atlas Repository at the Institute for Systems Biology (ISB)

Read an mzXML file into a MATLAB structure.

```
out = mzxmlread('results_1.mzxml')

out =

    scan: [2000x1 struct]
   mzXML: [1x1 struct]
   index: [1x1 struct]
```

View the first scan in the mzXML file by creating separate variables containing the mass-to-charge ratio (mz_ratio) and intensity (Y) values respectively. Then plot these values.

```
mz_ratio = out.scan(1).peaks.mz(1:2:end);
Y = out.scan(1).peaks.mz(2:2:end);
stem(mz_ratio,Y,'marker','none')
```
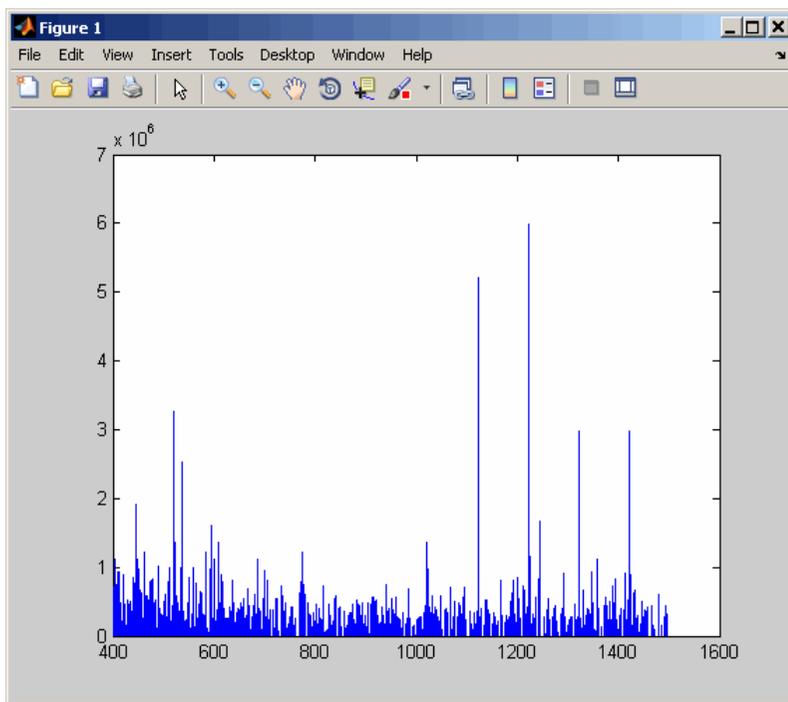

Extract One or Multiple Scans from an mzXML Structure

In this example, the file results_2.mzxml is not provided. You can find sample mzXML files at:

- The Sashimi Project
- Peptide Atlas Repository at the Institute for Systems Biology (ISB)

Read an mzXML file into a MATLAB structure, extracting a scan at index 1000.

```
out1 = mzxmlread('results_2.mzxml','ScanIndices',1000)
```

```
out1 =

     scan: [1x1 struct]
    mzXML: [1x1 struct]
    index: [1x1 struct]
```

Read an mzXML file into a MATLAB structure, extracting multiple scans at indices 1000, 1500, and 2000.

```
out2 = mzxmlread('results_2.mzxml','ScanIndices',[1000 1500 2000])

out2 =

     scan: [3x1 struct]
    mzXML: [1x1 struct]
    index: [1x1 struct]
```

Read an mzXML file into a MATLAB structure, extracting a range of scans from indices 1000 to 2000.

```
out3 = mzxmlread('results_2.mzxml','ScanIndices',[1000:2000])

out3 =

    scan: [1001x1 struct]
    mzXML: [1x1 struct]
    index: [1x1 struct]
```

Input Arguments

myFile — Input file
string containing mzXML file name

Input file, specified as a string containing an mzXML file name. The file must conform to the mzXML 2.1 or earlier specifications. You can read the mzXML 2.1 specification here:

http://sashimi.sourceforge.net/schema_revision/mzXML_2.1/Doc/mzXML_2.1_tutorial.pdf

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single

quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Levels',3,'TimeRange',[5.0 10.0]`

`'Levels'` — Spectra levels
positive integer | vector of integers

Spectra levels, specified as a positive integer or vector of integers indicating which scans to extract scans from myFile. By default, `mzxmlread` reads all spectra levels.

For summary information about the levels of spectra in an mzXML file, use the `mzxmlinfo` function.

If you are using the `'Levels'` name-value pair argument, then you cannot use `'TimeRange'` or `'ScanIndices'`.

Example: `'Levels',5`

`'TimeRange'` — Range of time
Two-element numeric array

Range of time, specified as a two-element numeric array, such as `[Start End]` indicating which scans to extract from myFile. The `Start` and `End` scalar values must be between the `startTime` and `endTime` attributes of the `msRun` element in myFile. The `Start` scalar value must be less than `End`. By default, `mzxmlread` reads all scans.

For summary information about the time ranges in an mzXML file, use the `mzxmlinfo` function.

If you are using `'TimeRange'` name-value pair argument, then you cannot use `'Levels'` or `'ScanIndices'`.

Example: `'TimeRange',[5.1 10.2]`

`'ScanIndices'` — Scan indices
positive integer | vector of positive integers

Scan indices, specified as a positive integer or vector of positive integers indicating which scans to extract from myFile. Use an integer to specify a single scan, or a vector of integers to specify multiple scans. By default, `mzxmlread` reads all scans.

For summary information about the time ranges in an mzXML file, use the `mzxmlinfo` function.

If you are using the `'ScanIndices'` name-value pair argument, then you cannot use `'Levels'` or `'TimeRange'`.

Example: `'ScanIndices',7000`

`'Verbose'` — Verbose mode
`true` (default) | `1` | `false` | `0`

Verbose mode, specified as `true` (1), or `false` (0). When `'Verbose'` is set to `true`, `mzxmlread` displays the progress while reading myFile.

Example: `'Verbose',false`

Output Arguments

`mzXMLStruct` — Structure from mzXML file
MATLAB structure

Structure from an mzXML file, returned as a MATLAB structure. `mzXMLStruct` has the following fields:

Field	Description
scan	Structure array containing the data pertaining to each individual scan, such as mass spectrometry level, total ion current, polarity, precursor mass (when it applies), and the spectrum data.
index	Structure containing indices to the positions of scan elements in the XML document.
mzXML	Structure containing all of the following: • Information in the root element of the mzXML schema, such as instrument details, experiment details, and preprocessing methods • URLs pointing to schemas for each scan • Indexing approach • Digital signature calculated for the current instance of the document

More About

Tips

LC/MS data analysis requires extended amounts of memory from the operating system.

- If you receive errors related to memory, try the following:
  - Increase the virtual memory (swap space) for your operating system (using a recommended initial size of 3,069 and a maximum size of 16,368) as described in "Memory Usage".
  - Set the 3 GB switch (32-bit Windows XP only) as described in "Memory Usage".
- If you receive errors related to Java heap space, increase your heap space:
  - If you have MATLAB 7.10 (R2010a) or later, see the following:
    "Java Heap Memory Preferences"
  - If you have MATLAB 7.9 (R2009b) or earlier, see the following:
    http://www.mathworks.com/support/solutions/data/1-18I2C.html

See Also

jcampread | mzxml2peaks | mzxmlinfo | tgspcread | xmlread

nbintest

Unpaired hypothesis test for short-read count data with small sample sizes

Syntax

```
test = nbintest(X,Y)
test = nbintest(X,Y,Name,Value)
```

Description

`test = nbintest(X,Y)` performs a hypothesis test that two independent samples of short-read count data, in each row of X and Y, come from distributions with equal means under the assumptions that:

- Short-read counts are modeled using the negative binomial distribution.
- Variance and mean of data in each row are linked through a regression function along all the rows.

X and Y must have the same number of rows and at least 2 columns, but not necessarily the same number of columns. Rows of X and Y correspond to variables, features, or genes, such as measurements of gene expression for different genes. Columns are usually time points or patients.

test is a `NegativeBinomialTest` object with two-sided p-values stored in the `pValue` property.

Use this function when you want to perform an unpaired hypothesis test for short-read count data (from high-throughput assays such as RNA-Seq or ChIP-Seq) with small sample sizes (in the order of tens at most). For instance, use this function to decide if observed differences in read counts between two conditions are significant for given genes.

`test = nbintest(X,Y,Name,Value)` uses additional options specified by one or more Name,Value pair arguments.

Note: It is recommended that you use the diagnostic plots of the `NegativeBinomialTest` object returned by `nbintest` before interpreting the p-values. These plots allow you to see if the model assumption is correct, and the variance link used is appropriate for the data.

Examples

Perform unpaired hypothesis test for short-read count data

This example shows how to perform an unpaired hypothesis test for synthetic short-read count data from two different biological conditions.

The data in this example contains synthetic gene count data for 5000 genes, representing two different biological conditions, such as diseased and normal cells. For each condition, there are five samples. Only 10% of the genes (500 genes) are differentially expressed. Specifically, half of them (250 genes) are exactly 3-fold overexpressed. The other 250 genes are 3-fold underexpressed. The rest of the gene expression data is generated from the same negative binomial distribution for both conditions. Each sample also has a different size factor (that is, the coverage or sampling depth).

Load the data.

```
clear all
load(fullfile(matlabroot,'examples','bioinfo','nbintest_data.mat'),'K','HO');
```

The variable K contains gene count data. The rows represent genes, and the columns represent samples. In this case, the first five columns represent samples from the first condition. The other five columns represent samples from the second condition. Display the first few rows of K.

```
K(1:5,:)
```

```
ans =

  Columns 1 through 6

       13683       14140        8281       14309       12208        8045
       16028       16805        9813       16486       14076        9901
         814         862         492         910         758         521
```

```
        15870        16453         9857        16454        14267         9671
         9422         9393         5734         9598         8174         5381

  Columns 7 through 10

         9446        11317        14597        14592
        10927        13348        16999        17036
          573          753          870          936
        10997        13624        17151        17205
         6315         7752         9869         9795
```

In this example, the null hypothesis is true when the gene is not differentially expressed. The variable H0 contains boolean indicators that indicate for which genes the null hypothesis is true (marked as 1). In other words, H0 contains known labels that you will use later to compare with predicted results.

```
sum(H0)
```

```
ans =

       4500
```

Out of 5000 genes, 4500 are not differentially expressed in this synthetic data.

Run an unpaired hypothesis test for samples from two conditions using nbintest. The assumption is that the data came from a negative binomial distribution, where the variance is linked to the mean via a locally-regressed smooth function of the mean as described in [1] by setting 'VarianceLink' to 'LocalRegression'.

```
tLocal = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','LocalRegression');
```

Use plotVarianceLink to plot a scatter plot for each experimental condition (for X and Y conditions), with the sample variance on the common scale versus the estimate of the condition-dependent mean. Use a linear scale for both axes. Include curves for all other linkage options by setting 'Compare' to true.

```
plotVarianceLink(tLocal,'Scale','linear','Compare',true)
```

The `Identity` line represents the Poisson model, where the variance is identical to the mean as described in [3]. Observe that the data seems to be overdispersed (that is, most points are above the `Identity` line). The `Constant` line represents the negative binomial model, where the variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. The `Local Regression` and `Constant` linkage options appear to fit better with the overdispersed data.

Use `plotChiSquaredFit` to assess the goodness-of-fit for variance regression. It plots the empirical CDF (ecdf) of the chi-squared probabilities. The probabilities are the ratio between the observed and the estimated variance stratified by short-read count levels into five equal-sized bins.

```
plotChiSquaredFit(tLocal)
```

Residuals ECDF Plot for Y

Each figure shows five ecdf curves. Each curve represents one of the five short-read count levels. For instance, the blue line represents the ecdf curve for a low short-read counts between 0 and 1264. The red line represents high counts (more than 11438).

One way to interpret the curves is to check if the ecdf curves are above the diagonal line. If they are above the line, then the variance is overestimated. If they are below the line, then the variance is underestimated. In both figures, the variance seems to be correctly estimated for higher counts (that is, the red line follows the diagonal line), but slightly overestimated for lower count levels.

To assess the performance of the hypothesis test, construct a confusion matrix using the known labels and the predicted p-values.

```
confusionmat(H0,(tLocal.pValue > .001))
```

```
ans =

         493            7
           5         4495
```

Out of 500 differentially expressed genes, 493 are correctly predicted (true positives) and 7 of them are incorrectly predicted as not-differentially expressed genes (false negatives). Out of 4500 genes that are not differentially expressed, 4495 are correctly predicted (true negatives) and 5 of them are incorrectly predicted as differentially expressed genes (false positives).

For a comparison, run the hypothesis test again assuming that counts are modeled by the Poisson distribution, where the variance is identical to the mean.

```
tPoisson = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','Identity');
```

Plot the ecdf curves. Observe that all the curves are below the diagonal line, implying that the variance is underestimated. Therefore, the negative binomial model fits the data better.

```
plotChiSquaredFit(tPoisson)
```

Input Arguments

X — Gene expression values from the first experimental condition
matrix | table

Gene expression values from the first experimental condition, specified as a matrix or table. For instance, X can represent gene expression values from cancer cells.

Note: X and Y must have the same number of rows and at least 2 columns, but not necessarily the same number of columns. Rows of X and Y correspond to genes (or

features), such as measurements of gene expression for different genes. Columns are usually time points or patients.

Y — Gene expression values from the second experimental condition
matrix | table

Gene expression values from the second experimental condition, specified as a matrix or table. For instance, Y can represent gene expression values from normal cells.

Note: X and Y must have the same number of rows and at least 2 columns, but not necessarily the same number of columns. Rows of X and Y correspond to genes (or features), such as measurements of gene expression for different genes. Columns are time points or patients.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example:

`'VarianceLink'` — Linkage type between the variance and mean
`'LocalRegression'` (default) | `'Constant'` | `'Identity'`

Linkage type between the variance and mean, specified as a comma-separated pair consisting of `'VarianceLink'` and a string. This table summarizes the available linkage options.

Linkage Option	Description
`'LocalRegression`	The variance is the sum of the shot noise term (mean) and a locally regressed nonparametric smooth function of the mean as described in [1]. This option is the default. Use this option if your data is overdispersed and has more than 1000 rows (genes).
`'Constant'`	The variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. This method uses all the rows in the data to estimate the constant. Use

Linkage Option	Description
	this option if your data is overdispersed and has less than 1000 rows.
`'Identity'`	The variance is equal to the mean as described in [3]. Counts are therefore modeled by the Poisson distribution individually for each row of X and Y. Use this option if your data has few genes and the regression between the variance and mean is not possible because of very small number of samples or replicates. This option is not recommended for overdispersed data.

Example: `'VarianceLink','Constant'`

`'PooledVariance'` — Logical flag to pool variance across both conditions
`false` (default) | `true`

Logical flag to pool variance across both conditions, specified as `true` or `false`. By default, the variance is estimated separately for each condition.

Example: `'PooledVariance',true`

`'SizeFactor'` — Size (scaling) factor of each column in X and Y
`[ ]` (default) | cell array of two vectors

Size (scaling) factor of each column in X and Y, specified as a cell array of two vectors such as `{SX,SY}`. `SX` and `SY` are numeric vectors with sizes equal to `size(X,2)` and `size(Y,2)`. `SX`, `SY`, or both can be a scalar indicating that all columns share the same size factor.

In a high-throughput sequencing library, the size factor is an estimation of the coverage or the sampling depth. The default is an empty array `[ ]`, meaning the size factor is estimated as the median of the ratio of the sample's counts to the geometric mean of each row in X or Y. Rows with zero geometric mean are ignored.

Example: `'SizeFactor',{[1.2,0.5,0.8],[0.8,1.1,1.5]}`

Output Arguments

`test` — Hypothesis test results
NegativeBinomialTest object

Hypothesis test results, returned as a `NegativeBinomialTest` object. Use this object to create diagnostic plots and access p-values.

More About

- "Negative Binomial Distribution"

References

[1] Anders, S., and Huber, W. (2010). Differential Expression Analysis for Sequence Count Data. Genome Biology, 11(10):R106.

[2] Robinson, M.D., and Smyth, G.K. (2008). Small-sample Estimation of Negative Binomial Dispersion, with Applications to SAGE data. Biostatistics, 9:321-332.

[3] Marioni, J.C., Mason, C.E., Mane, S.M., Stephens, M., and Gilad, Y. (2008). RNA-seq: an Assessment of Technical Reproducibility and Comparison with Gene Expression Arrays. Genome Research, 16:1509-1517.

See Also

`mattest` | `NegativeBinomialTest` | `plotChiSquaredFit` | `plotVarianceLink`

Using NegativeBinomialTest Objects

Unpaired hypothesis test result

A `NegativeBinomialTest` object, returned by the `nbintest` function, contains the results of an unpaired hypothesis test for short-read count data with small sample sizes. Use this object to access p-values of the test or to create diagnostic plots.

Examples

Perform unpaired hypothesis test for short-read count data

This example shows how to perform an unpaired hypothesis test for synthetic short-read count data from two different biological conditions.

The data in this example contains synthetic gene count data for 5000 genes, representing two different biological conditions, such as diseased and normal cells. For each condition, there are five samples. Only 10% of the genes (500 genes) are differentially expressed. Specifically, half of them (250 genes) are exactly 3-fold overexpressed. The other 250 genes are 3-fold underexpressed. The rest of the gene expression data is generated from the same negative binomial distribution for both conditions. Each sample also has a different size factor (that is, the coverage or sampling depth).

Load the data.

```
clear all
load(fullfile(matlabroot,'examples','bioinfo','nbintest_data.mat'),'K','HO');
```

The variable K contains gene count data. The rows represent genes, and the columns represent samples. In this case, the first five columns represent samples from the first condition. The other five columns represent samples from the second condition. Display the first few rows of K.

```
K(1:5,:)


ans =

  Columns 1 through 6
```

```
       13683         14140          8281         14309         12208          8045
       16028         16805          9813         16486         14076          9901
         814           862           492           910           758           521
       15870         16453          9857         16454         14267          9671
        9422          9393          5734          9598          8174          5381

  Columns 7 through 10

        9446         11317         14597         14592
       10927         13348         16999         17036
         573           753           870           936
       10997         13624         17151         17205
        6315          7752          9869          9795
```

In this example, the null hypothesis is true when the gene is not differentially expressed. The variable H0 contains boolean indicators that indicate for which genes the null hypothesis is true (marked as 1). In other words, H0 contains known labels that you will use later to compare with predicted results.

```
sum(H0)
```

```
ans =

      4500
```

Out of 5000 genes, 4500 are not differentially expressed in this synthetic data.

Run an unpaired hypothesis test for samples from two conditions using nbintest. The assumption is that the data came from a negative binomial distribution, where the variance is linked to the mean via a locally-regressed smooth function of the mean as described in [1] by setting 'VarianceLink' to 'LocalRegression'.

```
tLocal = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','LocalRegression');
```

Use plotVarianceLink to plot a scatter plot for each experimental condition (for X and Y conditions), with the sample variance on the common scale versus the estimate of the condition-dependent mean. Use a linear scale for both axes. Include curves for all other linkage options by setting 'Compare' to true.

```
plotVarianceLink(tLocal,'Scale','linear','Compare',true)
```

The `Identity` line represents the Poisson model, where the variance is identical to the mean as described in [3]. Observe that the data seems to be overdispersed (that is, most points are above the `Identity` line). The `Constant` line represents the negative binomial model, where the variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. The `Local Regression` and `Constant` linkage options appear to fit better with the overdispersed data.

Use `plotChiSquaredFit` to assess the goodness-of-fit for variance regression. It plots the empirical CDF (ecdf) of the chi-squared probabilities. The probabilities are the ratio between the observed and the estimated variance stratified by short-read count levels into five equal-sized bins.

```
plotChiSquaredFit(tLocal)
```

Residuals ECDF Plot for Y

Each figure shows five ecdf curves. Each curve represents one of the five short-read count levels. For instance, the blue line represents the ecdf curve for a low short-read counts between 0 and 1264. The red line represents high counts (more than 11438).

One way to interpret the curves is to check if the ecdf curves are above the diagonal line. If they are above the line, then the variance is overestimated. If they are below the line, then the variance is underestimated. In both figures, the variance seems to be correctly estimated for higher counts (that is, the red line follows the diagonal line), but slightly overestimated for lower count levels.

To assess the performance of the hypothesis test, construct a confusion matrix using the known labels and the predicted p-values.

```
confusionmat(H0,(tLocal.pValue > .001))
```

```
ans =

        493              7
          5           4495
```

Out of 500 differentially expressed genes, 493 are correctly predicted (true positives) and 7 of them are incorrectly predicted as not-differentially expressed genes (false negatives). Out of 4500 genes that are not differentially expressed, 4495 are correctly predicted (true negatives) and 5 of them are incorrectly predicted as differentially expressed genes (false positives).

For a comparison, run the hypothesis test again assuming that counts are modeled by the Poisson distribution, where the variance is identical to the mean.

```
tPoisson = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','Identity');
```

Plot the ecdf curves. Observe that all the curves are below the diagonal line, implying that the variance is underestimated. Therefore, the negative binomial model fits the data better.

```
plotChiSquaredFit(tPoisson)
```

Residuals ECDF Plot for X

Properties

`pValue` — Two-sided p-values
column vector

Two-sided p-values, specified as a column vector, for every row of the inputs to
`nbintest`.

`VarianceLink` — Linkage type between the variance and mean
`'LocalRegression'` (default) | `'Constant'` | `'Identity'`

Linkage type between the variance and mean, specified as a string. This table summarizes the available linkage options.

Linkage Option	Description
'LocalRegression'	The variance is the sum of the shot noise term (mean) and a locally regressed nonparametric smooth function of the mean as described in [1]. This option is the default. Use this option if your data contains several rows (genes), such as more than 1000 rows.
'Constant'	The variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. This method uses all the rows in the data to estimate the constant. Use this option if your data has fewer rows, that is, less than 1000 rows, and is overdispersed.
'Identity'	The variance is equal to the mean as described in [3]. Counts are therefore modeled by the Poisson distribution individually for each row of X and Y. Use this option to compare the results of the other two options.

`PooledVariance` — Logical flag to pool variance between both conditions
0 (default) | 1

Logical flag to pool variance between both conditions, specified as 1 (`true`) or 0 (`false`). The default is 0, meaning the variance is estimated separately for each condition.

`SizeFactors` — Size (scaling) factor of each column in X and Y
cell array of two vectors

Size (scaling) factor of each column in X and Y, specified as a cell array of two vectors, such as {SX,SY}. SX and SY are numeric vectors with sizes equal to `size(X,2)` and `size(Y,2)`.

Note: These properties are read-only. Run `nbintest` to change them.

Object Functions
plotVarianceLink plotChiSquaredFit

Create Object

A `NegativeBinomialTest` object is returned by the `nbintest` function, and you cannot construct this object directly.

References

[1] Anders, S., and Huber, W. (2010). Differential Expression Analysis for Sequence Count Data. Genome Biology, 11(10):R106.

[2] Robinson, M.D., and Smyth, G.K. (2008). Small-sample Estimation of Negative Binomial Dispersion, with Applications to SAGE data. Biostatistics, 9:321-332.

[3] Marioni, J.C., Mason, C.E., Mane, S.M., Stephens, M., and Gilad, Y. (2008). RNA-seq: an Assessment of Technical Reproducibility and Comparison with Gene Expression Arrays. Genome Research, 16:1509-1517.

See Also
`mattest` | `nbintest`

More About
· "Negative Binomial Distribution"

ndims (DataMatrix)

Return number of dimensions in DataMatrix object

Syntax

N = ndims(*DMObj*)

Input Arguments

DMObj	DataMatrix object, such as created by DataMatrix (object constructor).

Output Arguments

N	Positive integer representing the number of dimensions in *DMObj*. The number of dimensions in a DataMatrix object is always 2.

Description

N = ndims(*DMObj*) returns the number of dimensions in *DMObj*, a DataMatrix object. The number of dimensions in a DataMatrix object is always 2.

More About

- DataMatrix object

See Also

DataMatrix

ne (DataMatrix)

Test DataMatrix objects for inequality

Syntax

```
T = ne(DMObj1, DMObj2)
T = DMObj1 ~= DMObj2
T = ne(DMObj1, B)
T = DMObj1 ~= B
T = ne(B, DMObj1)
T = B ~= DMObj1
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

T	Logical matrix of the same size as *DMObj1* and *DMObj2* or *DMObj1* and *B*. It contains logical 1 (true) where elements in the first input are not equal to the corresponding element in the second input, and logical 0 (false) when they are equal.

Description

T = ne(*DMObj1*, *DMObj2*) or the equivalent *T* = *DMObj1* ~= *DMObj2* compares each element in DataMatrix object *DMObj1* to the corresponding element in DataMatrix object *DMObj2*, and returns *T*, a logical matrix of the same size as *DMObj1* and *DMObj2*, containing logical 1 (true) where elements in *DMObj1* are not equal to the corresponding element in *DMObj2*, and logical 0 (false) when they are equal. *DMObj1* and *DMObj2*

must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). *DMObj1* and *DMObj2* can have different Name properties.

T = ne(*DMObj1*, *B*) or the equivalent *T* = *DMObj1* ~= *B* compares each element in DataMatrix object *DMObj1* to the corresponding element in *B*, a numeric or logical array, and returns *T*, a logical matrix of the same size as *DMObj1* and *B*, containing logical 1 (true) where elements in *DMObj1* are not equal to the corresponding element in *B*, and logical 0 (false) when they are equal. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

T = ne(*B*, *DMObj1*) or the equivalent *T* = *B* ~= *DMObj1* compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are not equal to the corresponding element in *DMObj1*, and logical 0 (false) when they are equal. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls *T* = ne(*X*, *Y*) for the syntax *T* = *X* ~= *Y* when *X* or *Y* is a DataMatrix object.

More About

- DataMatrix object

See Also

DataMatrix | eq

ngsbrowser

Open NGS Browser to visualize and explore short-read sequence alignments

Syntax

```
ngsbrowser
```

Description

`ngsbrowser` opens the NGS Browser app, from which you can import:

- A single reference sequence from a FASTA-formatted file
- Short-read sequence alignment data from SAM- or BAM-formatted files or `BioMap` objects
- Annotation features from GFF- or GTF-formatted files

You can then visualize and explore the alignments and feature annotations.

Examples

Create a BioMap Object and Import it to the NGS Browser

This example shows how to create a BioMap object and display in the NGS Browser.

Create a BioMap object from a SAM-formatted file.

```
b = BioMap('ex1.sam');
```

Display the object in the NGS Browser.

```
ngsbrowser(b)
```

- Identifying Differentially Expressed Genes from RNA-Seq Data
- Exploring Protein-DNA Binding Sites from Paired-End ChIP-Seq Data

More About

Tips

- Use the NGS Browser to compare the alignment of multiple data sets to a common reference sequence.
- Use the NGS Browser to investigate regions of interest in the short-read alignment determined by various analyses, such as RNA-Seq, ChIP-Seq, and genetic variation analyses.

- "Visualize and Investigate Short-Read Alignments"
- Bowtie
- Burrows-Wheeler Aligner
- SAMtools
- NCBI Genome Database

See Also

BioMap | seqalignviewer | seqviewer

nmercount

Count n-mers in nucleotide or amino acid sequence

Syntax

Nmer = nmercount(*Seq, Length*)
Nmer = nmercount(*Seq, Length, C*)

Input Arguments

Seq	One of the following: • String of codes specifying a nucleotide sequence or amino acid sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers or the table Mapping Amino Acid Letter Codes to Integers. • MATLAB structure containing a `Sequence` field that contains a nucleotide sequence or an amino acid sequence, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, `getgenbank`, `getgenpept`, `genpeptread`, `getpdb`, or `pdbread`.
Length	Integer specifying the length of n-mer to count.

Output Arguments

Nmer	Cell array containing the n-mer counts in *Seq*.

Description

Nmer = nmercount(*Seq, Length*) counts the n-mers or patterns of a specific length in *Seq*, a nucleotide sequence or amino acid sequence, and returns the n-mer counts in a cell array.

Nmer = nmercount(*Seq*, *Length*, *C*) returns only the n-mers with cardinality of at least *C*.

Examples

1 Use the `getgenpept` function to retrieve the amino acid sequence for the human insulin receptor.

```
S = getgenpept('AAA59174','SequenceOnly',true);
```

2 Count the number of four-mers in the amino acid sequence and display the first 20 rows in the cell array.

```
nmers = nmercount(S,4);
nmers(1:20,:)

ans =
    'APES'    [2]
    'DFRD'    [2]
    'ESLK'    [2]
    'FRDL'    [2]
    'GNYS'    [2]
    'LKEL'    [2]
    'SHCQ'    [2]
    'SLKD'    [2]
    'SVRI'    [2]
    'TDYL'    [2]
    'TSLA'    [2]
    'TVIN'    [2]
    'VING'    [2]
    'VPLD'    [2]
    'YALV'    [2]
    'AAAA'    [1]
    'AAAP'    [1]
    'AAEI'    [1]
    'AAEL'    [1]
    'AAFP'    [1]
```

See Also

aacount | basecount | codoncount | dimercount

nt2aa

Convert nucleotide sequence to amino acid sequence

Syntax

SeqAA = nt2aa(*SeqNT*)
SeqAA = nt2aa(..., 'Frame', *FrameValue*, ...)
SeqAA = nt2aa(..., 'GeneticCode', *GeneticCodeValue*, ...)
SeqAA = nt2aa(..., 'AlternativeStartCodons',
AlternativeStartCodonsValue, ...)
SeqAA = nt2aa(..., 'ACGTOnly', *ACGTOnlyValue*, ...)

Input Arguments

SeqNT	One of the following:
	• String of single-letter codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers.
	• Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes.
	• MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, or `getgenbank`
	Note: Hyphens are valid only if the codon to which it belongs represents a gap, that is, the codon contains all hyphens. Example: `ACT---TGA`

	Tip Do not use a sequence with hyphens if you specify `'all'` for *FrameValue*.
FrameValue	Integer or string specifying a reading frame in the nucleotide sequence. Choices are 1, 2, 3, or `'all'`. Default is 1. If *FrameValue* is `'all'`, then *SeqAA* is a 3-by-1 cell array.
GeneticCodeValue	Integer or string specifying a genetic code number or code name from the table Genetic Code. Default is 1 or `'Standard'`. **Tip** If you use a code name, you can truncate the name to the first two letters of the name.
AlternativeStartCodonsValue	Controls the translation of alternative codons. Choices are `true` (default) or `false`.
ACGTOnlyValue	Controls the behavior of ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, and N) and unknown characters. *ACGTOnlyValue* can be `true` (default) or `false`. • If `true`, then the function errors if any of these characters are present. • If `false`, then the function tries to resolve ambiguities. If it cannot, it returns X for the affected codon.

Output Arguments

SeqAA	Amino acid sequence specified by a string of single-letter codes.

Description

SeqAA = nt2aa(*SeqNT*) converts a nucleotide sequence, specified by *SeqNT*, to an amino acid sequence, returned in *SeqAA*, using the standard genetic code.

SeqAA = nt2aa(*SeqNT*, ...'*PropertyName*', *PropertyValue*, ...) calls nt2aa with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

SeqAA = nt2aa(..., 'Frame', *FrameValue*, ...) converts a nucleotide sequence for a specific reading frame to an amino acid sequence. Choices are 1, 2, 3, or 'all'. Default is 1. If *FrameValue* is 'all', then output *SeqAA* is a 3-by-1 cell array.

SeqAA = nt2aa(..., 'GeneticCode', *GeneticCodeValue*, ...) specifies a genetic code to use when converting a nucleotide sequence to an amino acid sequence. *GeneticCodeValue* can be an integer or string specifying a code number or code name from the table Genetic Code. Default is 1 or 'Standard'. The amino acid to nucleotide codon mapping for the Standard genetic code is shown in the table Standard Genetic Code.

Tip If you use a code name, you can truncate the name to the first two letters of the name.

SeqAA = nt2aa(..., 'AlternativeStartCodons', *AlternativeStartCodonsValue*, ...) controls the translation of alternative start codons. By default, *AlternativeStartCodonsValue* is set to true, and if the first codon of a sequence is a known alternative start codon, the codon is translated to methionine.

If this option is set to false, then an alternative start codon at the start of a sequence is translated to its corresponding amino acid in the genetic code that you specify, which might not necessarily be methionine. For example, in the human mitochondrial genetic code, AUA and AUU are known to be alternative start codons.

For more information about alternative start codons, see:

www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=t#SG1

Genetic Code

Code Number	Code Name
1	Standard
2	Vertebrate Mitochondrial
3	Yeast Mitochondrial
4	Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma
5	Invertebrate Mitochondrial
6	Ciliate, Dasycladacean, and Hexamita Nuclear
9	Echinoderm Mitochondrial
10	Euplotid Nuclear
11	Bacterial and Plant Plastid
12	Alternative Yeast Nuclear
13	Ascidian Mitochondrial
14	Flatworm Mitochondrial
15	Blepharisma Nuclear
16	Chlorophycean Mitochondrial
21	Trematode Mitochondrial
22	Scenedesmus Obliquus Mitochondrial
23	Thraustochytrium Mitochondrial

Standard Genetic Code

Amino Acid Name	Amino Acid Code	Nucleotide Codon
Alanine	A	GCT GCC GCA GCG
Arginine	R	CGT CGC CGA CGG AGA AGG
Asparagine	N	ATT AAC
Aspartic acid (Aspartate)	D	GAT GAC
Cysteine	C	TGT TGC
Glutamine	Q	CAA CAG

Amino Acid Name	Amino Acid Code	Nucleotide Codon
Glutamic acid (Glutamate)	E	GAA GAG
Glycine	G	GGT GGC GGA GGG
Histidine	H	CAT CAC
Isoleucine	I	ATT ATC ATA
Leucine	L	TTA TTG CTT CTC CTA CTG
Lysine	K	AAA AAG
Methionine	M	ATG
Phenylalanine	F	TTT TTC
Proline	P	CCT CCC CCA CCG
Serine	S	TCT TCC TCA TCG AGT AGC
Threonine	T	ACT ACC ACA ACG
Tryptophan	W	TGG
Tyrosine	Y	TAT, TAC
Valine	V	GTT GTC GTA GTG
Asparagine or Aspartic acid (Aspartate)	B	Random codon from D and N
Glutamine or Glutamic acid (Glutamate)	Z	Random codon from E and Q
Unknown amino acid (any amino acid)	X	Random codon
Translation stop	*	TAA TAG TGA
Gap of indeterminate length	-	- - -
Unknown character (any character or symbol not in table)	?	???

SeqAA = nt2aa(..., 'ACGTOnly', *ACGTOnlyValue*, ...) controls the behavior of ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, and N) and unknown

characters. *ACGTOnlyValue* can be `true` (default) or `false`. If `true`, then the function errors if any of these characters are present. If `false`, then the function tries to resolve ambiguities. If it cannot, it returns X for the affected codon.

Examples

Converting the ND1 Gene

1 Use the `getgenbank` function to retrieve genomic information for the human mitochondrion from the GenBank database and store it in a MATLAB structure .

```
mitochondria = getgenbank('NC_012920')

mitochondria =

                 LocusName: 'NC_012920'
       LocusSequenceLength: '16569'
      LocusNumberofStrands: ''
             LocusTopology: 'circular'
         LocusMoleculeType: 'DNA'
     LocusGenBankDivision: 'PRI'
    LocusModificationDate: '05-MAR-2010'
                Definition: 'Homo sapiens mitochondrion, complete genome.'
                 Accession: 'NC_012920 AC_000021'
                   Version: 'NC_012920.1'
                        GI: '251831106'
                   Project: []
                    DBLink: 'Project:30353'
                  Keywords: []
                   Segment: []
                    Source: 'mitochondrion Homo sapiens (human)'
              SourceOrganism: [4x65 char]
                 Reference: {1x7 cell}
                   Comment: [24x67 char]
                  Features: [933x74 char]
                       CDS: [1x13 struct]
                  Sequence: [1x16569 char]
                 SearchURL: [1x70 char]
               RetrieveURL: [1x104 char]
```

2 Determine the name and location of the first gene in the human mitochondrion.

```
mitochondria.CDS(1).gene
```

```
ans =

ND1

mitochondria.CDS(1).location

ans =

3307..4262
```

3 Extract the sequence for the ND1 gene from the nucleotide sequence.

```
ND1gene = mitochondria.Sequence(3307:4262);
```

4 Convert the ND1 gene on the human mitochondria genome to an amino acid sequence using the Vertebrate Mitochondrial genetic code.

```
protein1 = nt2aa(ND1gene,'GeneticCode', 2);
```

5 Use the `getgenpept` function to retrieve the same amino acid sequence from the GenPept database.

```
protein2 = getgenpept('YP_003024026', 'SequenceOnly', true);
```

6 Use the `isequal` function to compare the two amino acid sequences.

```
isequal (protein1, protein2)

ans =

     1
```

Converting the ND2 Gene

1 Use the `getgenbank` function to retrieve the nucleotide sequence for the human mitochondrion from the GenBank database.

```
mitochondria = getgenbank('NC_012920');
```

2 Determine the name and location of the second gene in the human mitochondrion.

```
mitochondria.CDS(2).gene

ans =

ND2

mitochondria.CDS(2).location
```

```
ans =

4470..5511
```

3 Extract the sequence for the ND2 gene from the nucleotide sequence.

```
ND2gene = mitochondria.Sequence(4470:5511);
```

4 Convert the ND2 gene on the human mitochondria genome to an amino acid sequence using the Vertebrate Mitochondrial genetic code.

```
protein1 = nt2aa(ND2gene,'GeneticCode', 2);
```

Note: In the ND2gene nucleotide sequence, the first codon is ATT, which is translated to M, while the subsequent ATT codons are translated to I. If you set 'AlternativeStartCodons' to false, then the first ATT codon is translated to I, the corresponding amino acid in the Vertebrate Mitochondrial genetic code.

---

5 Use the getgenpept function to retrieve the same amino acid sequence from the GenPept database.

```
protein2 = getgenpept('YP_003024027', 'SequenceOnly', true);
```

6 Use the isequal function to compare the two amino acid sequences.

```
isequal (protein1, protein2)

ans =

    1
```

Converting a Sequence with Ambiguous Characters

If you have a sequence with ambiguous or unknown nucleotide characters, you can set the 'ACGTOnly' property to false to have the nt2aa function try to resolve them:

```
nt2aa('agttgccgacgcgcncar','ACGTOnly', false)

ans =

SCRRAQ
```

See Also

aa2nt | aminolookup | baselookup | codonbias | dnds | dndsml | geneticcode | isotopicdist | revgeneticcode | seqviewer

nt2int

Convert nucleotide sequence from letter to integer representation

Syntax

```
SeqInt = nt2int(SeqChar)
SeqInt = nt2int(SeqChar, ...'Unknown', UnknownValue, ...)
SeqInt = nt2int(SeqChar, ...'ACGTOnly', ACGTOnlyValue, ...)
```

Input Arguments

SeqChar	One of the following: • String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers. Integers are arbitrarily assigned to IUB/IUPAC letters. • MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, or `getgenbank`.
UnknownValue	Integer to represent unknown nucleotides. Choices are integers ≥ 0 and ≤ 255. Default is `0`.
ACGTOnlyValue	Controls the prohibition of ambiguous nucleotides. Choices are `true` or `false` (default). If *ACGTOnlyValue* is `true`, you can enter only the characters A, C, G, T, and U.

Output Arguments

SeqInt	Nucleotide sequence specified by a row vector of integers.

Description

SeqInt = nt2int(*SeqChar*) converts *SeqChar*, a string of codes specifying a nucleotide sequence, to *SeqInt*, a row vector of integers specifying the same nucleotide sequence. For valid codes, see the table Mapping Nucleotide Letter Codes to Integers. Unknown characters (characters not in the table) are mapped to 0. Gaps represented with hyphens are mapped to 16.

SeqInt = nt2int(*SeqChar*, ...'*PropertyName*', *PropertyValue*, ...) calls nt2int with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

SeqInt = nt2int(*SeqChar*, ...'Unknown', *UnknownValue*, ...) specifies an integer to represent unknown nucleotides. *UnknownValue* can be an integer ≥ 0 and ≤ 255. Default is 0.

SeqInt = nt2int(*SeqChar*, ...'ACGTOnly', *ACGTOnlyValue*, ...) controls the prohibition of ambiguous nucleotides (N, R, Y, K, M, S, W, B, D, H, and V). Choices are true or false (default). If *ACGTOnlyValue* is true, you can enter only the characters A, C, G, T, and U.

Mapping Nucleotide Letter Codes to Integers

Nucleotide	Code	Integer
Adenosine	A	1
Cytidine	C	2
Guanine	G	3
Thymidine	T	4
Uridine (if 'Alphabet' set to 'RNA')	U	4
Purine (A or G)	R	5
Pyrimidine (T or C)	Y	6
Keto (G or T)	K	7
Amino (A or C)	M	8
Strong interaction (3 H bonds) (G or C)	S	9

Nucleotide	Code	Integer
Weak interaction (2 H bonds) (A or T)	W	10
Not A (C or G or T)	B	11
Not C (A or G or T)	D	12
Not G (A or C or T)	H	13
Not T or U (A or C or G)	V	14
Any nucleotide (A or C or G or T or U)	N	15
Gap of indeterminate length	-	16
Unknown (any character not in table)	*	0 (default)

Examples

Converting a Simple Sequence

Convert a nucleotide sequence from letters to integers.

```
s = nt2int('ACTGCTAGC')

s =
    1    2    4    3    2    4    1    3    2
```

Converting a Random Sequence

1 Create a random character string to represent a nucleotide sequence.

```
SeqChar = randseq(20)

SeqChar =

TTATGACGTTATTCTACTTT
```

2 Convert the nucleotide sequence from letter to integer representation.

```
SeqInt = nt2int(SeqChar)

SeqInt =

  Columns 1 through 13
    4    4    1    4    3    1    2    3    4    4    1    4    4
```

```
Columns 14 through 20
   2    4    1    2    4    4    4
```

See Also
aa2int | baselookup | int2aa | int2nt

ntdensity

Plot density of nucleotides along sequence

Syntax

```
ntdensity(SeqNT)
Density = ntdensity(SeqNT)
... = ntdensity(..., 'Window', WindowValue, ...)
[Density, HighCG] = ntdensity(..., 'CGThreshold',
CGThresholdValue, ...)
```

Arguments

SeqNT	One of the following: • String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers. • Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes. • MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `emblread`, `fastaread`, `fastqread`, `genbankread`, `getembl`, or `getgenbank`. **Note:** Although you can submit a sequence with nucleotides other than A, C, G, and T, `ntdensity` plots only A, C, G, and T.
WindowValue	Value that specifies the window length for the density calculation. Default is `length(SeqNT)/20`.
CGThresholdValue	Controls the return of indices for regions where the CG content of *SeqNT* is greater than *CGThresholdValue*. Default is 5.

Description

ntdensity(*SeqNT*) plots the density of nucleotides A, C, G, and T in sequence *SeqNT*.

Density = ntdensity(*SeqNT*) returns a MATLAB structure with the density of nucleotides A, C, G, and T.

... = ntdensity(*SeqNT*, ...'*PropertyName*', *PropertyValue*, ...) calls ntdensity with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = ntdensity(..., 'Window', *WindowValue*, ...) uses a window of length *WindowValue* for the density calculation. Default *WindowValue* is length(*SeqNT*)/20.

[*Density*, *HighCG*] = ntdensity(..., 'CGThreshold', *CGThresholdValue*, ...) returns indices for regions where the CG content of *SeqNT* is greater than *CGThresholdValue*. Default *CGThresholdValue* is 5.

Examples

1 Create a random character string to represent a nucleotide sequence.

```
s = randseq(1000, 'alphabet', 'dna');
```

2 Plot the density of nucleotides along the sequence.

```
ntdensity(s)
```
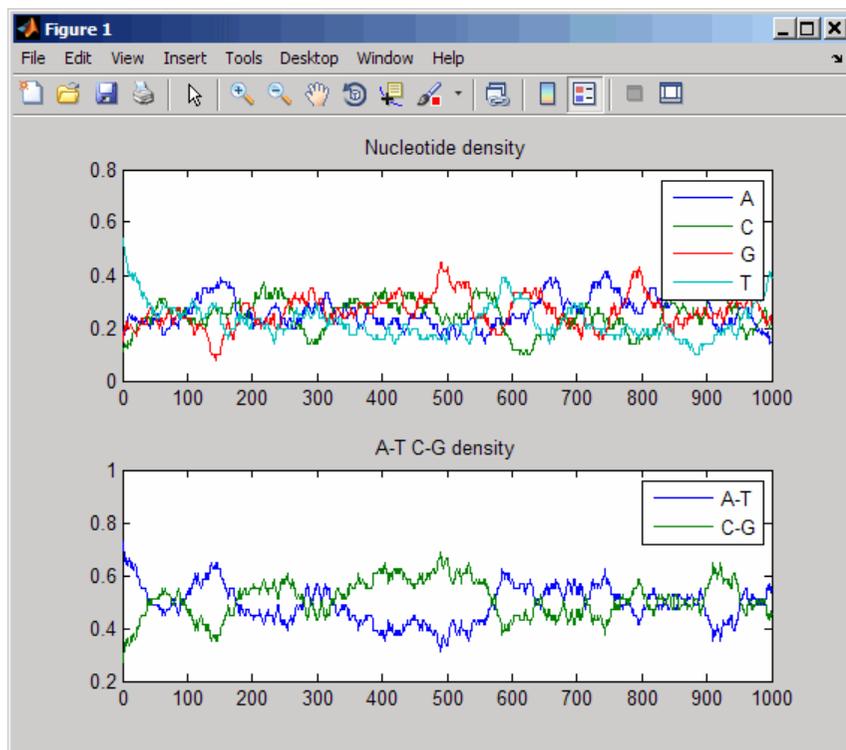
See Also
basecount | codoncount | cpgisland | dimercount | filter

nuc44

Return NUC44 scoring matrix for nucleotide sequences

Syntax

ScoringMatrix = nuc44
[*ScoringMatrix*, *MatrixInfo*] = nuc44

Description

ScoringMatrix = nuc44 returns the scoring matrix. The nuc44 scoring matrix uses ambiguous nucleotide codes and probabilities rounded to the nearest integer.

Scale = 0.277316

Expected score = -1.7495024, Entropy = 0.5164710 bits

Lowest score = -4, Highest score = 5

Order: A C G T R Y K M S W B D H V N

[*ScoringMatrix*, *MatrixInfo*] = nuc44 returns a structure with information about the matrix with fields Name and Order.

Note: The NUC44 scoring matrix is supplied by NCBI and is used by the BLAST suite of programs. For more information, see ftp://ftp.ncbi.nih.gov/blast/matrices/.

See Also

blosum | dayhoff | gonnet | localalign | nwalign | pam | swalign

num2goid

Convert numbers to Gene Ontology IDs

Syntax

GOIDs = num2goid(*X*)

Description

GOIDs = num2goid(*X*) converts the numbers in *X* to strings with Gene Ontology IDs. IDs are seven-digit numbers preceded by the prefix GO:, which is the standard used by the Gene Ontology database.

Examples

Get the Gene Ontology IDs of the following numbers.

```
t = [5575 5622 5623 5737 5840 30529 43226 43228 43229 43232 43234];
ids = num2goid(t)
```

Columns 1 through 4

```
    'GO:0005575'    'GO:0005622'    'GO:0005623'    'GO:0005737'
```

  Columns 5 through 8

```
    'GO:0005840'    'GO:0030529'    'GO:0043226'    'GO:0043228'
```

  Columns 9 through 11

```
    'GO:0043229'    'GO:0043232'    'GO:0043234'
```

See Also

geneont.getancestors | geneont.getmatrix | geneont | goannotread |
geneont | geneont.getdescendants | geneont.getrelatives

numel (DataMatrix)

Return number of elements in DataMatrix object

Syntax

```
N = numel(DMObj)
Ns = numel(DMObj, Index1, Index2)
```

Input Arguments

DMObj	DataMatrix object, such as created by `DataMatrix` (object constructor).
Index1	A row or range of rows in *DMObj* specified by a positive integer or a range using the format `x:y`, where `x` is the first row and `y` is the last row.
Index2	A column or range of columns in *DMObj* specified by a positive integer or a range using the format `x:y`, where `x` is the first column and `y` is the last column.

Output Arguments

N	Positive integer representing the number of elements in *DMObj*, a DataMatrix object.
Ns	Positive integer representing the number of subscripted elements in *DMObj*, a DataMatrix object.

Description

N = `numel(DMObj)` returns 1. To find the number of elements in *DMObj*, a DataMatrix object, use either of the following syntaxes:

```
prod(size(DMObj))
```

```
numel(DMObj,':',':')
```

Ns = numel(*DMObj, Index1, Index2*) returns the number of subscripted elements in *DMObj*, a DataMatrix object. *Index1* specifies a row or range of rows in *DMObj*. *Index2* specifies a column or range of columns in *DMObj*.

More About

·   DataMatrix object

See Also

```
DataMatrix
```

nwalign

Globally align two sequences using Needleman-Wunsch algorithm

Syntax

```
Score = nwalign(Seq1,Seq2)
[Score, Alignment] = nwalign(Seq1,Seq2)
[Score, Alignment, Start] = nwalign(Seq1,Seq2)
... = nwalign(Seq1,Seq2, ...'Alphabet', AlphabetValue, ...)
... =
nwalign(Seq1,Seq2, ...'ScoringMatrix', ScoringMatrixValue, ...)
... = nwalign(Seq1,Seq2, ...'Scale', ScaleValue, ...)
... = nwalign(Seq1,Seq2, ...'GapOpen', GapOpenValue, ...)
... = nwalign(Seq1,Seq2, ...'ExtendGap', ExtendGapValue, ...)
... = nwalign(Seq1,Seq2, ...'Glocal', GlocalValue, ...)
... = nwalign(Seq1,Seq2, ...'Showscore', ShowscoreValue, ...)
```

Input Arguments

Seq1, *Seq2*	Amino acid or nucleotide sequences. Enter any of the following: • Character string of letters representing amino acids or nucleotides, such as returned by `int2aa` or `int2nt` • Vector of integers representing amino acids or nucleotides, such as returned by `aa2int` or `nt2int` • Structure containing a `Sequence` field --- **Tip** For help with letter and integer representations of amino acids and nucleotides, see Amino Acid Lookup or Nucleotide Lookup.
AlphabetValue	String specifying the type of sequence. Choices are `'AA'` (default) or `'NT'`.
ScoringMatrixValue	Either of the following:

- String specifying the scoring matrix to use for the global alignment. Choices for amino acid sequences are:

  - `'BLOSUM62'`
  - `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'`
  - `'BLOSUM100'`
  - `'PAM10'` increasing by 10 up to `'PAM500'`
  - `'DAYHOFF'`
  - `'GONNET'`

  Default is:

  - `'BLOSUM50'` — When *AlphabetValue* equals `'AA'`
  - `'NUC44'` — When *AlphabetValue* equals `'NT'`

  Note: The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the `'Scale'` property to specify an additional scale factor to convert the output score from bits to another unit.

- Matrix representing the scoring matrix to use for the global alignment, such as returned by the `blosum`, `pam`, `dayhoff`, `gonnet`, or `nuc44` function.

  Note: If you use a scoring matrix that you created or was created by one of the above functions, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix. You can use the `'Scale'` property to specify a scale factor to convert the output score to another unit.

Note: If you need to compile `nwalign` into a stand-alone application or software component using

	MATLAB Compiler, use a matrix instead of a string for *ScoringMatrixValue*.
ScaleValue	Positive value that specifies a scale factor that is applied to the output score. For example, if the output score is initially determined in bits, and you enter `log(2)` for *ScaleValue*, then `nwalign` returns *Score* in nats. Default is `1`, which does not change the units of the output score. **Note:** If the `'ScoringMatrix'` property also specifies a scale factor, then `nwalign` uses it first to scale the output score, then applies the scale factor specified by *ScaleValue* to rescale the output score. **Tip** Before comparing alignment scores from multiple alignments, ensure the scores are in the same units. You can use the `'Scale'` property to control the units of the output scores.
GapOpenValue	Positive value specifying the penalty for opening a gap in the alignment. Default is `8`.
ExtendGapValue	Positive value specifying the penalty for extending a gap using the affine gap penalty scheme. **Note:** If you specify this value, `nwalign` uses the affine gap penalty scheme, that is, it scores the first gap using the *GapOpenValue* and scores subsequent gaps using the *ExtendGapValue*. If you do not specify this value, `nwalign` scores all gaps equally, using the *GapOpenValue* penalty.
GlocalValue	Controls the return of a semiglobal or "glocal" alignment. In a semiglobal alignment, gap penalties at the end of the sequences are null. Choices are `true` or `false` (default).

ShowscoreValue	Controls the display of the scoring space and the winning path of the alignment. Choices are `true` or `false` (default).

Output Arguments

Score	Optimal global alignment score in bits.
Alignment	3-by-N character array showing the two sequences, *Seq1* and *Seq2*, in the first and third rows, and symbols representing the optimal global alignment for them in the second row.
Start	2-by-1 vector of indices indicating the starting point in each sequence for the alignment. Because this is a global alignment, *Start* is always `[1;1]`.

Description

Score = nwalign(*Seq1*,*Seq2*) returns the optimal global alignment score in bits. The scale factor used to calculate the score is provided by the scoring matrix.

[*Score, Alignment*] = nwalign(*Seq1,Seq2*) returns a 3-by-N character array showing the two sequences, *Seq1* and *Seq2*, in the first and third rows, and symbols representing the optimal global alignment for them in the second row. The symbol | indicates amino acids or nucleotides that match exactly. The symbol : indicates amino acids or nucleotides that are related as defined by the scoring matrix (nonmatches with a zero or positive scoring matrix value).

[*Score, Alignment, Start*] = nwalign(*Seq1,Seq2*) returns a 2-by-1 vector of indices indicating the starting point in each sequence for the alignment. Because this is a global alignment, *Start* is always `[1;1]`.

... = nwalign(*Seq1,Seq2*, ...'*PropertyName*', *PropertyValue*, ...) calls nwalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = nwalign(*Seq1*,*Seq2*, ...'Alphabet', *AlphabetValue*, ...) specifies the type of sequences. Choices are 'AA' (default) or 'NT'.

... = nwalign(*Seq1*,*Seq2*, ...'ScoringMatrix', *ScoringMatrixValue*, ...) specifies the scoring matrix to use for the global alignment. Default is:

- 'BLOSUM50' — When *AlphabetValue* equals 'AA'
- 'NUC44' — When *AlphabetValue* equals 'NT'

... = nwalign(*Seq1*,*Seq2*, ...'Scale', *ScaleValue*, ...) specifies a scale factor that is applied to the output score, thereby controlling the units of the output score. Choices are any positive value.

... = nwalign(*Seq1*,*Seq2*, ...'GapOpen', *GapOpenValue*, ...) specifies the penalty for opening a gap in the alignment. Choices are any positive value. Default is 8.

... = nwalign(*Seq1*,*Seq2*, ...'ExtendGap', *ExtendGapValue*, ...) specifies the penalty for extending a gap using the affine gap penalty scheme. Choices are any positive value.

... = nwalign(*Seq1*,*Seq2*, ...'Glocal', *GlocalValue*, ...) controls the return of a semiglobal or "glocal" alignment. In a semiglobal alignment, gap penalties at the end of the sequences are null. Choices are true or false (default).

... = nwalign(*Seq1*,*Seq2*, ...'Showscore', *ShowscoreValue*, ...) controls the display of the scoring space and winning path of the alignment. Choices are true or false (default).
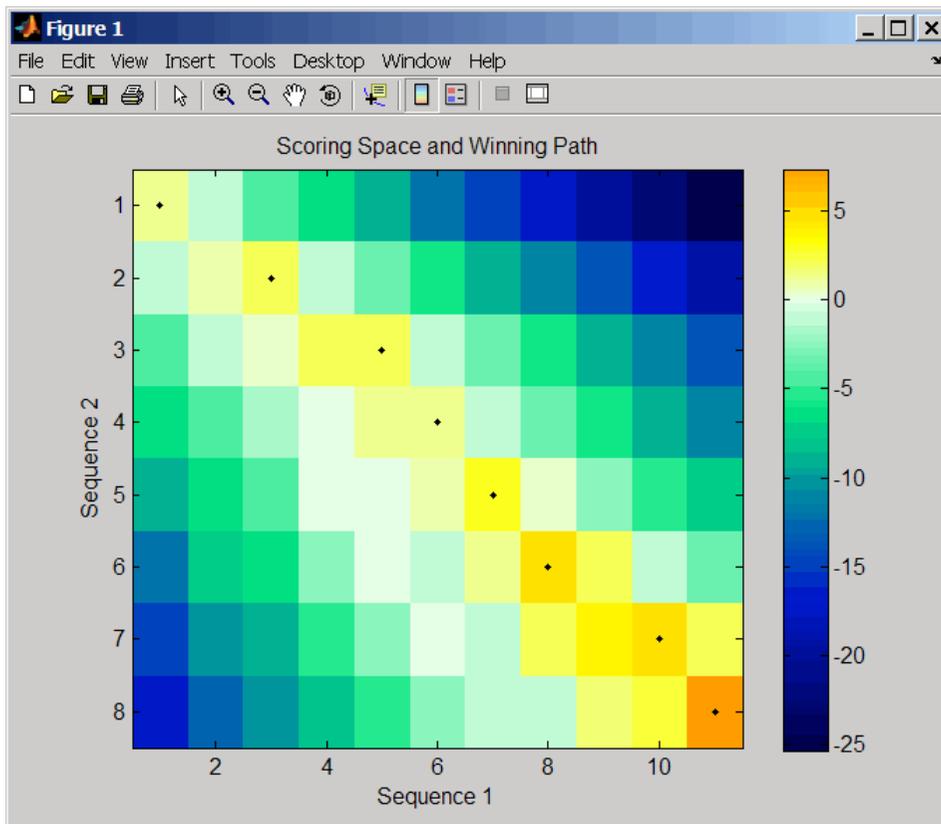
The scoring space is a heat map displaying the best scores for all the partial alignments of two sequences. The color of each (n1,n2) coordinate in the scoring space represents the best score for the pairing of subsequences Seq1(1:n1) and Seq2(1:n2), where n1 is a position in Seq1 and n2 is a position in Seq2. The best score for a pairing of specific subsequences is determined by scoring all possible alignments of the subsequences by summing matches and gap penalties.

The winning path is represented by black dots in the scoring space, and it illustrates the pairing of positions in the optimal global alignment. The color of the last point (lower right) of the winning path represents the optimal global alignment score for the two sequences and is the *Score* output returned by nwalign.

Note: The scoring space visually indicates if there are potential alternate winning paths, which is useful when aligning sequences with big gaps. Visual patterns in the scoring space can also indicate a possible sequence rearrangement.

Examples

1 Globally align two amino acid sequences using the `BLOSUM50` (default) scoring matrix and the default values for the `GapOpen` and `ExtendGap` properties. Return the optimal global alignment score in bits and the alignment character array.

```
[Score, Alignment] = nwalign('VSPAGMASGYD','IPGKASYD')
Score =

    7.3333

Alignment =

VSPAGMASGYD
: | | || ||
I-P-GKAS-YD
```

2 Globally align two amino acid sequences specifying the `PAM250` scoring matrix and a gap open penalty of `5`.

```
[Score, Alignment] = nwalign('IGRHRYHIGG','SRYIGRG',...
                             'scoringmatrix','pam250',...
                             'gapopen',5)
Score =

    2.3333

Alignment =

IGRHRYHIG-G
 :  || || |
-S--RY-IGRG
```

3 Globally align two amino acid sequences returning the *Score* in nat units (nats) by specifying a scale factor of `log(2)`.

```
[Score, Alignment] = nwalign('HEAGAWGHEE','PAWHEAE','Scale',log(2))

Score =
```

```
     0.2310

  Alignment =

  HEAGAWGHE-E
     || || |
  --P-AW-HEAE
```

References

[1] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). Biological Sequence
        Analysis (Cambridge University Press).

See Also

aa2int | aminolookup | baselookup | blosum | dayhoff | gonnet | int2aa |
int2nt | localalign | multialign | nt2aa | nt2int | nuc44 | pam | profalign
| seqdotplot | showalignment | swalign

oligoprop

Calculate sequence properties of DNA oligonucleotide

Syntax

```
SeqProperties = oligoprop(SeqNT)
SeqProperties = oligoprop(SeqNT, ...'Salt', SaltValue, ...)
SeqProperties = oligoprop(SeqNT, ...'Temp', TempValue, ...)
SeqProperties = oligoprop(SeqNT, ...'Primerconc',
PrimerconcValue, ...)
SeqProperties = oligoprop(SeqNT, ...'HPBase', HPBaseValue, ...)
SeqProperties = oligoprop(SeqNT, ...'HPLoop', HPLoopValue, ...)
SeqProperties = oligoprop(SeqNT, ...'Dimerlength',
DimerlengthValue, ...)
```

Input Arguments

SeqNT	DNA oligonucleotide sequence represented by any of the following:
	• Character string containing the letters A, C, G, T, or N
	• Vector of integers containing the integers 1, 2, 3, 4, or 15
	• Structure containing a Sequence field that contains a nucleotide sequence
SaltValue	Value that specifies a salt concentration in moles/liter for melting temperature calculations. Default is 0.05 moles/liter.
TempValue	Value that specifies the temperature in degrees Celsius for nearest-neighbor calculations of free energy. Default is 25 degrees Celsius.
PrimerconcValue	Value that specifies the concentration in moles/liter for melting temperature calculations. Default is 50e-6 moles/liter.
HPBaseValue	Value that specifies the minimum number of paired bases that form the neck of the hairpin. Default is 4 base pairs.

HPLoopValue	Value that specifies the minimum number of bases that form the loop of a hairpin. Default is 2 bases.
DimerlengthValue	Value that specifies the minimum number of aligned bases between the sequence and its reverse. Default is 4 bases.

Output Arguments

SeqProperties	Structure containing the sequence properties for a DNA oligonucleotide.

Description

SeqProperties = oligoprop(*SeqNT*) returns the sequence properties for a DNA oligonucleotide as a structure with the following fields:

Field	Description
GC	Percent GC content for the DNA oligonucleotide. Ambiguous N characters in *SeqNT* are considered to potentially be any nucleotide. If *SeqNT* contains ambiguous N characters, GC is the midpoint value, and its uncertainty is expressed by GCdelta.
GCdelta	The difference between GC (midpoint value) and either the maximum or minimum value GC could assume. The maximum and minimum values are calculated by assuming all N characters are G/C or not G/C, respectively. Therefore, GCdelta defines the possible range of GC content.
Hairpins	H-by-length(*SeqNT*) matrix of characters displaying all potential hairpin structures for the sequence *SeqNT*. Each row is a potential hairpin structure of the sequence, with the hairpin forming nucleotides designated by capital letters. H is the number of potential hairpin structures for the sequence. Ambiguous N characters in *SeqNT* are considered to potentially complement any nucleotide.
Dimers	D-by-length(*SeqNT*) matrix of characters displaying all potential dimers for the sequence *SeqNT*. Each row is a potential dimer of the sequence, with the self-dimerizing

Field	Description
	nucleotides designated by capital letters. D is the number of potential dimers for the sequence. Ambiguous N characters in *SeqNT* are considered to potentially complement any nucleotide.
MolWeight	Molecular weight of the DNA oligonucleotide. Ambiguous N characters in *SeqNT* are considered to potentially be any nucleotide. If *SeqNT* contains ambiguous N characters, MolWeight is the midpoint value, and its uncertainty is expressed by MolWeightdelta.
MolWeightdelta	The difference between MolWeight (midpoint value) and either the maximum or minimum value MolWeight could assume. The maximum and minimum values are calculated by assuming all N characters are G or C, respectively. Therefore, MolWeightdelta defines the possible range of molecular weight for *SeqNT*.
Tm	A vector with melting temperature values, in degrees Celsius, calculated by six different methods, listed in the following order: • Basic (Marmur et al., 1962) • Salt adjusted (Howley et al., 1979) • Nearest-neighbor (Breslauer et al., 1986) • Nearest-neighbor (SantaLucia Jr. et al., 1996) • Nearest-neighbor (SantaLucia Jr., 1998) • Nearest-neighbor (Sugimoto et al., 1996) Ambiguous N characters in *SeqNT* are considered to potentially be any nucleotide. If *SeqNT* contains ambiguous N characters, Tm is the midpoint value, and its uncertainty is expressed by Tmdelta.
Tmdelta	A vector containing the differences between Tm (midpoint value) and either the maximum or minimum value Tm could assume for each of the six methods. Therefore, Tmdelta defines the possible range of melting temperatures for *SeqNT*.

Field	Description
Thermo	4-by-3 matrix of thermodynamic calculations. The rows correspond to nearest-neighbor parameters from: • Breslauer et al., 1986 • SantaLucia Jr. et al., 1996 • SantaLucia Jr., 1998 • Sugimoto et al., 1996 The columns correspond to: • delta H — Enthalpy in kilocalories per mole, kcal/mol • delta S — Entropy in calories per mole-degrees Kelvin, cal/(K)(mol) • delta G — Free energy in kilocalories per mole, kcal/mol Ambiguous N characters in *SeqNT* are considered to potentially be any nucleotide. If *SeqNT* contains ambiguous N characters, Thermo is the midpoint value, and its uncertainty is expressed by Thermodelta.
Thermodelta	4-by-3 matrix containing the differences between Thermo (midpoint value) and either the maximum or minimum value Thermo could assume for each calculation and method. Therefore, Thermodelta defines the possible range of thermodynamic values for *SeqNT*.

SeqProperties = oligoprop(*SeqNT*, ...'*PropertyName*', *PropertyValue*, ...) calls oligoprop with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

SeqProperties = oligoprop(*SeqNT*, ...'Salt', *SaltValue*, ...) specifies a salt concentration in moles/liter for melting temperature calculations. Default is 0.05 moles/liter.

SeqProperties = oligoprop(*SeqNT*, ...'Temp', *TempValue*, ...) specifies the temperature in degrees Celsius for nearest-neighbor calculations of free energy. Default is 25 degrees Celsius.

SeqProperties = oligoprop(*SeqNT*, ...'Primerconc', *PrimerconcValue*, ...) specifies the concentration in moles/liter for melting temperatures. Default is 50e-6 moles/liter.

SeqProperties = oligoprop(*SeqNT*, ...'HPBase', *HPBaseValue*, ...) specifies the minimum number of paired bases that form the neck of the hairpin. Default is 4 base pairs.

SeqProperties = oligoprop(*SeqNT*, ...'HPLoop', *HPLoopValue*, ...) specifies the minimum number of bases that form the loop of a hairpin. Default is 2 bases.

SeqProperties = oligoprop(*SeqNT*, ...'Dimerlength', *DimerlengthValue*, ...) specifies the minimum number of aligned bases between the sequence and its reverse. Default is 4 bases.

Examples

Calculating Properties for a DNA Sequence

1 Create a random sequence.

```
seq = randseq(25)

seq =

TAGCTTCATCGTTGACTTCTACTAA
```

2 Calculate sequence properties of the sequence.

```
S1 = oligoprop(seq)

S1 =

               GC: 36
          GCdelta: 0
         Hairpins: [0x25 char]
            Dimers: 'tAGCTtcatcgttgacttctactaa'
        MolWeight: 7.5820e+003
   MolWeightdelta: 0
               Tm: [52.7640 60.8629 62.2493 55.2870 54.0293 61.0614]
```

```
       Tmdelta: [0 0 0 0 0 0]
         Thermo: [4x3 double]
    Thermodelta: [4x3 double]
```

3 List the thermodynamic calculations for the sequence.

```
S1.Thermo

ans =

 -178.5000  -477.5700   -36.1125
 -182.1000  -497.8000   -33.6809
 -190.2000  -522.9000   -34.2974
 -191.9000  -516.9000   -37.7863
```

Calculating Properties for a DNA Sequence with Ambiguous Characters

1 Calculate sequence properties of the sequence ACGTAGAGGACGTN.

```
S2 = oligoprop('ACGTAGAGGACGTN')

S2 =

               GC: 53.5714
          GCdelta: 3.5714
         Hairpins: 'ACGTagaggACGTn'
            Dimers: [3x14 char]
         MolWeight: 4.3329e+003
    MolWeightdelta: 20.0150
                Tm: [38.8357 42.2958 57.7880 52.4180 49.9633 55.1330]
           Tmdelta: [1.4643 1.4643 10.3885 3.4633 0.2829 3.8074]
            Thermo: [4x3 double]
       Thermodelta: [4x3 double]
```

2 List the potential dimers for the sequence.

```
S2.Dimers

ans =

ACGTagaggacgtn
ACGTagaggACGTn
acgtagagGACGTN
```

More About

- isoelectric
- molweight

- ntdensity
- randseq

References

[1] Breslauer, K.J., Frank, R., Blöcker, H., and Marky, L.A. (1986). Predicting DNA duplex stability from the base sequence. Proceedings of the National Academy of Science USA *83*, 3746–3750.

[2] Chen, S.H., Lin, C.Y., Cho, C.S., Lo, C.Z., and Hsiung, C.A. (2003). Primer Design Assistant (PDA): A web-based primer design tool. Nucleic Acids Research *31(13)*, 3751–3754.

[3] Howley, P.M., Israel, M.A., Law, M., and Martin, M.A. (1979). A rapid method for detecting and mapping homology between heterologous DNAs. Evaluation of polyomavirus genomes. The Journal of Biological Chemistry *254(11)*, 4876–4883.

[4] Marmur, J., and Doty, P. (1962). Determination of the base composition of deoxyribonucleic acid from its thermal denaturation temperature. Journal Molecular Biology *5*, 109–118.

[5] Panjkovich, A., and Melo, F. (2005). Comparison of different melting temperature calculation methods for short DNA sequences. Bioinformatics *21(6)*, 711–722.

[6] SantaLucia Jr., J., Allawi, H.T., and Seneviratne, P.A. (1996). Improved Nearest-Neighbor Parameters for Predicting DNA Duplex Stability. Biochemistry *35*, 3555–3562.

[7] SantaLucia Jr., J. (1998). A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proceedings of the National Academy of Science USA *95*, 1460–1465.

[8] Sugimoto, N., Nakano, S., Yoneyama, M., and Honda, K. (1996). Improved thermodynamic parameters and helix initiation factor to predict stability of DNA duplexes. Nucleic Acids Research *24(22)*, 4501–4505.

[9] http://www.basic.northwestern.edu/biotools/oligocalc.html for weight calculations.

See Also
palindromes

palindromes

Find palindromes in sequence

Syntax

```
[Position, Length] = palindromes(SeqNT)
[Position, Length, Pal] = palindromes(SeqNT)
... = palindromes(SeqNT, ..., 'Length', LengthValue, ...)
... = palindromes(SeqNT, ..., 'Complement', ComplementValue, ...)
```

Arguments

SeqNT	One of the following: • String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers. • Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes. • MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `emblread`, `fastaread`, `fastqread`, `genbankread`, `getembl`, or `getgenbank`.
LengthValue	Integer specifying a minimum length for palindromes. Default is `6`.
ComplementValue	Controls the return of complementary palindromes, that is, where the elements match their complementary pairs `A`-`T` (or `U`) and `C`-`G` instead of an exact nucleotide match. Choices are `true` or `false` (default).

Description

[*Position*, *Length*] = palindromes(*SeqNT*) finds all palindromes in sequence *SeqNT* with a length greater than or equal to 6, and returns the starting indices, *Position*, and the lengths of the palindromes, *Length*.

[*Position*, *Length*, *Pal*] = palindromes(*SeqNT*) also returns a cell array, *Pal*, of the palindromes.

... = palindromes(*SeqNT*, ...'*PropertyName*', *PropertyValue*, ...) calls palindromes with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = palindromes(*SeqNT*, ..., 'Length', *LengthValue*, ...) finds all palindromes longer than or equal to *LengthValue*. Default is 6.

... = palindromes(*SeqNT*, ..., 'Complement', *ComplementValue*, ...) controls the return of complementary palindromes, that is, where the elements match their complementary pairs A-T (or A-U) and C-G instead of an exact nucleotide match. Choices for *ComplementValue* are true or false (default).

Examples

Find the palindromes in a simple nucleotide sequence.

```
[p,l,s] = palindromes('GCTAGTAACGTATATATAAT')

p =
    11
    12
l =
     7
     7
s =
    'TATATAT'
    'ATATATA'
```

Find the complementary palindromes in a simple nucleotide sequence.

```
[pc,lc,sc] = palindromes('TAGCTTGTCACTGAGGCCA',...
                          'Complement',true)
pc =
     8
lc =
     7
sc =
    'TCACTGA'
```

Find the palindromes in a random nucleotide sequence.

```
a = randseq(100)

a =
TAGCTTCATCGTTGACTTCTACTAA
AAGCAAGCTCCTGAGTAGCTGGCCA
AGCGAGCTTGCTTGTGCCCGGCTGC
GGCGGTTGTATCCTGAATACGCCAT

[pos,len,pal]=palindromes(a)

pos =
    74
len =
     6
pal =
    'GCGGCG'
```

See Also

seqcomplement | seqrcomplement | seqreverse | seqshowwords | regexp |
strfind

pam

Return Point Accepted Mutation (PAM) scoring matrix

Syntax

```
ScoringMatrix = pam(N)
[ScoringMatrix, MatrixInfo] = pam(N)
... = pam(N, ...'Extended', ExtendedValue, ...)
... = pam(N, ...'Order', OrderValue, ...)
```

Arguments

N	Integer specifying the PAM scoring matrix to return. Choices are `10:10:500`. **Tip** Entering a larger value for *N* allows for sequence alignments with larger evolutionary distances.
ExtendedValue	Controls the return of the ambiguous characters (B, Z, and X), and the stop character (`*`), in addition to the `20` standard amino acid characters. Choices are `true` or `false` (default).
OrderValue	String that controls the order of amino acids in the scoring matrix. Choices are a string with at least the `20` standard amino acids. The default order of the output is A R N D C Q E G H I L K M F P S T W Y V B Z X `*`. If *OrderValue* does not contain the characters B, Z, X, and `*`, then these characters are not returned.

Description

ScoringMatrix = `pam(N)` returns the PAM*N* scoring matrix for amino acid sequences.

[*ScoringMatrix, MatrixInfo*] = pam(*N*) returns a structure with information about the PAM matrix. The fields in the structure are Name, Scale, Entropy, Expected, and Order.

... = pam(*N*, ...'*PropertyName*', *PropertyValue*, ...) calls pam with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = pam(*N*, ...'Extended', *ExtendedValue*, ...) controls the return of the ambiguous characters (B, Z, and X), and the stop character (*), in addition to the 20 standard amino acid characters. Choices are true or false (default).

... = pam(*N*, ...'Order', *OrderValue*, ...) controls the order of amino acids in the returned scoring matrix. Choices are a string with at least the 20 standard amino acids. The default ordering of the output is A R N D C Q E G H I L K M F P S T W Y V B Z X *. If *OrderValue* does not contain the extended characters B, Z, X, and *, then these characters are not returned.

PAM50 substitution matrix in 1/2 bit units, Expected score = -3.70, Entropy = 2.00 bits, Lowest score = -13, Highest score = 13.

PAM250 substitution matrix in 1/3 bit units, Expected score = -0.844, Entropy = 0.354 bits, Lowest score = -8, Highest score = 17.

Examples

Return the PAM50 matrix.

```
PAM50 = pam(50)
```

Return the PAM250 matrix and specify the order of amino acids in the matrix.

```
PAM250 = pam(250,'Order','CSTPAGNDEQHRKMILVFYW')
```

See Also

blosum | dayhoff | gonnet | localalign | nuc44 | nwalign | swalign

pdbdistplot

Visualize intermolecular distances in Protein Data Bank (PDB) file

Syntax

```
pdbdistplot(PDBid)
pdbdistplot(PDBid, Distance)
```

Arguments

PDBid	Any of the following: • String specifying a unique identifier for a protein structure record • Name of a variable for a MATLAB structure containing PDB information for a molecular structure, such as returned by `getpdb` or `pdbread`. • Name of file containing PDB information for a molecular structure, such as created by `getpdb` with the `'ToFile'` property. **Note:** Each structure in the PDB database is represented by a four-character alphanumeric identifier. For example, `4hhb` is the identification code for hemoglobin.
Distance	Threshold distance in angstroms shown on a spy plot. Default is `7`.

Description

`pdbdistplot` displays the distances between atoms and amino acids in a PDB structure.

`pdbdistplot(PDBid)` retrieves information for the structure specified by *PDBid* from the Protein Data Bank (PDB) database. Creates a heat map showing interatom distances and a spy plot showing the residues where the minimum distances apart are less than `7` angstroms.

pdbdistplot(*PDBid, Distance*) specifies the threshold distance shown on a spy plot. Default is 7.

Examples

Display a heat map of the interatom distances and a spy plot at 7 angstroms of the protein cytochrome C from albacore tuna.

```
pdbdistplot('5CYT');
```
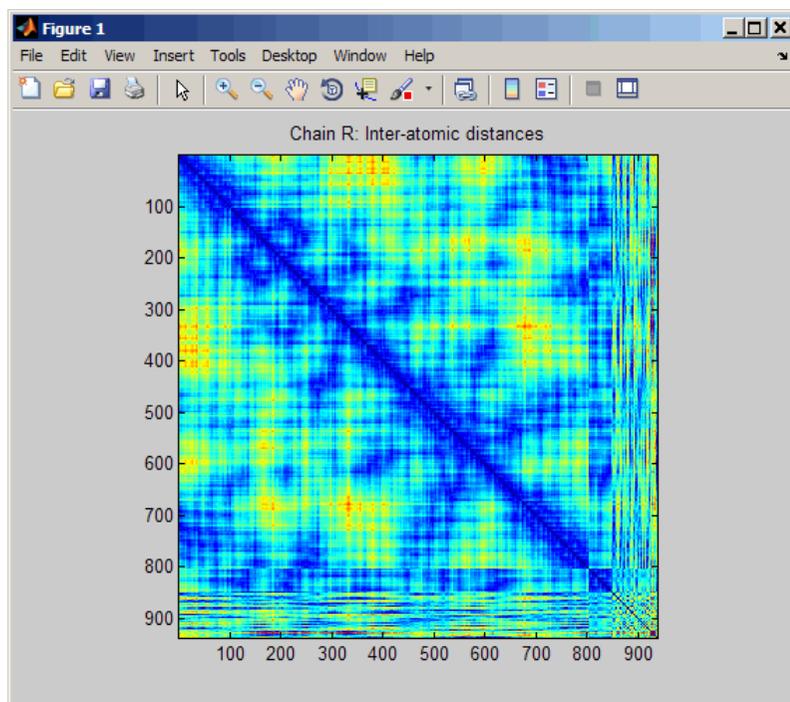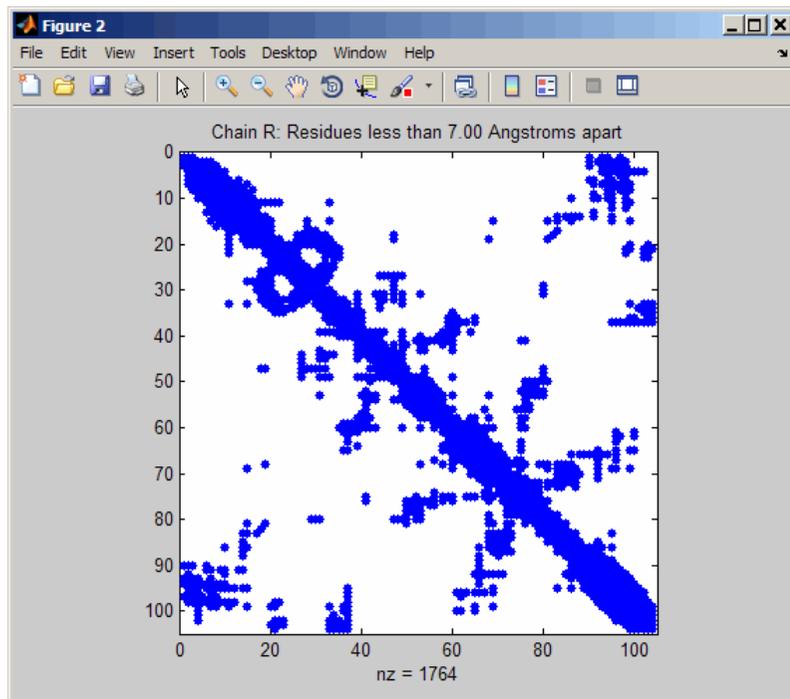
Display a spy plot at 10 angstroms of the same structure.

```
pdbdistplot('5CYT',10);
```
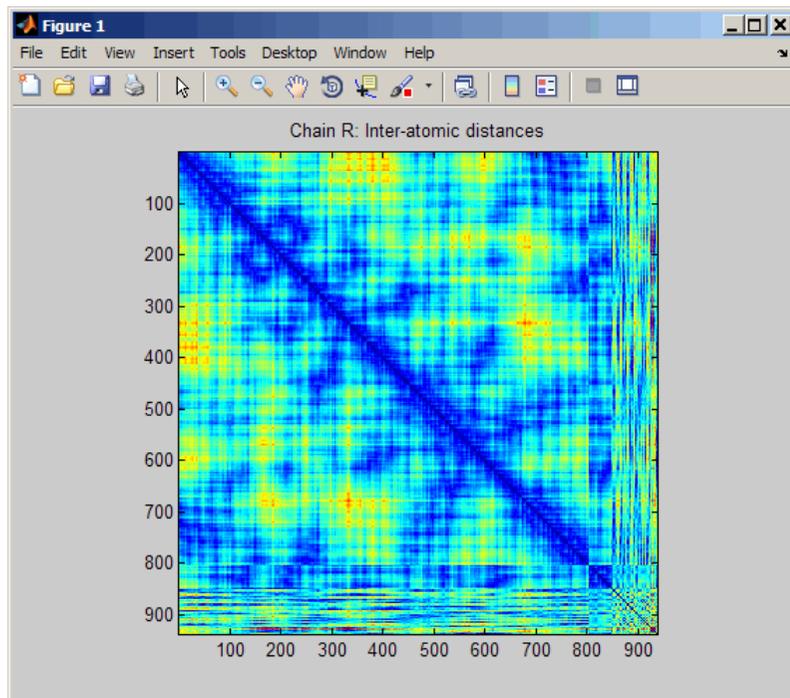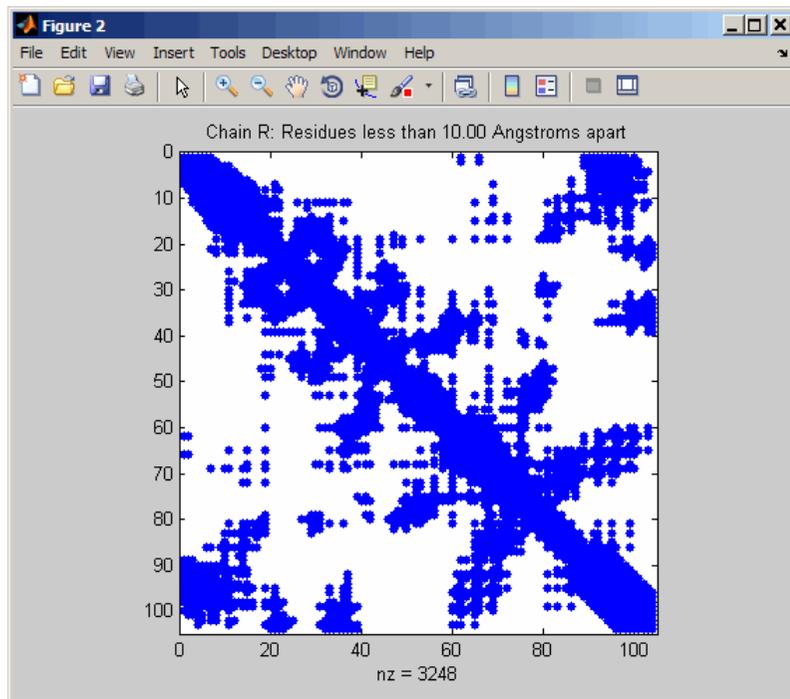
See Also

getpdb | molviewer | pdbread | proteinplot | ramachandran

pdbread

Read data from Protein Data Bank (PDB) file

Syntax

PDBStruct = pdbread(*File*)
PDBStruct = pdbread(*File*, 'ModelNum', *ModelNumValue*)

Input Arguments

File	Either of the following: • String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a Protein Data Bank (PDB)-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder. • MATLAB character array that contains the text of a PDB-formatted file. **Tip** You can use the getpdb function with the 'ToFile' property to retrieve protein structure data from the PDB database and create a PDB-formatted file.
ModelNumValue	Positive integer specifying a model in a PDB-formatted file.

Output Arguments

PDBStruct	MATLAB structure containing a field for each PDB record.

Description

The Protein Data Bank (PDB) database is an archive of experimentally determined 3-D biological macromolecular structure data. For more information about the PDB format, see:

http://www.wwpdb.org/documentation/format23/v2.3.html

PDBStruct = pdbread(*File*) reads the data from PDB-formatted text file *File* and stores the data in the MATLAB structure, *PDBStruct*, which contains a field for each PDB record. The following table summarizes the possible PDB records and the corresponding fields in the MATLAB structure *PDBStruct*:

PDB Database Record	Field in the MATLAB Structure
HEADER	Header
OBSLTE	Obsolete
TITLE	Title
CAVEAT	Caveat
COMPND	Compound
SOURCE	Source
KEYWDS	Keywords
EXPDTA	ExperimentData
AUTHOR	Authors
REVDAT	RevisionDate
SPRSDE	Superseded
JRNL	Journal
REMARK 1	Remark1
REMARK *N* **Note:** *N* equals 2 through 999.	Remark*n* **Note:** *n* equals 2 through 999.
DBREF	DBReferences
SEQADV	SequenceConflicts
SEQRES	Sequence

PDB Database Record	Field in the MATLAB Structure
FTNOTE	Footnote
MODRES	ModifiedResidues
HET	Heterogen
HETNAM	HeterogenName
HETSYN	HeterogenSynonym
FORMUL	Formula
HELIX	Helix
SHEET	Sheet
TURN	Turn
SSBOND	SSBond
LINK	Link
HYDBND	HydrogenBond
SLTBRG	SaltBridge
CISPEP	CISPeptides
SITE	Site
CRYST1	Cryst1
ORIGXn	OriginX
SCALEn	Scale
MTRIXn	Matrix
TVECT	TranslationVector
MODEL	Model
ATOM	Atom
SIGATM	AtomSD
ANISOU	AnisotropicTemp
SIGUIJ	AnisotropicTempSD
TER	Terminal
HETATM	HeterogenAtom

PDB Database Record	Field in the MATLAB Structure
CONECT	Connectivity

PDBStruct = pdbread(*File*, 'ModelNum', *ModelNumValue*) reads only the model specified by *ModelNumValue* from the PDB-formatted text file *File* and stores the data in the MATLAB structure *PDBStruct*. If *ModelNumValue* does not correspond to an existing mode number in *File*, then pdbread reads the coordinate information of all the models.

The Sequence Field

The Sequence field is also a structure containing sequence information in the following subfields:

- NumOfResidues
- ChainID
- ResidueNames — Contains the three-letter codes for the sequence residues.
- Sequence — Contains the single-letter codes for the sequence residues.

Note: If the sequence has modified residues, then the ResidueNames subfield might not correspond to the standard three-letter amino acid codes. In this case, the Sequence subfield will contain the modified residue code in the position corresponding to the modified residue. The modified residue code is provided in the ModifiedResidues field.

The Model Field

The Model field is also a structure or an array of structures containing coordinate information. If the MATLAB structure contains one model, the Model field is a structure containing coordinate information for that model. If the MATLAB structure contains multiple models, the Model field is an array of structures containing coordinate information for each model. The Model field contains the following subfields:

- Atom
- AtomSD
- AnisotropicTemp
- AnisotropicTempSD

- Terminal
- HeterogenAtom

The Atom Field

The Atom field is also an array of structures containing the following subfields:

- AtomSerNo
- AtomName
- altLoc
- resName
- chainID
- resSeq
- iCode
- X
- Y
- Z
- occupancy
- tempFactor
- segID
- element
- charge
- AtomNameStruct — Contains three subfields: chemSymbol, remoteInd, and branch.

Examples

1 Use the getpdb function to retrieve structure information from the Protein Data Bank (PDB) for the nicotinic receptor protein with identifier 1abt, and then save the data to the PDB-formatted file nicotinic_receptor.pdb in the MATLAB Current Folder.

```
getpdb('1abt', 'ToFile', 'nicotinic_receptor.pdb');
```

2 Read the data from the `nicotinic_receptor.pdb` file into a MATLAB structure `pdbstruct`.

```
pdbstruct = pdbread('nicotinic_receptor.pdb');
```

3 Read only the second model from the `nicotinic_receptor.pdb` file into a MATLAB structure `pdbstruct_Model2`.

```
pdbstruct_Model2 = pdbread('nicotinic_receptor.pdb', 'ModelNum', 2);
```

4 View the atomic coordinate information in the model fields of both MATLAB structures `pdbstruct` and `pdbstruct_Model2`.

```
pdbstruct.Model

ans =

1x4 struct array with fields:
    MDLSerNo
    Atom
    Terminal

pdbstruct_Model2.Model

ans =

    MDLSerNo: 2
        Atom: [1x1205 struct]
    Terminal: [1x2 struct]
```

5 Read the data from a URL into a MATLAB structure, `gfl_pdbstruct`.

```
gfl_pdbstruct = pdbread('http://www.rcsb.org/pdb/files/1gfl.pdb');
```

See Also
genpeptread | getpdb | molviewer | pdbdistplot | pdbsuperpose |
pdbtransform | pdbwrite

pdbsuperpose

Superpose 3-D structures of two proteins

Syntax

```
pdbsuperpose(PDB1, PDB2)
Dist = pdbsuperpose(PDB1, PDB2)
[Dist, RMSD] = pdbsuperpose(PDB1, PDB2)
[Dist, RMSD, Transf] = pdbsuperpose(PDB1, PDB2)
[Dist, RMSD, Transf, PBD2TX] = pdbsuperpose(PDB1, PDB2)
... = pdbsuperpose(..., 'ModelNum', ModelNumValue, ...)
... = pdbsuperpose(..., 'Scale', ScaleValue, ...)
... = pdbsuperpose(..., 'Translate', TranslateValue, ...)
... = pdbsuperpose(..., 'Reflection', ReflectionValue, ...)
... = pdbsuperpose(..., 'SeqAlign', SeqAlignValue, ...)
... = pdbsuperpose(..., 'Segment', SegmentValue, ...)
... = pdbsuperpose(..., 'Apply', ApplyValue, ...)
... = pdbsuperpose(..., 'Display', DisplayValue, ...)
```

Input Arguments

PDB1, *PDB2*	Protein structures represented by any of the following:
	• String specifying a unique identifier for a protein structure record in the Protein Data Bank (PDB) database.
	• Variable containing a PDB-formatted MATLAB structure, such as returned by getpdb or pdbread.
	• String specifying a file name or, a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.
ModelNumValue	Two-element numeric array whose elements correspond to models in *PDB1* and *PDB2* respectively when *PDB1* or *PDB2* contains multiple models. It specifies the models to consider in

	the superposition. By default, the first model in each structure is considered.
ScaleValue	Specifies whether to include a scaling component in the linear transformation. Choices are `true` or `false` (default).
TranslateValue	Specifies whether to include a translation component in the linear transformation. Choices are `true` (default) or `false`.
ReflectionValue	Specifies whether to include a reflection component in the linear transformation. Choices are: • `true` — Include reflection component. • `false` — Exclude reflection component. • `'best'` — Default. May or may not include the reflection component, depending on the best fit solution.
SeqAlignValue	Specifies whether to perform a local sequence alignment and then use only the portions of the structures corresponding to the segments that align to compute the linear transformation. Choices are `true` (default) or `false`. **Note:** If you set the `'SeqAlign'` property to `true`, you can also specify the following properties used by the `swalign` function: • `'ScoringMatrix'` • `'GapOpen'` • `'ExtendGap'` For more information on these properties, see `swalign`.

SegmentValue	Specifies the boundaries and the chain of two subsequences to consider for computing the linear transformation. *SegmentValue* is a cell array of strings with the following format: `{'start1-stop1:chain1', 'start2-stop2:chain2'}` You can omit the boundaries to indicate the entire chain, such as in `{'chain1', 'start2-stop2:chain2'}`. You can specify only one pair of segments at any given time, and the specified segments are assumed to contain the same number of alpha carbon atoms.
ApplyValue	Specifies the extent to which the linear transformation should be applied. Choices are: • `'all'` — Default. Apply the linear transformation to the entire PDB2 structure. • `'chain'` — Apply the linear transformation to the specified chain only. • `'segment'` — Apply the linear transformation to the specified segment only.
DisplayValue	Specifies whether to display the original *PDB1* structure and the resulting transformed *PDB2TX* structure in the Molecule Viewer window using the `molviewer` function. Each structure is represented as a separate model. Choices are `true` (default) or `false`.

Output Arguments

Dist	Value representing a dissimilarity measure given by the sum of the squared errors between *PDB1* and *PDB2*. For more information, see `procrustes` in the Statistics and Machine Learning Toolbox documentation.
RMSD	Scalar representing the root mean square distance between the coordinates of the *PDB1* structure and the transformed *PDB2* structure, considering only the atoms used to compute the linear transformation.

Transf	Linear transformation computed to superpose the chain of *PDB2* to the chain of *PDB1*. *Transf* is a MATLAB structure with the following fields: • T — Orthogonal rotation and reflection component. • b — Scale component. • c — Translation component. **Note:** Only alpha carbon atom coordinates are used to compute the linear transformation. **Tip** You can use the *Transf* output as input to the pdbtransform function.
PDB2TX	PDB-formatted MATLAB structure that represents the coordinates in the transformed *PDB2* protein structure.

Description

pdbsuperpose(*PDB1*, *PDB2*) computes and applies a linear transformation to superpose the coordinates of the protein structure represented in *PDB2* to the coordinates of the protein structure represented in *PDB1*. *PDB1* and *PDB2* are protein structures represented by any of the following:

- String specifying a unique identifier for a protein structure record in the PDB database.
- Variable containing a PDB-formatted MATLAB structure, such as returned by getpdb or pdbread.
- String specifying a file name or a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.

Alpha carbon atom coordinates of single chains for each structure are considered to compute the linear transformation (translation, reflection, orthogonal rotation, and scaling). By default, the first chain in each structure is considered to compute the transformation, and the transformation is applied to the entire molecule. By default, the

original *PDB1* structure and the resulting transformed *PDB2* structure are displayed as separate models in the Molecule Viewer window using the `molviewer` function.

Dist = pdbsuperpose(*PDB1, PDB2*) returns a dissimilarity measure given by the sum of the squared errors between *PDB1* and *PDB2*. For more information, see `procrustes`.

[*Dist, RMSD*] = pdbsuperpose(*PDB1, PDB2*) also returns *RMSD*, the root mean square distance between the coordinates of the *PDB1* structure and the transformed *PDB2* structure, considering only the atoms used to compute the linear transformation.

[*Dist, RMSD, Transf*] = pdbsuperpose(*PDB1, PDB2*) also returns *Transf*, the linear transformation computed to superpose the chain of *PDB2* to the chain of *PDB1*. *Transf* is a MATLAB structure with the following fields:

- `T` — Orthogonal rotation and reflection component.
- `b` — Scale component.
- `c` — Translation component.

Note: Only alpha carbon atom coordinates are used to compute the linear transformation.

[*Dist, RMSD, Transf, PBD2TX*] = pdbsuperpose(*PDB1, PDB2*) also returns *PBD2TX*, a PDB-formatted MATLAB structure that represents the coordinates in the transformed *PDB2* protein structure.

... = pdbsuperpose(..., '*PropertyName*', *PropertyValue*, ...) calls `pdbsuperpose` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = pdbsuperpose(..., 'ModelNum', *ModelNumValue*, ...) specifies the models to consider in the superposition when *PDB1* or *PDB2* contains multiple models. *ModelNumValue* is a two-element numeric array whose elements correspond to the models in *PDB1* and *PDB2* respectively. By default, the first model in each structure is considered.

... = pdbsuperpose(..., 'Scale', *ScaleValue*, ...) specifies whether to include a scaling component in the linear transformation. Choices are true or false (default).

... = pdbsuperpose(..., 'Translate', *TranslateValue*, ...) specifies whether to include a translation component in the linear transformation. Choices are true (default) or false.

... = pdbsuperpose(..., 'Reflection', *ReflectionValue*, ...) specifies whether to include a reflection component in the linear transformation. Choices are true (include reflection component), false (exclude reflection component), or 'best' (may or may not include the reflection component, depending on the best fit solution). Default is 'best'.

... = pdbsuperpose(..., 'SeqAlign', *SeqAlignValue*, ...) specifies whether to perform a local sequence alignment and then use only the portions of the structures corresponding to the segments that align to compute the linear transformation. Choices are true (default) or false.

Note: If you set the 'SeqAlign' property to true, you can also specify the following properties used by the swalign function:

- 'ScoringMatrix'
- 'GapOpen'
- 'ExtendGap'

For more information on these properties, see swalign.

... = pdbsuperpose(..., 'Segment', *SegmentValue*, ...) specifies the boundaries and the chain of two subsequences to consider for computing the linear transformation. *SegmentValue* is a cell array of strings with the following format: {'start1-stop1:chain1', 'start2-stop2:chain2'}. You can omit the boundaries to indicate the entire chain, such as in {'chain1', 'start2-stop2:chain2'}. You can specify only one pair of segments at any given time, and the specified segments are assumed to contain the same number of alpha carbon atoms.

... = pdbsuperpose(..., 'Apply', *ApplyValue*, ...) specifies the extent to which the linear transformation should be applied. Choices are 'all' (apply the linear transformation to the entire PDB2 structure), 'chain' (apply the linear transformation

to the specified chain only), or `'segment'` (apply the linear transformation to the specified segment only). Default is `'all'`.

`... = pdbsuperpose(..., 'Display', DisplayValue, ...)` specifies whether to display the original *PDB1* structure and the resulting transformed *PDB2TX* structure in the Molecule Viewer window using the `molviewer` function. Each structure is represented as a separate model. Choices are `true` (default) or `false`.

Examples

Superposing Two Hemoglobin Structures

1   Use the `getpdb` function to retrieve protein structure data from the Protein Data Bank (PDB) database for two hemoglobin structures.

```
str1 = getpdb('1dke');
str2 = getpdb('4hhb');
```

2   Superpose the first model of the two hemoglobin structures, applying the transformation to the entire molecule.

```
d = pdbsuperpose(str1, str2, 'model', [1 1], 'apply', 'all');
```

3   Superpose the two hemoglobin structures (each containing four chains), computing and applying the linear transformation chain by chain. Do not display the structures.

```
strtx = str2;
chainList1 = {str1.Sequence.ChainID};
chainList2 = {str2.Sequence.ChainID};
for i = 1:4
    [d(i), rmsd(i), tr(i), strtx] = pdbsuperpose(str1, strtx,  ...
                            'segment', {chainList1{i}; chainList2{i}}, ...
                            'apply', 'chain', 'display', false);
end
```

Superposing Two Chains of a Thioredoxin Structure

Superpose chain B on chain A of a thioredoxin structure (PDBID = 2trx), and then apply the transformation only to chain B.

```
[d, rmsd, tr] = pdbsuperpose('2trx', '2trx', 'segment', {'A', 'B'}, ...
                            'apply', 'chain')

d =

    0.0028
```

```
rmsd =

    0.6604

tr =

    T: [3x3 double]
    b: 1
    c: [109x3 double]
```

Superposing Two Calmodulin Structures

Superpose two calmodulin structures according to the linear transformation obtained using two 20 residue-long segments.

```
pdbsuperpose('1a29', '1cll', 'segment', {'10-30:A', '10-30:A'})

ans =

    0.1945
```

See Also

getpdb | molviewer | pdbread | pdbtransform | swalign | procrustes

pdbtransform

Apply linear transformation to 3-D structure of molecule

Syntax

```
pdbtransform(PDB, Transf)
PDBTX = pdbtransform(PDB, Transf)
... = pdbtransform(..., 'ModelNum', ModelNumValue, ...)
... = pdbtransform(..., 'Segment', SegmentValue, ...)
```

Input Arguments

PDB	Protein structure represented by any of the following: • String specifying a unique identifier for a protein structure record in the Protein Data Bank (PDB) database. • Variable containing a PDB-formatted MATLAB structure, such as returned by `getpdb` or `pdbread`. • String specifying a file name or a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.
Transf	MATLAB structure representing a linear transformation, which is applied to the coordinates of the molecule represented by *PDB*. *Transf* contains the following fields: • `T` — Orthogonal rotation and reflection component. • `b` — Scale component. • `c` — Translation component. **Tip** You can use the *Transf* structure returned by the `pdbsuperpose` function as input.

ModelNumValue	Positive integer that specifies the model to which to apply the transformation, when *PDB* contains multiple models. By default, the first model is considered.
SegmentValue	Specifies the extent to which the linear transformation is applied. *SegmentValue* can be either: • `'all'` — The transformation is applied to the entire PDB input. • String specifying the boundaries and the chain to consider. It uses either of the following formats: `'start-stop:chain'` or `'chain'`. Omitting the boundaries indicates the entire chain.

Output Arguments

PDBTX	Transformed PDB-formatted MATLAB structure.

Description

pdbtransform(*PDB*, *Transf*) applies the linear transformation specified in *Transf*, a MATLAB structure representing a linear transformation, to the coordinates of the molecule represented by *PDB*, which can be any of the following:

- String specifying a unique identifier for a protein structure record in the PDB database.
- Variable containing a PDB-formatted MATLAB structure, such as returned by getpdb or pdbread.
- String specifying a file name or a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.

PDBTX = pdbtransform(*PDB*, *Transf*) returns *PDBTX*, the transformed PDB-formatted MATLAB structure.

... = pdbtransform(...*'PropertyName'*, *PropertyValue*, ...) calls pdbtransform with optional properties that use property name/property value pairs.

You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`... = pdbtransform(..., 'ModelNum', ModelNumValue, ...)` specifies the model to which to apply the transformation, when *PDB* contains multiple models. *ModelNumValue* is a positive integer. By default, the first model is considered.

`... = pdbtransform(..., 'Segment', SegmentValue, ...)` specifies the extent to which the linear transformation is applied. *SegmentValue* can be either:

- `'all'` — The transformation is applied to the entire PDB input.
- String specifying the boundaries and the chain to consider. It uses either of the following formats: `'start-stop:chain'` or `'chain'`. Omitting the boundaries indicates the entire chain.

Examples

1 Create a MATLAB structure that defines a linear transformation.

```
transf.T = eye(3);  transf.b = 1;  transf.c = [11.8 -2.8 -32.3];
```

2 Apply the linear transformation to chain B in the thioredoxin structure, with a PDB identifier of 2trx.

```
pdbtx = pdbtransform('2trx', transf, 'segment', 'B');
```

See Also

getpdb | molviewer | pdbread | pdbsuperpose | procrustes

pdbwrite

Write to file using Protein Data Bank (PDB) format

Syntax

```
pdbwrite(File, PDBStruct)
PDBArray = pdbwrite(File, PDBStruct)
```

Input Arguments

File	String specifying either a file name or a path and file name for saving the PDB-formatted data. If you specify only a file name, the file is saved to the MATLAB Current Folder.
	Tip After you save the MATLAB structure to a local PDB-formatted file, you can use the molviewer function to display and manipulate a 3-D image of the structure.
PDBStruct	MATLAB structure containing 3-D protein structure coordinate data, created initially by using the getpdb or pdbread functions.
	Note: You can edit this structure to modify its 3-D protein structure data. The coordinate information is stored in the Model field of *PDBStruct*.

Output Arguments

PDBArray	Character array in which each row corresponds to a line in a PDB record.

Description

pdbwrite(*File*, *PDBStruct*) writes the contents of the MATLAB structure *PDBStruct* to a PDB-formatted file (ASCII text file) whose path and file name are specified by *File*. In the output file, *File*, the atom serial numbers are preserved. The atomic coordinate records are ordered according to their atom serial numbers.

Tip After you save the MATLAB structure to a local PDB-formatted file, you can use the molviewer function to display and manipulate a 3-D image of the structure.

PDBArray = pdbwrite(*File*, *PDBStruct*) saves the formatted PDB record, converted from the contents of the MATLAB structure *PDBStruct*, to *PDBArray*, a character array in which each row corresponds to a line in a PDB record.

Note: You can edit *PDBStruct* to modify its 3-D protein structure data. The coordinate information is stored in the Model field of *PDBStruct*.

Examples

1   Use the getpdb function to retrieve structure information from the Protein Data Bank (PDB) for the green fluorescent protein with identifier 1GFL , and store the data in the MATLAB structure gflstruct.

```
gflstruct = getpdb('1GFL');
```

2   Find the *x*-coordinate of the first atom.

```
gflstruct.Model.Atom(1).X

ans =

   -14.0930
```

3   Edit the *x*-coordinate of the first atom.

```
gflstruct.Model.Atom(1).X = -18;
```

> **Note:** Do not add or remove any `Atom` fields, because the `pdbwrite` function does not allow the number of elements in the structure to change.

---

4 Write the modified MATLAB structure `gflstruct` to a new PDB-formatted file `modified_gfl.pdb` in the `Work` folder on your `C` drive.

```
pdbwrite('c:\work\modified_gfl.pdb', gflstruct);
```

5 Use the `pdbread` function to read the modified PDB file into a MATLAB structure, then confirm that the *x*-coordinate of the first atom has changed.

```
modified_gflstruct = pdbread('c:\work\modified_gfl.pdb')
modified_gflstruct.Model.Atom(1).X

ans =

   -18
```

See Also

getpdb | molviewer | pdbread

pdist (phytree)

Calculate pairwise patristic distances in phytree object

Syntax

```
D = pdist(Tree)
[D, C] = pdist(Tree)
pdist(..., 'Nodes', NodesValue, ...)
pdist(..., 'Squareform', SquareformValue, ...)
pdist(..., 'Criteria', CriteriaValue, ...)
```

Arguments

Tree	phytree object created by phytree function (object constructor) or phytreeread function.
NodesValue	String that specifies the nodes included in the computation. Choices are 'leaves' (default) or 'all'.
SquareformValue	Controls the creation of a square matrix. Choices are true or false (default).
CriteriaValue	String that specifies the criteria used to relate pairs. Choices are 'distance' (default) or 'levels'.

Description

D = pdist(*Tree*) returns D, a vector containing the patristic distances between every possible pair of leaf nodes of *Tree*, a phylogenetic tree object. The patristic distances are computed by following paths through the branches of the tree and adding the patristic branch distances originally created with the seqlinkage function.

The output vector *D* is arranged in the order `((2,1), (3,1), ..., (M,1), (3,2), ..., (M,2), ..., (M,M-1))` (the lower-left triangle of the full M-by-M distance matrix). To get the distance between the Ith and Jth nodes (I > J), use the formula `D((J-1)*(M-J/2)+I-J)`. M is the number of leaves.

[*D, C*] = pdist(*Tree*) returns in *C*, the index of the closest common parent nodes for every possible pair of query nodes.

pdist(..., '*PropertyName*', *PropertyValue*, ...) calls pdist with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

pdist(..., 'Nodes', *NodesValue*, ...) specifies the nodes included in the computation. Choices are 'leaves' (default) or 'all'. When *NodesValue* is 'leaves', the output is ordered as before, but M is the total number of nodes in the tree (NumLeaves+NumBranches).

pdist(..., 'Squareform', *SquareformValue*, ...) controls the creation of a square matrix. Choices are true or false (default). When *SquareformValue* is true, pdist converts the output into a square-formatted matrix, so that D(I,J) denotes the distance between the Ith and the Jth nodes. The output matrix is symmetric and has a zero diagonal.

pdist(..., 'Criteria', *CriteriaValue*, ...) changes the criteria used to relate pairs. *CriteriaValue* can be 'distance' (default) or 'levels'.

Examples

1   Read a phylogenetic tree file into a phytree object.

    ```
    tr = phytreeread('pf00002.tree')
    ```

2   Calculate the tree distances between pairs of leaves.

    ```
    dist = pdist(tr,'nodes','leaves','squareform',true)
    ```

More About

•   phytree object

See Also

phytree | phytreeread | phytreeviewer | seqlinkage | seqpdist

pfamhmmread

Read data from PFAM HMM-formatted file

Syntax

HMMStruct = pfamhmmread(*File*)

Input Arguments

File	Either of the following:
	• String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a PFAM HMM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.
	• MATLAB character array that contains the text of a PFAM-HMM-formatted file.
	Tip You can use the `gethmmprof` function with the `'ToFile'` property to retrieve HMM profile information from the PFAM database and create a PFAM HMM-formatted file.

Output Arguments

HMMStruct	MATLAB structure containing information from a PFAM HMM-formatted file.

Description

Note: `pfamhmmread` reads PFAM-HMM formatted files, from file format version HMMER2.0 to HMMER3/b.

HMMStruct = pfamhmmread(*File*) reads *File*, a PFAM HMM-formatted file, and converts it to *HMMStruct*, a MATLAB structure containing the following fields corresponding to parameters of an HMM profile:

Field	Description
Name	The protein family name (unique identifier) of the HMM profile record in the PFAM database.
PfamAccessionNumber	The protein family accession number of the HMM profile record in the PFAM database.
ModelDescription	Description of the HMM profile.
ModelLength	The length of the profile (number of MATCH states).
Alphabet	The alphabet used in the model, `'AA'` or `'NT'`. **Note:** `AlphaLength` is 20 for `'AA'` and 4 for `'NT'`.
MatchEmission	Symbol emission probabilities in the MATCH states. The format is a matrix of size `ModelLength`-by-`AlphaLength`, where each row corresponds to the emission distribution for a specific MATCH state.
InsertEmission	Symbol emission probabilities in the INSERT state. The format is a matrix of size `ModelLength`-by-`AlphaLength`, where each row corresponds to the emission distribution for a specific INSERT state.
NullEmission	Symbol emission probabilities in the MATCH and INSERT states for the NULL model. The format is a 1-by-`AlphaLength` row vector. **Note:** NULL probabilities are also known as the background probabilities.
BeginX	BEGIN state transition probabilities. Format is a 1-by-(`ModelLength` + 1) row vector: `[B->D1 B->M1 B->M2 B->M3 .... B->Mend]`

Field	Description

`MatchX`	MATCH state transition probabilities.
	
	Format is a 4-by-(`ModelLength - 1`) matrix:
	
	```[M1->M2 M2->M3 ... M[end-1]->Mend;```
	``` M1->I1 M2->I2 ... M[end-1]->I[end-1];```
	``` M1->D2 M2->D3 ... M[end-1]->Dend;```
	``` M1->E M2->E ... M[end-1]->E]```

`InsertX`	INSERT state transition probabilities.
	
	Format is a 2-by-(`ModelLength - 1`) matrix:
	
	```[I1->M2 I2->M3 ... I[end-1]->Mend;```
	``` I1->I1 I2->I2 ... I[end-1]->I[end-1]]```

`DeleteX`	DELETE state transition probabilities.
	
	Format is a 2-by-(`ModelLength - 1`) matrix:
	
	```[D1->M2 D2->M3 ... D[end-1]->Mend ;```
	``` D1->D2 D2->D3 ... D[end-1]->Dend]```

`FlankingInsertX`	Flanking insert states (N and C) used for LOCAL profile alignment.
	
	Format is a 2-by-2 matrix:
	
	```[N->B C->T ;```
	``` N->N C->C]```

`LoopX`	Loop states transition probabilities used for multiple hits alignment.
	
	Format is a 2-by-2 matrix:
	
	```[E->C J->B ;```
	``` E->J J->J]```

`NullX`	Null transition probabilities used to provide scores with log-odds values also for state transitions. Format is a 2-by-1 column vector: ```[G->F ; G->G]```

For more information on HMM profile models, see "HMM Profile Model" on page 1-976.

Examples

Read a locally saved PFAM HMM-formatted file into a MATLAB structure.

```
pfamhmmread('pf00002.ls')

ans =

                    Name: '7tm_2'
    PfamAccessionNumber: 'PF00002.15'
        ModelDescription: '7 transmembrane receptor (Secretin family)'
             ModelLength: 293
                Alphabet: 'AA'
           MatchEmission: [293x20 double]
          InsertEmission: [293x20 double]
            NullEmission: [1x20 double]
                  BeginX: [294x1 double]
                  MatchX: [292x4 double]
                 InsertX: [292x2 double]
                 DeleteX: [292x2 double]
         FlankingInsertX: [2x2 double]
                   LoopX: [2x2 double]
                   NullX: [2x1 double]
```

See Also

gethmmalignment | gethmmprof | hmmprofalign | hmmprofstruct | showhmmprof

phytree object

Data structure containing phylogenetic tree

Description

A phytree object is a data structure containing a phylogenetic tree. Phylogenetic trees are binary rooted trees, which means that each branch is the parent of two other branches, two leaves, or one branch and one leaf. A phytree object can be ultrametric or nonultrametric.

Method Summary

Following are methods of a phytree object:

cluster (phytree)

> Validate clusters in phylogenetic tree

get (phytree)

> Retrieve information about phylogenetic tree object

getbyname (phytree)

> Branches and leaves from phytree object

getcanonical (phytree)

> Calculate canonical form of phylogenetic tree

getmatrix (phytree)

> Convert phytree object into relationship matrix

getnewickstr (phytree)

> Create Newick-formatted string

pdist (phytree)

> Calculate pairwise patristic distances in phytree object

plot (phytree)

> Draw phylogenetic tree

prune (phytree)	
	Remove branch nodes from phylogenetic tree
reorder (phytree)	
	Reorder leaves of phylogenetic tree
reroot (phytree)	
	Change root of phylogenetic tree
select (phytree)	
	Select tree branches and leaves in phytree object
subtree (phytree)	
	Extract phylogenetic subtree
view (phytree)	
	View phylogenetic tree
weights (phytree)	
	Calculate weights for phylogenetic tree

Property Summary

Note: You cannot modify these properties directly. You can access these properties using the `get` method.

Property	Description
NumLeaves	Number of leaves
NumBranches	Number of branches
NumNodes	Number of nodes (NumLeaves + NumBranches)
Pointers	Branch to leaf/branch connectivity list
Distances	Edge length for every leaf/branch
LeafNames	Names of the leaves
BranchNames	Names of the branches
NodeNames	Names of all the nodes

See Also

phytree | cluster | getbyname | getmatrix | pdist | plot | reroot | subtree | weights | phytreeread | phytreeviewer | phytreewrite | seqlinkage | seqneighjoin | seqpdist | get | getcanonical | getnewickstr | prune | select | view

phytree

Create phytree object

Syntax

Tree = phytree(*B*)
Tree = phytree(*B*, *D*)
Tree = phytree(*B*, *C*)
Tree = phytree(*BC*)
Tree = phytree(..., *N*)
Tree = phytree

Arguments

B	Numeric array of size [NUMBRANCHES X 2] in which every row represents a branch of the tree. It contains two pointers to the branch or leaf nodes, which are its children.
C	Column vector with distances for every branch.
D	Column vector with distances from every node to their parent branch.
BC	Combined matrix with pointers to branches or leaves, and distances of branches.
N	Cell array with the names of leaves and branches.

Description

Tree = phytree(*B*) creates an ultrametric phylogenetic tree object. In an ultrametric phylogenetic tree object, all leaves are the same distance from the root.

B is a numeric array of size [NUMBRANCHES X 2] in which every row represents a branch of the tree and it contains two pointers to the branch or leaf nodes, which are its children.

Leaf nodes are numbered from 1 to NUMLEAVES and branch nodes are numbered from NUMLEAVES + 1 to NUMLEAVES + NUMBRANCHES. Note that because only binary trees are allowed, NUMLEAVES = NUMBRANCHES + 1.

Branches are defined in chronological order (for example, `B(i,:) > NUMLEAVES + i`). As a consequence, the first row can only have pointers to leaves, and the last row must represent the root branch. Parent-child distances are set to `1`, unless the child is a leaf and to satisfy the ultrametric condition of the tree its distance is increased.

Given a tree with three leaves and two branches as an example.
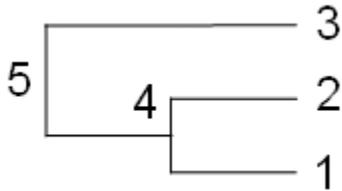
In the MATLAB Command Window, type

```
B = [1 2 ; 3 4]

 B =

     1     2
     3     4

tree = phytree(B)

 Phylogenetic tree object with 3 leaves (2 branches)

view(tree)
```
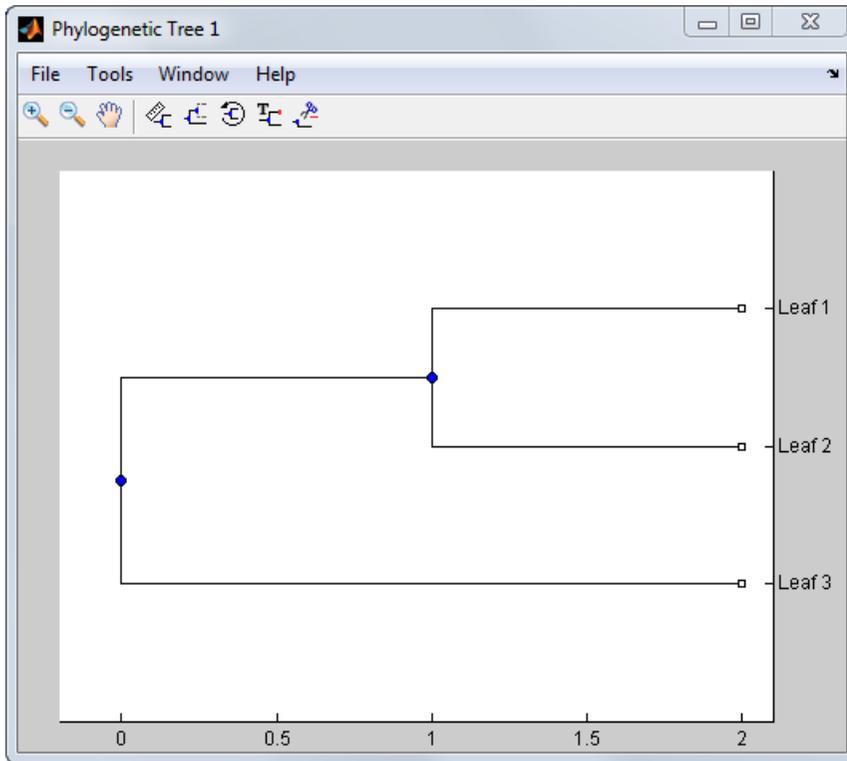
Tree = phytree(*B*, *D*) creates an additive (ultrametric or nonultrametric) phylogenetic tree object with branch distances defined by *D*. *D* is a numeric array of size [NUMNODES X 1] with the distances of every child node (leaf or branch) to its parent branch equal to NUMNODES = NUMLEAVES + NUMBRANCHES. The last distance in *D* is the distance of the root node and is meaningless.

```
b = [ 1 2 ; 3 4 ]

b =

     1      2
     3      4

d = [1; 2; 1.5; 1; 0]

d =
```

```
        1.0000
        2.0000
        1.5000
        1.0000
             0
```

view(phytree(b,d))
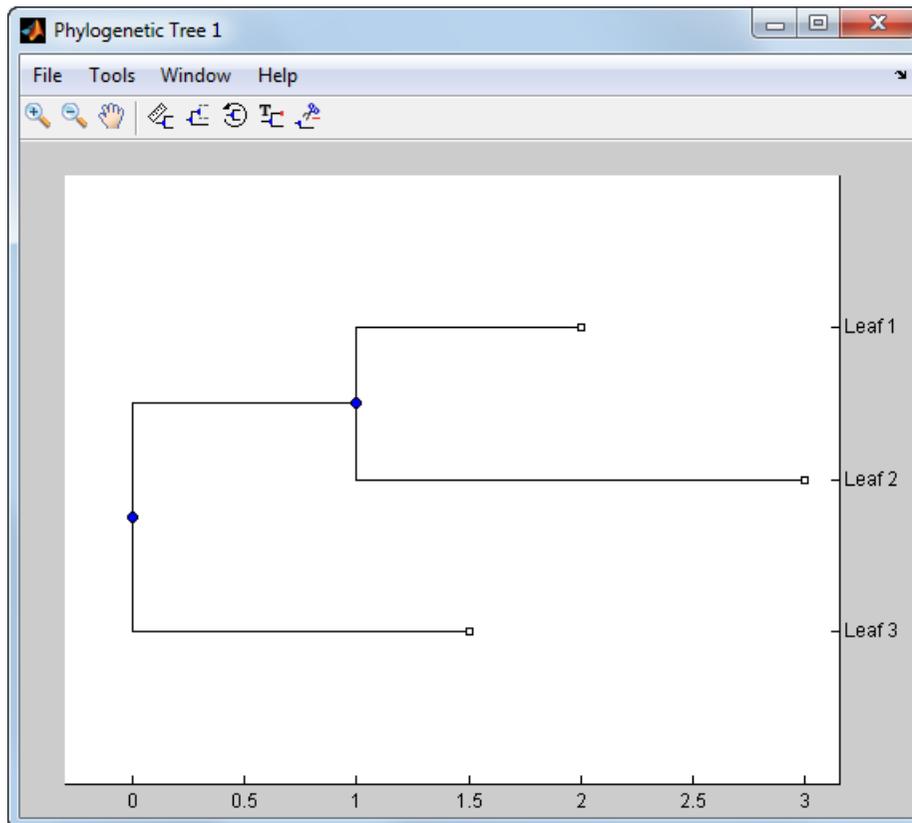
Tree = phytree(*B*, *C*) creates an ultrametric phylogenetic tree object with distances between branches and leaves defined by C. C is a numeric array of size [NUMBRANCHES X 1], which contains the distance from each branch to the leaves. In ultrametric trees, all of the leaves are at the same location (same distance to the root).

```
b = [1 2 ; 3 4]

b =

      1      2
      3      4

c = [1 4]'

c =

      1
      4
```

```
view(phytree(b,c))
```
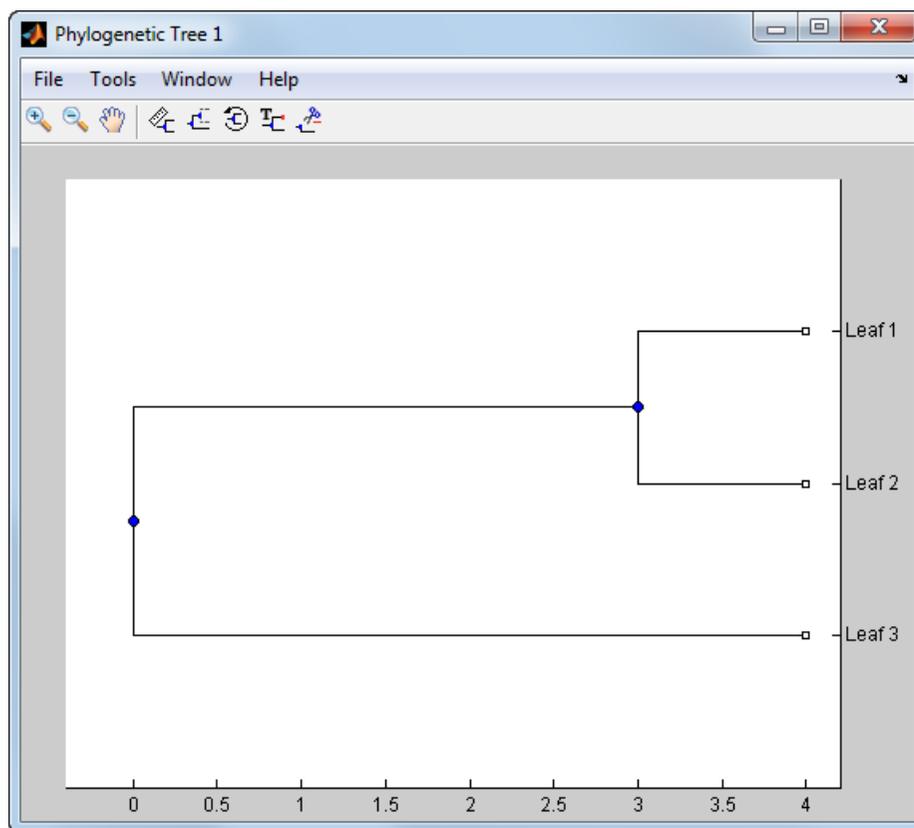
Tree = phytree(*BC*) creates an ultrametric phylogenetic binary tree object with branch pointers in BC(:,[1 2]) and branch coordinates in BC(:,3). Same as phytree(B,C).

Tree = phytree(..., *N*) specifies the names for the leaves and/or the branches. *N* is a cell of strings. If NUMEL(N)==NUMLEAVES, then the names are assigned chronologically to the leaves. If NUMEL(N)==NUMBRANCHES, the names are assigned to the branch nodes. If NUMEL(N)==NUMLEAVES + NUMBRANCHES, all the nodes are named. Unassigned names default to 'Leaf #' and/or 'Branch #' as required.

Tree = phytree creates an empty phylogenetic tree object.

Examples

Create a Phylogenetic Tree

This example shows how to create a phylogenetic tree from a multiple sequence alignment file.

Read a multiple sequence alignment file.

```
Sequences = multialignread('aagag.aln');
```

Calculate the distance between each pair of sequences.

```
distances = seqpdist(Sequences);
```

Construct a phylogenetic tree object from the pairwise distances calculated previously.

```
tree = seqlinkage(distances);
```

View the phylogenetic tree.

```
phytreeviewer(tree)
```
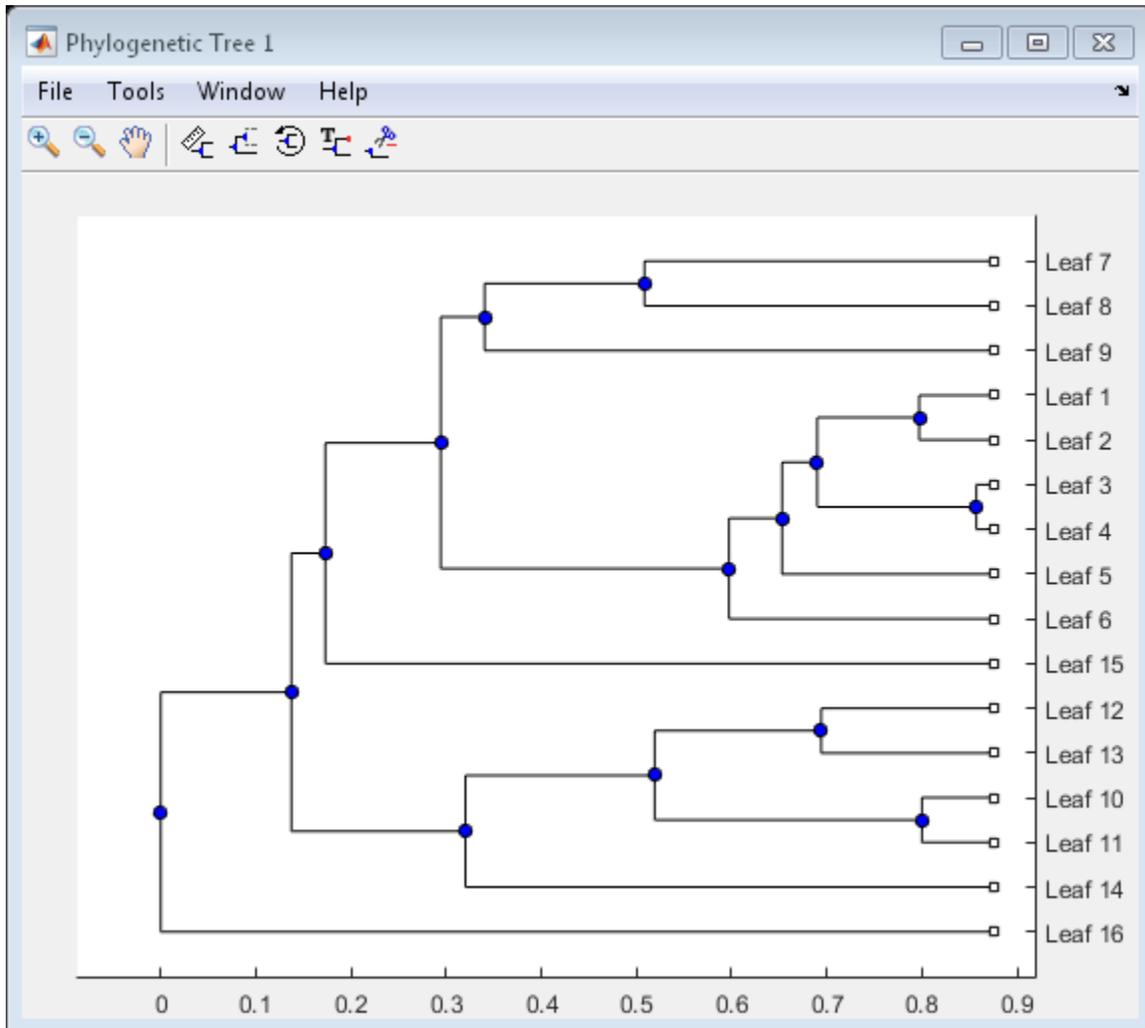
More About

·   phytree object

See Also

phytreeread | cluster | getbyname | getmatrix | pdist | plot | reroot | subtree | weights | phytreeviewer | phytreewrite | seqlinkage | seqneighjoin | seqpdist | get | getcanonical | getnewickstr | prune | select | view

phytreeread

Read phylogenetic tree file

Syntax

Tree = phytreeread(*File*)
[*Tree, Boot*]= phytreeread(*File*)

Input Arguments

File	Newick-formatted tree files (ASCII text file). Enter a file name, a path and file name, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text for a file.

Output Arguments

Tree	phytree object created with the function phytree.
Boot	Column vector of bootstrap values for each tree node specified in *File*. If *File* does not specify a bootstrap value for a node, it returns a NaN value for that node. phytreeread considers the following values in *File* to be bootstrap values:
	• Values within square brackets ([ ]) after the branch or leaf node lengths
	• Values that appear instead of branch or leaf node labels

Description

Tree = phytreeread(*File*) reads a Newick-formatted tree file and returns a phytree object containing data from the file.

The NEWICK tree format can be found at

http://evolution.genetics.washington.edu/phylip/newicktree.html

---

Note: This implementation allows only binary trees. Non-binary trees are translated into a binary tree with extra branches of length 0.

[*Tree, Boot*]= phytreeread(*File*) returns *Boot*, a column vector of bootstrap values for each tree node specified in *File*. If *File* does not specify a bootstrap value for a node, it returns a NaN value for that node. phytreeread considers the following values in *File* to be bootstrap values:

- Values within square brackets ([ ]) after the branch or leaf node lengths
- Values that appear instead of branch or leaf node labels

Examples

```
tr = phytreeread('pf00002.tree')

 Phylogenetic tree object with 33 leaves (32 branches)
```

More About

- phytree object

See Also

phytree | gethmmtree | phytreeviewer | phytreewrite

phytreeviewer

Visualize, edit, and explore phylogenetic tree data

Syntax

```
phytreeviewer
phytreeviewer(Tree)
phytreeviewer(File)
```

Description

`phytreeviewer` opens the Phylogenetic Tree app that allows you to view, edit, and explore phylogenetic tree data.

`phytreeviewer(Tree)` loads a `phytree` object Tree into the app.

`phytreeviewer(File)` loads data from a Newick or ClustalW tree formatted file into the app.

Input Arguments

`Tree` — Phylogenetic tree
Phytree object

Phylogenetic tree, specified as a `Phytree` object created with the functions `phytree` or `phytreeread`.

`File` — Newick or ClustalW tree formatted file
string

Newick or ClustalW tree formatted file, specified as a string containing the file name, a path, a URL or a MATLAB character array that contains the text for a Newick file.

Examples

View a Phylogenetic Tree

This example shows how to view a phylogenetic tree.

Load a sample phylogenetic tree.

```
tr= phytreeread('pf00002.tree')

    Phylogenetic tree object with 33 leaves (32 branches)
```

View the phylogenetic tree.

```
phytreeviewer(tr)
```
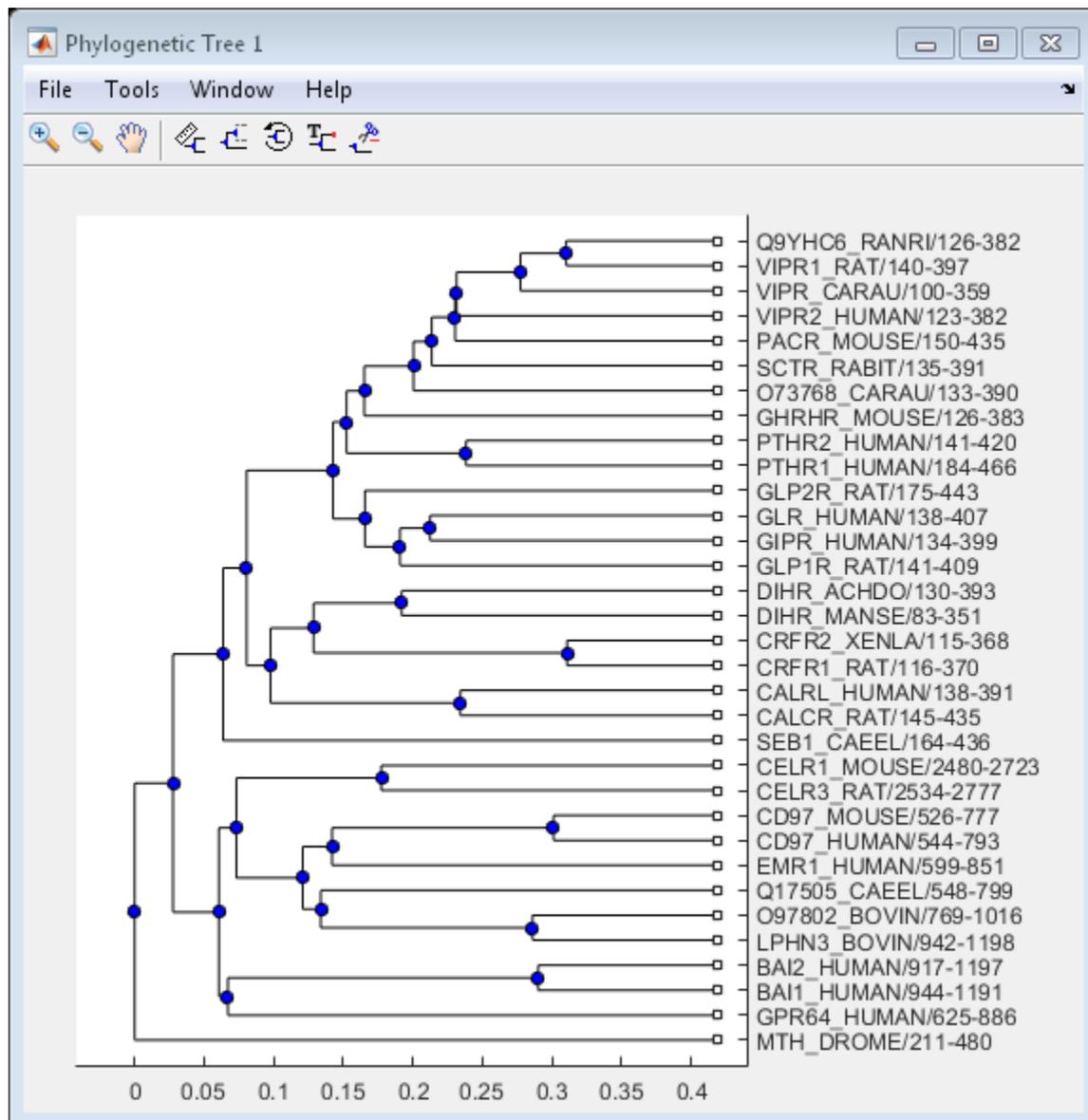
Alternatively, you can click Phylogenetic Tree on the **Apps** tab to open the app, and view the phylogenetic tree object `tr` .

See Also

phytree | cluster | view | phytreeread | phytreewrite | plot

phytreewrite

Write phylogenetic tree object to Newick-formatted file

Syntax

```
phytreewrite(File, Tree)
phytreewrite(Tree)
phytreewrite(..., 'Distances', DistancesValue, ...)
phytreewrite(..., 'BranchNames', BranchNamesValue, ...)
```

Arguments

File	String specifying a Newick-formatted file. Enter either a file name or a path and file name supported by your operating system (ASCII text file).
Tree	Phylogenetic tree object, either created with `phytree` (object constructor function) or imported using the `phytreeread` function.

Description

`phytreewrite(`*File*`, `*Tree*`)` copies the contents of a `phytree` object from the MATLAB workspace to a file. Data in the file uses the Newick format for describing trees.

The Newick tree format can be found at

`http://evolution.genetics.washington.edu/phylip/newicktree.html`

`phytreewrite(`*Tree*`)` opens the Save Phylogenetic Tree As dialog box for you to enter or select a file name.

`phytreewrite(..., '`*PropertyName*`', `*PropertyValue*`, ...)` calls `phytreewrite` with optional properties that use property name/property value pairs.

You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/ property value pairs are as follows:

phytreewrite(..., 'Distances', *DistancesValue*, ...) specifies whether to exclude the distances from the output. *DistancesValue* can be true (default) or false.

phytreewrite(..., 'BranchNames', *BranchNamesValue*, ...) specifies whether to exclude the branch names from the output. *BranchNamesValue* can be true (default) or false.

Examples

Read tree data from a Newick-formatted file.

```
tr = phytreeread('pf00002.tree')

 Phylogenetic tree object with 33 leaves (32 branches)
```

Remove all the mouse proteins and view the pruned tree.

```
ind = getbyname(tr,'mouse');
tr = prune(tr,ind);
view(tr)
```
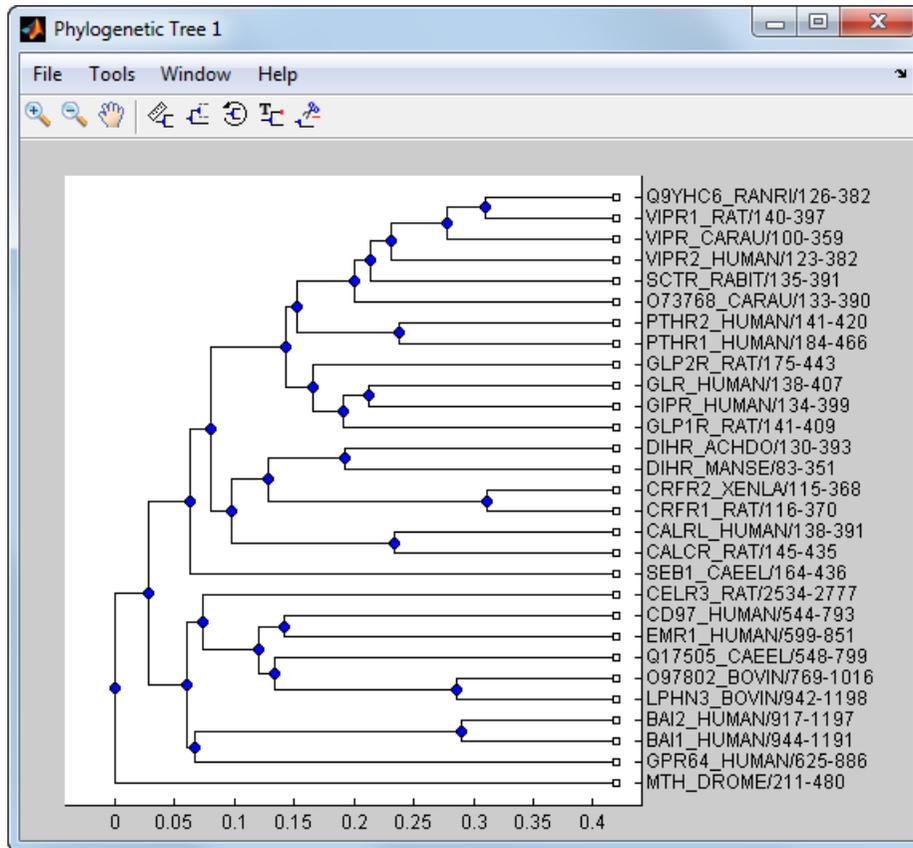
Write pruned tree data to a file.

```
phytreewrite('newtree.tree',tr)
```

More About

- `getnewickstr`

See Also

`multialignwrite` | `phytree object` | `phytree` | `phytreeread` | `phytreeviewer` | `seqlinkage`

plot (clustergram)

Render clustergram and dendrograms for clustergram object

Syntax

```
plot(CGObject)
plot(CGObject, HFig)
HFig = plot(...)
```

Arguments

CGObject	Clustergram object created with the function `clustergram`.
HFig	Handle to a MATLAB Figure window.

Description

`plot(CGObject)` renders a heat map and dendrograms for *CGObject*, a clustergram object, in a MATLAB Figure window.

`plot(CGObject, HFig)` renders a heat map and dendrograms for *CGObject*, a clustergram object, in a MATLAB Figure window with the handle *HFig*.

`HFig = plot(...)` returns the handle to the figure. The graphic properties are stored as application data in the figure handle.

Examples

Plot the clustergram object created in the first two steps of the "Examples" on page 1-447 section of the `clustergram` function reference page.

```
plot(cgo)
```

More About

· clustergram object

See Also
```
clustergram | addXLabel | get | set | addTitle | addYLabel | view
```

plot (DataMatrix)

Draw 2-D line plot of DataMatrix object

Syntax

```
plot(DMObj1)
plot(DMObj1, DMObj2)
plot(..., LineSpec)
```

Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by DataMatrix (object constructor). **Note:** If both *DMObj1* and *DMObj2* are input arguments, one of the inputs can be a MATLAB numeric array.
LineSpec	String specifying a line style, marker symbol, and color of the plotted lines. For more information on these specifiers, see LineSpec.

Description

plot(*DMObj1*) plots the columns of a DataMatrix object *DMObj1* versus their index.

plot(*DMObj1*, *DMObj2*) plots the data from *DMObj1* and *DMObj2*, two DataMatrix objects, or one DataMatrix object and one MATLAB numeric array.

- If *DMObj1* and *DMObj2* are both vectors, they must have the same number of elements, and plot plots one vector versus the other vector, creating a single line.
- If one is a vector and one a scalar, plot plots discrete points vertically or horizontally, at the scalar value.
- If one is a vector and one a matrix, the number of elements in the vector must equal either the number of rows or the number of columns in the matrix, and plot plots the vector versus each row or column in the matrix.

- If both are matrices, they must have the same size (number of rows and columns), and `plot` plots each column in *DMObj1* versus the corresponding column in *DMObj2*.

`plot(..., `*LineSpec*`)` plots all lines as defined by *LineSpec*, a character string specifying a line style, marker symbol, and/or color.

Note: For a list of line style, marker, and color specifiers, see `LineSpec`.

More About

- DataMatrix object

See Also
`DataMatrix` | `plot`

plot (HeatMap)

Render heat map for HeatMap object

Syntax

```
plot(HMObject)
plot(HMObject, HFig)
HFig = plot(...)
```

Arguments

HMObject	HeatMap object created with the function `HeatMap`.
HFig	Handle to a MATLAB Figure window.

Description

`plot(HMObject)` renders a heat map for *HMObject*, a HeatMap object, in a MATLAB Figure window.

`plot(HMObject, HFig)` renders a heat map for *HMObject*, a HeatMap object, in a MATLAB Figure window with the handle *HFig*.

HFig = `plot(...)` returns the handle to the figure. The graphic properties are stored as application data in the figure handle.

Examples

Plot the HeatMap object created in the "Examples" on page 1-   section of the `HeatMap` function reference page.

```
plot(hmo)
```

More About

- HeatMap object

See Also

HeatMap | addXLabel | view | addTitle | addYLabel

plotChiSquaredFit

Plot goodnesss-of-fit for variance regression

Syntax

```
plotChiSquaredFit(test)
plotChiSquaredFit(test,Name,Value)
H = plotChiSquaredFit( ___ )
```

Description

`plotChiSquaredFit(test)` plots the empirical CDF of the chi-squared probabilities of the ratio between the observed and the estimated variance stratified by count levels into five equal-sized bins. Use this plot to assess the goodness-of-fit.

test, an output of the `nbintest` function, is a `NegativeBinomialTest` object. It contains results from an unpaired hypothesis test for two independent samples.

Note: If the `'VarianceLink'` name-value pair argument was set to `'Identity'` when you ran `nbintest`, then the chi-squared probability is computed using the ratio between the observed variance to the mean.

`plotChiSquaredFit(test,Name,Value)` uses a name-value pair argument.

`H = plotChiSquaredFit( ___ )` returns handles to axes.

Examples

Perform unpaired hypothesis test for short-read count data

This example shows how to perform an unpaired hypothesis test for synthetic short-read count data from two different biological conditions.

The data in this example contains synthetic gene count data for 5000 genes, representing two different biological conditions, such as diseased and normal cells. For each condition, there are five samples. Only 10% of the genes (500 genes) are differentially expressed. Specifically, half of them (250 genes) are exactly 3-fold overexpressed. The other 250 genes are 3-fold underexpressed. The rest of the gene expression data is generated from the same negative binomial distribution for both conditions. Each sample also has a different size factor (that is, the coverage or sampling depth).

Load the data.

```
clear all
load(fullfile(matlabroot,'examples','bioinfo','nbintest_data.mat'),'K','H0');
```

The variable K contains gene count data. The rows represent genes, and the columns represent samples. In this case, the first five columns represent samples from the first condition. The other five columns represent samples from the second condition. Display the first few rows of K.

```
K(1:5,:)
```

```
ans =

  Columns 1 through 6

     13683       14140        8281       14309       12208        8045
     16028       16805        9813       16486       14076        9901
       814         862         492         910         758         521
     15870       16453        9857       16454       14267        9671
      9422        9393        5734        9598        8174        5381

  Columns 7 through 10

      9446       11317       14597       14592
     10927       13348       16999       17036
       573         753         870         936
     10997       13624       17151       17205
      6315        7752        9869        9795
```

In this example, the null hypothesis is true when the gene is not differentially expressed. The variable H0 contains boolean indicators that indicate for which genes the null hypothesis is true (marked as 1). In other words, H0 contains known labels that you will use later to compare with predicted results.

```
sum(H0)
```

```
ans =

        4500
```

Out of 5000 genes, 4500 are not differentially expressed in this synthetic data.

Run an unpaired hypothesis test for samples from two conditions using `nbintest`. The assumption is that the data came from a negative binomial distribution, where the variance is linked to the mean via a locally-regressed smooth function of the mean as described in [1] by setting `'VarianceLink'` to `'LocalRegression'`.

```
tLocal = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','LocalRegression');
```

Use `plotVarianceLink` to plot a scatter plot for each experimental condition (for X and Y conditions), with the sample variance on the common scale versus the estimate of the condition-dependent mean. Use a linear scale for both axes. Include curves for all other linkage options by setting `'Compare'` to `true`.

```
plotVarianceLink(tLocal,'Scale','linear','Compare',true)
```

The `Identity` line represents the Poisson model, where the variance is identical to the mean as described in [3]. Observe that the data seems to be overdispersed (that is, most points are above the `Identity` line). The `Constant` line represents the negative binomial model, where the variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. The `Local Regression` and `Constant` linkage options appear to fit better with the overdispersed data.

Use `plotChiSquaredFit` to assess the goodness-of-fit for variance regression. It plots the empirical CDF (ecdf) of the chi-squared probabilities. The probabilities are the ratio between the observed and the estimated variance stratified by short-read count levels into five equal-sized bins.

```
plotChiSquaredFit(tLocal)
```

Residuals ECDF Plot for Y

Each figure shows five ecdf curves. Each curve represents one of the five short-read count levels. For instance, the blue line represents the ecdf curve for a low short-read counts between 0 and 1264. The red line represents high counts (more than 11438).

One way to interpret the curves is to check if the ecdf curves are above the diagonal line. If they are above the line, then the variance is overestimated. If they are below the line, then the variance is underestimated. In both figures, the variance seems to be correctly estimated for higher counts (that is, the red line follows the diagonal line), but slightly overestimated for lower count levels.

To assess the performance of the hypothesis test, construct a confusion matrix using the known labels and the predicted p-values.

```
confusionmat(H0,(tLocal.pValue > .001))
```

```
ans =

        493               7
          5            4495
```

Out of 500 differentially expressed genes, 493 are correctly predicted (true positives) and 7 of them are incorrectly predicted as not-differentially expressed genes (false negatives). Out of 4500 genes that are not differentially expressed, 4495 are correctly predicted (true negatives) and 5 of them are incorrectly predicted as differentially expressed genes (false positives).

For a comparison, run the hypothesis test again assuming that counts are modeled by the Poisson distribution, where the variance is identical to the mean.

```
tPoisson = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','Identity');
```

Plot the ecdf curves. Observe that all the curves are below the diagonal line, implying that the variance is underestimated. Therefore, the negative binomial model fits the data better.

```
plotChiSquaredFit(tPoisson)
```

Residuals ECDF Plot for X

Residuals ECDF Plot for Y

Input Arguments

`test` — Unpaired hypothesis test result
NegativeBinomialTest object (default)

Unpaired hypothesis test results, specified as a NegativeBinomialTest object. test is returned by the nbintest function.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'NumBins',4`

`'NumBins'` — Number of equal-sized bins

5 (default) | positive integer

Number of equal-sized bins, specified as a comma-separated pair consisting of `'NumBins'` and a positive integer.

Example: `'NumBins',3`

Output Arguments

`H` — Handles to axes

vector of handles

Handles to axes, specified as a vector of handles.

See Also

`mattest` | `nbintest` | `NegativeBinomialTest` | `plotVarianceLink`

plotPerPositionCountByQuality

Class: BioReadQualityStatistics

Plot fractions of reads with Phred scores in ranges

Syntax

```
plotPerPositionCountByQuality(QSObj)
H = plotPerPositionCountByQuality(QSObj)
```

Description

`plotPerPositionCountByQuality(QSObj)` generates a line plot displaying the fractions of all reads that have Phred scores in the ranges of 0–10, 11–20, 21–30, and 31–40 at each base position for the `BioReadQualityStatistics` object `QSObj`.

`H = plotPerPositionCountByQuality(QSObj)` returns the handle `H` to the axes object containing the generated plot.

Input Arguments

QSObj

`BioReadQualityStatistics` object.

Default:

Output Arguments

H

Handle to axes object containing generated plot.

Examples

Plot Fractions of Reads Stratified by Phread Scores

Create a `BioReadQualityStatistics` object from a FASTQ file using only the first 40 characters of each read with a minimum average quality score of 5.

```
QSObj = BioReadQualityStatistics('SRR005164_1_50.fastq','FilterLength',40,...
                                 'QualityScoreThreshold',5);
```

Plot the fractions of all reads with Phred scores in the ranges of 0-10, 11-20, 21-30, and 31-40 at each base position.

```
plotPerPositionCountByQuality(QSObj)


ans = 

  Axes (Quality Stratification) with properties:

             XLim: [0.1000 41]
             YLim: [0 100]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Use GET to show all properties
```
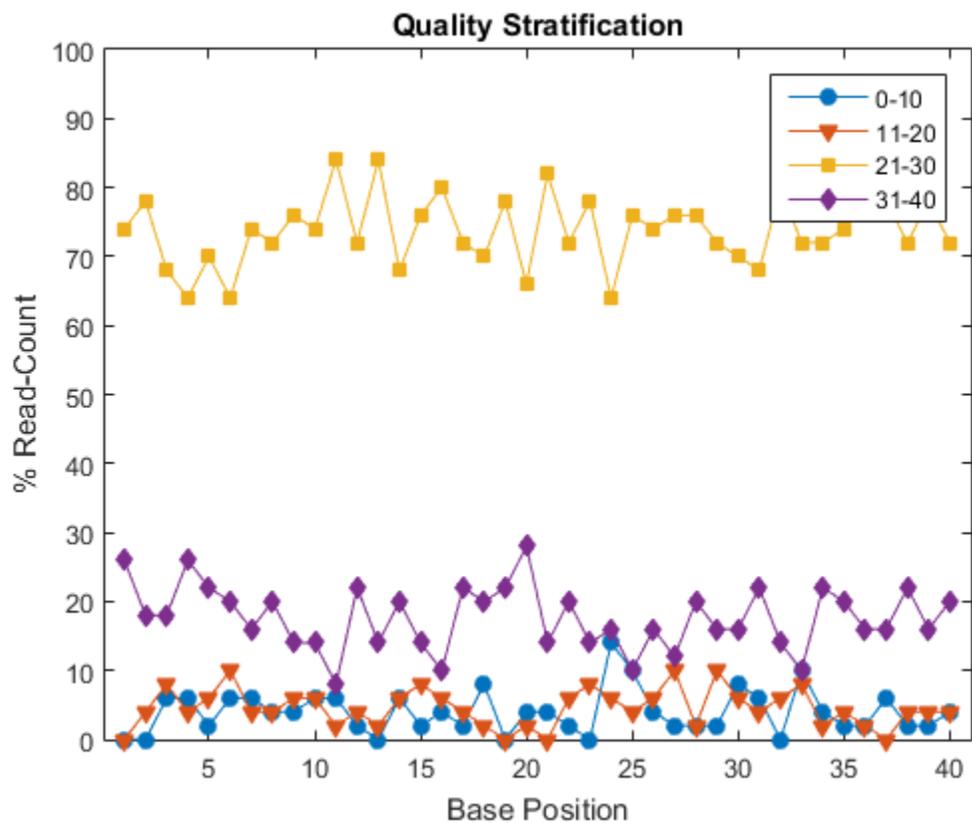
See Also

BioReadQualityStatistics | setQuality | BioRead | BioMap

plotPerPositionGC

Class: BioReadQualityStatistics

Plot percentages of G or C nucleotides at each base position

Syntax

```
plotPerPositionGC(QSObj)
H = plotPerPositionGC(QSObj)
```

Description

plotPerPositionGC(QSObj) generates a line plot displaying the percentages of G or C nucleotides at each base position for the BioReadQualityStatistics object QSObj.

H = plotPerPositionGC(QSObj) returns the handle H to the axes object containing the generated plot.

Input Arguments

QSObj

BioReadQualityStatistics object.

Default:

Output Arguments

H

Handle to axes object containing generated plot.

1-1437

Examples

Plot Percentages of G or C Nucleotide

Create a `BioReadQualityStatistics` object from a FASTQ file using only the first 40 characters of each read with a minimum average quality score of 5.

```
QSObj = BioReadQualityStatistics('SRR005164_1_50.fastq', ...
                                 'FilterLength', 40, ...
                                 'QualityScoreThreshold', 5);
```

Plot percentages of G or C nucleotide at each base position.

```
plotPerPositionGC(QSObj)


ans =

  Axes (GC Content) with properties:

             XLim: [0.1000 41]
             YLim: [0 100]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Use GET to show all properties
```
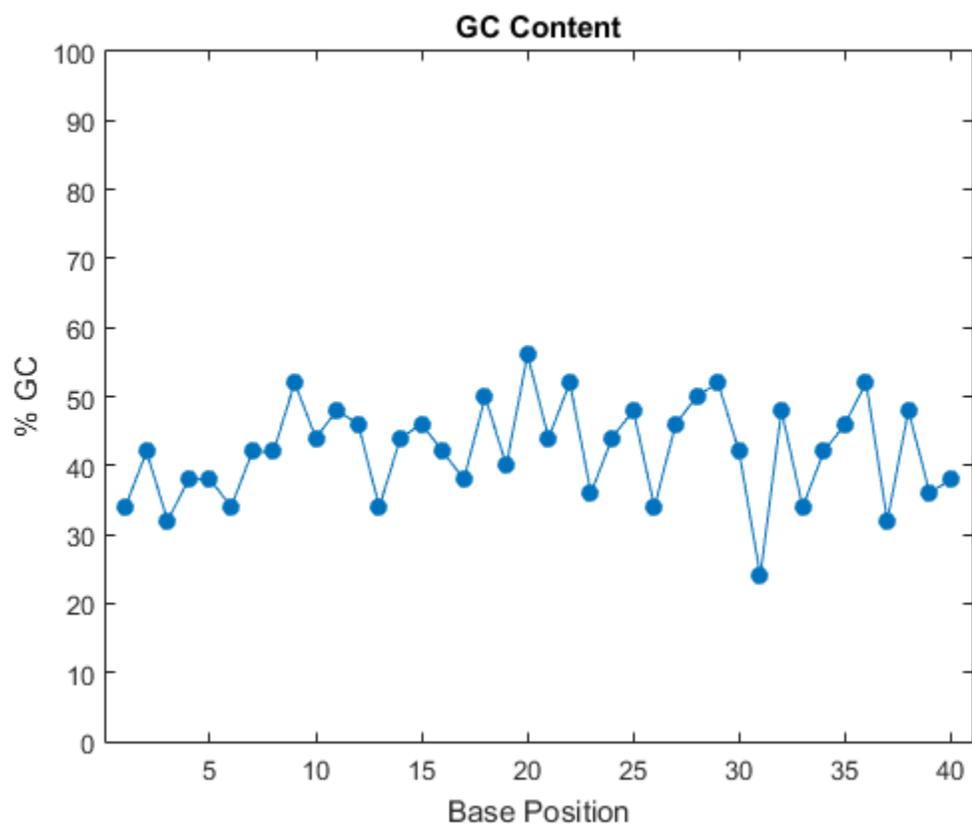
See Also
BioReadQualityStatistics | setQuality | BioRead | BioMap

plotPerPositionQuality

Class: BioReadQualityStatistics

Plot Phred score distributions

Syntax

```
plotPerPositionQuality(QSObj)
H = plotPerPositionQuality(QSObj)
```

Description

`plotPerPositionQuality(QSObj)` displays a series of box plots showing the Phred quality score distribution at each base position for the `BioReadQualityStatistics` object `QSObj`. Each box plot shows the median Phred score, the 25th and 75th percentiles, and the most extreme scores that are not considered outliers. An outlier is defined as a point that is more than 1.5 times the interquartile distance from the median.

`H = plotPerPositionQuality(QSObj)` returns the handle `H` to the axes object containing the generated plot.

Input Arguments

`QSObj`

`BioReadQualityStatistics` object.

Default:

Output Arguments

`H`

Handle to axes object containing generated plot.

Examples

Plot Phred score distributions

Create a `BioReadQualityStatistics` object from a FASTQ file using only the first 40 characters of each read with a minimum average quality score of 5.

```
QSObj = BioReadQualityStatistics('SRR005164_1_50.fastq', ...
                                 'FilterLength', 40, ...
                                 'QualityScoreThreshold', 5);
```

Plot Phred quality scores at given base positions.

```
plotPerPositionQuality(QSObj)


ans =

  Axes (Quality Scores) with properties:

            XLim: [0.1000 41]
            YLim: [0 40]
          XScale: 'linear'
          YScale: 'linear'
   GridLineStyle: '-'
        Position: [0.1300 0.1100 0.7750 0.8150]
           Units: 'normalized'

  Use GET to show all properties
```

Quality Scores
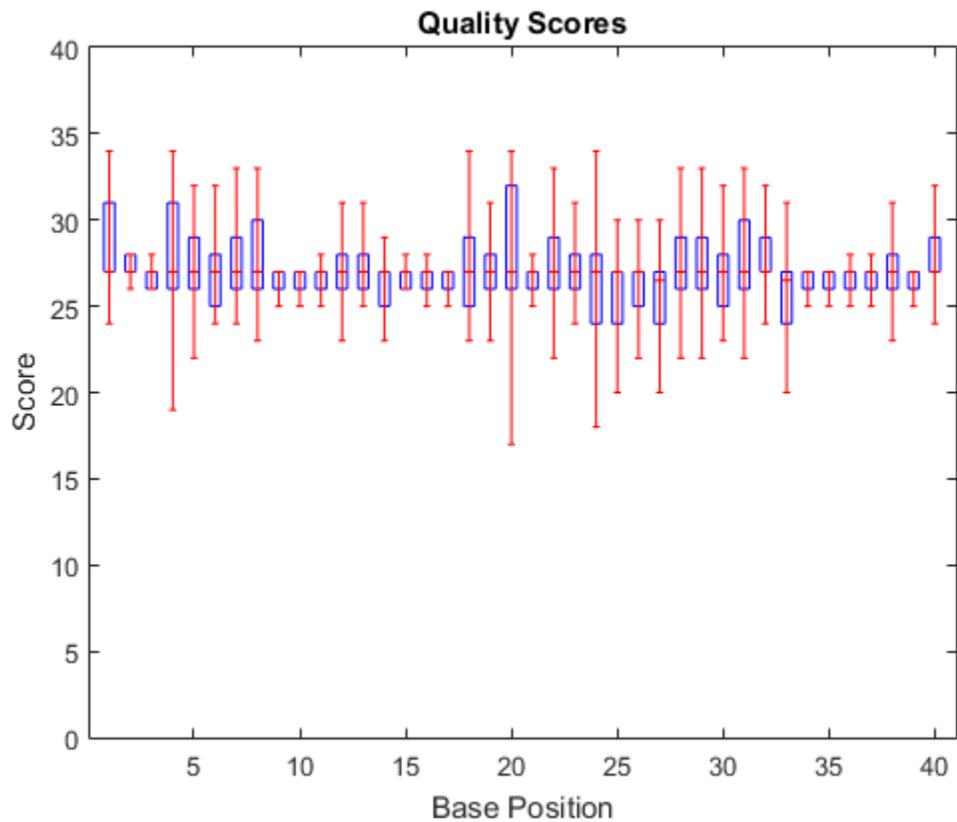
See Also

`BioReadQualityStatistics` | `setQuality` | `BioRead` | `BioMap`

plotPerSequenceGC

Class: BioReadQualityStatistics

Plot G or C nucleotide distribution

Syntax

```
plotPerSequenceGC(QSObj)
H = plotPerSequenceGC(QSObj)
```

Description

plotPerSequenceGC(QSObj) displays a bar graph showing the distribution of G or C content for short-read sequences of the BioReadQualityStatistics object QSObj.

H = plotPerSequenceGC(QSObj) returns the handle H to the axes object containing the generated plot.

Input Arguments

QSObj

BioReadQualityStatistics object.

Default:

Output Arguments

H

Handle to axes object containing generated plot.

Examples

Plot G or C Nucleotide Distribution

Create a `BioReadQualityStatistics` object from a FASTQ file using only the first 40 characters of each read with a minimum average quality score of 5.

```
QSObj = BioReadQualityStatistics('SRR005164_1_50.fastq','FilterLength',40,...
                                 'QualityScoreThreshold',5);
```

Plot G or C distribution.

```
plotPerSequenceGC(QSObj)
```

```
ans =

  Axes (GC Distribution) with properties:

             XLim: [0 100]
             YLim: [0 18]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Use GET to show all properties
```
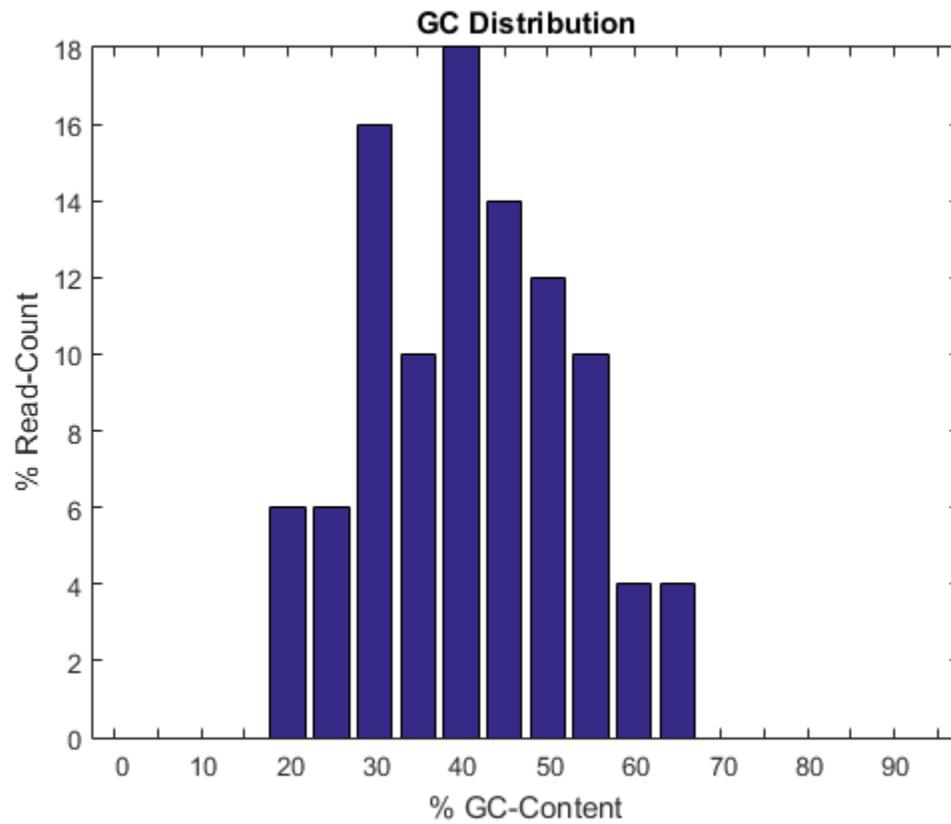
See Also

BioReadQualityStatistics | setQuality | BioRead | BioMap

plotPerSequenceQuality

Class: BioReadQualityStatistics

Plot distribution of average quality scores

Syntax

```
plotPerSequenceQuality(QSObj)
H = plotPerSequenceQuality(QSObj)
```

Description

`plotPerSequenceQuality(QSObj)` plots the distribution of average quality scores for short-read sequences of the `BioReadQualityStatistics` object `QSObj`.

`H = plotPerSequenceQuality(QSObj)` returns the handle `H` to the axes object containing the generated plot.

Input Arguments

QSObj

`BioReadQualityStatistics` object.

Default:

Output Arguments

H

Handle to axes object containing generated plot.

Examples

Plot the Distribution of Average Quality Scores

Create a `BioReadQualityStatistics` object from a FASTQ file using only the first 40 characters of each read with a minimum average quality score of 5.

```
QSObj = BioReadQualityStatistics('SRR005164_1_50.fastq','FilterLength',40,...
                                 'QualityScoreThreshold',5);
```

Display the distribution of average quality scores.

```
plotPerSequenceQuality(QSObj)


ans =

  Axes (Quality Distribution) with properties:

             XLim: [0 40]
             YLim: [0 50]
           XScale: 'linear'
           YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Use GET to show all properties
```
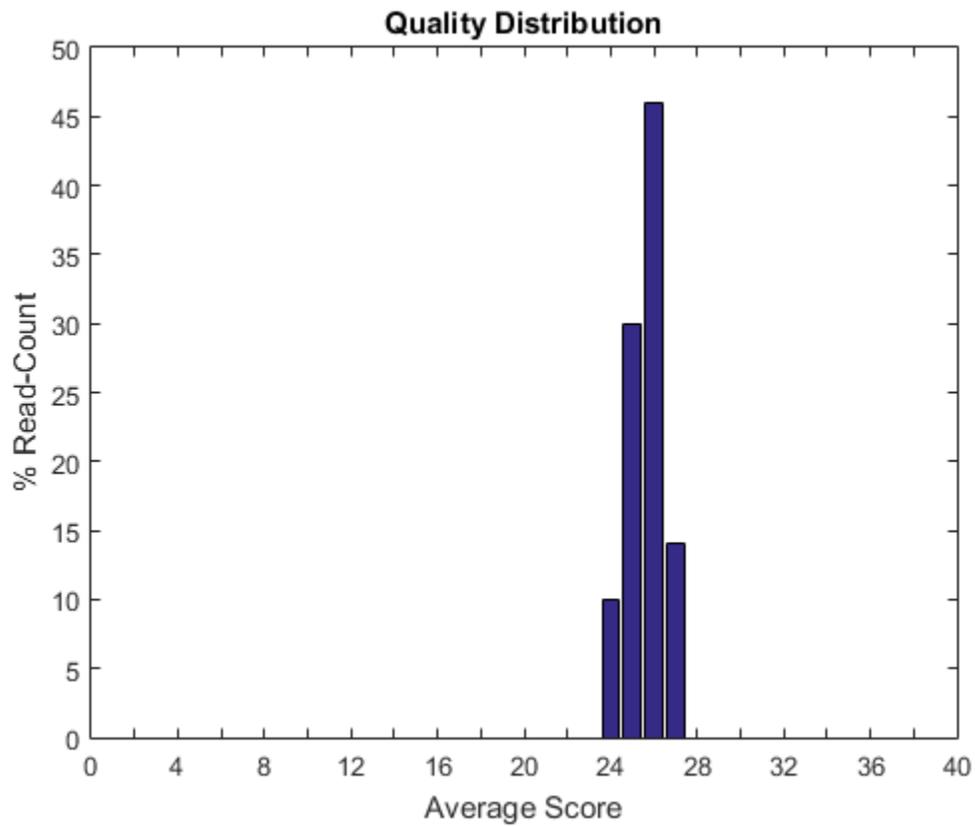
See Also

`BioReadQualityStatistics` | `setQuality` | `BioRead` | `BioMap`

plot (phytree)

Draw phylogenetic tree

Syntax

```
plot(Tree)
plot(Tree, ActiveBranches)
H = plot(...)
plot(..., 'Type', TypeValue, ...)
plot(..., 'Orientation', OrientationValue, ...)
plot(..., 'Rotation', RotationValue, ...)
plot(..., 'BranchLabels', BranchLabelsValue, ...)
plot(..., 'LeafLabels', LeafLabelsValue, ...)
plot(..., 'TerminalLabels', TerminalLabelsValue, ...)
plot(..., 'LLRotation', LLRotationValue, ...)
```

Input Arguments

Tree	Phylogenetic tree object created, such as created with the `phytree` constructor function.
ActiveBranches	Logical array of size `numBranches`-by-1 indicating the active branches, which are displayed in the Figure window.
TypeValue	String specifying a method for drawing the phylogenetic tree. Choices are: • `'square'` (default) • `'angular'` • `'radial'` • `'equalangle'` • `'equaldaylight'`
OrientationValue	String specifying the position of the root node, and hence the orientation of a phylogram or cladogram tree, when the `'Type'` property is `'square'` or `'angular'`. Choices are:

	• `'left'` (default) • `'right'` • `'top'` • `'bottom'`
RotationValue	Scalar between `0` (default) and `360` specifying rotation angle (in degrees) of the phylogenetic tree in the Figure window, when the `'Type'` property is `'radial'`, `'equalangle'`, or `'equaldaylight'`.
BranchLabelsValue	Controls the display of branch labels next to branch nodes. Choices are `true` or `false` (default).
LeafLabelsValue	Controls the display of leaf labels next to leaf nodes. Choices are `true` or `false`. Default is: • `true` — When the `'Type'` property is `'radial'`, `'equalangle'`, or `'equaldaylight'` • `false` — When the `'Type'` property is `'square'` or `'angular'`
TerminalLabels	Controls the display of terminal labels over the axis tick labels, when the `'Type'` property is `'square'` or `'angular'`. Choices are `true` (default) or `false`.
LLRotationValue	Controls the rotation of leaf labels so that the text aligns to the root node, when the `'Type'` property is `'radial'`, `'equalangle'`, or `'equaldaylight'`. Choices are `true` or `false` (default).

Output Arguments

H	Structure with handles to seven graph elements. The structure includes the following fields: • axes • BranchLines • BranchDots • LeafDots

- branchNodeLabels
- leafNodeLabels
- terminalNodeLabels

Tip Use the set function with the handles in this structure and their related properties to modify the plot. For more information on the properties you can modify using the axes handle, see Axes Properties. For more information on the properties you can modify using the BranchLines, BranchDots, or LeafDots handle, see Primitive Line Properties. For more information on the properties you can modify using the branchNodeLabels, leafNodeLabels, or terminalNodeLabels handle, see Text Properties.

Description

plot(*Tree*) draws a phylogenetic tree object into a figure as a phylogram. The significant distances between branches and nodes are in the horizontal direction. Vertical distances are arbitrary and have no significance.

plot(*Tree*, *ActiveBranches*) hides the nonactive branches and all of their descendants in the Figure window. *ActiveBranches* is a logical array of size numBranches-by-1 indicating the active branches.

H = plot(...) returns a structure with handles to seven graph elements.

plot(..., 'Type', *TypeValue*, ...) specifies a method for rendering the phylogenetic tree. Choices are as follows.

Rendering Type	Description
`'square'` (default)	

Rendering Type	Description
'angular'	
'radial'	

Rendering Type	Description
`'equalangle'`	
	Tip This rendering type hides the significance of the root node and emphasizes clusters, thereby making it useful for visually assessing clusters and detecting outliers.
`'equaldaylight'`	
	Tip This rendering type hides the significance of the root node and emphasizes clusters, thereby making it useful for visually assessing clusters and detecting outliers.

plot(..., 'Orientation', *OrientationValue*, ...) specifies the orientation of the root node, and hence the orientation of a phylogram or cladogram phylogenetic tree in the Figure window, when the 'Type' property is 'square' or 'angular'.

plot(..., 'Rotation', *RotationValue*, ...) specifies the rotation angle (in degrees) of the phylogenetic tree in the Figure window, when the 'Type' property is 'radial', 'equalangle', or 'equaldaylight'. Choices are any scalar between 0 (default) and 360.

plot(..., 'BranchLabels', *BranchLabelsValue*, ...) hides or displays branch labels next to the branch nodes. Choices are true or false (default).

plot(..., 'LeafLabels', *LeafLabelsValue*, ...) hides or displays leaf labels next to the leaf nodes. Choices are true or false. Default is:

- true — When the 'Type' property is 'radial', 'equalangle', or 'equaldaylight'
- false — When the 'Type' property is 'square' or 'angular'

plot(..., 'TerminalLabels', *TerminalLabelsValue*, ...) hides or displays terminal labels over the axis tick labels, when the 'Type' property is 'square' or 'angular'. Choices are true (default) or false.

plot(..., 'LLRotation', *LLRotationValue*, ...) controls the rotation of leaf labels so that the text aligns to the root node, when the 'Type' property is 'radial', 'equalangle', or 'equaldaylight'. Choices are true or false (default).

Examples

```
% Create a phytree object from a file
tr = phytreeread('pf00002.tree')
% Plot the tree and return a structure with handles to the
% graphic elements of the phytree object
h = plot(tr,'Type','radial')

% Modify the font size and color of the leaf node labels
% by using one of the handles in the return structure
set(h.leafNodeLabels,'FontSize',6,'Color',[1 0 0])
```

More About

- phytree object

See Also
```
phytree | cluster | phytreeread | phytreeviewer | seqlinkage |
seqneighjoin | view
```

plotSummary

Class: BioRead

Plot summary statistics of BioRead object

Syntax

```
plotSummary(BRObj)
plotSummary(BRObj,Name,Value)
[H,qsObj] = plotSummary( ___ )
```

Description

`plotSummary(BRObj)` generates a summary statistics figure containing six plots about the average quality score for each base position, average quality score distribution, read count percentage for each base position, percentage of GC content for each base position, GC content distribution, and nucleotide distribution.

`plotSummary(BRObj,Name,Value)` generates a summary statistics figure using additional options specified by one or more name-value pair arguments.

`[H,qsObj] = plotSummary( ___ )` returns a column vector `H` of handles to the axes in the generated figure and a `BioReadQualityStatistics` object `qsObj` using any of the input arguments from the previous syntaxes.

Input Arguments

BRObj

`BioRead` or `BioMap` object.

Default:

Name-Value Pair Arguments

'Encoding'

String specifying the format used for the characters encoding the sequence information and quality scores in a sequence file. Supported formats are `'Sanger'`, `'Illumina13'`, `'Illumina15'`, `'Illumina18'`, and `'Solexa'`.

Default: `'Illumina18'`

'FilterLength'

Positive integer (`n`) specifying the first `n` characters of each read to be used. Use an empty array to supress filtering.

Default: [ ]

'QualityScoreThreshold'

Scalar value specifying a minimum average quality threshold for a read to be considered. Any read with an average score of less than the specified threshold value is ignored. The default value is $-\text{Inf}$, which causes all reads to be considered.

Default: `-Inf`

Note: If `'FilterLength'` is set to `L` and `'QualityScoreThreshold'` is set to `T`, then a read is discarded if the average quality of the first `L` characters of the read is less than `T`.

Output Arguments

`H`

Column vector of handles to the axes in the generated figure.

`qsObj`

`BioReadQualityStatistics` object that stores the data represented by the graphs.

Examples

Plot Summary Statistics of BioRead Object

Create a BioRead object from a FASTQ file.

```
BRObj = BioRead('SRR005164_1_50.fastq');
```

Plot summary statistics of the BioRead object BRObj using the first 40 characters of each read with the minimum average quality score of 5.

```
plotSummary(BRObj,'FilterLength',40,'QualityScoreThreshold',5)
```
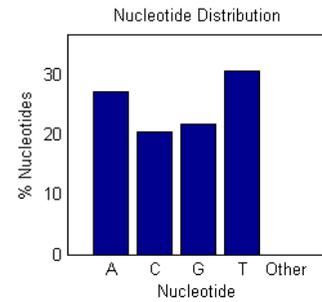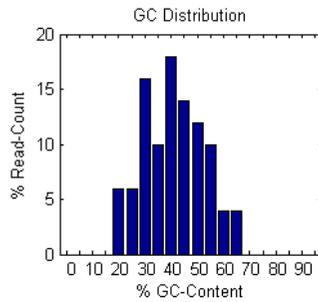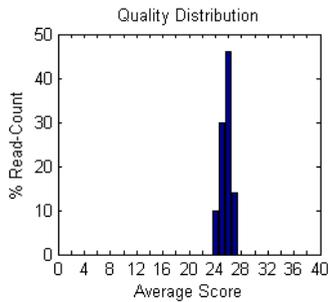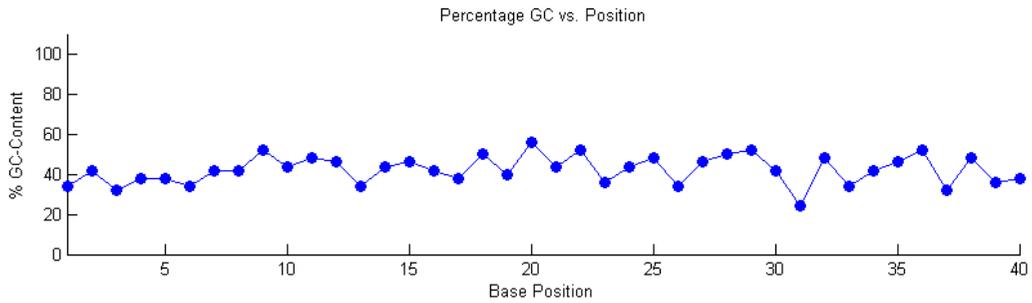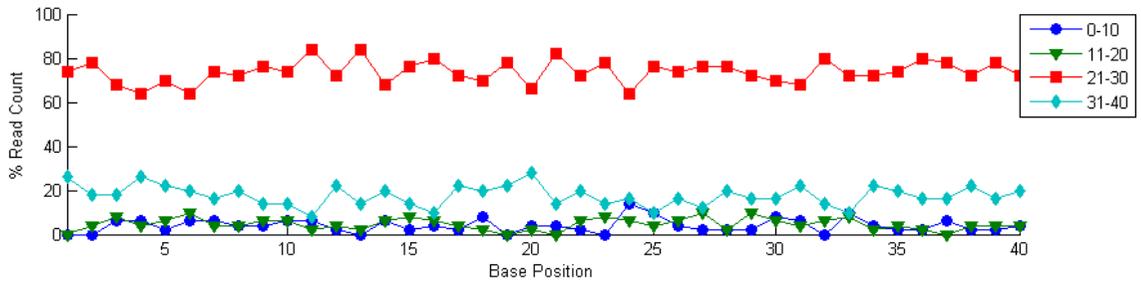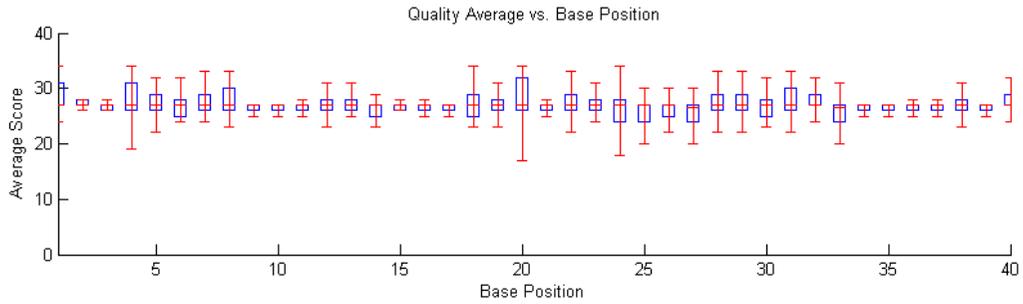
See Also
BioReadQualityStatistics | setQuality | BioRead | BioMap

plotSummary

Class: BioReadQualityStatistics

Plot summary statistics of a BioReadQualityStatistics object

Syntax

```
plotSummary(QSObj)
H = plotSummary(QSObj)
```

Description

plotSummary(QSObj) plots summary statistics of the BioReadQualityStatistics object QSObj that contains six plots about the average quality score for each base position, average quality score distribution, read count percentage for each base position, percentage of GC content for each base position, GC content distribution, and nucleotide distribution.

H = plotSummary(QSObj) returns the handle H to axes object containing generated plot.

Input Arguments

QSObj

BioReadQualityStatistics object.

Default:

Output Arguments

H

Handle to axes object containing generated plot.

Examples

Plot Summary Statistics of BioReadQualityStatistics Object

Create a `BioReadQualityStatistics` object from a FASTQ file using only the first 40 characters of each read with a minimum average quality score of 5.

```
QSObj = BioReadQualityStatistics('SRR005164_1_50.fastq','FilterLength',40,...
                                 'QualityScoreThreshold',5);
```

Plot summary statistics of `QSObj`.

```
plotSummary(QSObj);
```
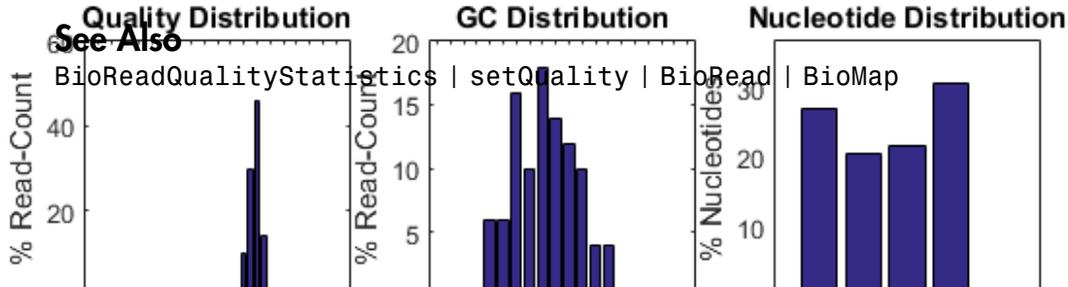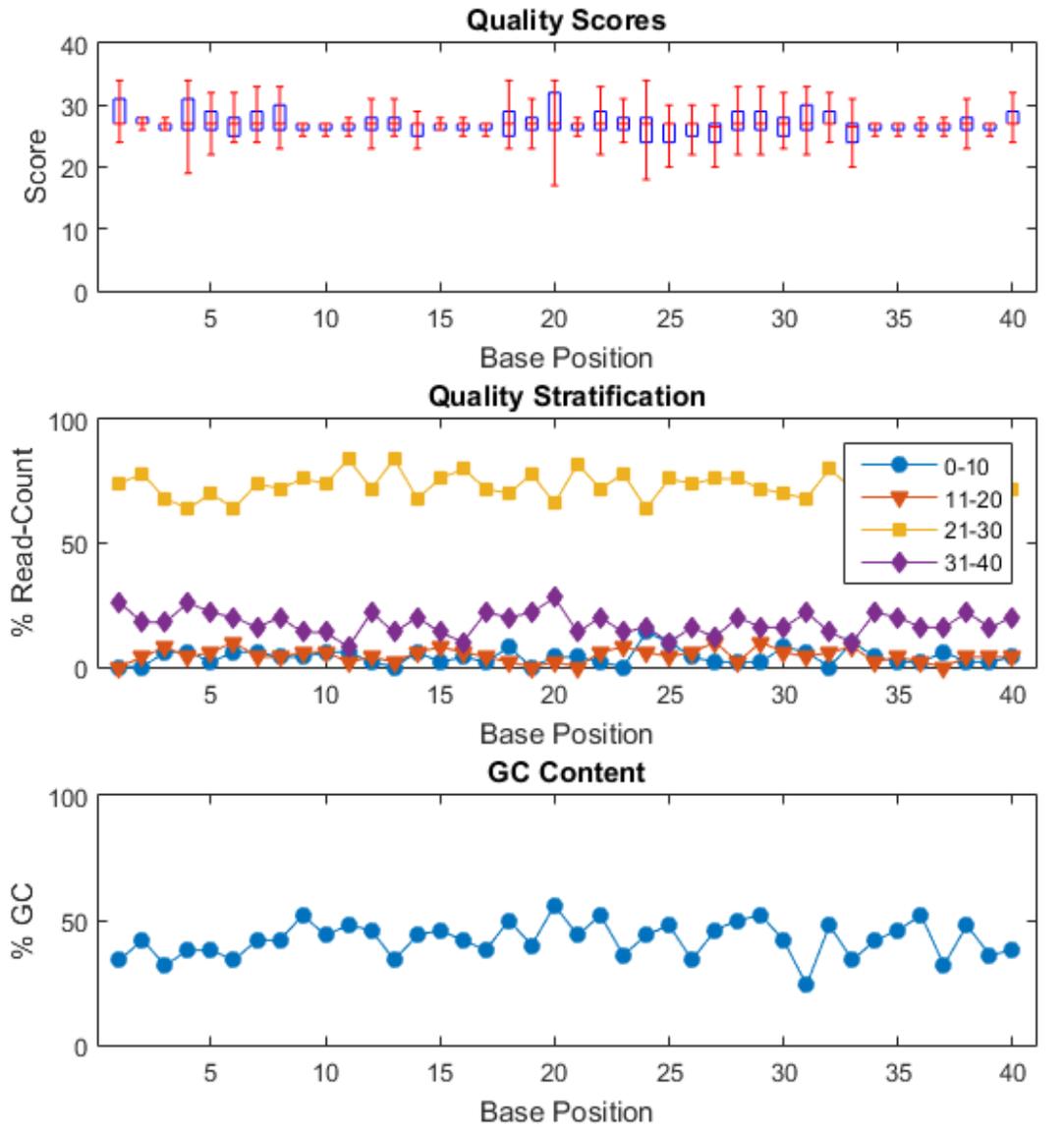
plotTotalGC

Class: BioReadQualityStatistics

Plot distribution of all nucleotides of short-read sequences

Syntax

```
plotTotalGC(QSObj)
H = plotTotalGC(QSObj)
```

Description

plotTotalGC(QSObj) plots the distribution of all nucleotides of short-read sequences in the BioReadQualityStatistics object QSObj.

H = plotTotalGC(QSObj) returns the handle H to axes object containing generated plot.

Input Arguments

QSObj

BioReadQualityStatistics object.

Default:

Output Arguments

H

Handle to axes object containing generated plot.

Examples

Plot Distribution of All Nucleotides

Create a `BioReadQualityStatistics` object from a FASTQ file using only the first 40 characters of each read with a minimum average quality score of 5.

```
QSObj = BioReadQualityStatistics('SRR005164_1_50.fastq','FilterLength',40,...
                                 'QualityScoreThreshold',5);
```

Display the distribution of all nucleotides.

```
plotTotalGC(QSObj);
```
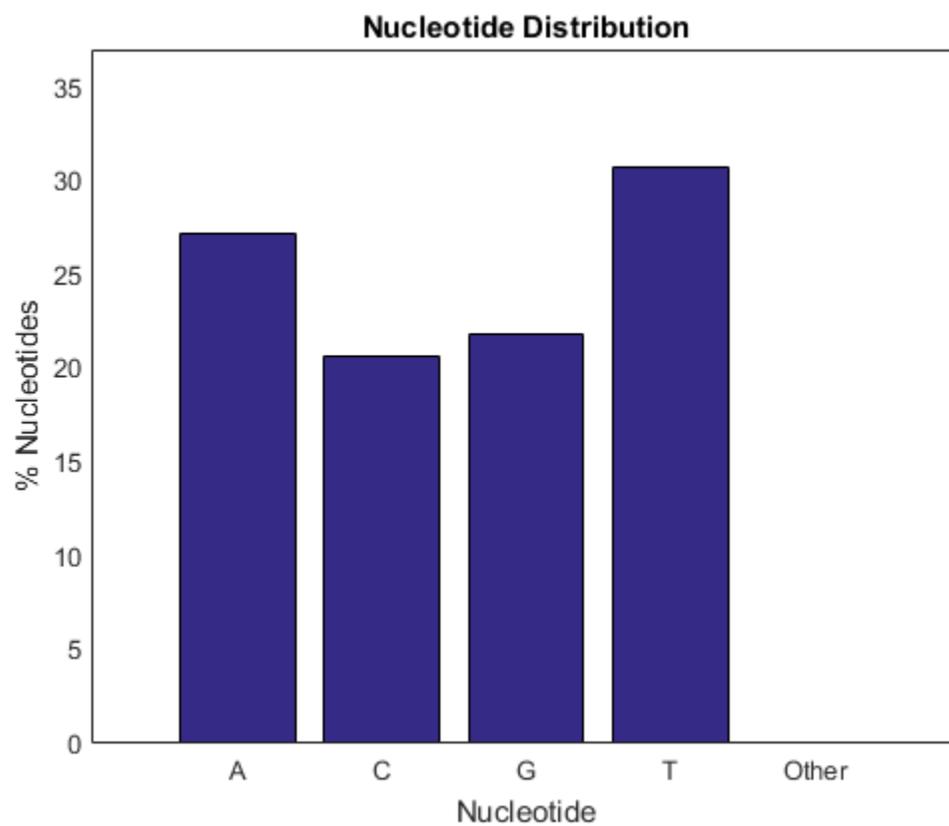
See Also

BioReadQualityStatistics | setQuality | BioRead | BioMap

plotVarianceLink

Plot the sample variance versus the estimate of the condition-dependent mean

Syntax

```
plotVarianceLink(test)
plotVarianceLink(test,Name,Value)
H = plotVarianceLink( ___ )
```

Description

`plotVarianceLink(test)` displays one scatter plot for each experimental condition with the sample variance on the common scale versus the estimate of the condition-dependent mean.

test, an output of the `nbintest` function, is a `NegativeBinomialTest` object, containing results from an unpaired hypothesis test for two independent samples.

If the `'PooledVariance'` name-value pair argument was set to `true` when you ran `nbintest`, then `plotVarianceLink` plots only one scatter plot. The function also plots the variance regression according to the model specified by the `'VarianceLink'` name-value pair argument of `nbintest`.

`plotVarianceLink(test,Name,Value)` uses one or more name-value pair arguments.

`H = plotVarianceLink( ___ )` returns handles to axes.

Examples

Perform unpaired hypothesis test for short-read count data

This example shows how to perform an unpaired hypothesis test for synthetic short-read count data from two different biological conditions.

The data in this example contains synthetic gene count data for 5000 genes, representing two different biological conditions, such as diseased and normal cells. For each condition,

there are five samples. Only 10% of the genes (500 genes) are differentially expressed. Specifically, half of them (250 genes) are exactly 3-fold overexpressed. The other 250 genes are 3-fold underexpressed. The rest of the gene expression data is generated from the same negative binomial distribution for both conditions. Each sample also has a different size factor (that is, the coverage or sampling depth).

Load the data.

```
clear all
load(fullfile(matlabroot,'examples','bioinfo','nbintest_data.mat'),'K','H0');
```

The variable K contains gene count data. The rows represent genes, and the columns represent samples. In this case, the first five columns represent samples from the first condition. The other five columns represent samples from the second condition. Display the first few rows of K.

```
K(1:5,:)
```

```
ans =

  Columns 1 through 6

       13683       14140        8281       14309       12208        8045
       16028       16805        9813       16486       14076        9901
         814         862         492         910         758         521
       15870       16453        9857       16454       14267        9671
        9422        9393        5734        9598        8174        5381

  Columns 7 through 10

        9446       11317       14597       14592
       10927       13348       16999       17036
         573         753         870         936
       10997       13624       17151       17205
        6315        7752        9869        9795
```

In this example, the null hypothesis is true when the gene is not differentially expressed. The variable H0 contains boolean indicators that indicate for which genes the null hypothesis is true (marked as 1). In other words, H0 contains known labels that you will use later to compare with predicted results.

```
sum(H0)
```

```
ans =

        4500
```

Out of 5000 genes, 4500 are not differentially expressed in this synthetic data.

Run an unpaired hypothesis test for samples from two conditions using `nbintest`. The assumption is that the data came from a negative binomial distribution, where the variance is linked to the mean via a locally-regressed smooth function of the mean as described in [1] by setting `'VarianceLink'` to `'LocalRegression'`.

```
tLocal = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','LocalRegression');
```

Use `plotVarianceLink` to plot a scatter plot for each experimental condition (for X and Y conditions), with the sample variance on the common scale versus the estimate of the condition-dependent mean. Use a linear scale for both axes. Include curves for all other linkage options by setting `'Compare'` to `true`.

```
plotVarianceLink(tLocal,'Scale','linear','Compare',true)
```

Variance Link on X

The `Identity` line represents the Poisson model, where the variance is identical to the mean as described in [3]. Observe that the data seems to be overdispersed (that is, most points are above the `Identity` line). The `Constant` line represents the negative binomial model, where the variance is the sum of the shot noise term (mean) and a constant multiplied by the squared mean as described in [2]. The `Local Regression` and `Constant` linkage options appear to fit better with the overdispersed data.

Use `plotChiSquaredFit` to assess the goodness-of-fit for variance regression. It plots the empirical CDF (ecdf) of the chi-squared probabilities. The probabilities are the ratio between the observed and the estimated variance stratified by short-read count levels into five equal-sized bins.

```
plotChiSquaredFit(tLocal)
```

Residuals ECDF Plot for X

Each figure shows five ecdf curves. Each curve represents one of the five short-read count levels. For instance, the blue line represents the ecdf curve for a low short-read counts between 0 and 1264. The red line represents high counts (more than 11438).

One way to interpret the curves is to check if the ecdf curves are above the diagonal line. If they are above the line, then the variance is overestimated. If they are below the line, then the variance is underestimated. In both figures, the variance seems to be correctly estimated for higher counts (that is, the red line follows the diagonal line), but slightly overestimated for lower count levels.

To assess the performance of the hypothesis test, construct a confusion matrix using the known labels and the predicted p-values.

```
confusionmat(H0,(tLocal.pValue > .001))
```

```
ans =

      493           7
        5        4495
```

Out of 500 differentially expressed genes, 493 are correctly predicted (true positives) and 7 of them are incorrectly predicted as not-differentially expressed genes (false negatives). Out of 4500 genes that are not differentially expressed, 4495 are correctly predicted (true negatives) and 5 of them are incorrectly predicted as differentially expressed genes (false positives).

For a comparison, run the hypothesis test again assuming that counts are modeled by the Poisson distribution, where the variance is identical to the mean.

```
tPoisson = nbintest(K(:,1:5),K(:,6:10),'VarianceLink','Identity');
```

Plot the ecdf curves. Observe that all the curves are below the diagonal line, implying that the variance is underestimated. Therefore, the negative binomial model fits the data better.

```
plotChiSquaredFit(tPoisson)
```

Residuals ECDF Plot for X

Input Arguments

`test` — Unpaired hypothesis test result
NegativeBinomialTest object (default)

Unpaired hypothesis test results, specified as a `NegativeBinomialTest` object. test is returned by the `nbintest` function.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'  '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Compare',true,'Scale','linear'`

`'Compare'` — Logical flag to add a curve with the variance link for other models
`false` (default) | `true`

Logical flag to add a curve with the variance link for other models, specified as a comma-separated pair consisting of `'Compare'` and `true` or `false`. When it is set to `true`, the plot shows curves for all the available linkage options, that is, `'LocalRegression'`, `'Constant'`, and `'Identity'`.

Example: `'Compare',true`

`'Scale'` — Scale for both axes
`'log'` (default) | `'linear'`

Scale for both axes, specified as a comma-separated pair consisting of `'Scale'` and `'log'` or `'linear'`.

Example: `'Scale','linear'`

Output Arguments

H — Handles to axes
vector of handles

Handles to axes, specified as a vector of handles.

See Also
mattest | nbintest | NegativeBinomialTest | plotChiSquaredFit

plus (DataMatrix)

Add DataMatrix objects

Syntax

DMObjNew = plus(*DMObj1*, *DMObj2*)
DMObjNew = *DMObj1* + *DMObj2*
DMObjNew = plus(*DMObj1*, *B*)
DMObjNew = *DMObj1* + *B*
DMObjNew = plus(*B*, *DMObj1*)
DMObjNew = *B* + *DMObj1*

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by DataMatrix (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

DMObjNew	DataMatrix object created by addition.

Description

DMObjNew = plus(*DMObj1*, *DMObj2*) or the equivalent *DMObjNew* = *DMObj1* + *DMObj2* performs an element-by-element addition of the DataMatrix objects *DMObj1* and *DMObj2* and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = `plus(`*DMObj1,* *B*`)` or the equivalent *DMObjNew* = *DMObj1* + *B* performs an element-by-element addition of *DMObj1*, a DataMatrix object, and *B*, a numeric or logical array, and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

DMObjNew = `plus(`*B,* *DMObj1*`)` or the equivalent *DMObjNew* = *B* + *DMObj1* performs an element-by-element addition of *B*, a numeric or logical array, and *DMObj1*, a DataMatrix object, and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

Note: Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls *DMObjNew* = `plus(`*X, Y*`)` for the syntax *DMObjNew* = *X* + *Y* when *X* or *Y* is a DataMatrix object.

More About

· DataMatrix object

See Also
`DataMatrix` | `minus`

power (DataMatrix)

Array power DataMatrix objects

Syntax

```
DMObjNew = power(DMObj1, DMObj2)
DMObjNew = DMObj1 .^ DMObj2
DMObjNew = power(DMObj1, B)
DMObjNew = DMObj1 .^ B
DMObjNew = power(B, DMObj1)
DMObjNew = B .^ DMObj1
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

DMObjNew	DataMatrix object created by array power.

Description

DMObjNew = `power(`*DMObj1*`, `*DMObj2*`)` or the equivalent *DMObjNew* = *DMObj1* `.^` *DMObj2* performs an element-by-element power of the DataMatrix objects *DMObj1* and *DMObj2* and places the results in *DMObjNew*, another DataMatrix object. In other words, `power` raises each element in *DMObj1* by the corresponding element in *DMObj2*. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = power(*DMObj1*, *B*) or the equivalent *DMObjNew* = *DMObj1* .^ *B* performs an element-by-element power of the DataMatrix object *DMObj1* and *B*, a numeric or logical array, and places the results in *DMObjNew*, another DataMatrix object. In other words, power raises each element in *DMObj1* by the corresponding element in *B*. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

DMObjNew = power(*B*, *DMObj1*) or the equivalent *DMObjNew* = *B* .^ *DMObj1* performs an element-by-element power of *B*, a numeric or logical array, and the DataMatrix object *DMObj1*, and places the results in *DMObjNew*, another DataMatrix object. In other words, power raises each element in *B* by the corresponding element in *DMObj1*. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

Note: Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls *DMObjNew* = power(*X*, *Y*) for the syntax *DMObjNew* = *X* .^ *Y* when *X* or *Y* is a DataMatrix object.

More About

- DataMatrix object

See Also
DataMatrix | times

probelibraryinfo

Create table of probe set library information

Syntax

ProbeInfo = probelibraryinfo(*CELStruct*, *CDFStruct*)

Input Arguments

CELStruct	Structure created by the affyread function from an Affymetrix CEL file.
CDFStruct	Structure created by the affyread function from an Affymetrix CDF library file associated with the CEL file.

Output Arguments

ProbeInfo	Three-column matrix with the same number of rows as the Probes field of the *CELStruct*. • Column 1 — Probe set ID/name to which the probe belongs. (Probes that do not belong to a probe set in the CDF library file have probe set ID/name equal to 0.) • Column 2 — Contains the probe pair number. • Column 3 — Indicates if the probe is a perfect match (1) or mismatch (-1) probe.

Description

ProbeInfo = probelibraryinfo(*CELStruct*, *CDFStruct*) creates a table of information linking the probe data from *CELStruct*, a structure created from an Affymetrix CEL file, with probe set information from *CDFStruct*, a structure created from an Affymetrix CDF file.

Note: Affymetrix probe pair indexing is 0-based, while MATLAB software indexing is 1-based. The output from `probelibraryinfo` is 1-based.

Examples

The following example uses a sample CEL file and the CDF library file from the *E. coli* Antisense Genome array, which you can download from:

```
http://www.affymetrix.com/support/technical/sample_data/demo_data.affx
```

After you download the sample data, you will need the Affymetrix Data Transfer Tool to extract the CEL file from a DTT file. You can download the Affymetrix Data Transfer Tool from:

```
http://www.affymetrix.com/browse/products.jsp?productId=131431&navMode=34000&navAction=
```

The following example assumes that the `Ecoli-antisense-121502.CEL` file is stored on the MATLAB search path or in the current folder. It also assumes that the associated CDF library file, `Ecoli_ASv2.CDF`, is stored at `D:\Affymetrix\LibFiles\Ecoli`.

1 Read the contents of a CEL file into a MATLAB structure.

```
celStruct = affyread('Ecoli-antisense-121502.CEL');
```

2 Read the contents of a CDF file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

3 Extract probe set library information.

```
ProbeInfo = probelibraryinfo(celStruct, cdfStruct);
```

4 Determine the probe set to which the 1104th probe belongs.

```
cdfStruct.ProbeSets(ProbeInfo(1104,1)).Name

ans =

thrA_b0002_at
```

See Also

affyread | celintensityread | probesetlink | probesetlookup | probesetplot | probesetvalues

probesetlink

Display probe set information on NetAffx Web site

Syntax

```
probesetlink(AffyStruct, PS)
URL = probesetlink(AffyStruct, PS)
probesetlink(AffyStruct, PS, ...'Source', SourceValue, ...)
probesetlink(AffyStruct, PS, ...'Browser', BrowserValue, ...)
URL = probesetlink(AffyStruct, PS, ...'NoDisplay',
NoDisplayValue, ...)
```

Input Arguments

AffyStruct	Structure created by the `affyread` function from an Affymetrix CHP file or an Affymetrix CDF library file.
PS	Probe set index or the probe set ID/name.
SourceValue	Controls the linking to the data source (for example, GenBank or Flybase) for the probe set (instead of linking to the NetAffx™ Web site). Choices are `true` or `false` (default). **Note:** This property requires the GIN library file associated with the CHP or CDF file to be located in the same folder as the CDF library file.
BrowserValue	Controls the display of the probe set information in your system's default Web browser. Choices are `true` or `false` (default).
NoDisplayValue	Controls the return of *URL* without opening a Web browser. Choices are `true` or `false` (default).

Output Arguments

URL	URL for the probe set information.

Description

probesetlink(*AffyStruct*, *PS*) opens a Web Browser window displaying information on the NetAffx Web site about a probe set specified by *PS*, a probe set index or the probe set ID/name, and *AffyStruct*, a structure created from an Affymetrix CHP file or Affymetrix CDF library file.

URL = probesetlink(*AffyStruct*, *PS*) also returns the URL (linking to the NetAffx Web site) for the probe set information.

probesetlink(*AffyStruct*, *PS*, ...'*PropertyName*', *PropertyValue*, ...) calls probesetlink with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

probesetlink(*AffyStruct*, *PS*, ...'Source', *SourceValue*, ...) controls the linking to the data source (for example, GenBank or Flybase) for the probe set (instead of linking to the NetAffx Web site). Choices are true or false (default).

Note: The 'Source' property requires the GIN library file associated with the CHP or CDF file to be located in the same folder as the CDF library file.

---

probesetlink(*AffyStruct*, *PS*, ...'Browser', *BrowserValue*, ...) controls the display of the probe set information in your system's default Web browser. Choices are true or false (default).

URL = probesetlink(*AffyStruct*, *PS*, ...'NoDisplay', *NoDisplayValue*, ...) controls the return of the URL without opening a Web browser. Choices are true or false (default).

Note: The NetAffx Web site requires you to register and provide a user name and password.

Examples

The following example uses a sample CHP file and the CDF library file from the *E. coli* Antisense Genome array, which you can download from:

```
http://www.affymetrix.com/support/technical/sample_data/demo_data.affx
```

After you download the sample data, you will need the Affymetrix Data Transfer Tool to extract the CHP file from a DTT file. You can download the Affymetrix Data Transfer Tool from:

```
http://www.affymetrix.com/browse/products.jsp?productId=131431&navMode=34000&navAction=
```

The following example assumes that the `Ecoli-antisense-121502.CHP` file is stored on the MATLAB search path or in the current folder. It also assumes that the associated CDF library file, `Ecoli_ASv2.CDF`, is stored at `D:\Affymetrix\LibFiles\Ecoli`.

1 Read the contents of a CHP file into a MATLAB structure.

```
chpStruct = affyread('Ecoli-antisense-121502.CHP',...
              'D:\Affymetrix\LibFiles\Ecoli');
```

2 Display information from the NetAffx Web site for the `argG_b3172_at` probe set.

```
probesetlink(chpStruct,'argG_b3172_at')
```

See Also

`affyread` | `celintensityread` | `probelibraryinfo` | `probesetlookup` | `probesetplot` | `probesetvalues`

probesetlookup

Look up information for Affymetrix probe set

Syntax

PSStruct = probesetlookup(*AffyStruct*, *ID*)

Input Arguments

AffyStruct	Structure created by the affyread function from an Affymetrix CHP file or an Affymetrix CDF library file for expression assays.
ID	String or cell array of strings specifying one or more probe set IDs/ names or gene IDs.

Output Arguments

PSStruct	Structure or array of structures containing the following fields for a probe set: • Identifier — Gene ID associated with the probe set • ProbeSetName — Probe set ID/name • CDFIndex — Index into the CDF structure for the probe set • GINIndex — Index into the GIN structure for the probe set • Description — Description of the probe set • Source — Source(s) of the probe set • SourceURL — Source URL(s) for the probe set

Description

PSStruct = probesetlookup(*AffyStruct*, *ID*) returns a structure or an array of structures containing information for an Affymetrix probe set specified by ID, a string

or cell array of strings specifying one or more probe set IDs/names or gene IDs, and by *AffyStruct*, a structure created from an Affymetrix CHP file or Affymetrix CDF library file for expression assays.

Note: This function works with CHP files and CDF files for expression assays only. It requires that the GIN library file associated with the CHP file or CDF file to be located in the same folder as the CDF library file.

Examples

The following example uses the CDF library file from the *E. coli* Antisense Genome array, which you can download from:

```
http://www.affymetrix.com/support/technical/sample_data/demo_data.affx
```

The following example assumes that the `Ecoli_ASv2.CDF` library file is stored at `D:\Affymetrix\LibFiles\Ecoli`.

1 Read the contents of a CDF library file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

2 Look up the gene ID (`Identifier`) associated with the `argG_b3172_at` probe set.

```
probesetlookup(cdfStruct,'argG_b3172_at')

ans =

     Identifier: '3315278'
    ProbeSetName: 'argG_b3172_at'
        CDFIndex: 5213
        GINIndex: 3074
     Description: [1x82 char]
          Source: 'NCBI EColi Genome'
        SourceURL: [1x74 char]
```

See Also

affyread | celintensityread | probelibraryinfo | probesetlink | probesetplot | probesetvalues | rmabackadj

probesetplot

Plot Affymetrix probe set intensity values

Syntax

```
probesetplot(CELStruct, CDFStruct, PS)
probesetplot(CELStruct, CDFStruct, PS, ...'GeneName',
GeneNameValue, ...)
probesetplot(CELStruct, CDFStruct, PS, ...'Field', FieldValue, ...)
probesetplot(CELStruct, CDFStruct, PS, ...'ShowStats',
ShowStatsValue, ...)
```

Arguments

CELStruct	Structure created by the `affyread` function from an Affymetrix CEL file.
CDFStruct	Structure created by the `affyread` function from an Affymetrix CDF library file associated with the CEL file.
PS	Probe set index or the probe set ID/name.
GeneNameValue	Controls whether the probe set name or the gene name is used for the title of the plot. Choices are `true` or `false` (default). **Note:** The `'GeneName'` property requires the GIN library file associated with the CEL and CDF files to be located in the same folder as the CDF library file from which *CDFStruct* was created.
FieldValue	String specifying the type of data to plot. Choices are: • `'Intensity'` (default) • `'StdDev'` • `'Background'` • `'Pixels'`

	• `'Outlier'`
ShowStatsValue	Controls whether the mean and standard deviation lines are included in the plot. Choices are `true` or `false` (default).

Description

`probesetplot(`*CELStruct*`, `*CDFStruct*`, `*PS*`)` plots the PM (perfect match) and MM (mismatch) intensity values for a specified probe set. *CELStruct* is a structure created by the `affyread` function from an Affymetrix CEL file. *CDFStruct* is a structure created by the `affyread` function from an Affymetrix CDF library file associated with the CEL file. *PS* is the probe set index or the probe set ID/name.

Note: MATLAB software uses 1-based indexing for probe set numbers, while the Affymetrix CDF file uses 0-based indexing for probe set numbers. For example, `CDFStruct.ProbeSets(1)` has a `ProbeSetNumber` of `0` in the `ProbePairs` field.

`probesetplot(`*CELStruct*`, `*CDFStruct*`, `*PS*`, ...'`*PropertyName*`', `*PropertyValue*`, ...)` calls `probesetplot` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`probesetplot(`*CELStruct*`, `*CDFStruct*`, `*PS*`, ...'GeneName', `*GeneNameValue*`, ...)` controls whether the probe set name or the gene name is used for the title of the plot. Choices are `true` or `false` (default).

Note: The `'GeneName'` property requires the GIN library file associated with the CEL and CDF files to be located in the same folder as the CDF library file from which *CDFStruct* was created.

`probesetplot(`*CELStruct*`, `*CDFStruct*`, `*PS*`, ...'Field', `*FieldValue*`, ...)` specifies the type of data to plot. Choices are:

- `'Intensity'` (default)
- `'StdDev'`

- `'Background'`
- `'Pixels'`
- `'Outlier'`

probesetplot(*CELStruct*, *CDFStruct*, *PS*, ...'ShowStats', *ShowStatsValue*, ...) controls whether the mean and standard deviation lines are included in the plot. Choices are `true` or `false` (default).

Examples

The following example uses a sample CEL file and the CDF library file from the *E. coli* Antisense Genome array, which you can download from:

```
http://www.affymetrix.com/support/technical/sample_data/demo_data.affx
```

After you download the sample data, you will need the Affymetrix Data Transfer Tool to extract the CEL file from a DTT file. You can download the Affymetrix Data Transfer Tool from:

```
http://www.affymetrix.com/browse/products.jsp?productId=131431&navMode=34000&navAction=
```

The following example assumes that the `Ecoli-antisense-121502.CEL` file is stored on the MATLAB search path or in the current folder. It also assumes that the associated CDF library file, `Ecoli_ASv2.CDF`, is stored at `D:\Affymetrix\LibFiles\Ecoli`.

1 Read the contents of a CEL file into a MATLAB structure.

```
celStruct = affyread('Ecoli-antisense-121502.CEL');
```

2 Read the contents of a CDF file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

3 Plot the PM and MM intensity values of the `argG_b3172_at` probe set, including the mean and standard deviation.

```
probesetplot(celStruct, cdfStruct, 'argG_b3172_at','showstats', true)
```
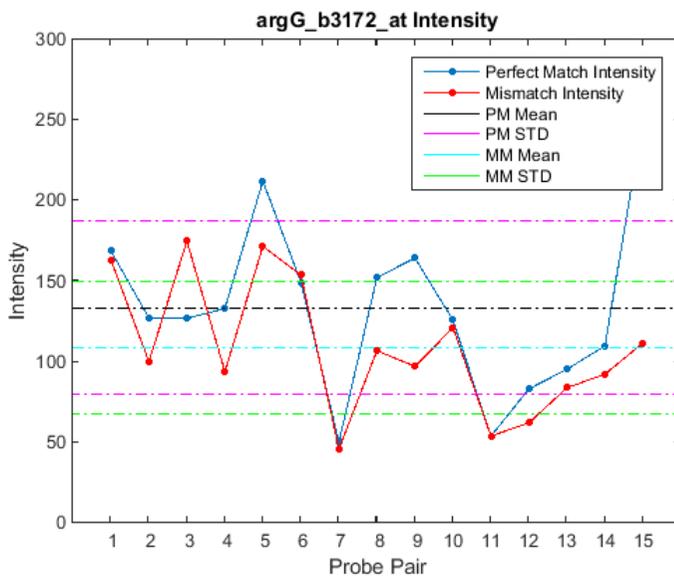
See Also
affyread | celintensityread | probesetlink | probesetlookup |
probesetvalues

probesetvalues

Create table of Affymetrix probe set intensity values

Syntax

```
PSValues = probesetvalues(CELStruct, CDFStruct, PS)
PSValues = probesetvalues(CELStruct, CDFStruct, PS, 'Background',
BackgroundValue)
ColumnNames = probesetvalues
```

Input Arguments

CELStruct	Structure created by the `affyread` function from an Affymetrix CEL file.
CDFStruct	Structure created by the `affyread` function from an Affymetrix CDF library file associated with the CEL file.
PS	Probe set index or the probe set ID/name.
BackgroundValue	Controls the background correction in the calculation. Choices are: • `true` (default) — Background values from the `Background` field in the *PSValues* matrix are used to calculate the probe intensity values. • `false` — Background values are not calculated. • A vector of precalculated background values (such as returned by the `zonebackadj` function) whose length is equal to the number of probes in *CELStruct*. These background values are used to calculate the probe intensity values. --- **Tip** Including background correction in the calculation of the probe intensity values can be slow. Therefore, setting `'Background'` to `false` can speed up the calculation.

| | However, the values returned in the `'Background'` field of the *PSValues* matrix will be zero. |

Output Arguments

| *PSValues* | Twenty-column matrix with one row for each probe pair in the probe set. |
| *ColumnNames* | Cell array of strings containing the column names of the *PSValues* matrix. This is returned only when you call `probesetvalues` with no input arguments. |

Description

PSValues = probesetvalues(*CELStruct*, *CDFStruct*, *PS*) creates a table of intensity values for *PS*, a probe set, from the probe-level data in *CELStruct*, a structure created by the `affyread` function from an Affymetrix CEL file. *PS* is a probe set index or probe set ID/name from *CDFStruct*, a structure created by the `affyread` function from an Affymetrix CDF library file associated with the CEL file. *PSValues* is a twenty-column matrix with one row for each probe pair in the probe set. The columns correspond to the following fields.

Column	Field	Description
1	`'ProbeSetNumber'`	Number identifying the probe set to which the probe pair belongs.
2	`'ProbePairNumber'`	Index of the probe pair within the probe set.
3	`'UseProbePair'`	This field is for backward compatibility only and is not currently used.
4	`'Background'`	Estimated background of probe intensity values of the probe pair.
5	`'PMPosX'`	*x*-coordinate of the perfect match probe.
6	`'PMPosY'`	*y*-coordinate of the perfect match probe.
7	`'PMIntensity'`	Intensity value of the perfect match probe.
8	`'PMStdDev'`	Standard deviation of intensity value of the perfect match probe.

Column	Field	Description
9	`'PMPixels'`	Number of pixels in the cell containing the perfect match probe.
10	`'PMOutlier'`	True/false flag indicating if the perfect match probe was marked as an outlier.
11	`'PMMasked'`	True/false flag indicating if the perfect match probe was masked.
12	`'MMPosX'`	x-coordinate of the mismatch probe.
13	`'MMPosY'`	y-coordinate of the mismatch probe.
14	`'MMIntensity'`	Intensity value of the mismatch probe.
15	`'MMStdDev'`	Standard deviation of intensity value of the mismatch probe.
16	`'MMPixels'`	Number of pixels in the cell containing the mismatch probe.
17	`'MMOutlier'`	True/false flag indicating if the mismatch probe was marked as an outlier.
18	`'MMMasked'`	True/false flag indicating if the mismatch probe was masked.
19	`'GroupNumber'`	Number identifying the group to which the probe pair belongs. For expression arrays, this is always 1. For genotyping arrays, this is typically 1 (allele A, sense), 2 (allele B, sense), 3 (allele A, antisense), or 4 (allele B, antisense).
20	`'Direction'`	Number identifying the direction of the probe pair. 1 = sense and 2 = antisense.

Note: MATLAB software uses 1-based indexing for probe set numbers, while the Affymetrix CDF file uses 0-based indexing for probe set numbers. For example, `CDFStruct.ProbeSets(1)` has a `ProbeSetNumber` of 0 in the `ProbePairs` field.

PSValues = probesetvalues(*CELStruct*, *CDFStruct*, *PS*, 'Background', *BackgroundValue*) controls the background correction in the calculation. *BackgroundValue* can be:

- true (default) — Background values from the Background field in the *PSValues* matrix are used to calculate the probe intensity values.

- false — Background values are not calculated.

- A vector of precalculated background values (such as returned by the zonebackadj function) whose length is equal to the number of probes in *CELStruct*. These background values are used to calculate the probe intensity values.

Tip Including background correction in the calculation of the probe intensity values can be slow. Therefore, setting 'Background' to false can speed up the calculation. However, the values returned in the 'Background' field of the *PSValues* matrix will be zero.

ColumnNames = probesetvalues returns a cell array of strings containing the column names of the *PSValues* matrix. *ColumnNames* is returned only when you call probesetvalues without input arguments. The information contained in *ColumnNames* is common to all Affymetrix GeneChip arrays.

Examples

The following example uses a sample CEL file and the CDF library file from the *E. coli* Antisense Genome array, which you can download from:

http://www.affymetrix.com/support/technical/sample_data/demo_data.affx

After you download the sample data, you will need the Affymetrix Data Transfer Tool to extract the CEL file from a DTT file. You can download the Affymetrix Data Transfer Tool from:

http://www.affymetrix.com/browse/products.jsp?productId=131431&navMode=34000&navAction=

The following example assumes that the Ecoli-antisense-121502.CEL file is stored on the MATLAB search path or in the current folder. It also assumes that the associated CDF library file, Ecoli_ASv2.CDF, is stored at D:\Affymetrix\LibFiles\Ecoli.

1 Read the contents of a CEL file into a MATLAB structure.

    celStruct = affyread('Ecoli-antisense-121502.CEL');

2 Read the contents of a CDF file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

3 Use the `zonebackadj` function to return a matrix or cell array of vectors containing the estimated background values for each probe.

```
[baData,zones,background] = zonebackadj(celStruct,'cdf',cdfStruct);
```

4 Create a table of intensity values for the `argG_b3172_at` probe set.

```
psvals = probesetvalues(celStruct, cdfStruct, 'argG_b3172_at',...
         'background',background);
```

See Also
affyread | celintensityread | probelibraryinfo | probesetlink | probesetlookup | probesetplot | rmabackadj | zonebackadj

profalign

Align two profiles using Needleman-Wunsch global alignment

Syntax

```
Prof = profalign(Prof1, Prof2)
[Prof, H1, H2] = profalign(Prof1, Prof2)
profalign(..., 'ScoringMatrix', ScoringMatrixValue, ...)
profalign(..., 'GapOpen', {G1Value, G2Value}, ...)
profalign(..., 'ExtendGap', {E1Value, E2Value}, ...)
profalign(..., 'ExistingGapAdjust', ExistingGapAdjustValue, ...)
profalign(..., 'TerminalGapAdjust', TerminalGapAdjustValue, ...)
profalign(..., 'ShowScore', ShowScoreValue, ...)
```

Description

Prof = profalign(*Prof1*, *Prof2*) returns a new profile (*Prof*) for the optimal global alignment of two profiles (*Prof1*, *Prof2*). The profiles (*Prof1*, *Prof2*) are numeric arrays of size `[(4 or 5 or 20 or 21) x Profile Length]` with counts or weighted profiles. Weighted profiles are used to down-weight similar sequences and up-weight divergent sequences. The output profile is a numeric matrix of size `[(5 or 21) x New Profile Length]` where the last row represents gaps. Original gaps in the input profiles are preserved. The output profile is the result of adding the aligned columns of the input profiles.

[*Prof, H1, H2*] = profalign(*Prof1*, *Prof2*) returns pointers that indicate how to rearrange the columns of the original profiles into the new profile.

profalign(..., '*PropertyName*', *PropertyValue*, ...) calls profalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

profalign(..., 'ScoringMatrix', *ScoringMatrixValue*, ...) defines the scoring matrix to be used for the alignment.

ScoringMatrixValue can be either of the following:

- String specifying the scoring matrix to use for the alignment. Choices for amino acid sequences are:

  - `'BLOSUM62'`
  - `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'`
  - `'BLOSUM100'`
  - `'PAM10'` increasing by 10 up to `'PAM500'`
  - `'DAYHOFF'`
  - `'GONNET'`

  Default is:

  - `'BLOSUM50'` — When *AlphabetValue* equals `'AA'`
  - `'NUC44'` — When *AlphabetValue* equals `'NT'`

  Note: The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the `'Scale'` property to specify an additional scale factor to convert the output score from bits to another unit.

- Matrix representing the scoring matrix to use for the alignment, such as returned by the `blosum`, `pam`, `dayhoff`, `gonnet`, or `nuc44` function.

  Note: If you use a scoring matrix that you created or was created by one of the above functions, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix.

Note: If you need to compile `profalign` into a stand-alone application or software component using MATLAB Compiler, use a matrix instead of a string for *ScoringMatrixValue*.

`profalign(..., 'GapOpen', {G1Value, G2Value}, ...)` sets the penalties for opening a gap in the first and second profiles respectively. *G1Value* and *G2Value* can be either scalars or vectors. When using a vector, the number of elements is one more than the length of the input profile. Every element indicates the position specific penalty for opening a gap between two consecutive symbols in the sequence. The first and the last

elements are the gap penalties used at the ends of the sequence. The default gap open penalties are {10,10}.

profalign(..., 'ExtendGap', {*E1Value*, *E2Value*}, ...) sets the penalties for extending a gap in the first and second profile respectively. *E1Value* and *E2Value* can be either scalars or vectors. When using a vector, the number of elements is one more than the length of the input profile. Every element indicates the position specific penalty for extending a gap between two consecutive symbols in the sequence. The first and the last elements are the gap penalties used at the ends of the sequence. If ExtendGap is not specified, then extensions to gaps are scored with the same value as GapOpen.

profalign(..., 'ExistingGapAdjust', *ExistingGapAdjustValue*, ...), if *ExistingGapAdjustValue* is false, turns off the automatic adjustment based on existing gaps of the position-specific penalties for opening a gap. When *ExistingGapAdjustValue* is true (default), for every profile position, profalign proportionally lowers the penalty for opening a gap toward the penalty of extending a gap based on the proportion of gaps found in the contiguous symbols and on the weight of the input profile.

profalign(..., 'TerminalGapAdjust', *TerminalGapAdjustValue*, ...), when *TerminalGapAdjustValue* is true, adjusts the penalty for opening a gap at the ends of the sequence to be equal to the penalty for extending a gap. Default is false.

profalign(..., 'ShowScore', *ShowScoreValue*, ...), when *ShowScoreValue* is true, displays the scoring space and the winning path.

Examples

1 Read in sequences and create profiles.

```
ma1 = ['RGTANCDMQDA';'RGTAHCDMQDA';'RRRAPCDL-DA'];
ma2 = ['RGTHCDLADAT';'RGTACDMADAA'];
p1  = seqprofile(ma1,'gaps','all','counts',true);
p2  = seqprofile(ma2,'counts',true);
```

2 Merge two profiles into a single one by aligning them.

```
p = profalign(p1,p2);
seqlogo(p)
```

3 Use the output pointers to generate the multiple alignment.

```
[p, h1, h2] = profalign(p1,p2);
```

```
ma = repmat('-',5,12);
ma(1:3,h1) = ma1;
ma(4:5,h2) = ma2;
disp(ma)
```

4 Increase the gap penalty before cysteine in the second profile.

```
gapVec = 10 + [p2(aa2int('C'),:) 0] * 10
p3 = profalign(p1,p2,'gapopen',{10,gapVec});
seqlogo(p3)
```

5 Add a new sequence to a profile without inserting new gaps into the profile.

```
gapVec = [0 inf(1,11) 0];
p4 = profalign(p3,seqprofile('PLHFMSVLWDVQQWP'),...
               'gapopen',{gapVec,10});
seqlogo(p4)
```

More About

- hmmprofalign

- nwalign

- seqprofile

See Also

multialign | seqconsensus

proteinplot

Open Protein Plot window to investigate properties of amino acid sequence

Syntax

```
proteinplot
proteinplot (SeqAA)
```

Arguments

SeqAA	Either of the following: • String of single-letter codes specifying an amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers. Unknown characters are mapped to 0. • MATLAB structure containing a `Sequence` field that contains an amino acid sequence, such as returned by `fastaread`, `getgenpept`, `genpeptread`, `getpdb`, or `pdbread`.

Description

The Protein Plot window lets you analyze and compare properties of a single amino acid sequence. It displays smoothed line plots of various properties such as the hydrophobicity of the amino acids in the sequence.

`proteinplot` opens the Protein Plot window.

`proteinplot (SeqAA)` opens the Protein Plot window and loads *SeqAA*, an amino acid sequence, into the window.

Tip You can analyze and compare properties of an amino acid sequence from the MATLAB command line also by using the `proteinpropplot` function.

Examples

Importing Sequences into the Protein Plot Window

You can import a sequence into the Protein Plot window from the MATLAB command line.

1 Retrieve an amino acid sequence from the Protein Data Bank (PDB) database.

```
prion = getpdb('1HJM', 'SEQUENCEONLY', true);
```

2 Load the amino acid sequence into the Protein Plot window.

```
proteinplot(prion)
```

The Protein Plot window opens, and the sequence appears in the **Sequence** text box.
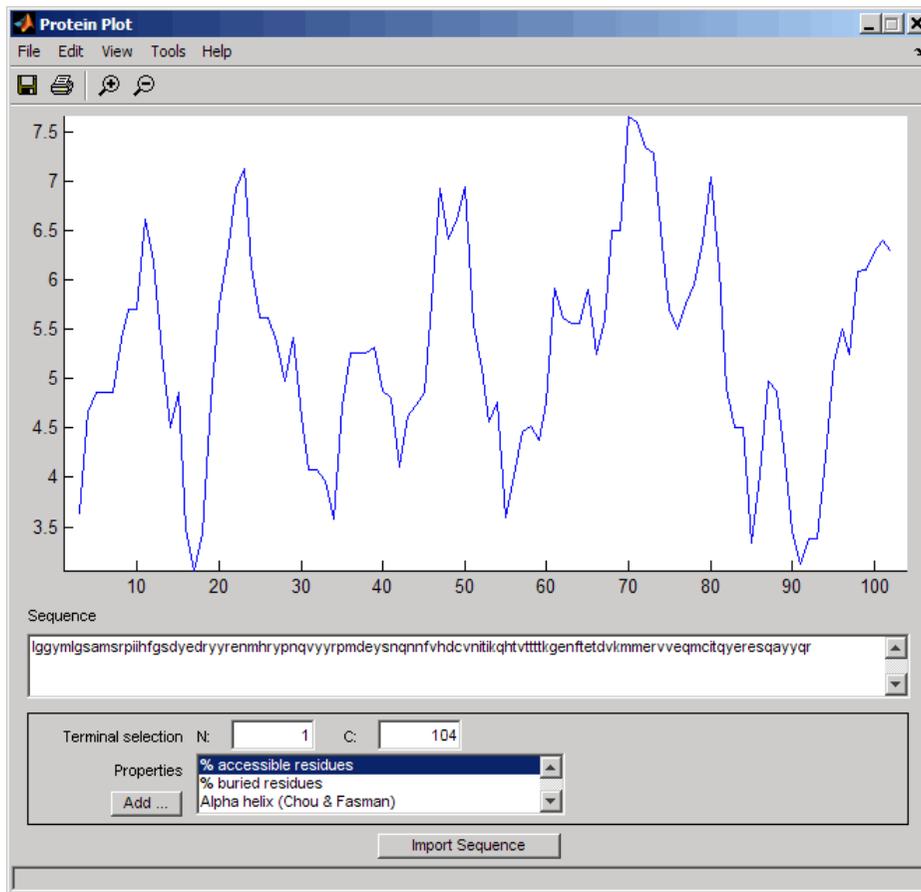
You can import a sequence after the Protein Plot window is open by doing either of the following:

- Type or paste an amino acid sequence into the **Sequence** text box.
- Click the **Import Sequence** button to open the Import dialog box From the **Import From** list, select one of the following:

  - **Workspace** — To select a variable from the MATLAB Workspace
  - **Text File** — To select a text file
  - **FASTA File** — To select a FASTA-formatted file

- **GenPept File** — To select a GenPept-formatted file
- **GenPept Database** — To specify an accession number in the GenPept database

Viewing Properties of Amino Acids

Select a property from the **Properties** drop-down list box to display a smoothed plot of the property values along the sequence. You can select multiple properties from the list by holding down **Shift** or **Ctrl** while selecting properties. When you select two properties, the plots are displayed using a PLOTYY-style layout, with one *y*-axis on the left and one on the right. For all other selections, a single *y*-axis is displayed. When displaying one or two properties, the *y* values displayed are the actual property values. When displaying three or more properties, the values are normalized to the range 0–1.

Accessing Information About the Properties

You can access information about the properties from the **Help** menu.

1   Select **Help > References**. The Help browser opens with a list of properties and references.

2   Scroll down to locate the property of interest.

Using Other Features in the Protein Plot Window

The **Terminal Selection** boxes (N and C) let you choose to plot only part of the sequence. By default, all of the sequence is plotted.

You can add your own properties by clicking on the **Add** button next to the **Properties** list. This opens a Property dialog box that lets you specify the value for each of the amino acids. The **Display Text** box lets you specify the text that will be displayed in the **Properties** list on the main Protein Plot window. You can also save the property values to a file for future use by typing a file name in the **Filename** text box.

The default smoothing method is an unweighted linear moving average with a window length of five residues. You can change this by selecting **Edit > Filter Window Options**. The dialog box lets you select the **Window Size** from 5 to 29 residues. Increasing the window size produces a smoother plot. You can modify the shape of the smoothing window by changing the **Edge Weight** factor. And you can choose the smoothing function to be a linear moving average, an exponential moving average or a linear Lowess smoothing.

The **File** menu lets you import a sequence, save the plot that you have created to a Figure file, export the data values in the figure to a workspace variable or to a MAT-file, export the figure to a normal Figure window for customizing, or print the figure.

The **Edit** menu lets you create a new property, to reset the property values to the default values, and to modify the smoothing parameters with the **Configuration Values** menu item.

The **View** menu lets you turn the toolbar on and off, and to add a legend to the plot.

The **Tools** menu lets you zoom in and zoom out of the plot, to view **Data Statistics** such as mean, minimum and maximum values of the plot, and to normalize the values of the plot from 0 to 1.

The **Help** menu lets you view this document and to see the references for the sequence properties included with the Protein Plot window.

See Also

aacount | atomiccomp | molviewer | molweight | pdbdistplot | proteinpropplot | seqviewer | plotyy

proteinpropplot

Plot properties of amino acid sequence

Syntax

```
proteinpropplot (SeqAA)
proteinpropplot(SeqAA, ...'PropertyTitle', PropertyTitleValue, ...)
proteinpropplot(SeqAA, ...'Startat', StartatValue, ...)
proteinpropplot(SeqAA, ...'Endat', EndatValue, ...)
proteinpropplot(SeqAA, ...'Smoothing', SmoothingValue, ...)
proteinpropplot(SeqAA, ...'EdgeWeight', EdgeWeightValue, ...)
proteinpropplot(SeqAA, ...'WindowLength', WindowLengthValue, ...)
```

Arguments

SeqAA	Amino acid sequence. Enter any of the following: • Character string of letters representing an amino acid • Vector of integers representing an amino acid, such as returned by `aa2int` • Structure containing a `Sequence` field that contains an amino acid sequence, such as returned by `getembl`, `getgenpept`, or `getpdb`
PropertyTitleValue	String that specifies the property to plot. Default is `Hydrophobicity (Kyte & Doolittle)`. To display a list of properties to plot, enter a empty string for *PropertyTitleValue*. For example, type: `proteinpropplot(sequence, 'propertytitle', '')` --- **Tip** To access references for the properties, view the proteinpropplot file.

StartatValue	Integer that specifies the starting point for the plot from the N-terminal end of the amino acid sequence *SeqAA*. Default is 1.
EndatValue	Integer that specifies the ending point for the plot from the N-terminal end of the amino acid sequence *SeqAA*. Default is length(*SeqAA*).
SmoothingValue	String the specifies the smoothing method. Choices are: • linear (default) • exponential • lowess
EdgeWeightValue	Value that specifies the edge weight used for linear and exponential smoothing methods. Decreasing this value emphasizes peaks in the plot. Choices are any value ≥0 and ≤1. Default is 1.
WindowLengthValue	Integer that specifies the window length for the smoothing method. Increasing this value gives a smoother plot that shows less detail. Default is 11.

Description

proteinpropplot (*SeqAA*) displays a plot of the hydrophobicity (Kyte and Doolittle, 1982) of the residues in sequence SeqAA.

proteinpropplot(*SeqAA*, ...'*PropertyName*', *PropertyValue*, ...) calls proteinpropplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

proteinpropplot(*SeqAA*, ...'PropertyTitle', *PropertyTitleValue*, ...) specifies a property to plot for the amino acid sequence *SeqAA*. Default is Hydrophobicity (Kyte & Doolittle). To display a list of possible properties to plot, enter an empty string for *PropertyTitleValue*. For example, type:

proteinpropplot(sequence, 'propertytitle', '')

Tip To access references for the properties, view the proteinpropplot file.

---

proteinpropplot(*SeqAA*, ...'Startat', *StartatValue*, ...) specifies the starting point for the plot from the N-terminal end of the amino acid sequence *SeqAA*. Default is 1.

proteinpropplot(*SeqAA*, ...'Endat', *EndatValue*, ...) specifies the ending point for the plot from the N-terminal end of the amino acid sequence *SeqAA*. Default is length(*SeqAA*).

proteinpropplot(*SeqAA*, ...'Smoothing', *SmoothingValue*, ...) specifies the smoothing method. Choices are:

- linear (default)
- exponential
- lowess

proteinpropplot(*SeqAA*, ...'EdgeWeight', *EdgeWeightValue*, ...) specifies the edge weight used for linear and exponential smoothing methods. Decreasing this value emphasizes peaks in the plot. Choices are any value ≥0 and ≤1. Default is 1.

proteinpropplot(*SeqAA*, ...'WindowLength', *WindowLengthValue*, ...) specifies the window length for the smoothing method. Increasing this value gives a smoother plot that shows less detail. Default is 11.

Examples

Plotting Hydrophobicity

1 Use the getpdb function to retrieve a protein sequence.

```
prion = getpdb('1HJM', 'SEQUENCEONLY', true);
```

2 Plot the hydrophobicity (Kyte and Doolittle, 1982) of the residues in the sequence.

```
proteinpropplot(prion)
```
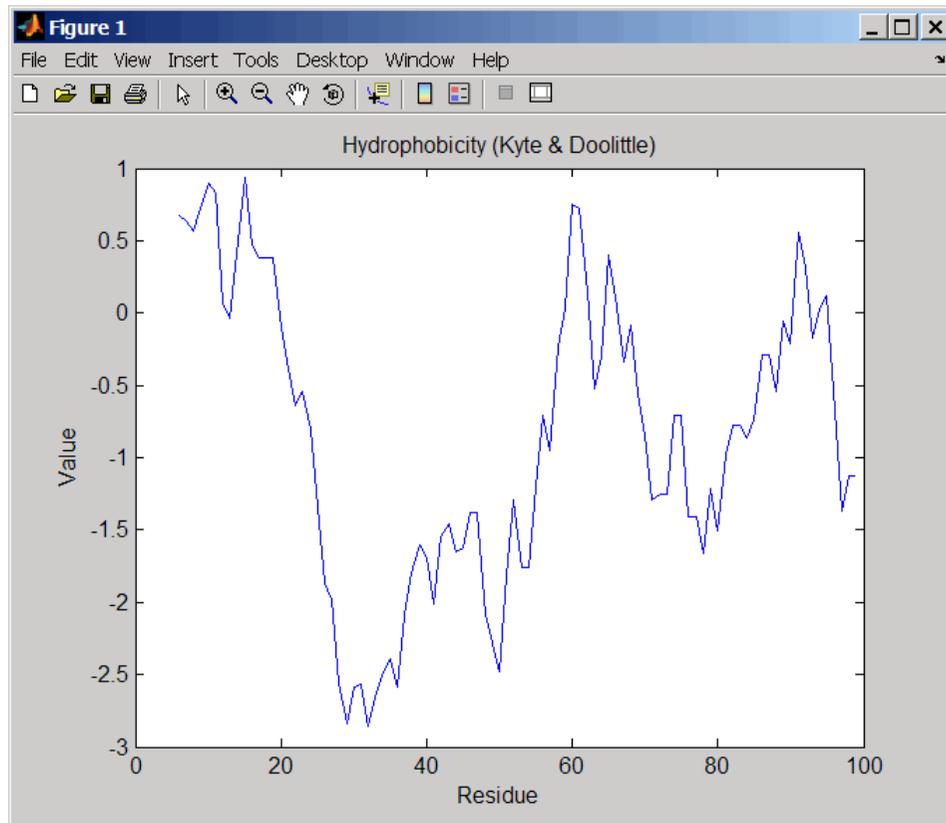
Plotting Parallel Beta Strand

1   Use the `getgenpept` function to retrieve a protein sequence.

```
s = getgenpept('aad50640');
```

2   Plot the conformational preference for parallel beta strand for the residues in the sequence.

```
proteinpropplot(s,'propertytitle','Parallel beta strand')
```
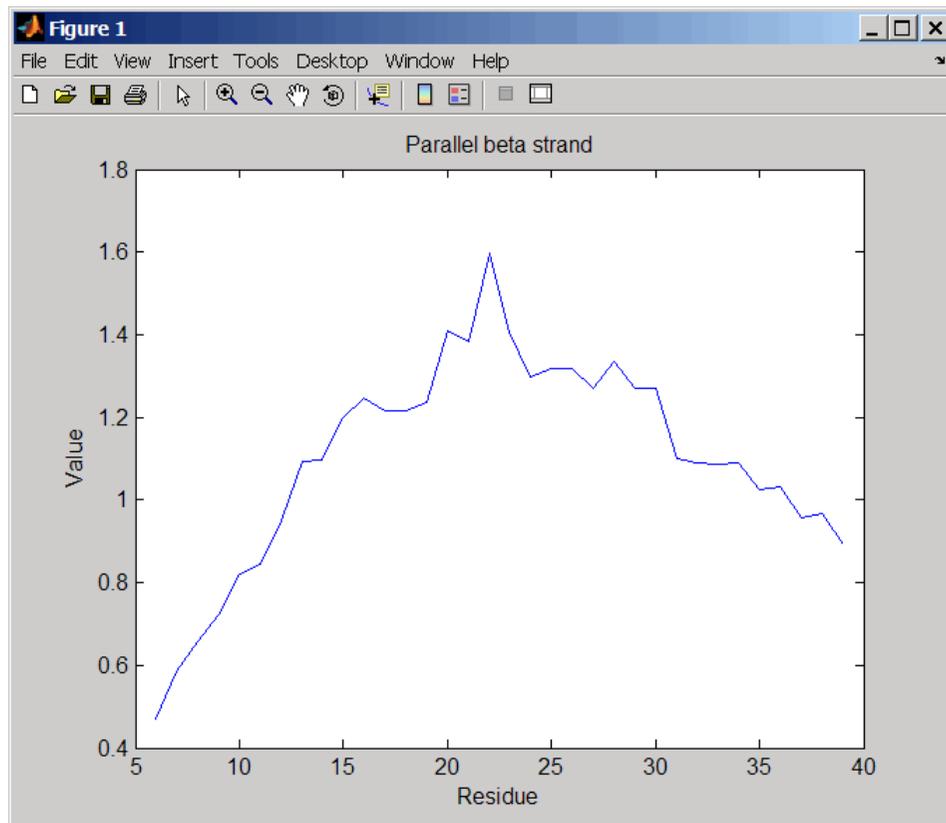
References

[1] Kyte, J., and Doolittle, R.F. (1982). A simple method for displaying the hydropathic character of a protein. J Mol Biol *157(1)*, 105–132.

See Also

aacount | atomiccomp | molviewer | molweight | pdbdistplot | proteinplot | ramachandran | seqviewer | plotyy

prune (phytree)

Remove branch nodes from phylogenetic tree

Syntax

T2 = prune(*T1*, *Nodes*)
T2 = prune(*T1*, *Nodes*, 'Mode','Exclusive')

Arguments

T1	Phylogenetic object created with the phytree constructor function.
Nodes	Nodes to remove from tree.
Mode	Property to control the method of pruning. Enter either 'Inclusive' or 'Exclusive'. The default value is 'Inclusive'.

Description

T2 = prune(*T1*, *Nodes*) removes the nodes listed in the vector *Nodes* from the tree *T1*. prune removes any branch or leaf nodes listed in *Nodes* and all their descendants from the tree *T1*, and returns the modified tree *T2*. The parent nodes are connected to the 'brothers' as required. Nodes in the tree are labeled as [1:numLeaves] for the leaves and as [numLeaves+1:numLeaves+numBranches] for the branches. *Nodes* can also be a logical array of size [numLeaves+numBranches x 1] indicating the nodes to be removed.

T2 = prune(*T1*, *Nodes*, 'Mode','Exclusive') changes the Mode property for pruning to 'Exclusive' and removes only the descendants of the nodes listed in the vector *Nodes*. Nodes that do not have a predecessor become leaves in the list *Nodes*. In this case, pruning is the process of reducing a tree by turning some branch nodes into leaf nodes, and removing the leaf nodes under the original branch.

Examples

Load a phylogenetic tree created from a protein family

```
tr = phytreeread('pf00002.tree');
view(tr)
```
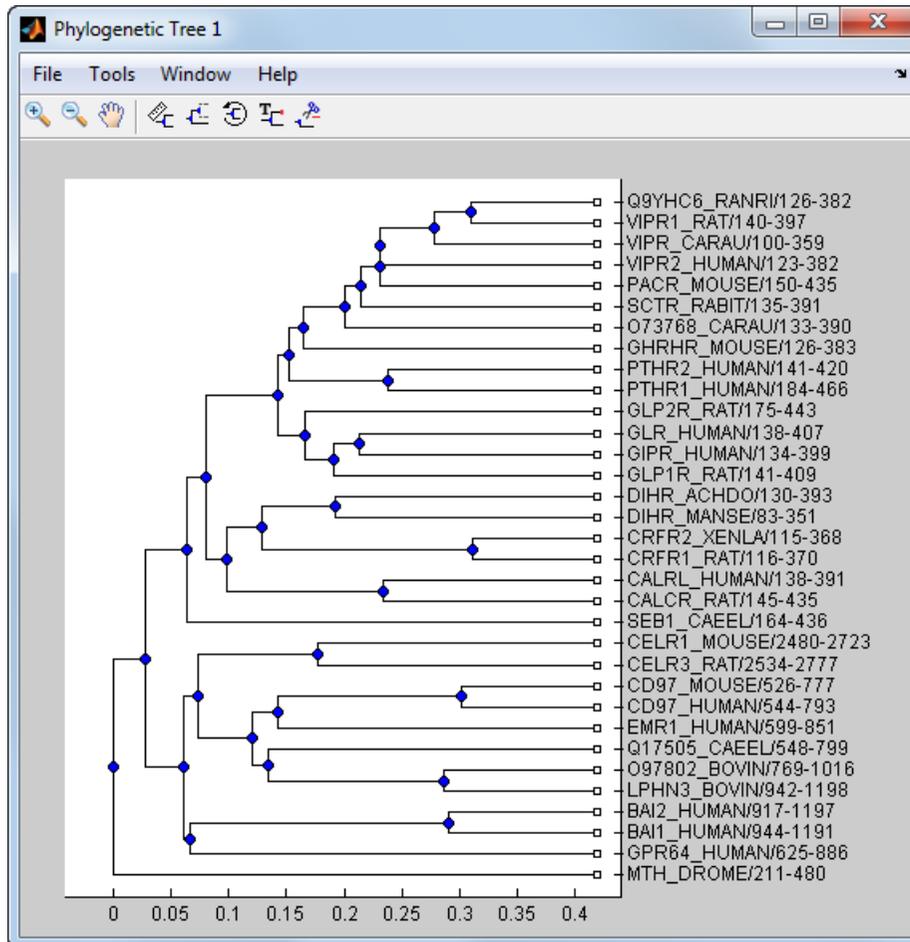

Remove all the 'mouse' proteins

```
ind = getbyname(tr,'mouse');
```

```
tr = prune(tr,ind);
view(tr)
```
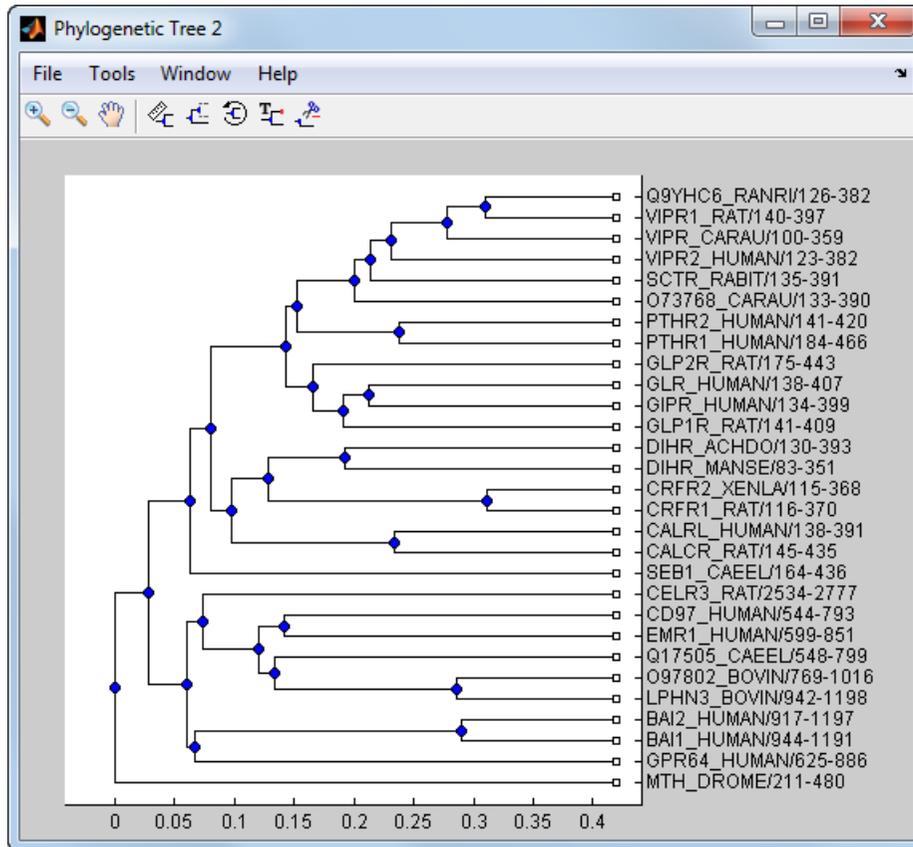

Remove potential outliers in the tree

```
[sel,sel_leaves] = select(tr,'criteria','distance',...
                             'threshold',.3,...
                             'reference','leaves',...
                             'exclude','leaves',...
                             'propagate','toleaves');
tr = prune(tr,~sel_leaves)
view(tr)
```
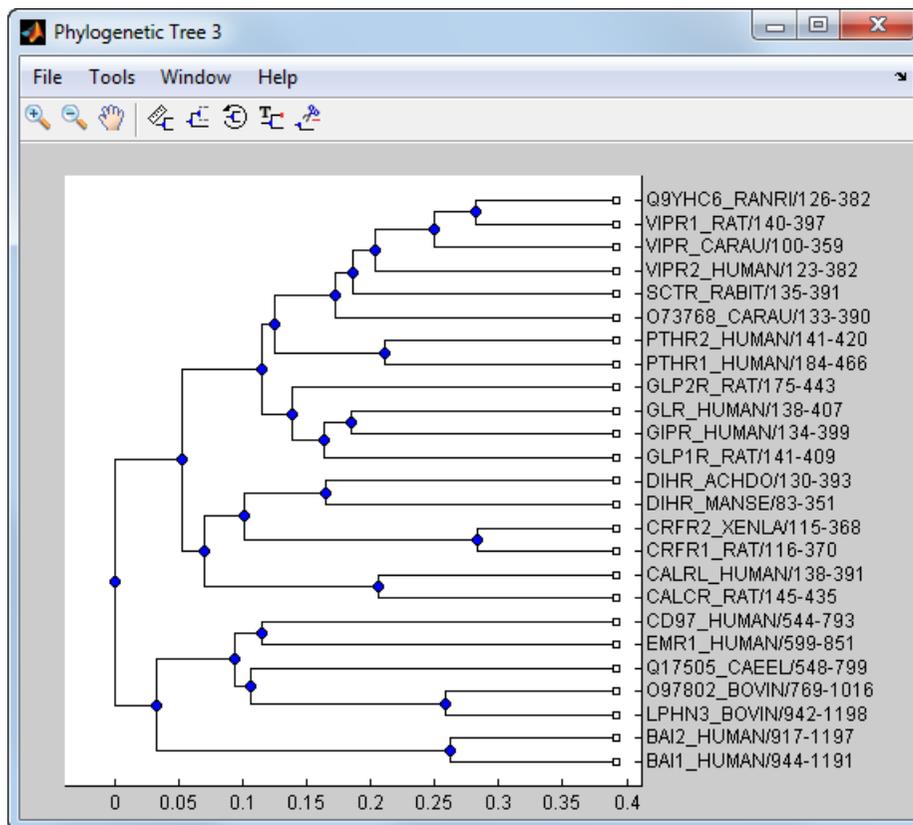
More About

- phytree object

See Also

phytree | get | phytreeviewer | select

pubMedID

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set PubMed IDs in ExpressionSet object

Syntax

PMIDs = pubMedID(*ESObj*)
NewESObj = pubMedID(*ESObj*, *NewPMIDs*)

Description

PMIDs = pubMedID(*ESObj*) returns a string or cell array of strings containing the PubMed IDs from a MIAME object in an ExpressionSet object.

NewESObj = pubMedID(*ESObj*, *NewPMIDs*) replaces the PubMed IDs in the MIAME object in *ESObj*, an ExpressionSet object, with *NewPMIDs*, a string or cell array of strings specifying new PubMed IDs, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the bioma.ExpressionSet class.

Default:

NewPMIDs

String or cell array of strings containing new PubMed IDs.

Default:

Output Arguments

PMIDs

String or cell array of strings containing the PubMed IDs from a MIAME object in an ExpressionSet object.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing the PubMed IDs.

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1- section of the `bioma.ExpressionSet` class reference page. Retrieve the PubMed identifiers stored in the MIAME object stored in the ExpressionSet object:

```
% Retrieve PubMed IDs from the MIAME object
PMIDs = pubMedID(ESObj)
```

See Also
bioma.ExpressionSet | bioma.data.MIAME

How To
· "Managing Gene Expression Data in Objects"

Related Links
· http://www.ncbi.nlm.nih.gov/pubmed/

quantilenorm

Quantile normalization over multiple arrays

Syntax

```
NormData = quantilenorm(Data)
NormData = quantilenorm(...,'MEDIAN', true)
NormData = quantilenorm(...,'DISPLAY', true)
```

Description

NormData = quantilenorm(*Data*), where the columns of *Data* correspond to separate chips, normalizes the distributions of the values in each column.

Note: If *Data* contains NaN values, then *NormData* will also contain NaN values at the corresponding positions.

NormData = quantilenorm(...,'MEDIAN', true) takes the median of the ranked values instead of the mean.

NormData = quantilenorm(...,'DISPLAY', true) plots the distributions of the columns and of the normalized data.

Examples

```
load yeastdata
normYeastValues = quantilenorm(yeastvalues,'display',1);
```

See Also

affygcrma | affyrma | malowess | manorm | rmabackadj | rmasummary

ramachandran

Draw Ramachandran plot for Protein Data Bank (PDB) data

Syntax

```
ramachandran(PDBid)
ramachandran(File)
ramachandran(PDBStruct)
RamaStruct = ramachandran(...)
ramachandran(..., 'Chain', ChainValue, ...)
ramachandran(..., 'Plot', PlotValue, ...)
ramachandran(..., 'Model', ModelValue, ...)
ramachandran(..., 'Glycine', GlycineValue, ...)
ramachandran(..., 'Regions', RegionsValue, ...)
ramachandran(..., 'RegionDef', RegionDefValue, ...)
```

Input Arguments

PDBid	String specifying a unique identifier for a protein structure record in the PDB database.
	Note: Each structure in the PDB database is represented by a four-character alphanumeric identifier. For example, 4hhb is the identifier for hemoglobin.
File	String specifying a file name or a path and file name. The referenced file is a Protein Data Bank (PDB)-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Directory.
PDBStruct	MATLAB structure containing PDB-formatted data, such as returned by getpdb or pdbread.
ChainValue	String or cell array of strings that specifies the chain(s) to compute the torsion angles for and plot. Choices are:

	• `'All'` (default) — Torsion angles for all chains are computed and plotted. • A string specifying the chain ID, which is case sensitive. • A cell array of strings specifying chain IDs, which are case sensitive.
PlotValue	String specifying how to plot chains. Choices are: • `'None'` — Plots nothing. • `'Separate'` — Plots torsion angles for all specified chains in separate plots. • `'Combined'` (default) — Plots torsion angles for all specified chains in one combined plot.
ModelValue	Integer that specifies the structure model to consider. Default is 1.
GlycineValue	Controls the highlighting of glycine residues with a circle in the plot. Choices are `true` or `false` (default).
RegionsValue	Controls the drawing of Ramachandran reference regions in the plot. Choices are `true` or `false` (default). The default regions are core right-handed alpha, core beta, core left-handed alpha, and allowed, with the core regions corresponding to data points of preferred values of psi/phi angle pairs, and the allowed regions corresponding to possible, but disfavored values of psi/phi angle pairs, based on simple energy considerations. The boundaries of these default regions are based on the calculations by Morris et al., 1992. **Note:** If using the default colormap, red = right-handed core alpha, core beta, and core left-handed alpha, while yellow = allowed.

RegionDefValue	MATLAB structure or array of structures (if specifying multiple regions) containing information (name, color, and boundaries) for custom reference regions in a Ramachandran plot. Each structure must contain the following fields: • Name — String specifying a name for the region. • Color — String or three-element numeric vector of RGB values specifying a color for the region in the plot. • Patch — A 2-by-N matrix of values, the first row containing torsion angle phi (Φ) values, and the second row containing torsion angle psi (Ψ) values. When psi/phi angle pairs are plotted, the data points specify boundaries for the region. N is the number of data points needed to define the region. **Tip** If you specify custom reference regions in which a smaller region is contained or covered by a larger region, list the structure for the smaller region first in the array so that it is plotted last and visible in the plot.

Output Arguments

RamaStruct	MATLAB structure or array of structures (if protein contains multiple chains). Each structure contains the following fields: • Angles • ResidueNum • ResidueName • Chain • HPoints For descriptions of the fields, see the following table.

Description

A Ramachandran plot is a plot of the torsion angle phi, Φ, (torsion angle between the C-N-CA-C atoms) versus the torsion angle psi, Ψ, (torsion angle between the N-CA-C-N atoms) for each residue of a protein sequence.

ramachandran(*PDBid*) generates the Ramachandran plot for the protein specified by the PDB database identifier *PDBid*.

ramachandran(*File*) generates the Ramachandran plot for the protein specified by *File*, a PDB-formatted file.

ramachandran(*PDBStruct*) generates the Ramachandran plot for the protein stored in *PDBStruct*, a MATLAB structure containing PDB-formatted data, such as returned by getpdb or pdbread.

RamaStruct = ramachandran(...) returns a MATLAB structure or array of structures (if protein contains multiple chains). Each structure contains the following fields.

Field	Description
Angles	Three-column matrix containing the torsion angles phi (Φ), psi (Ψ), and omega (ω) for each residue in the sequence, ordered by residue sequence number. The number of rows in the matrix is equal to the number of rows in the ResidueNum column vector, which can be used to determine which residue corresponds to each row in the Angles matrix.
	Note: The Angles matrix contains a row for each number in the range of residue sequence numbers, including residue sequence numbers missing from the PDB file. Rows corresponding to residue sequence numbers missing from the PDB file contain the value NaN.
ResidueNum	Column vector containing the residue sequence numbers from the PDB file.
	Note: The ResidueNum vector starts with one of the following:

Field	Description
	• The lowest residue sequence number (if the lowest residue sequence number is negative or zero) • The number 1 (if the lowest residue sequence number is positive) The `ResidueNum` vector ends with the highest residue sequence number and includes all numbers in the range, including residue sequence numbers missing from the PDB file. The angles listed in the `Angles` matrix are in the same order as the residue sequence numbers in the `ResidueNum` vector. Therefore, you can use the `ResidueNum` vector to determine which residue corresponds to each row in the `Angles` matrix.
`ResidueName`	Column vector containing the residue names for the protein.
`Chain`	A string specifying the chains in the protein.
`HPoints`	Handle to the data points in the plot.

`ramachandran(..., 'PropertyName', PropertyValue, ...)` calls `ramachandran` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`ramachandran(..., 'Chain', ChainValue, ...)` specifies the chain(s) to compute the torsion angles for and plot. Choices are:

- `'All'` (default) — Torsion angles for all chains are computed and plotted.
- A string specifying the chain ID, which is case sensitive.
- A cell array of strings specifying chain IDs, which are case sensitive.

`ramachandran(..., 'Plot', PlotValue, ...)` specifies how to plot chains. Choices are:

- `'None'` — Plots nothing.
- `'Separate'` — Plots torsion angles for all specified chains in separate plots.
- `'Combined'` (default) — Plots torsion angles for all specified chains in one combined plot.

`ramachandran(..., 'Model', `*`ModelValue`*`, ...)` specifies the structure model to consider. Default is `1`.

`ramachandran(..., 'Glycine', `*`GlycineValue`*`, ...)` controls the highlighting of glycine residues with a circle in the plot. Choices are `true` or `false` (default).

`ramachandran(..., 'Regions', `*`RegionsValue`*`, ...)` controls the drawing of Ramachandran reference regions in the plot. Choices are `true` or `false` (default).

The default regions are core right-handed alpha, core beta, core left-handed alpha, and allowed, with the core regions corresponding to data points of preferred values of psi/phi angle pairs, and the allowed regions corresponding to possible, but disfavored values of psi/phi angle pairs, based on simple energy considerations. The boundaries of these default regions are based on the calculations by Morris et al., 1992.

Note: If using the default colormap, then red = core right-handed alpha, core beta, and core left-handed alpha, while yellow = allowed.

`ramachandran(..., 'RegionDef', `*`RegionDefValue`*`, ...)` specifies information (name, color, and boundary) for custom reference regions in a Ramachandran plot. *`RegionDefValue`* is a MATLAB structure or array of structures containing the following fields:

- `Name` — String specifying a name for the region.
- `Color` — String or three-element numeric vector of RGB values specifying a color for the region in the plot.
- `Patch` — A 2-by-N matrix of values, the first row containing torsion angle phi (Φ) values, and the second row containing torsion angle psi (Ψ) values. When psi/phi angle pairs are plotted, the data points specify a boundary for the region. N is the number of data points needed to define the region.

Tip If you specify custom reference regions in which a smaller region is contained or covered by a larger region, list the structure for the smaller region first in the array so that it is plotted last and visible in the plot.

Examples

Drawing a Ramachandran Plot

Draw the Ramachandran plot for the human serum albumin complexed with octadecanoic acid, which has a PDB database identifier of `1E7I`.

```
ramachandran('1E7I')
```
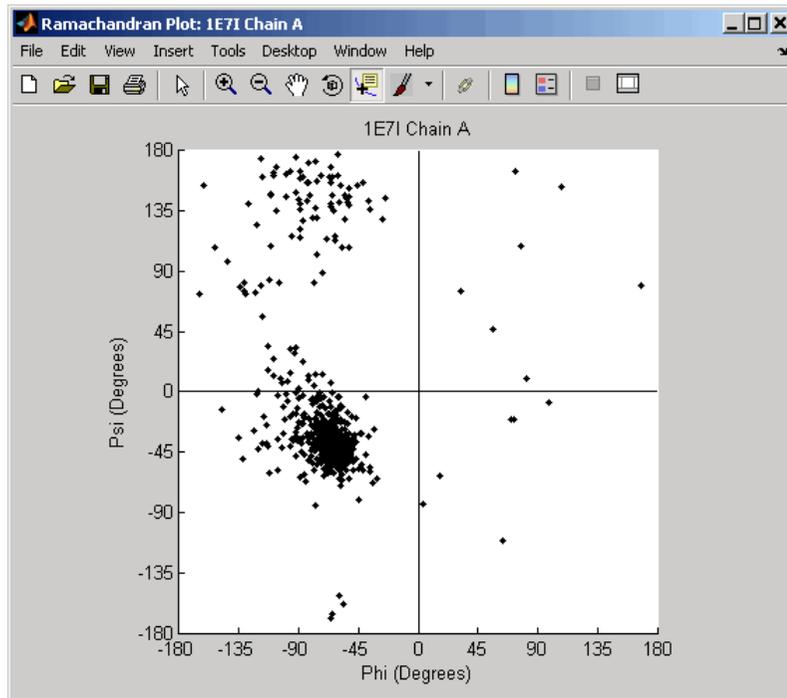

Drawing a Ramachandran Plot for a Specific Chain

1   Use the `getpdb` function to retrieve protein structure data for the human growth hormone from the PDB database, and save the information to a file.

```
getpdb('1a22','ToFile','1a22.pdb');
```

2   Compute the torsion angles and draw the Ramachandran plot for chain A of the human growth hormone, represented in the pdb file, `1a22.pdb`.

```
ChainA1a22Struct = ramachandran('1a22.pdb','chain','A')
```

```
ChainA1a22Struct =

         Angles: [191x3 double]
     ResidueNum: [191x1 double]
    ResidueName: {191x1 cell}
          Chain: 'A'
         HPoints: 370.0012
```
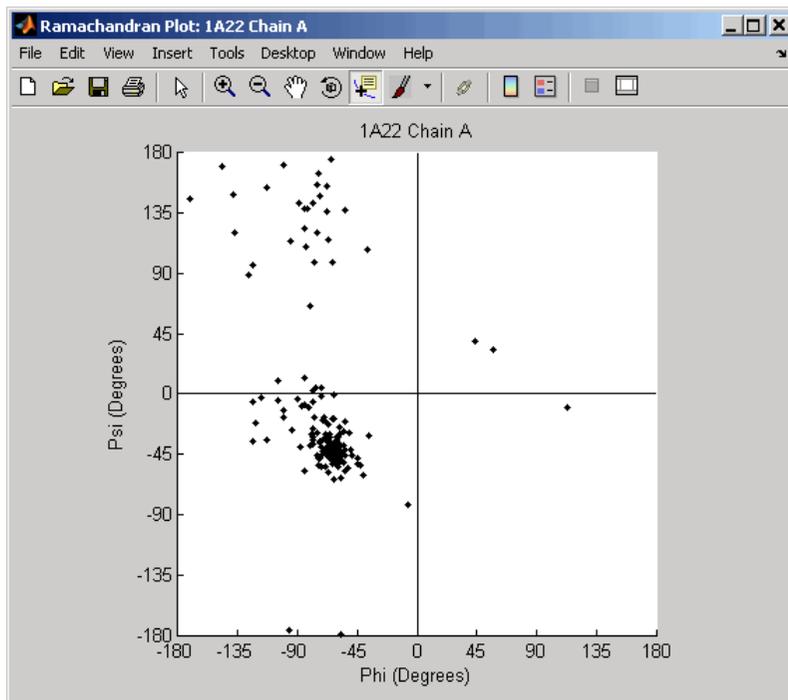

Drawing Ramachandran Plots with Highlighted Glycine Residues and Ramachandran Regions

1 Use the `getpdb` function to retrieve protein structure data for the human growth hormone from the PDB database, and store the information in a structure.

```
Struct1a22 = getpdb('1a22');
```

2 Draw a combined Ramachandran plot for all chains of the human growth hormone, represented in the pdb structure, `1a22Struct`. Highlight the glycine residues (with a circle), and draw the reference Ramachandran regions in the plot.

```
ramachandran(Struct1a22,'glycine',true,'regions',true);
```
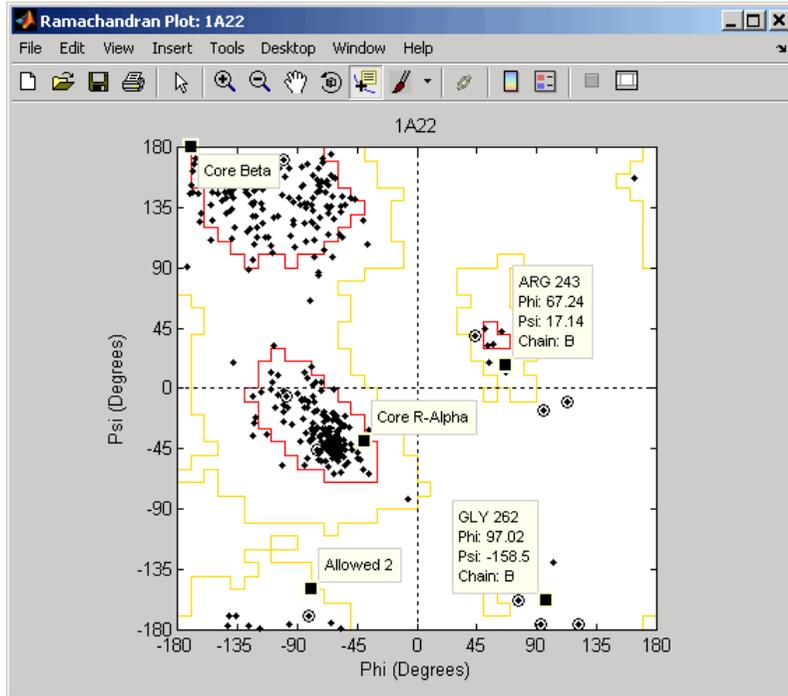
Tip Click a data point to display a data tip with information about the residue. Click a region to display a data tip defining the region. Press and hold the **Alt** key to display multiple data tips.

---

3 Draw a separate Ramachandran plot for each chain of the human growth hormone, represented in the pdb structure, `1a22Struct`. Highlight the glycine residues (with a circle) and draw the reference Ramachandran regions in the plot.

```
ramachandran(Struct1a22,'plot','separate','chain','all',...
              'glycine',true,'regions',true)
```
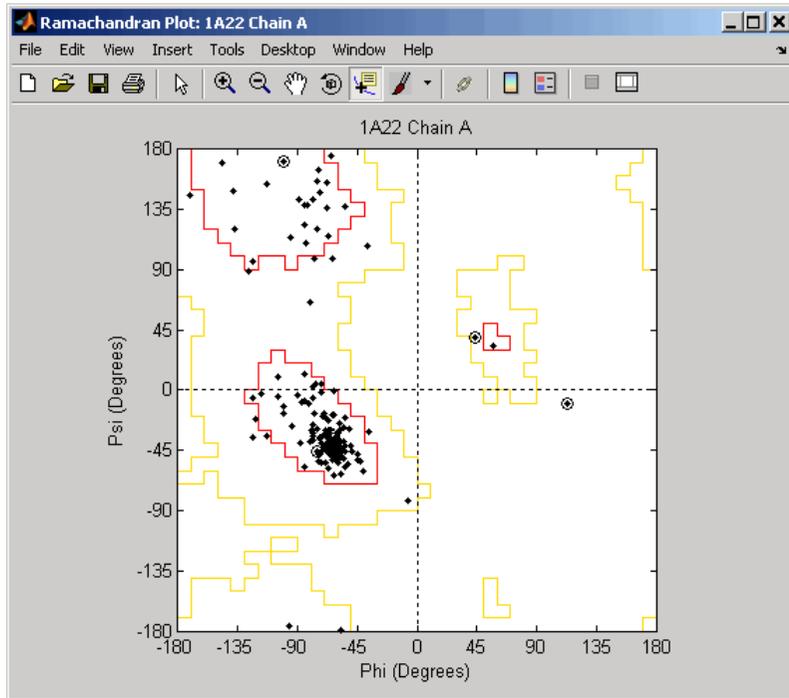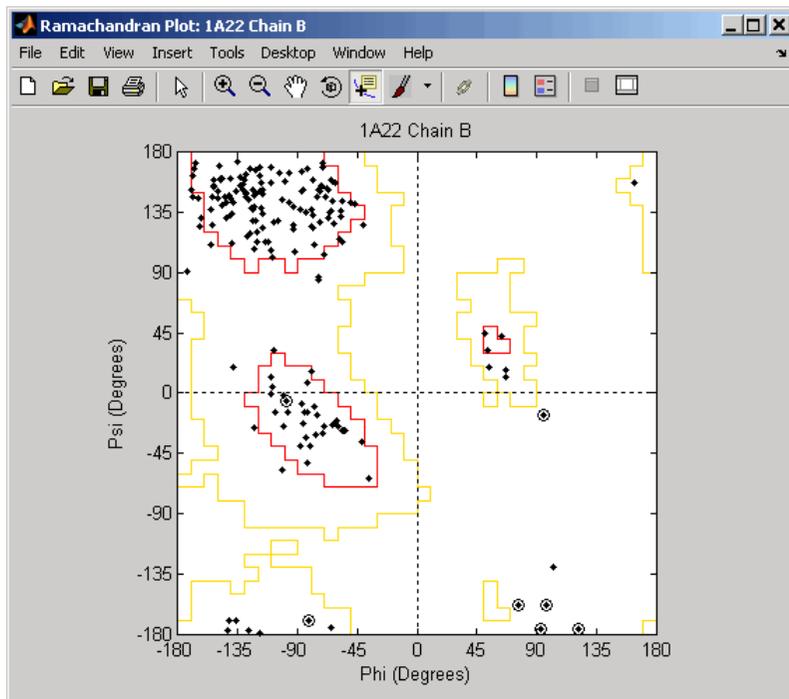
Writing a Tab-Delimited Report File from a Ramachandran Structure

1   Create an array of two structures containing torsion angles for chains A and D in the Calcium/Calmodulin-dependent protein kinase, which has a PDB database identifier of 1hkx.

```
a = ramachandran('1hkx', 'chain', {'A', 'D'})

a =

1x2 struct array with fields:
    Angles
    ResidueNum
    ResidueName
    Chain
    HPoints
```

2 Write a tab-delimited report file containing torsion angles phi (Φ) and psi (Ψ) for chains A and D in the Calcium/Calmodulin-dependent protein kinase.

```
fid = fopen('rama_1hkx_report.txt', 'wt');

for c = 1:numel(a)
    for i = 1:length(a(c).Angles)
        if ~all(isnan(a(c).Angles(i,:)))
            fprintf(fid,'%s\t%d\t%s\t%f\t%f\n', a(c).Chain, ...
                a(c).ResidueNum(i), a(c).ResidueName{i}, ...
                a(c).Angles(i,1:2));
        end
    end
end

fclose(fid);
```

3 View the file you created in the MATLAB Editor.

```
edit rama_1hkx_report.txt
```

```
C:\Work\rama_1hkx_report.txt

1   A   336 THR -107.589510 23.018406
2   A   337 THR -55.719152  -67.612222
3   A   338 ILE -65.153695  163.796933
4   A   339 GLU 53.092557   74.553413
5   A   340 ASP 156.881039  -38.848811
6   A   341 GLU -76.392611  -19.197177
7   A   342 ASP -93.845157  -25.807018

plain text file                 Ln 53   Col 35   OVR
```
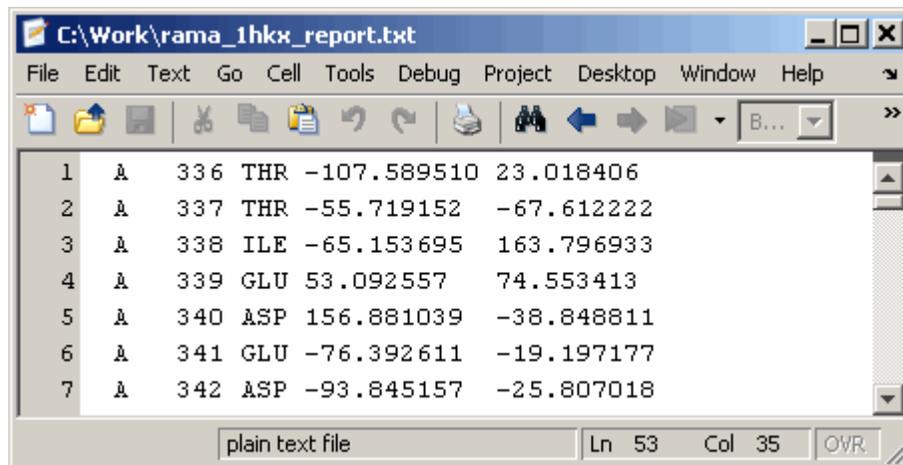
References

[1] Morris, A.L., MacArthur, M.W., Hutchinson, E.G., and Thornton, J.M. (1992). Stereochemical Quality of Protein Structure Coordinates. PROTEINS: Structure, Function, and Genetics *12*, 345–364.

See Also
getpdb | molviewer | pdbdistplot | pdbread | proteinpropplot

randfeatures

Generate randomized subset of features

Syntax

```
[IDX, Z] = randfeatures(X, Group, 'PropertyName', PropertyValue...)
randfeatures(..., 'Classifier', C)
randfeatures(..., 'ClassOptions', CO)
randfeatures(..., 'PerformanceThreshold', PT)
randfeatures(..., 'ConfidenceThreshold', CT)
randfeatures(..., 'SubsetSize', SS)
randfeatures(..., 'PoolSize', PS)
randfeatures(..., 'NumberOfIndices', N)
randfeatures(..., 'CrossNorm', CN)
randfeatures(..., 'Verbose', VerboseValue)
```

Description

`[IDX, Z] = randfeatures(X, Group, 'PropertyName', PropertyValue...)` performs a randomized subset feature search reinforced by classification. `randfeatures` randomly generates subsets of features used to classify the samples. Every subset is evaluated with the apparent error. Only the best subsets are kept, and they are joined into a single final pool. The cardinality for every feature in the pool gives the measurement of the significance.

`X` contains the training samples. Every column of `X` is an observed vector. `Group` contains the class labels. `Group` can be a numeric vector or a cell array of strings; `numel(Group)` must be the same as the number of columns in `X`, and `numel(unique(Group))` must be greater than or equal to `2`. `Z` is the classification significance for every feature. `IDX` contains the indices after sorting `Z`; i.e., the first one points to the most significant feature.

`randfeatures(..., 'Classifier', C)` sets the classifier. Options are

```
'da'    (default)   Discriminant analysis
'knn'               K nearest neighbors
```

randfeatures(..., 'ClassOptions', CO) is a cell with extra options for the selected classifier. When you specify the discriminant analysis model ('da') as a classifier, randfeatures uses the classify function with its default parameters. For the KNN classifier, randfeatures uses fitcknn with the following default options. {'Distance','correlation','NumNeighbors',5}.

randfeatures(..., 'PerformanceThreshold', PT) sets the correct classification threshold used to pick the subsets included in the final pool. For the 'da' model, the default is 0.8. For the 'knn' model, the default is 0.7.

randfeatures(..., 'ConfidenceThreshold', CT) uses the posterior probability of the discriminant analysis to invalidate classified subvectors with low confidence. When using the 'da' model, the default is 0.95.^(number of classes). When using the 'knn' model, the default is 1, meaning any classified subvector must have all k neighbors classified to the same class in order to be kept in the pool.

randfeatures(..., 'SubsetSize', SS) sets the number of features considered in every subset. Default is 20.

randfeatures(..., 'PoolSize', PS) sets the targeted number of accepted subsets for the final pool. Default is 1000.

randfeatures(..., 'NumberOfIndices', N) sets the number of output indices in IDX. Default is the same as the number of features.

randfeatures(..., 'CrossNorm', CN) applies independent normalization across the observations for every feature. Cross-normalization ensures comparability among different features, although it is not always necessary because the selected classifier properties might already account for this. Options are

```
'none' (default)   Intensities are not cross-normalized.
'meanvar'          x_new = (x - mean(x))/std(x)
'softmax'          x_new = (1+exp((mean(x)-x)/std(x)))^-1
'minmax'           x_new = (x - min(x))/(max(x)-min(x))
```

randfeatures(..., 'Verbose', VerboseValue), when Verbose is true, turns off verbosity. Default is true.

Examples

Find a reduced set of genes that is sufficient for classification of all the cancer types in the t-matrix NCI60 data set. Load sample data.

```
load NCI60tmatrix
```

Select features.

```
I = randfeatures(X,GROUP,'SubsetSize',15,'Classifier','da');
```

Test features with a linear discriminant classifier.

```
C = classify(X(I(1:25),:)',X(I(1:25),:)',GROUP);
cp = classperf(GROUP,C);
cp.CorrectRate

ans =

    1
```

References

[1] Li, L., Umbach, D.M., Terry, P., and Taylor, J.A. (2003). Application of the GA/KNN method to SELDI proteomics data. PNAS. 20, 1638-1640.

[2] Liu, H., Motoda, H. (1998). Feature Selection for Knowledge Discovery and Data Mining, Kluwer Academic Publishers.

[3] Ross, D.T. et.al. (2000). Systematic Variation in Gene Expression Patterns in Human Cancer Cell Lines. Nature Genetics. 24 (3), 227-235.

See Also
classperf | crossvalind | rankfeatures | classify | sequentialfs | svmclassify

randseq

Generate random sequence from finite alphabet

Syntax

Seq = randseq(*SeqLength*)
Seq = randseq(*SeqLength*, ...'Alphabet', *AlphabetValue*, ...)
Seq = randseq(*SeqLength*, ...'Weights', *WeightsValue*, ...)
Seq =
randseq(*SeqLength*, ...'FromStructure', *FromStructureValue*, ...)
Seq = randseq(*SeqLength*, ...'Case', *CaseValue*, ...)
Seq = randseq(*SeqLength*, ...'DataType', *DataTypeValue, ...*)

Arguments

SeqLength	Integer that specifies the number of nucleotides or amino acids in the random sequence .
AlphabetValue	String that specifies the alphabet for the sequence. Choices are 'dna'(default), 'rna', or 'amino'.
WeightsValue	Property to specify a weighted random sequence.
FromStructureValue	Property to specify a weighted random sequence using output structures from the functions from basecount, dimercount, codoncount, or aacount.
CaseValue	String that specifies the case of letters in a sequence when Alphabet is 'char'. Choices are 'upper' (default) or 'lower'.
DataTypeValue	String that specifies the data type for a sequence. Choices are 'char'(default) for letter sequences, and 'uint8' or 'double' for numeric sequences. Creates a sequence as an array of *DataType*.

Description

Seq = randseq(*SeqLength*) creates a random sequence with a length specified by *SeqLength*.

Seq = randseq(*SeqLength*, ...'*PropertyName*', *PropertyValue*, ...) calls randseq with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Seq = randseq(*SeqLength*, ...'Alphabet', *AlphabetValue*, ...) generates a sequence from a specific alphabet.

Seq = randseq(*SeqLength*, ...'Weights', *WeightsValue*, ...) creates a weighted random sequence where the ith letter of the sequence alphabet is selected with weight W(i). The weight vector is usually a probability vector or a frequency count vector. Note that the ith element of the nucleotide alphabet is given by int2nt(i), and the ith element of the amino acid alphabet is given by int2aa(i).

Seq = randseq(*SeqLength*, ...'FromStructure', *FromStructureValue*, ...) creates a weighted random sequence with weights given by the output structure from basecount, dimercount, codoncount, or aacount.

Seq = randseq(*SeqLength*, ...'Case', *CaseValue*, ...) specifies the case for a letter sequence.

Seq = randseq(*SeqLength*, ...'DataType', *DataTypeValue, ...*) specifies the data type for the sequence array.

Examples

Generate a random DNA sequence.

```
randseq(20)

ans =
TAGCTGGCCAAGCGAGCTTG
```

Generate a random RNA sequence.

```
randseq(20,'alphabet','rna')
```

```
ans =
GCUGCGGCGGUUGUAUCCUG
```

Generate a random protein sequence.

```
randseq(20,'alphabet','amino')
```

```
ans =
DYKMCLYEFGMFGHFTGHKK
```

See Also
hmmgenerate | rand | randsample | randperm

rankfeatures

Rank key features by class separability criteria

Syntax

```
[IDX, Z] = rankfeatures(X, Group)
[IDX, Z] = rankfeatures(X, Group, ...'Criterion',
CriterionValue, ...)
[IDX, Z] = rankfeatures(X, Group, ...'CCWeighting', ALPHA, ...)
[IDX, Z] = rankfeatures(X, Group, ...'NWeighting', BETA, ...)
[IDX, Z] = rankfeatures(X, Group, ...'NumberOfIndices', N, ...)
[IDX, Z] = rankfeatures(X, Group, ...'CrossNorm', CN, ...)
```

Description

[*IDX*, *Z*] = rankfeatures(*X*, *Group*) ranks the features in *X* using an independent evaluation criterion for binary classification. *X* is a matrix where every column is an observed vector and the number of rows corresponds to the original number of features. *Group* contains the class labels.

IDX is the list of indices to the rows in *X* with the most significant features. *Z* is the absolute value of the criterion used (see below).

Group can be a numeric vector or a cell array of strings; numel(Group) is the same as the number of columns in *X*, and numel(unique(Group)) is equal to 2.

[*IDX*, *Z*] = rankfeatures(*X*, *Group*, ...'*PropertyName*', *PropertyValue*, ...) calls rankfeatures with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

[*IDX*, *Z*] = rankfeatures(*X*, *Group*, ...'Criterion', *CriterionValue*, ...) sets the criterion used to assess the significance of every feature for separating two labeled groups. Choices are:

- `'ttest'` (default) — Absolute value two-sample t-test with pooled variance estimate.
- `'entropy'` — Relative entropy, also known as Kullback-Leibler distance or divergence.
- `'bhattacharyya'` — Minimum attainable classification error or Chernoff bound.
- `'roc'` — Area between the empirical receiver operating characteristic (ROC) curve and the random classifier slope.
- `'wilcoxon'` — Absolute value of the standardized u-statistic of a two-sample unpaired Wilcoxon test, also known as Mann-Whitney.

Note: `'ttest'`, `'entropy'`, and `'bhattacharyya'` assume normal distributed classes while `'roc'` and `'wilcoxon'` are nonparametric tests. All tests are feature independent.

[*IDX*, *Z*] = rankfeatures(*X*, *Group*, ...'CCWeighting', *ALPHA*, ...) uses correlation information to outweigh the *Z* value of potential features using *Z* * (1-*ALPHA**(RHO)), where RHO is the average of the absolute values of the cross-correlation coefficient between the candidate feature and all previously selected features. *ALPHA* sets the weighting factor. It is a scalar value between 0 and 1. When *ALPHA* is 0 (default) potential features are not weighted. A large value of RHO (close to 1) outweighs the significance statistic; this means that features that are highly correlated with the features already picked are less likely to be included in the output list.

[*IDX*, *Z*] = rankfeatures(*X*, *Group*, ...'NWeighting', *BETA*, ...) uses regional information to outweigh the *Z* value of potential features using *Z* * (1-exp(-(DIST/*BETA*).^2)), where DIST is the distance (in rows) between the candidate feature and previously selected features. *BETA* sets the weighting factor. It is greater than or equal to 0. When BETA is 0 (default) potential features are not weighted. A small DIST (close to 0) outweighs the significance statistics of only close features. This means that features that are close to already picked features are less likely to be included in the output list. This option is useful for extracting features from time series with temporal correlation.

BETA can also be a function of the feature location, specified using @ or an anonymous function. In both cases rankfeatures passes the row position of the feature to BETA() and expects back a value greater than or equal to 0.

Note: You can use `'CCWeighting'` and `'NWeighting'` together.

[*IDX*, *Z*] = rankfeatures(*X*, *Group*, ...'NumberOfIndices', *N*, ...) sets the number of output indices in *IDX*. Default is the same as the number of features when *ALPHA* and *BETA* are 0, or 20 otherwise.

[*IDX*, *Z*] = rankfeatures(*X*, *Group*, ...'CrossNorm', *CN*, ...) applies independent normalization across the observations for every feature. Cross-normalization ensures comparability among different features, although it is not always necessary because the selected criterion might already account for this. Choices are:

- 'none' (default) — Intensities are not cross-normalized.
- 'meanvar' — x_new = (x - mean(x))/std(x)
- 'softmax' — x_new = (1+exp((mean(x)-x)/std(x)))^-1
- 'minmax' — x_new = (x - min(x))/(max(x)-min(x))

Examples

Find a reduced set of genes to differentiate breast cancer cells

Find a reduced set of genes that is sufficient for differentiating breast cancer cells from all other types of cancer in the t-matrix NCI60 data set. Load sample data.

```
load NCI60tmatrix
```

Get a logical index vector to the breast cancer cells.

```
BC = GROUP == 8;
```

Select features.

```
I = rankfeatures(X,BC,'NumberOfIndices',12);
```

Test features with a linear discriminant classifier.

```
C = classify(X(I,:)',X(I,:)',double(BC));
cp = classperf(BC,C);
cp.CorrectRate

ans =

    1
```

Use cross-correlation weighting to further reduce the required number of genes.

```
I = rankfeatures(X,BC,'CCWeighting',0.7,'NumberOfIndices',8);
C = classify(X(I,:)',X(I,:)',double(BC));
cp = classperf(BC,C);
cp.CorrectRate

ans =

     1
```

Find the discriminant peaks of two groups of signals with Gaussian pulses modulated by two different sources.

```
load GaussianPulses
f = rankfeatures(y',grp,'NWeighting',@(x) x/10+5,'NumberOfIndices',5);
plot(t,y(grp==1,:),'b',t,y(grp==2,:),'g',t(f),1.35,'vr')
```
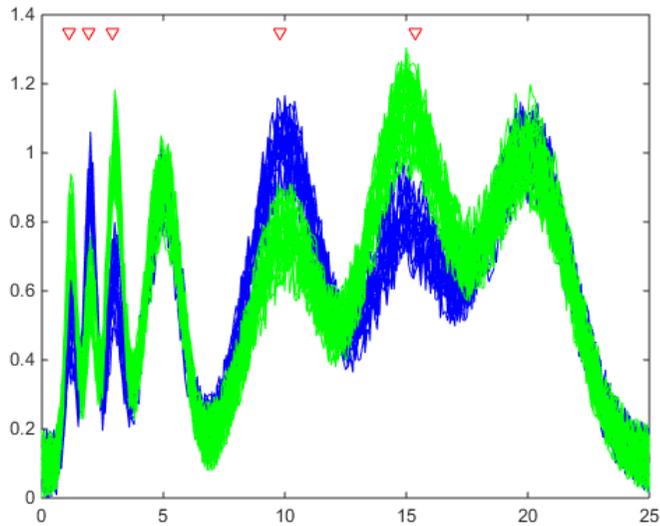

References

[1] Theodoridis, S., and Koutroumbas, K. (1999). Pattern Recognition, Academic Press, 341-342.

[2] Liu, H., Motoda, H. (1998). Feature Selection for Knowledge Discovery and Data Mining, Kluwer Academic Publishers.

[3] Ross, D.T. et.al. (2000). Systematic Variation in Gene Expression Patterns in Human Cancer Cell Lines. Nature Genetics. 24 (3), 227-235.

See Also

classperf | crossvalind | randfeatures | classify | sequentialfs | svmclassify

rdivide (DataMatrix)

Right array divide DataMatrix objects

Syntax

```
DMObjNew = rdivide(DMObj1, DMObj2)
DMObjNew = DMObj1 ./ DMObj2
DMObjNew = rdivide(DMObj1, B)
DMObjNew = DMObj1 ./ B
DMObjNew = rdivide(B, DMObj1)
DMObjNew = B ./ DMObj1
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

DMObjNew	DataMatrix object created by right array division.

Description

DMObjNew = rdivide(*DMObj1, DMObj2*) or the equivalent *DMObjNew* = *DMObj1* ./ *DMObj2* performs an element-by-element right array division of the DataMatrix objects *DMObj1* and *DMObj2* and places the results in *DMObjNew*, another DataMatrix object. In other words, rdivide divides each element in *DMObj1* by the corresponding element in *DMObj2*. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = `rdivide(`*DMObj1*`, `*B*`)` or the equivalent *DMObjNew* = *DMObj1* `./`
B performs an element-by-element right array division of the DataMatrix object
DMObj1 and *B*, a numeric or logical array, and places the results in *DMObjNew*, another
DataMatrix object. In other words, `rdivide` divides each element in *DMObj1* by the
corresponding element in *B*. *DMObj1* and *B* must have the same size (number of rows and
columns), unless *B* is a scalar. The size (number of rows and columns), row names, and
column names for *DMObjNew* are the same as *DMObj1*.

DMObjNew = `rdivide(`*B*`, `*DMObj1*`)` or the equivalent *DMObjNew* = *B* `./` *DMObj1*
performs an element-by-element right array division of *B*, a numeric or logical array, and
the DataMatrix object *DMObj1*, and places the results in *DMObjNew*, another DataMatrix
object. In other words, `rdivide` divides each element in *B* by the corresponding element
in *DMObj1*. *DMObj1* and *B* must have the same size (number of rows and columns), unless
B is a scalar. The size (number of rows and columns), row names, and column names for
DMObjNew are the same as *DMObj1*.

Note: Arithmetic operations between a scalar DataMatrix object and a nonscalar array
are not supported.

MATLAB calls *DMObjNew* = `rdivide(`*X*`, `*Y*`)` for the syntax *DMObjNew* = *X* `./` *Y*
when *X* or *Y* is a DataMatrix object.

More About

· DataMatrix object

See Also
`DataMatrix` | `times` | `ldivide`

read

Class: BioIndexedFile

Read one or more entries from source file associated with BioIndexedFile object

Syntax

```
Output = read(BioIFobj, Indices)
Output = read(BioIFobj, Key)
```

Description

Output = read(*BioIFobj*, *Indices*) reads the entries specified by *Indices* from the source file associated with *BioIFobj*, a BioIndexedFile object. *Indices* is a vector of positive integers specifying indices to entries in the source file. The read method reads and parses the entries using the function specified by the Interpreter property of the BioIndexedFile object. A one-to-one relationship exists between the number and order of elements in *Indices* and *Output*, even if *Indices* has repeated entries. *Output* is a structure or an array of structures containing the parsed data returned by the interpreter function.

Output = read(*BioIFobj*, *Key*) reads the entries specified by *Key* from the source file associated with *BioIFobj*, a BioIndexedFile object. *Key* is a string or cell array of strings specifying one or more keys to entries in the source file. The read method reads and parses the entries using the function specified by the Interpreter property of the BioIndexedFile object. If the keys in the source file are not unique, the read method reads all entries that match a specified key, all at the position of the key in the *Key* cell array. If the keys in the source file are unique, there is a one-to-one relationship between the number and order of elements in *Key* and *Output*.

Tips

Before using the read method, make sure the Interpreter property of the BioIndexedFile object is set appropriately. The Interpreter property is a handle to a function that parses the entries in the source file. The interpreter function must accept

a single string of one or more concatenated entries and return a structure or an array of structures containing the interpreted data.

If the BioIndexedFile object was created from a source file with an application-specific format such as `'SAM'`, `'FASTQ'`, or `'FASTA'`, the default `Interpreter` property is a handle to a function appropriate for that file type and typically does not require you to change it. If the BioIndexedFile object was created from a source file with a `'TABLE'`, `'MRTAB'`, or `'FLAT'` format, then the default `Interpreter` property is `[]`, which means the interpreter is an anonymous function in which the output is equivalent to the input.

For information on setting the `Interpreter` property, see `BioIndexedFile` class.

Input Arguments

BioIFobj

Object of the `BioIndexedFile` class.

Default:

Indices

Vector of positive integers specifying indices to entries in the source file associated with *BioIFobj*, the BioIndexedFile object. The number of elements in *Indices* must be less than or equal to the number of entries in the source file. There is a one-to-one relationship between the number and order of elements in *Indices* and *Output*, even if *Indices* has repeated entries.

Default:

Key

String or cell array of strings specifying one or more keys in the source file.

Default:

Output Arguments

Output

Structure or an array of structures containing the parsed data returned by the interpreter function.

Examples

Construct a BioIndexedFile object to access a table containing cross-references between gene names and gene ontology (GO) terms:

```
% Create variable containing full absolute path of source file
sourcefile = which('yeastgenes.sgd');
% Create a BioIndexedFile object from the source file. Indicate
% the source file is a tab-delimited file where contiguous rows
% with the same key are considered a single entry. Store the
% index file in the Current Folder. Indicate that keys are
% located in column 3 and that header lines are prefaced with !
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
                            'KeyColumn', 3, 'HeaderPrefix','!')
```

Read the GO term from all entries that are associated with the gene YAT2:

```
% Access entries that have the string YAT2 in their keys
YAT2_entries = getEntryByKey(gene2goObj, 'YAT2');
% Adjust the object interpreter to return only the column
% containing the GO term
gene2goObj.Interpreter = @(x) regexp(x,'GO:\d+','match')
% Parse the entries with a key of YAT2 and return all GO terms
% from those entries
GO_YAT2_entries = read(gene2goObj, 'YAT2')

GO_YAT2_entries =

'GO:0004092'  'GO:0005737'  'GO:0006066'  'GO:0006066'  'GO:0009437'
```

See Also
BioIndexedFile.getSubset | BioIndexedFile

How To

- "Work with Large Multi-Entry Text Files"

rebasecuts

Find restriction enzymes that cut nucleotide sequence

Syntax

```
[Enzymes, Sites] = rebasecuts(SeqNT)
rebasecuts(SeqNT, Group)
rebasecuts(SeqNT, [Q, R])
rebasecuts(SeqNT, S)
```

Input Arguments

SeqNT	Nucleotide sequence.
Group	Cell array of strings representing the names of valid restriction enzymes.
Q, *R*	Base positions that limit the search to all sites between base *Q* and base *R*.
S	Base position that limits the search to all sites after base *S*.

Output Arguments

Enzymes	Cell array with the names of restriction enzymes from REBASE®, the Restriction Enzyme Database.
Sites	Vector of cut sites identified with the base position number before every cut.

Description

[*Enzymes*, *Sites*] = rebasecuts(*SeqNT*) finds all the restriction enzymes that cut *SeqNT*, a nucleotide sequence.

rebasecuts(*SeqNT, Group)* limits the search to *Group*, a list of enzymes.

rebasecuts(*SeqNT,* [*Q, R*]) limits the search to those enzymes that cut after the base position specified by *Q* and before the base position specified by *R*.

rebasecuts(*SeqNT, S*) limits the search to those enzymes that cut just after the base position specified by *S*.

REBASE, the Restriction Enzyme Database, is a collection of information about restriction enzymes and related proteins. For more information about REBASE, see:

http://rebase.neb.com/rebase/rebase.html

Examples

1 Create a nucleotide sequence.

```
seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
```

2 Find all possible enzymes and cleavage sites in the sequence.

```
[enzymes, sites] = rebasecuts(seq)
```

3 Find where restriction enzymes CfoI and Tru9I cut the sequence.

```
[enzymes, sites] = rebasecuts(seq, {'CfoI','Tru9I'})

enzymes =

    'CfoI'
    'CfoI'
    'Tru9I'

sites =

    13
    39
    45
```

4 Find all possible enzymes that cut after base 7.

```
enzymes  = rebasecuts(seq, 7)

enzymes =
```

```
        'Csp6I'
        'CviQI'
        'RsaNI'
```

5 Find all possible enzymes that cut between bases 11 and 37.

```
enzymes  = rebasecuts(seq, [11 37])

enzymes =

        'AccII'
        'AspLEI'
        'BmiI'
        'Bsh1236I'
        'BspFNI'
        'BspLI'
        'BstFNI'
        'BstHHI'
        'BstUI'
        'CfoI'
        'FnuDII'
        'GlaI'
        'HhaI'
        'Hin6I'
        'HinP1I'
        'Hpy188I'
        'HspAI'
        'MvnI'
        'NlaIV'
        'PspN4I'
        'SetI'
```

References

[1] Roberts, R.J., Vincze, T., Posfai, J., and Macelis, D. (2007). REBASE—enzymes and genes for DNA restriction and modification. Nucl. Acids Res. *35*, D269–D270.

[2] Official REBASE Web site: http://rebase.neb.com.

See Also
`cleave` | `cleavelookup` | `restrict` | `seq2regexp` | `seqshowwords` | `regexp`

redbluecmap

Create red and blue colormap

Syntax

redbluecmap(*Length*)

Arguments

Length	Positive integer that specifies the length of (or the number of colors in) the colormap. Choices are positive integers ≥ 3 or ≤ 11. Default is 11.

Description

redbluecmap(*Length*) returns a *Length*-by-3 matrix containing a red and blue diverging color palette. Low values are dark blue, values in the center of the map are white, and high values are dark red. *Length* is a positive integer ≥ 3 and ≤ 11, which determines the number of colors in the colormap. Default is 11.

Examples

1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeastvalues, a matrix of gene expression data. Create a clustergram object and display the dendrograms and heat map from the gene expression data in the first 30 rows of the yeastvalues matrix.

```
load filteredyeastdata
cgo = clustergram(yeastvalues(1:30,:))
Clustergram object with 30 rows of nodes and 7 columns of nodes.
```
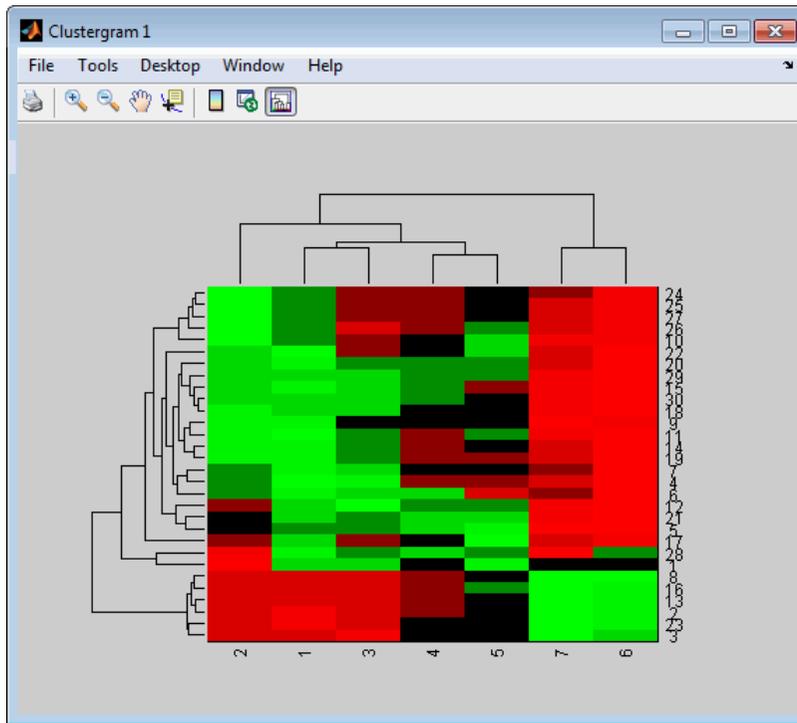
2 Reset the colormap of the heat map to redbluecmap.

```
set(cgo,'Colormap',redbluecmap);
```

References

[1] http://colorbrewer.org
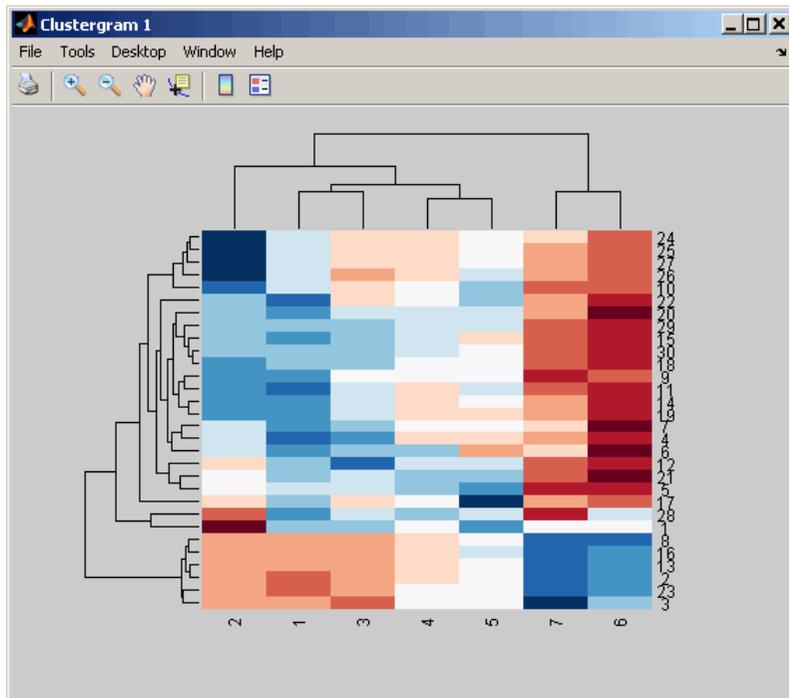
See Also
clustergram | redgreencmap | colormap | colormapeditor

redgreencmap

Create red and green colormap

Syntax

```
redgreencmap(Length)
redgreencmap(Length, 'Interpolation', InterpolationValue)
```

Arguments

Length	Length of the colormap. Enter either 256 or 64. Default is the length of the colormap of the current figure.
InterpolationValue	Property that lets you set the algorithm for color interpolation. Choices are: • `'linear'` • `'quadratic'` • `'cubic'` • `'sigmoid'` (default) **Note:** The sigmoid interpolation is tanh.

Description

redgreencmap(*Length*) returns a *Length*-by-3 matrix containing a red and green colormap. Low values are bright green, values in the center of the map are black, and high values are red. Enter either 256 or 64 for *Length*. If *Length* is empty, the length of the map will be the same as the length of the colormap of the current figure.

redgreencmap(*Length*, 'Interpolation', *InterpolationValue*) lets you set the algorithm for color interpolation. Choices are:

- `'linear'`
- `'quadratic'`

- `'cubic'`
- `'sigmoid'` (default)

Note: The sigmoid interpolation is tanh.

Examples

1 Create a MATLAB structure from the microarray data in a GenePix Results (GPR) file, then display an image of the `'F635 Median'` field.

```
pd = gprread('mouse_a1pd.gpr');
maimage(pd,'F635 Median')
```
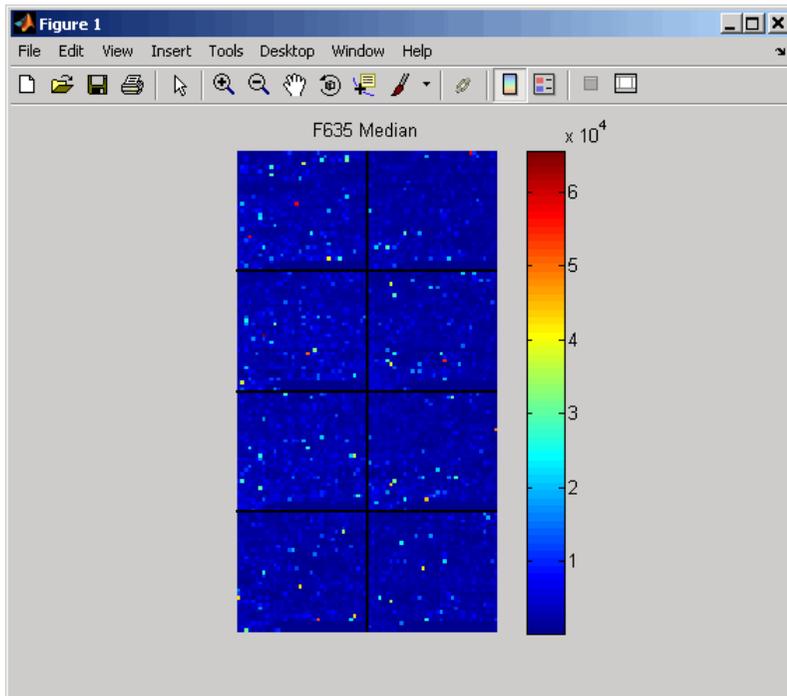

2 Reset the colormap of the current figure.

```
colormap(redgreencmap)
```
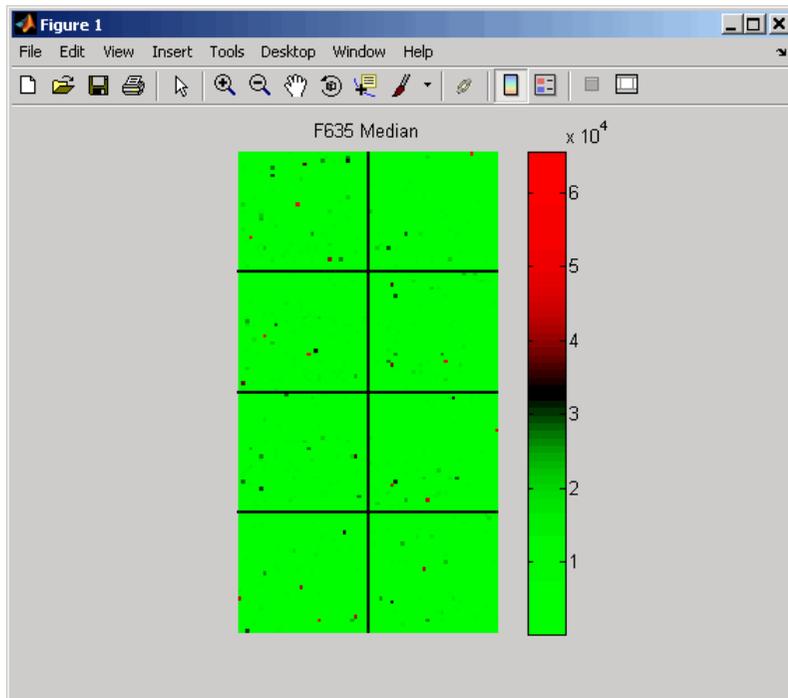
See Also
clustergram | redbluecmap | colormap | colormapeditor

reorder (phytree)

Reorder leaves of phylogenetic tree

Syntax

Tree1Reordered = reorder(*Tree1*, *Order*)
[*Tree1Reordered*, *OptimalOrder*] = reorder(*Tree1*, *Order*,
'Approximate', *ApproximateValue*)
[*Tree1Reordered*, *OptimalOrder*] = reorder(*Tree1*, *Tree2*)

Input Arguments

Tree1, *Tree2*	Phytree objects.
Order	Vector with position indices for each leaf.
ApproximateValue	Controls the use of the optimal leaf-ordering calculation to find the closest order possible to the suggested one without dividing the clades or producing crossing branches. Enter true to use the calculation. Default is false.

Output Arguments

Tree1Reordered	Phytree object with reordered leaves.
OptimalOrder	Vector of position indices for each leaf in *Tree1Reordered*, determined by the optimal leaf-ordering calculation.

Description

Tree1Reordered = reorder(*Tree1*, *Order*) reorders the leaves of the phylogenetic tree *Tree1*, without modifying its structure and distances, creating a new phylogenetic tree, *Tree1Reordered*. *Order* is a vector of position indices for each leaf. If *Order* is

invalid, that is, if it divides the clades (or produces crossing branches), then `reorder` returns an error message.

[*Tree1Reordered*, *OptimalOrder*] = reorder(*Tree1*, *Order*, 'Approximate', *ApproximateValue*) controls the use of the optimal leaf-ordering calculation, which finds the best approximate order closest to the suggested one, without dividing the clades or producing crossing branches. Enter `true` to use the calculation and return *Tree1Reordered*, the reordered tree, and *OptimalOrder*, a vector of position indices for each leaf in *Tree1Reordered*, determined by the optimal leaf-ordering calculation. Default is `false`.

[*Tree1Reordered*, *OptimalOrder*] = reorder(*Tree1*, *Tree2*) uses the optimal leaf-ordering calculation to reorder the leaves in *Tree1* such that it matches the order of leaves in *Tree2* as closely as possible, without dividing the clades or producing crossing branches. *Tree1Reordered* is the reordered tree, and *OptimalOrder* is a vector of position indices for each leaf in *Tree1Reordered*, determined by the optimal leaf-ordering calculation

Examples

Reordering Leaves Using a Valid Order

1   Create and view a phylogenetic tree.

```
b = [1 2; 3 4; 5 6; 7 8; 9 10];
tree = phytree(b)
    Phylogenetic tree object with 6 leaves (5 branches)
view(tree)
```

2   Reorder the leaves on the phylogenetic tree, and then view the reordered tree.

```
 treeReordered = reorder(tree, [5, 6, 3, 4, 1, 2])
 view(treeReordered)
```

Finding Best Approximate Order When Using an Invalid Order

1   Create a phylogenetic tree by reading a Newick-formatted tree file (ASCII text file).

```
tree = phytreeread('pf00002.tree')
    Phylogenetic tree object with 33 leaves (32 branches)
```

2   Create a row vector of the leaf names in alphabetical order.

```
[dummy,order] = sort(get(tree,'LeafNames'));
```

3 Reorder the phylogenetic tree to match as closely as possible the row vector of alphabetically ordered leaf names, without dividing the clades or having crossing branches.

```
treeReordered = reorder(tree,order,'approximate',true)
    Phylogenetic tree object with 33 leaves (32 branches)
```

4 View the original and the reordered phylogenetic trees.

```
view(tree)
view(treeReordered)
```

Reordering Leaves to Match Leaf Order in Another Phylogenetic Tree

1 Create a phylogenetic tree by reading sequence data from a FASTA file, calculating the pairwise distances between sequences, and then using the neighbor-joining method.

```
seqs = fastaread('pf00002.fa')

seqs =

33x1 struct array with fields:
    Header
    Sequence

dist = seqpdist(seqs,'method','jukes-cantor','indels','pair');
NJtree = seqneighjoin(dist,'equivar',seqs)
    Phylogenetic tree object with 33 leaves (32 branches)
```

2 Create another phylogenetic tree from the same sequence data and pairwise distances between sequences, using the single linkage method.

```
HCtree = seqlinkage(dist,'single',seqs)
    Phylogenetic tree object with 33 leaves (32 branches)
```

3 Use the optimal leaf-ordering calculation to reorder the leaves in HCtree such that it matches the order of leaves in NJtree as closely as possible, without dividing the clades or having crossing branches.

```
HCtree_reordered = reorder(HCtree,NJtree)
    Phylogenetic tree object with 33 leaves (32 branches)
```

4 View the reordered phylogenetic tree and the tree used to reorder it.

```
view(HCtree_reordered)
view(NJtree)
```

More About

- phytree object

See Also

phytree | getbyname | get | prune

reroot (phytree)

Change root of phylogenetic tree

Syntax

```
Tree2 = reroot(Tree1)
Tree2 = reroot(Tree1, Node)
Tree2 = reroot(Tree1, Node, Distance)
```

Arguments

Tree1	Phylogenetic tree (`phytree` object) created with the function `phytree`.
Node	Node index returned by the phytree object method `getbyname`.
Distance	Distance from the reference branch.

Description

Tree2 = `reroot(`*Tree1*`)` changes the root of a phylogenetic tree (*Tree1*) using a midpoint method. The midpoint is the location where the mean values of the branch lengths, on either side of the tree, are equalized. The original root is deleted from the tree.

Tree2 = `reroot(`*Tree1*`, `*Node*`)` changes the root of a phylogenetic tree (*Tree1*) to a branch node using the node index (*Node*). The new root is placed at half the distance between the branch node and its parent.

Tree2 = `reroot(`*Tree1*`, `*Node*`, `*Distance*`)` changes the root of a phylogenetic tree (*Tree1*) to a new root at a given distance (*Distance*) from the reference branch node (*Node*) toward the original root of the tree. Note: The new branch representing the root in the new tree (`Tree2`) is labeled `'Root'`.

Examples

1   Create an ultrametric tree.

```
tr_1 = phytree([5 7;8 9;6 11; 1 2;3 4;10 12;...
                14 16; 15 17;13 18])
plot(tr_1,'branchlabels',true)
```

A figure with the phylogenetic tree displays.
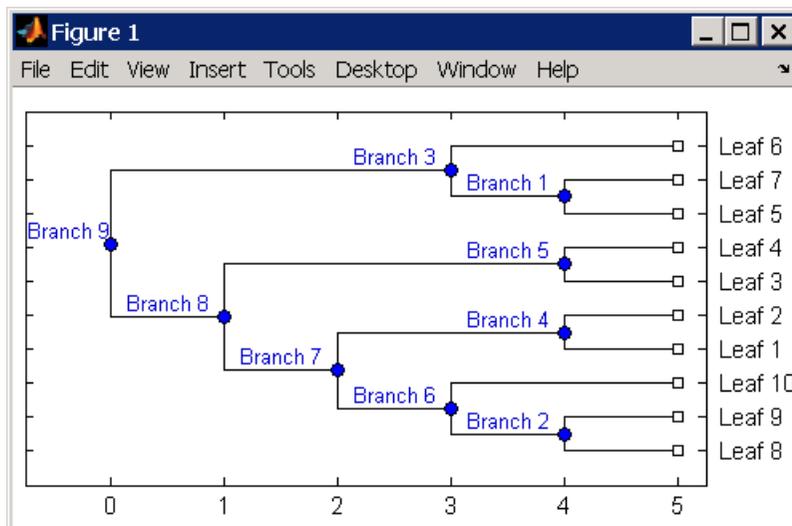
2   Place the root at `'Branch 7'`.

```
sel = getbyname(tr_1,'Branch 7');
tr_2 =  reroot(tr_1,sel)
plot(tr_2,'branchlabels',true)
```

A figure of a phylogenetic tree displays with the root moved to the center of branch 7.
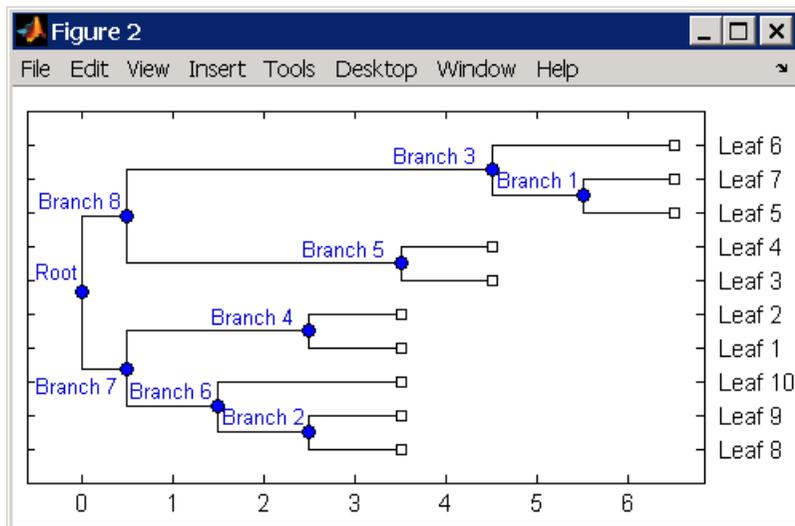
3 Move the root to a branch that makes the tree as ultrametric as possible.

```
tr_3 = reroot(tr_2)
plot(tr_3,'branchlabels',true)
```

A figure of the new tree displays with the root moved from the center of branch 7 to branch 8.
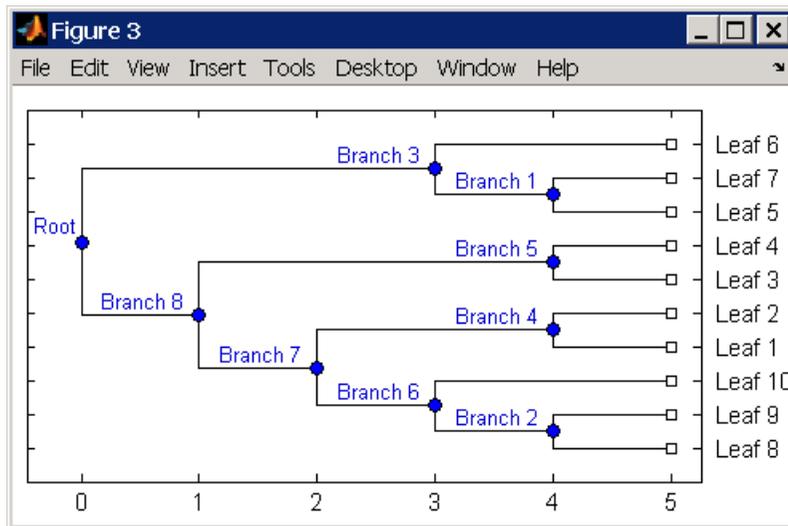
More About

- phytree object

See Also

phytree | getbyname | select | seqneighjoin | get | prune

restrict

Split nucleotide sequence at restriction site

Syntax

```
Fragments = restrict(SeqNT, Enzyme)
Fragments = restrict(SeqNT, NTPattern, Position)
[Fragments, CuttingSites] = restrict(...)
[Fragments, CuttingSites, Lengths] = restrict(...)
... = restrict(..., 'PartialDigest', PartialDigestValue)
```

Arguments

SeqNT	One of the following: • String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers. • Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes. • MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, or `getgenbank`.
Enzyme	String specifying a name of a restriction enzyme from REBASE, the Restriction Enzyme Database. --- **Tip** Some enzymes specify cutting rules for both a strand and its complement strand. `restrict` applies the cutting rule only for the 5' —> 3' strand. For a workaround to applying an enzyme cutting rule for both strands, see Splitting a Double-Stranded Nucleotide Sequence.

NTPattern	Short nucleotide sequence recognition pattern to search for in *SeqNT*, a larger sequence. *NTPattern* can be either of the following: • Character string • Regular expression
Position	Either of the following: • Integer specifying a position in the *SeqNT* to cut, relative to *NTPattern*. • Two-element vector specifying two positions in the *SeqNT* to cut, relative to *NTPattern*. **Note:** Position 0 corresponds to a cut before the first base of *NTPattern*.
PartialDigestValue	Value from 0 to 1 (default) specifying the probability that a cleavage site will be cut.

Description

Fragments = restrict(*SeqNT, Enzyme*) cuts *SeqNT*, a nucleotide sequence, into fragments at the restriction sites of *Enzyme*, a restriction enzyme. The restrict function stores the return values in *Fragments*, a cell array of sequences.

Fragments = restrict(*SeqNT, NTPattern, Position*) cuts *SeqNT*, a nucleotide sequence, into fragments at restriction sites specified by *NTPattern*, a nucleotide recognition pattern, and *Position*.

[*Fragments, CuttingSites*] = restrict(...) returns a numeric vector with the indices representing the cutting sites. The restrict function adds a 0 to the beginning of the *CuttingSites* vector so that the number of elements in *CuttingSites* equals the number of elements in *Fragments*. You can use *CuttingSites* + 1 to point to the first base of every fragment respective to the original sequence.

[*Fragments, CuttingSites, Lengths*] = restrict(...) returns a numeric vector with the lengths of every fragment.

`... = restrict(..., 'PartialDigest', `*`PartialDigestValue`*`)` simulates a partial digest where each restriction site in the sequence has a *PartialDigestValue* or probability of being cut.

REBASE, the Restriction Enzyme Database, is a collection of information about restriction enzymes and related proteins. For more information about REBASE or to search REBASE for the name of a restriction enzyme, see:

`http://rebase.neb.com/rebase/rebase.html`

Examples

Splitting a Nucleotide Sequence by Specifying an Enzyme

1 Enter a nucleotide sequence.

```
Seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
```

2 Use the restriction enzyme HspAI (which specifies a recognition sequence of GCGC and a cleavage position of 1) to cleave the nucleotide sequence.

```
fragmentsEnzyme = restrict(Seq,'HspAI')
```

MATLAB returns:

```
fragmentsEnzyme =

    'AGAGGGGTACG'
    'CGCTCTGAAAAGCGGGAACCTCGTGG'
    'CGCTTTATTAA'
```

Splitting a Nucleotide Sequence by Specifying a Pattern and Position

1 Enter a nucleotide sequence.

```
Seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
```

2 Use the sequence pattern GCGC with the point of cleavage at position 3 to cleave the nucleotide sequence.

```
fragmentsPattern = restrict(Seq,'GCGC',3)
```

MATLAB returns:

```
fragmentsPattern =

    'AGAGGGGTACGCG'
    'CTCTGAAAAGCGGGAACCTCGTGGCG'
    'CTTTATTAA'
```

Splitting a Nucleotide Sequence by Specifying a Regular Expression for the Pattern

1 Enter a nucleotide sequence.

```
Seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
```

2 Use a regular expression to specify the sequence pattern.

```
fragmentsRegExp = restrict(Seq,'GCG[^C]',3)
```

MATLAB returns:

```
fragmentsRegExp =

    'AGAGGGGTACGCGCTCTGAAAAGCG'
    'GGAACCTCGTGGCGCTTTATTAA'
```

Returning the Cutting Sites and Fragment Lengths

1 Enter a nucleotide sequence.

```
Seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
```

2 Capture the cutting sites and fragment lengths as well as the fragments.

```
[fragments, cut_sites, lengths] = restrict(Seq,'HspAI')
```

MATLAB returns:

```
fragments =
    'AGAGGGGTACG'
    'CGCTCTGAAAAGCGGGAACCTCGTGG'
    'CGCTTTATTAA'

cut_sites =
     0
    11
    37

lengths =
```

```
    11
    26
    11
```

Splitting a Double-Stranded Nucleotide Sequence

Some enzymes specify cutting rules for both a strand and its complement strand. `restrict` applies the cutting rule only for the 5' —> 3' strand. You can apply this rule manually for the complement strand.

1 Enter a nucleotide sequence.

```
seq = 'CCCGCNNNNNNN';
```

2 Use the `seqcomplement` function to determine the complement strand, which is in the 3' —> 5' direction.

```
seqc = seqcomplement(seq)
```

MATLAB returns:

```
seqc =

GGGCGNNNNNNN
```

3 Cut the first strand using the restriction enzyme `FauI` (which specifies a recognition sequence pattern of `CCCGC` and a cleavage position of `9`).

```
cuts_strand1 = restrict(seq, 'FauI')
```

MATLAB returns:

```
cuts_strand1 =

    'CCCGCNNNN'
    'NNN'
```

4 Cut the complement strand according the rule specified by `FauI` (which specifies a recognition sequence pattern of `GGGCG` with the point of cleavage at position `11`).

```
cuts_strand2 = restrict(seqc, 'GGGCG', 11)
```

MATLAB returns:

```
cuts_strand2 =

    'GGGCGNNNNN'
```

```
'N'
```

References

[1] Roberts, R.J., Vincze, T., Posfai, J., and Macelis, D. (2007). REBASE—enzymes and genes for DNA restriction and modification. Nucl. Acids Res. *35*, D269–D270.

[2] Official REBASE Web site: http://rebase.neb.com.

See Also

cleave | cleavelookup | rebasecuts | seq2regexp | seqcomplement | seqshowwords | regexp

revgeneticcode

Return reverse mapping (amino acid to nucleotide codon) for genetic code

Syntax

Map = revgeneticcode
Map = revgeneticcode(*GeneticCode*)
Map = revgeneticcode(..., 'Alphabet', *AlphabetValue*, ...)
Map = revgeneticcode(..., 'ThreeLetterCodes',
ThreeLetterCodesValue, ...)

Input Arguments

GeneticCode	Integer or string specifying a genetic code number or code name from the table Genetic Code. Default is 1 or 'Standard'.
	Tip If you use a code name, you can truncate the name to the first two letters of the name.
AlphabetValue	String specifying the nucleotide alphabet to use in the map. Choices are: • 'DNA' (default) — Uses the symbols A, C, G, and T. • 'RNA' — Uses the symbols A, C, G, and U.
ThreeLetterCodesValue	Controls the use of three-letter amino acid codes as field names in the return structure *Map*. Choices are true for three-letter codes or false for one-letter codes. Default is false.

Output Arguments

Map	Structure containing the reverse mapping of amino acids to nucleotide codons for the standard genetic

	code. The *Map* structure contains a field for each amino acid.

Description

Map = `revgeneticcode` returns a structure containing the reverse mapping of amino acids to nucleotide codons for the standard genetic code. The *Map* structure contains a field for each amino acid.

Map = `revgeneticcode`(*GeneticCode*) returns a structure containing the reverse mapping of amino acids to nucleotide codons for the specified genetic code. *GeneticCode* is either:

- An integer or string specifying a code number or code name from the table Genetic Code
- The `transl_table` (code) number from the NCBI Web page describing genetic codes:

  `http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c`

Tip If you use a code name, you can truncate the name to the first two letters of the name.

Map = `revgeneticcode`(..., '*PropertyName*', *PropertyValue*, ...) calls `revgeneticcode` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Map = `revgeneticcode`(..., 'Alphabet', *AlphabetValue*, ...) specifies the nucleotide alphabet to use in the map. *AlphabetValue* can be 'DNA', which uses the symbols A, C, G, and T, or 'RNA', which uses the symbols A, C, G, and U. Default is 'DNA'.

Map = `revgeneticcode`(..., 'ThreeLetterCodes', *ThreeLetterCodesValue*, ...) controls the use of three-letter amino acid codes as field names in the return structure *Map*. *ThreeLetterCodesValue* can be `true` for three-letter codes or `false` for one-letter codes. Default is `false`.

Genetic Code

Code Number	Code Name
1	`Standard`
2	`Vertebrate Mitochondrial`
3	`Yeast Mitochondrial`
4	`Mold`, `Protozoan`, `Coelenterate Mitochondrial`, and `Mycoplasma/Spiroplasma`
5	`Invertebrate Mitochondrial`
6	`Ciliate`, `Dasycladacean`, and `Hexamita Nuclear`
9	`Echinoderm Mitochondrial`
10	`Euplotid Nuclear`
11	`Bacterial` and `Plant Plastid`
12	`Alternative Yeast Nuclear`
13	`Ascidian Mitochondrial`
14	`Flatworm Mitochondrial`
15	`Blepharisma Nuclear`
16	`Chlorophycean Mitochondrial`
21	`Trematode Mitochondrial`
22	`Scenedesmus Obliquus Mitochondrial`
23	`Thraustochytrium Mitochondrial`

Examples

- Return the reverse mapping of amino acids to nucleotide codons for the `Standard` genetic code.

```
map = revgeneticcode

map =

    Name: 'Standard'
       A: {'GCT'  'GCC'  'GCA'  'GCG'}
       R: {'CGT'  'CGC'  'CGA'  'CGG'  'AGA'  'AGG'}
       N: {'AAT'  'AAC'}
```

```
        D: {'GAT'   'GAC'}
        C: {'TGT'   'TGC'}
        Q: {'CAA'   'CAG'}
        E: {'GAA'   'GAG'}
        G: {'GGT'   'GGC'   'GGA'   'GGG'}
        H: {'CAT'   'CAC'}
        I: {'ATT'   'ATC'   'ATA'}
        L: {'TTA'   'TTG'   'CTT'   'CTC'   'CTA'   'CTG'}
        K: {'AAA'   'AAG'}
        M: {'ATG'}
        F: {'TTT'   'TTC'}
        P: {'CCT'   'CCC'   'CCA'   'CCG'}
        S: {'TCT'   'TCC'   'TCA'   'TCG'   'AGT'   'AGC'}
        T: {'ACT'   'ACC'   'ACA'   'ACG'}
        W: {'TGG'}
        Y: {'TAT'   'TAC'}
        V: {'GTT'   'GTC'   'GTA'   'GTG'}
    Stops: {'TAA'   'TAG'   'TGA'}
   Starts: {'TTG'   'CTG'   'ATG'}
```

- Return the reverse mapping of amino acids to nucleotide codons for the Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma genetic code, using the rna alphabet.

```
moldmap = revgeneticcode(4,'Alphabet','rna');
```

- Return the reverse mapping of amino acids to nucleotide codons for the Flatworm Mitochondrial genetic code, using three-letter codes for the field names in the return structure.

```
wormmap = revgeneticcode('Flatworm Mitochondrial',...
                         'ThreeLetterCodes',true);
```

References

[1] NCBI Web page describing genetic codes:

http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c

See Also

aa2nt | aminolookup | baselookup | geneticcode | nt2aa

rmabackadj

Perform background adjustment on Affymetrix microarray probe-level data using Robust Multi-array Average (RMA) procedure

Syntax

```
BackAdjustedMatrix = rmabackadj(PMData)
BackAdjustedMatrix = rmabackadj(..., 'Method', MethodValue, ...)
BackAdjustedMatrix = rmabackadj(..., 'Truncate', TruncateValue, ...)
BackAdjustedMatrix = rmabackadj(..., 'Showplot', ShowplotValue, ...)
```

Input Arguments

PMData	Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)
MethodValue	Specifies the estimation method for the background adjustment model parameters. Enter either `'RMA'` (to use estimation method described by Bolstad, 2005) or `'MLE'` (to estimate the parameters using maximum likelihood). Default is `'RMA'`.
TruncateValue	Specifies the background noise model. Enter either `true` (use a truncated Gaussian distribution) or `false` (use a nontruncated Gaussian distribution). Default is `true`.
ShowplotValue	Controls the plotting of a histogram showing the distribution of PM probe intensity values (blue) and the convoluted probability distribution function (red), with estimated parameters mu, sigma and alpha. Enter either `'all'` (plot a histogram for each column or chip) or specify a subset of columns (chips) by entering the column number, list of numbers, or range of numbers.
	For example:
	• `..., 'Showplot', 3, ...)` plots the intensity values in column 3.

- ..., 'Showplot', [3,5,7], ...) plots the intensity values in columns 3, 5, and 7.

- ..., 'Showplot', 3:9, ...) plots the intensity values in columns 3 to 9.

Output Arguments

BackAdjustedMatrix	Matrix of background-adjusted probe intensity values.

Description

BackAdjustedMatrix = rmabackadj(*PMData*) returns the background adjusted values of probe intensity values in the matrix, *PMData*. Note that each row in *PMData* corresponds to a perfect match (PM) probe and each column in *PMData* corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.) Details on the background adjustment are described by Bolstad, 2005.

BackAdjustedMatrix = rmabackadj(..., '*PropertyName*', *PropertyValue*, ...) calls rmabackadj with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

BackAdjustedMatrix = rmabackadj(..., 'Method', *MethodValue*, ...) specifies the estimation method for the background adjustment model parameters. When *MethodValue* is 'RMA', rmabackadj implements the estimation method described by Bolstad, 2005. When *MethodValue* is 'MLE', rmabackadj estimates the parameters using maximum likelihood. Default is 'RMA'.

BackAdjustedMatrix = rmabackadj(..., 'Truncate', *TruncateValue*, ...) specifies the background noise model used. When *TruncateValue* is false, rmabackadj uses nontruncated Gaussian as the background noise model. Default is true.

BackAdjustedMatrix = rmabackadj(..., 'Showplot', *ShowplotValue*, ...) lets you plot a histogram showing the distribution of PM probe intensity values (blue)

and the convoluted probability distribution function (red), with estimated parameters mu, sigma and alpha. When *ShowplotValue* is `'all'`, rmabackadj plots a histogram for each column or chip. When *ShowplotValue* is a number, list of numbers, or range of numbers, rmabackadj plots a histogram for the indicated column number (chip).

For example:

- (..., `'Showplot'`, `3,...`) plots the intensity values in column 3 of *PMData*.
- (..., `'Showplot'`, `[3,5,7],...`) plots the intensity values in columns 3, 5, and 7 of *PMData*.
- (..., `'Showplot'`, `3:9,...`) plots the intensity values in columns 3 to 9 of *PMData*.
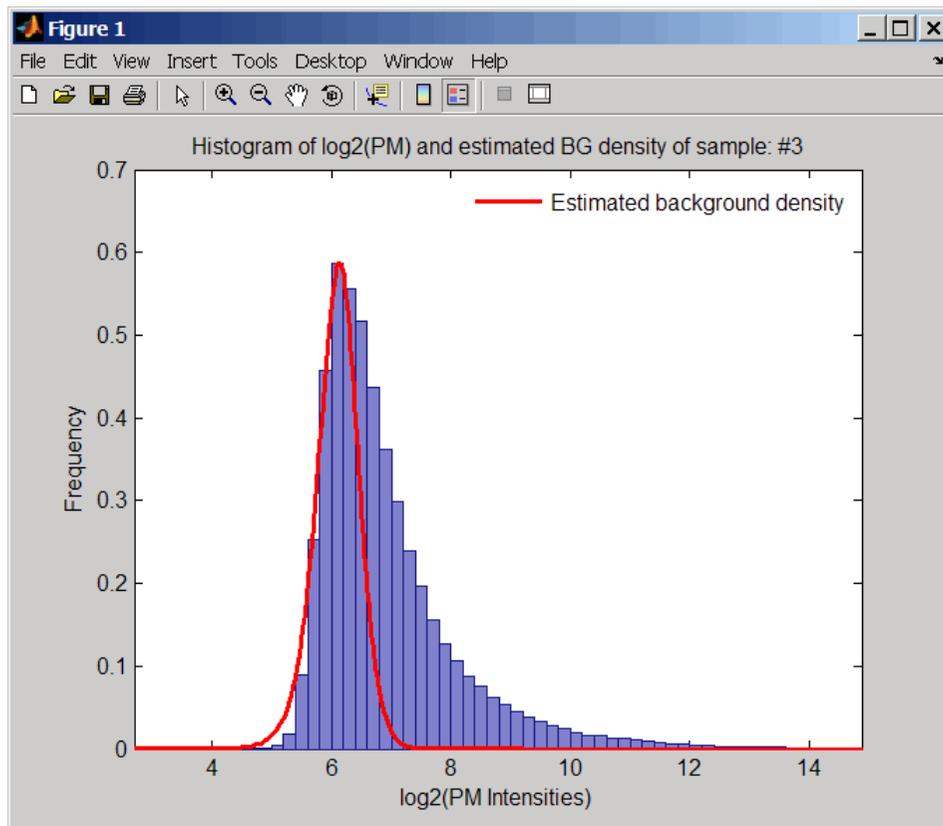
Examples

1   Load a MAT-file, included with the Bioinformatics Toolbox software, which contains Affymetrix probe-level data, including `pmMatrix`, a matrix of PM probe intensity values from multiple CEL files.

    ```
    load prostatecancerrawdata
    ```

2   Perform background adjustment on the PM probe intensity values in the matrix, `pmMatrix`, creating a new matrix, `BackgroundAdjustedMatrix`.

    ```
    BackgroundAdjustedMatrix = rmabackadj(pmMatrix);
    ```

3   Perform background adjustment on the PM probe intensity values in only column 3 of the matrix, `pmMatrix`, creating a new matrix, `BackgroundAdjustedChip3`.

    ```
    BackgroundAdjustedChip3 = rmabackadj(pmMatrix(:,3));
    ```

The `prostatecancerrawdata.mat` file used in the previous example contains data from Best et al., 2005.

References

[1] Irizarry, R.A., Hobbs, B., Collin, F., Beazer-Barclay, Y.D., Antonellis, K.J., Scherf, U., Speed, T.P. (2003). Exploration, Normalization, and Summaries of High Density Oligonucleotide Array Probe Level Data. Biostatistics *4*, 249–264.

[2] Bolstad, B. (2005). "affy: Built-in Processing Methods" http://www.bioconductor.org/ packages/2.1/bioc/vignettes/affy/ inst/doc/builtinMethods.pdf

[3] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

See Also

affyinvarsetnorm | affyread | affyrma | celintensityread | probelibraryinfo | probesetlink | probesetlookup | probesetvalues | quantilenorm | rmasummary

rmasummary

Calculate gene expression values from Affymetrix microarray probe-level data using Robust Multi-array Average (RMA) procedure

Syntax

ExpressionMatrix = rmasummary(*ProbeIndices*, *Data*)
ExpressionMatrix = rmasummary(*ProbeIndices*, *Data*, 'Output', *OutputValue*)

Arguments

ProbeIndices	Column vector of probe indices. The convention for probe indices is, for each probe set, to label each probe 0 to $N-1$, where N is the number of probes in the probe set.
	Tip Use the ProbeIndices field in the structure returned by celintensityread as the *ProbeIndices* input.
Data	Matrix of natural-scale intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)
	Tip Using a single-precision matrix for *Data* decreases memory usage.
	Tip You can use the matrix from the PMIntensities field in the structure returned by celintensityread as the *Data* input. However, first ensure the matrix has been background adjusted, using the rmabackadj or gcrmabackadj function, and normalized, using the quantilenorm function.

OutputValue	Specifies the scale of the returned gene expression values. *OutputValue* can be: • `'log'` • `'log2'` • `'log10'` • `'linear'` • `@functionname` In the last instance, the data is transformed as defined by the function *functionname*. Default is `'log2'`.

Description

ExpressionMatrix = rmasummary(*ProbeIndices*, *Data*) returns gene (probe set) expression values after calculating them from natural-scale probe intensities in the matrix *Data*, using the column vector of probe indices, *ProbeIndices*. Note that each row in *Data* corresponds to a perfect match (PM) probe, and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.) Note that the column vector *ProbeIndices* designates probes within each probe set by labeling each probe 0 to $N-1$, where N is the number of probes in the probe set. Note that each row in *ExpressionMatrix* corresponds to a gene (probe set) and each column in *ExpressionMatrix* corresponds to an Affymetrix CEL file, which represents a single chip.

For a given probe set *n*, with *J* probe pairs, let *Yijn* denote the background-adjusted, base 2 log transformed and quantile-normalized PM probe intensity value of chip *i* and probe *j*. *Yijn* follows a linear additive model:
$$Yijn = Uin + Ajn + Eijn; i = 1, ..., I; j = 1, ..., J; n = 1, ..., N$$

where:
Uin = Gene expression of the probe set *n* on chip *i*
Ajn = Probe affinity effect for the *j*th probe in the probe set
Eijn = Residual for the *j*th probe on the *i*th chip

The RMA method assumes $A1 + A2 + ... + AJ = 0$ for all probe sets. A robust procedure, median polish, estimates *Ui* as the log scale measure of expression.

Note: There is no column in *ExpressionMatrix* that contains probe set or gene information.

ExpressionMatrix = rmasummary(..., '*PropertyName*', *PropertyValue*, ...) calls rmasummary with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

ExpressionMatrix = rmasummary(*ProbeIndices*, *Data*, 'Output', *OutputValue*) specifies the scale of the returned gene expression values. *OutputValue* can be:

- 'log'
- 'log2'
- 'log10'
- 'linear'
- @*functionname*

In the last instance, the data is transformed as defined by the function *functionname*. Default is 'log2'.

Examples

1   Load a MAT-file, included with the Bioinformatics Toolbox software, which contains Affymetrix data variables, including pmMatrix, a matrix of PM probe intensity values from multiple CEL files.

    load prostatecancerrawdata

2   Perform background adjustment on the PM probe intensity values in the matrix, pmMatrix, using the rmabackadj function, thereby creating a new matrix, BackgroundAdjustedMatrix.

    BackgroundAdjustedMatrix = rmabackadj(pmMatrix);

3   Normalize the data in BackgroundAdjustedMatrix, using the quantilenorm function.

    NormMatrix = quantilenorm(BackgroundAdjustedMatrix);

4 Calculate gene expression values from the probe intensities in `NormMatrix`, creating a new matrix, `ExpressionMatrix`. (Use the `probeIndices` column vector provided to supply information on the probe indices.)

```
ExpressionMatrix = rmasummary(probeIndices, NormMatrix);
```

The `prostatecancerrawdata.mat` file used in the previous example contains data from Best et al., 2005.

References

[1] Irizarry, R.A., Hobbs, B., Collin, F., Beazer-Barclay, Y.D., Antonellis, K.J., Scherf, U., Speed, T.P. (2003). Exploration, Normalization, and Summaries of High Density Oligonucleotide Array Probe Level Data. Biostatistics. *4*, 249–264.

[2] Mosteller, F., and Tukey, J. (1977). Data Analysis and Regression (Reading, Massachusetts: Addison-Wesley Publishing Company), pp. 165–202.

[3] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. Clinical Cancer Research *11*, 6823–6834.

See Also

affygcrma | affyinvarsetnorm | affyrma | celintensityread | gcrmabackadj | mainvarsetnorm | malowess | manorm | quantilenorm | rmabackadj

rna2dna

Convert RNA sequence to DNA sequence

Syntax

SeqDNA = rna2dna(*SeqRNA*)

Arguments

SeqRNA	RNA sequence specified by any of the following: • Character string with the characters A, C, G, U, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N, • Row vector of integers from the table Mapping Nucleotide Integers to Letter Codes. • MATLAB structure containing a Sequence field that contains an RNA sequence, such as returned by fastaread, fastqread, emblread, getembl, genbankread, or getgenbank.

Description

SeqDNA = rna2dna(*SeqRNA*) converts an RNA sequence to a DNA sequence by converting any uracil nucleotides (U) in the RNA sequence to thymine nucleotides (T). The DNA sequence is returned in the same format as the RNA sequence. For example, if *SeqRNA* is a vector of integers, then so is *SeqDNA*.

Examples

Convert an RNA sequence to a DNA sequence.

```
rna2dna('ACGAUGAGUCAUGCUU')

ans =
```

ACGATGAGTCATGCTT

See Also
dna2rna | regexp | strrep

rnaconvert

Convert secondary structure of RNA sequence between bracket and matrix notations

Syntax

RNAStruct2 = rnaconvert(*RNAStruct*)

Input Arguments

RNAStruct	Secondary structure of an RNA sequence represented by either:
	• Bracket notation
	• Connectivity matrix
	Tip Use the rnafold function to create *RNAStruct*.

Output Arguments

RNAStruct2	Secondary structure of an RNA sequence represented by either:
	• **Bracket notation** — String of dots and brackets, where each dot represents an unpaired base, while a pair of equally nested, opening and closing brackets represents a base pair.
	• **Connectivity matrix** — Binary, upper-triangular matrix, where *RNAmatrix*(i, j) = 1 if and only if the *i*th residue in the RNA sequence *Seq* is paired with the *j*th residue of *Seq*.

Description

RNAStruct2 = rnaconvert(*RNAStruct*) returns *RNAStruct2*, the secondary structure of an RNA sequence, in matrix notation (if *RNAStruct* is in bracket notation), or in bracket notation (if *RNAStruct* is in matrix notation).

Examples

Converting from Bracket to Matrix Notation

1 Create a string representing a secondary structure of an RNA sequence in bracket notation.

```
Bracket = '(((..((((.......)))).((.....))).))).';
```

2 Convert the secondary structure to a connectivity matrix representation.

```
Matrix = rnaconvert(Bracket);
```

Converting from Matrix to Bracket Notation

1 Create a connectivity matrix representing a secondary structure of an RNA sequence.

```
Matrix2 = zeros(12);
Matrix2(1,12) = 1;
Matrix2(2,11) = 1;
Matrix2(3,10) = 1;
Matrix2(4,9) = 1;
```

2 Convert the secondary structure to bracket notation.

```
Bracket2 = rnaconvert(Matrix2)

Bracket2 =

((((....))))
```

See Also
`rnafold` | `rnaplot`

rnafold

Predict minimum free-energy secondary structure of RNA sequence

Syntax

```
rnafold(Seq)
RNAbracket = rnafold(Seq)
[RNAbracket, Energy] = rnafold(Seq)
[RNAbracket, Energy, RNAmatrix] = rnafold(Seq)
... = rnafold(Seq, ...'MinLoopSize', MinLoopSizeValue, ...)
... = rnafold(Seq, ...'NoGU', NoGUValue, ...)
... = rnafold(Seq, ...'Progress', ProgressValue, ...)
```

Input Arguments

Seq	Either of the following: • String specifying an RNA sequence. • MATLAB structure containing a `Sequence` field that specifies an RNA sequence.
MinLoopSizeValue	Integer specifying the minimum size of the loops (in bases) to be considered when computing the free energy. Default is `3`.
NoGUValue	Controls whether GU or UG pairs are forbidden to form. Choices are `true` or `false` (default).
ProgressValue	Controls the display of a progress bar during the computation of the minimum free-energy secondary structure. Choices are `true` or `false` (default).

Output Arguments

RNAbracket	String of dots and brackets indicating the bracket notation for the minimum-free energy secondary structure of an RNA sequence. In the bracket notation, each dot represents an

	unpaired base, while a pair of equally nested, opening and closing brackets represents a base pair.
Energy	Value specifying the energy (in kcal/mol) of the minimum free-energy secondary structure of an RNA sequence.
RNAmatrix	Connectivity matrix representing the minimum free-energy secondary structure of an RNA sequence. A binary, upper-triangular matrix where *RNAmatrix*(i, j) = 1 if and only if the *i*th residue in the RNA sequence *Seq* is paired with the *j*th residue of *Seq*.

Description

rnafold(*Seq*) predicts and displays the secondary structure (in bracket notation) associated with the minimum free energy for the RNA sequence, *Seq*, using the thermodynamic nearest-neighbor approach.

Note: For long sequences, this prediction can be time consuming. For example, a 600-nucleotide sequence can take several minutes, and sequences greater than 1000 nucleotides can take over 1 hour, depending on your system.

RNAbracket = rnafold(*Seq*) predicts and returns the secondary structure associated with the minimum free energy for the RNA sequence, *Seq*, using the thermodynamic nearest-neighbor approach. The returned structure, *RNAbracket*, is in bracket notation, that is a vector of dots and brackets, where each dot represents an unpaired base, while a pair of equally nested, opening and closing brackets represents a base pair.

[*RNAbracket, Energy*] = rnafold(*Seq*) also returns *Energy*, the energy value (in kcal/mol) of the minimum free-energy secondary structure of the RNA sequence.

[*RNAbracket, Energy, RNAmatrix*] = rnafold(*Seq*) also returns *RNAmatrix*, a connectivity matrix representing the secondary structure associated with the minimum free energy. *RNAmatrix* is an upper triangular matrix where *RNAmatrix*(i, j) = 1 if and only if the *i*th residue in the RNA sequence *Seq* is paired with the *j*th residue of *Seq*.

... = rnafold(*Seq*, ...'*PropertyName*', *PropertyValue*, ...) calls rnafold with optional properties that use property name/property value pairs. You can specify

one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = rnafold(*Seq*, ...'MinLoopSize', *MinLoopSizeValue*, ...) specifies the minimum size of the loops (in bases) to be considered when computing the free energy. Default is 3.

... = rnafold(*Seq*, ...'NoGU', *NoGUValue*, ...) controls whether GU or UG pairs are forbidden to form. Choices are true or false (default).

... = rnafold(*Seq*, ...'Progress', *ProgressValue*, ...) controls the display of a progress bar during the computation of the minimum free-energy secondary structure. Choices are true or false (default).

Examples

Determine the minimum free-energy secondary structure (in both bracket and matrix notation) and the energy value of the following RNA sequence:

```
seq = 'ACCCCCUCCUUCCUUGGAUCAAGGGGCUCAA';
[bracket, energy, matrix] = rnafold(seq);bracket

bracket =

..(((((...((....))...))))).....
```

References

[1] Wuchty, S., Fontana, W., Hofacker, I., and Schuster, P. (1999). Complete suboptimal folding of RNA and the stability of secondary structures. Biopolymers *49*, 145–165.

[2] Matthews, D., Sabina, J., Zuker, M., and Turner, D. (1999). Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. J. Mol. Biol. *288*, 911–940.

See Also
rnaconvert | rnaplot

rnaplot

Draw secondary structure of RNA sequence

Syntax

```
rnaplot(RNA2ndStruct)
ha = rnaplot(RNA2ndStruct)
[ha, H] = rnaplot(RNA2ndStruct)
rnaplot(RNA2ndStruct, ...'Sequence', SequenceValue, ...)
rnaplot(RNA2ndStruct, ...'Format', FormatValue, ...)
rnaplot(RNA2ndStruct, ...'Selection', SelectionValue, ...)
rnaplot(RNA2ndStruct, ...'ColorBy', ColorByValue, ...)
```

Input Arguments

RNA2ndStruct	Secondary structure of an RNA sequence represented by either: • String specifying bracket notation • Connectivity matrix **Tip** Use the `rnafold` function to create *RNA2ndStruct*.
SequenceValue	Sequence of the RNA secondary structure being plotted, specified by either of the following: • String of characters • Structure containing a `Sequence` field that contains an RNA sequence This information is used in the data tip displayed by clicking a base in the plot of the RNA secondary structure *RNA2ndStruct*. This information is required if you specify the `'Diagram'` format or if you specify to highlight any of the following paired selections: `'AU'`, `'UA'`, `'GC'`, `'CG'`, `'GU'` or `'UG'`.
FormatValue	String specifying the format of the plot. Choices are:

	• `'Circle'` (default) • `'Diagram'` • `'Dotdiagram'` • `'Graph'` • `'Mountain'` • `'Tree'` **Note:** If you specify `'Diagram'`, you must also use the `'Sequence'` property to provide the RNA sequence.

SelectionValue	Either of the following:
	
	• Numeric array specifying the indices of residues to highlight in the plot.
	• String specifying the subset of residues to highlight in the plot. Choices are:
	
	  • `'Paired'`
	  • `'Unpaired'`
	  • `'AU'` or `'UA'`
	  • `'GC'` or `'CG'`
	  • `'GU'` or `'UG'`
	
	Note: If you specify `'AU'`, `'UA'`, `'GC'`, `'CG'`, `'GU'`, or `'UG'`, you must also use the `'Sequence'` property to provide the RNA sequence.

ColorByValue	String specifying a color scheme for the plot. Choices are:
	• `'State'` (default) — Color by pair state: paired bases and unpaired bases.
	• `'Residue'` — Color by residue type (A, C, G, and U).
	• `'Pair'` — Color by pair type (AU/UA, GC/CG, and GU/UG).
	Note: If you specify `'residue'` or `'pair'`, you must also use the `'Sequence'` property to provide the RNA sequence.
	Note: Because internal nodes of a tree correspond to paired residues, you cannot specify `'residue'` if you specify `'Tree'` for the `'Format'` property.

Output Arguments

ha	Handle to the figure axis.
H	A structure of handles containing a subset of the following fields, based on what you specify for the `'Selection'` and `'ColorBy'` properties:
	• Paired
	• Unpaired
	• A
	• C
	• G
	• U
	• AU
	• GC
	• GU
	• Selected

Description

rnaplot(*RNA2ndStruct*) draws the RNA secondary structure specified by *RNA2ndStruct*, the secondary structure of an RNA sequence represented by a string specifying bracket notation or a connectivity matrix.

ha = rnaplot(*RNA2ndStruct*) returns *ha*, a handle to the figure axis.

[*ha, H*] = rnaplot(*RNA2ndStruct*) also returns *H*, a structure of handles, which you can use to graph elements in a MATLAB Figure window.

Tip Use the handles returned in *H* to change properties of the graph elements, such as color, marker size, and marker type.

H contains a subset of the following fields, based on what you specify for the 'Selection' and 'ColorBy' properties.

Field	Description
Paired	Handles to all paired residues
Unpaired	Handles to all unpaired residues
A	Handles to all A residues
C	Handles to all C residues
G	Handles to all G residues
U	Handles to all U residues
AU	Handles to all AU or UA pairs
GC	Handles to all GC or CG pairs
GU	Handles to all GU or UG pairs
Selected	Handles to all selected residues

rnaplot(*RNA2ndStruct*, ...'*PropertyName*', *PropertyValue*, ...) calls rnaplot with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

rnaplot(*RNA2ndStruct*, ...'Sequence', *SequenceValue*, ...) draws the RNA secondary structure specified by *RNA2ndStruct*, and annotates it with the sequence positions supplied by *SequenceValue*, the RNA sequence specified by a string of characters or a structure containing a Sequence field.

rnaplot(*RNA2ndStruct*, ...'Format', *FormatValue*, ...) draws the RNA secondary structure specified by *RNA2ndStruct*, using the format specified by *FormatValue*.

FormatValue is a string specifying the format of the plot. Choices are as follows.

Format	Description
'Circle' (default)	Each base is represented by a dot on the circumference of a circle of arbitrary size. Lines connect bases that pair with each other.
'Diagram'	Two-dimensional representation of the RNA secondary structure. Each base is represented and identified by a letter. The backbone and hydrogen bonds between base pairs are represented by lines.
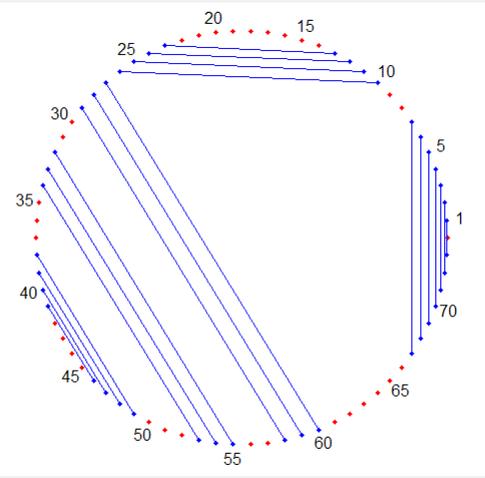
Format	Description
	Note: If you specify `'Diagram'`, you must also use the `'Sequence'` property to provide the RNA sequence.
`'Dotdiagram'`	Two-dimensional representation of the RNA secondary structure. Each base is represented and identified by a dot. The backbone and hydrogen bonds between base pairs are represented by lines.

Format	Description
`'Graph'`	Bases are displayed in their sequence position along the abscissa (*x*-axis) of a graph. Semi-elliptical lines connect bases that pair with each other. The height of the lines is proportional to the distance between paired bases.
`'Mountain'`	Each base is represented by a dot in a two-dimensional plot, where the base position is in the abscissa (*x*-axis) and the number of base pairs enclosing a given base is in the ordinate (*y*-axis).
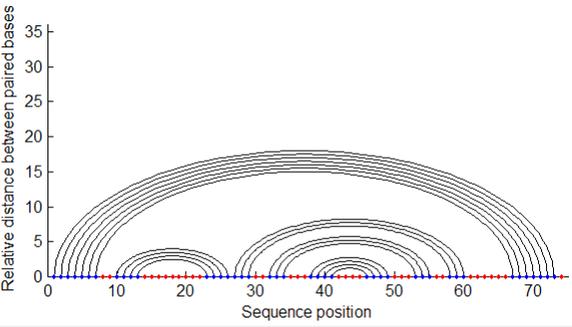
Format	Description
`'Tree'`	Each base is represented by a node in a tree graph. Leaf nodes indicate unpaired bases, while each internal node indicates a base pair. The tree root is a fictitious node, not associated with any base in the secondary structure.
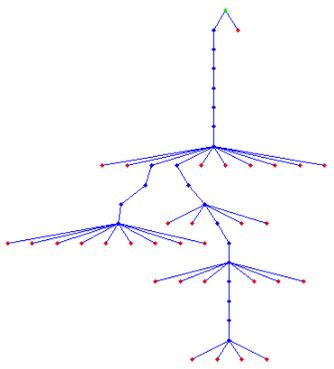
rnaplot(*RNA2ndStruct*, ...'Selection', *SelectionValue*, ...) draws the RNA secondary structure specified by *RNA2ndStruct*, highlighting a subset of residues specified by *SelectionValue*. *SelectionValue* can be either:

- Numeric array specifying the indices of residues to highlight in the plot.
- String specifying the subset of residues to highlight in the plot. Choices are:

  - `'Paired'`
  - `'Unpaired'`
  - `'AU'` or `'UA'`
  - `'GC'` or `'CG'`
  - `'GU'` or `'UG'`

Note: If you specify `'AU'`, `'UA'`, `'GC'`, `'CG'`, `'GU'`, or `'UG'`, you must also use the `'Sequence'` property to provide the RNA sequence.

---

rnaplot(*RNA2ndStruct*, ...'ColorBy', *ColorByValue*, ...) draws the RNA secondary structure specified by *RNA2ndStruct*, using a color scheme specified by *ColorByValue*, a string indicating a color scheme. Choices are:

- 'State' (default) — Color by pair state: paired bases and unpaired bases.

- 'Residue' — Color by residue type (A, C, G, and U).

- 'Pair' — Color by pair type (AU/UA, GC/CG, and GU/UG).

Note: If you specify 'Residue' or 'Pair', you must also use the 'Sequence' property to provide the RNA sequence.

Note: Because internal nodes of a tree correspond to paired residues, you cannot specify 'Residue' if you specify 'Tree' for the 'Format' property.

Examples

1 Determine the minimum free-energy secondary structure of an RNA sequence and plot it in circle format:

```
seq = 'GCGCCCGUAGCUCAAUUGGAUAGAGCGUUUGACUACGGAUCAAAAGGUUAGGGGUUCGACUCCUCUCGGGCGCG';
ss = rnafold(seq);
rnaplot(ss)
```
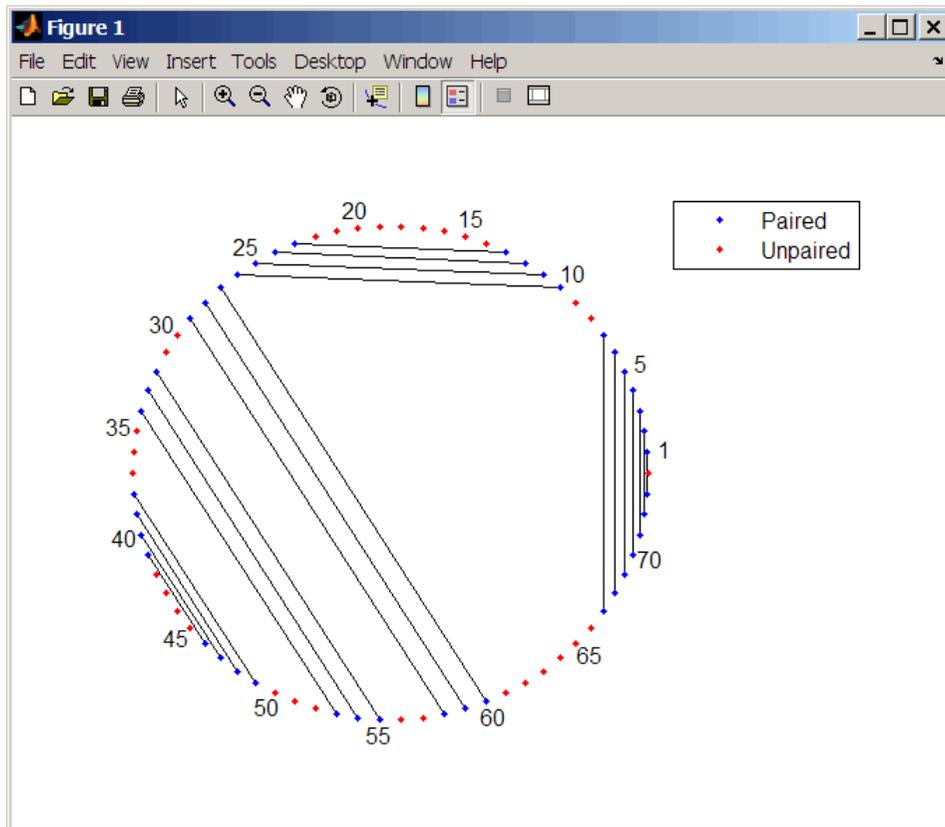
2 Plot the RNA sequence secondary structure in graph format and color it by pair type.

```
rnaplot(ss, 'sequence', seq, 'format', 'graph', 'colorby', 'pair')
```
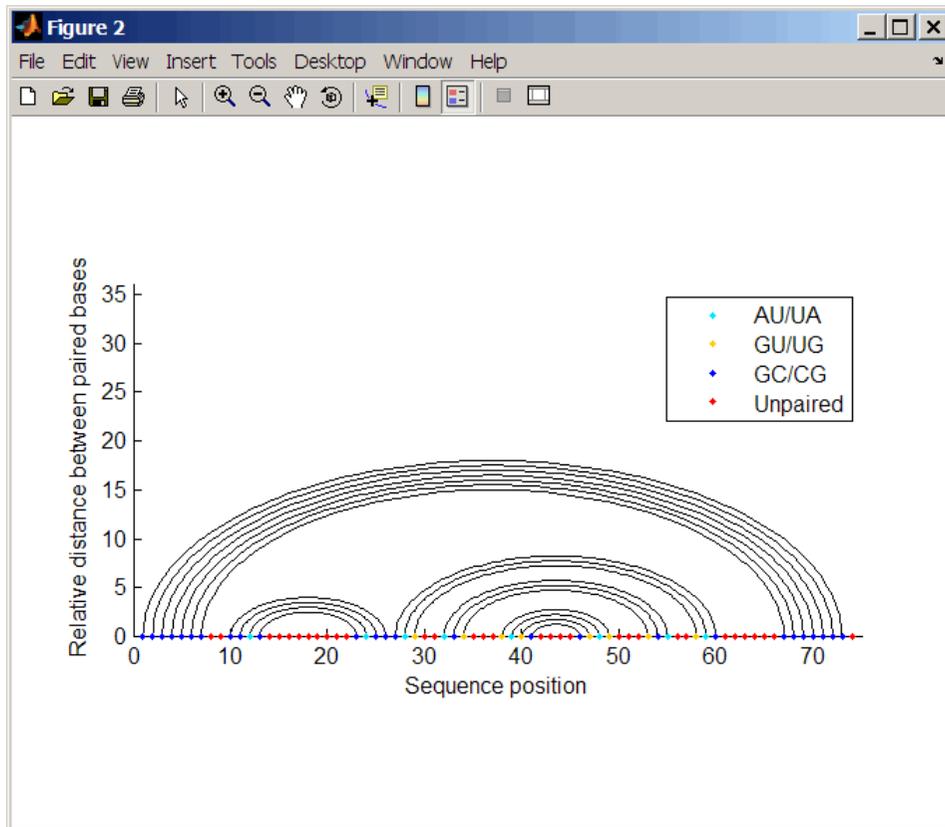
3 Plot the RNA sequence secondary structure in mountain format and color it by residue type. Use the handle to add a title to the plot.

```
ha = rnaplot(ss, 'sequence', seq, 'format', 'mountain',...
             'colorby', 'residue')
title(ha, 'Bacillus halodurans, tRNA Arg')
```
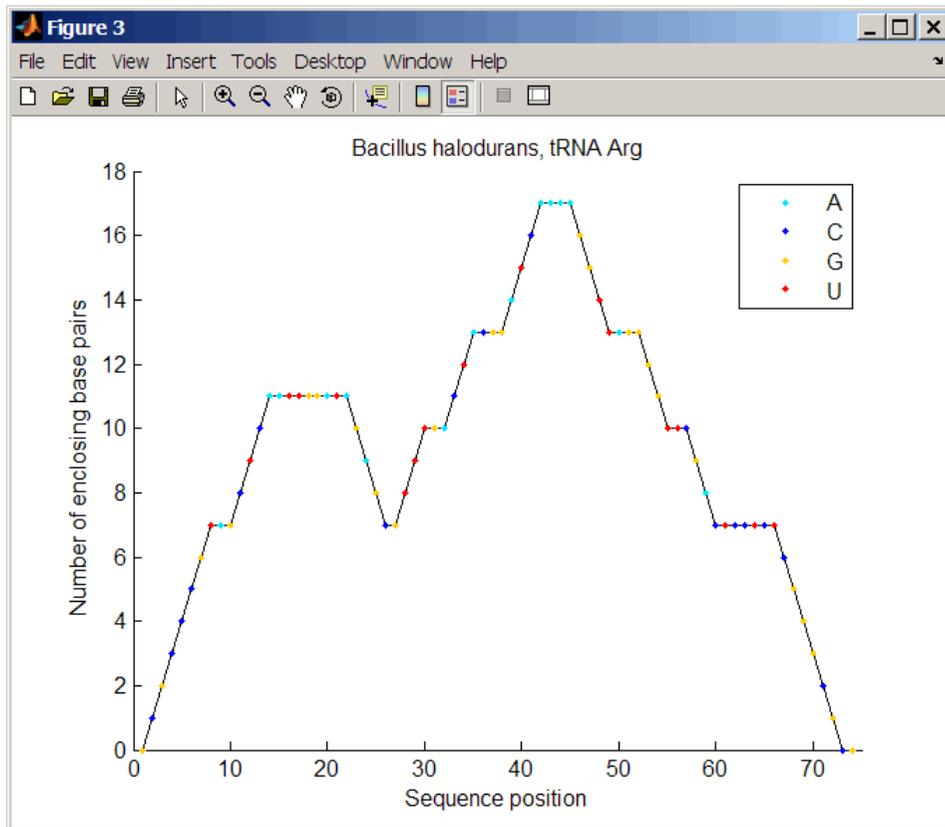
4 Mutate the first six positions in the sequence and observe the effect the change has on the secondary structure by highlighting the first six residues.

```
seqMut = seq;
seqMut(1:6) = 'AAAAAA';
ssMut = rnafold(seqMut);
rnaplot(ss, 'sequence', seq, 'format', 'dotdiagram', 'selection', 1:6);
rnaplot(ssMut, 'sequence', seqMut, 'format', 'dotdiagram', 'selection', 1:6);
```
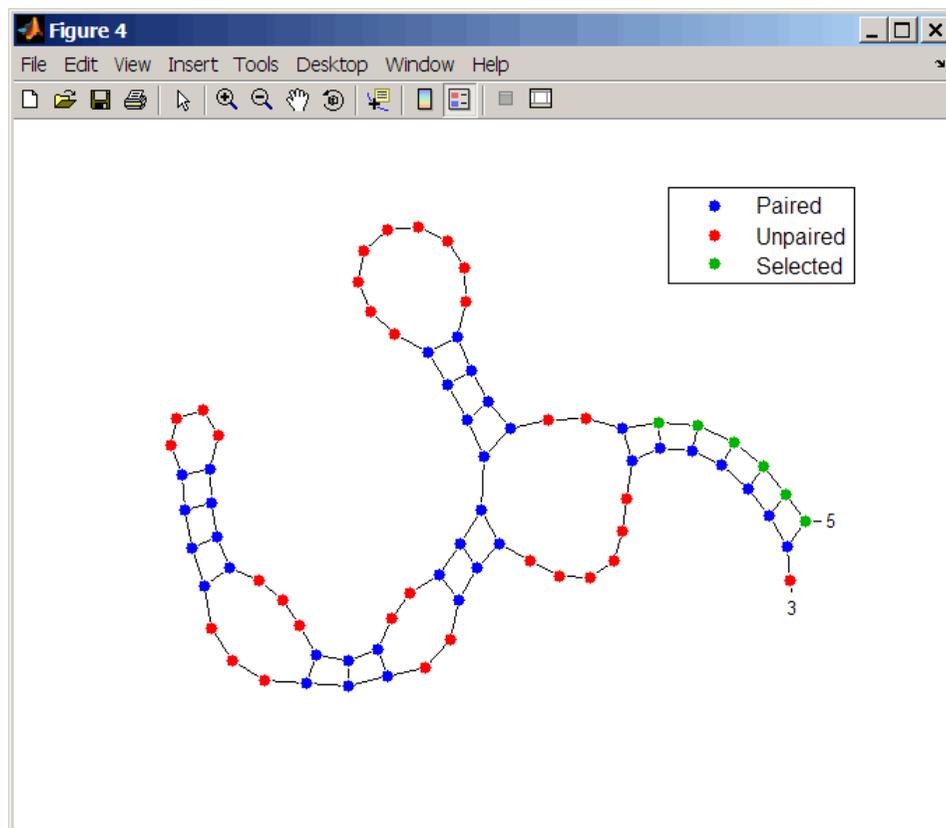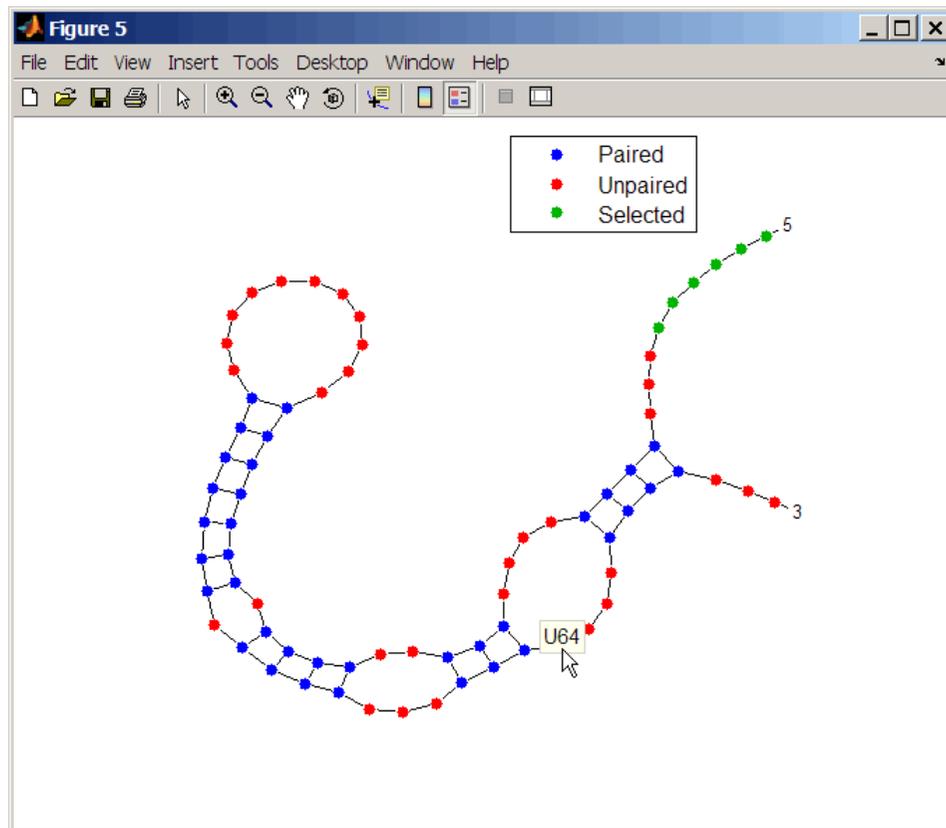
Tip If necessary, click-drag the legend to prevent it from covering the plot. Click a base in the plot to display a data tip with information on that base.

See Also
rnaconvert | rnafold

rownames (DataMatrix)

Retrieve or set row names of DataMatrix object

Syntax

```
ReturnRowNames = rownames(DMObj)
ReturnRowNames = rownames(DMObj, RowIndices)
DMObjNew = rownames(DMObj, RowIndices, RowNames)
```

Input Arguments

DMObj	DataMatrix object, such as created by `DataMatrix` (object constructor).
RowIndices	One or more rows in *DMObj*, specified by any of the following: • Positive integer • Vector of positive integers • String specifying a row name • Cell array of strings • Logical vector
RowNames	Row names specified by any of the following: • Numeric vector • Cell array of strings • Character array • Single string, which is used as a prefix for row names, with row numbers appended to the prefix • Logical `true` or `false` (default). If `true`, unique row names are assigned using the format `row1`, `row2`, `row3`, etc. If `false`, no row names are assigned.

	Note: The number of elements in *RowNames* must equal the number of elements in *RowIndices*.

Output Arguments

ReturnRowNames	String or cell array of strings containing row names in *DMObj*.
DMObjNew	DataMatrix object created with names specified by *RowIndices* and *RowNames*.

Description

ReturnRowNames = rownames(*DMObj*) returns *ReturnRowNames*, a cell array of strings specifying the row names in *DMObj*, a DataMatrix object.

ReturnRowNames = rownames(*DMObj*, *RowIndices*) returns the row names specified by *RowIndices*. *RowIndices* can be a positive integer, vector of positive integers, string specifying a row name, cell array of strings, or a logical vector.

DMObjNew = rownames(*DMObj*, *RowIndices*, *RowNames*) returns *DMObjNew*, a DataMatrix object with rows specified by *RowIndices* set to the names specified by *RowNames*. The number of elements in *RowIndices* must equal the number of elements in *RowNames*.

More About

· DataMatrix object

See Also
DataMatrix | colnames

saminfo

Return information about Sequence Alignment/Map (SAM) file

Syntax

```
InfoStruct = saminfo(File)
InfoStruct = saminfo(File,Name,Value)
```

Description

InfoStruct = saminfo(*File*) returns a MATLAB structure containing summary information about a SAM-formatted file.

InfoStruct = saminfo(File,Name,Value) returns a MATLAB structure with additional options specified by one or more Name,Value pair arguments.

Input Arguments

File

String specifying a file name or path and file name of a SAM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the Current Folder.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'NumOfReads'

Logical that controls the inclusion of a NumReads field in *InfoStruct*, the output structure.

> **Note:** Setting `NumOfReads` to `true` can significantly increase the time to create the output structure.

Default: `false`

`'ScanDictionary'`

Logical that controls the scanning of the SAM-formatted file to determine the reference names and the number of reads aligned to each reference. If `true`, the `ScannedDictionary` and `ScannedDictionaryCount` fields contain this information.

Default: `false`

Output Arguments

`InfoStruct`

MATLAB structure containing summary information about a SAM-formatted file. The structure contains these fields.

Field	Description
Filename	Name of the SAM-formatted file.
FilePath	Path to the file.
FileSize	Size of the file in bytes.
FileModDate	Modification date of the file.
NumReads*	Number of sequence reads in the file.
ScannedDictionary*	Cell array of strings specifying the names of the reference sequences in the SAM-formatted file.
ScannedDictionaryCount*	Cell array specifying the number of reads aligned to each reference sequence.
Header**	Structure containing the file format version, sort order, and group order.
SequenceDictionary**	Structure containing the: • Sequence name

Field	Description
	• Sequence length
	• Genome assembly identifier
	• MD5 checksum of sequence
	• URI of sequence
	• Species
ReadGroup**	Structure containing the:
	• Read group identifier
	• Sample
	• Library
	• Description
	• Platform unit
	• Predicted median insert size
	• Sequencing center
	• Date
	• Platform
Program**	Structure containing the:
	• Program name
	• Version
	• Command line

* — The `NumReads` field is empty if you do not set the `NumOfReads` name-value pair argument to `true`. The `ScannedDictionary` and `ScannedDictionaryCount` fields are empty if you do not set the `ScanDictionary` name-value pair argument to `true`.

** — These structures and their fields appear in the output structure only if they are in the SAM file. The information in these structures depends on the information in the SAM file.

Examples

Return information about the `ex1.sam` file included with Bioinformatics Toolbox:

```
info = saminfo('ex1.sam')

info =

                   Filename: 'ex1.sam'
                   FilePath: [1x89 char]
                   FileSize: 254270
                FileModDate: '12-May-2011 14:23:25'
                     Header: [1x1 struct]
         SequenceDictionary: [1x1 struct]
                  ReadGroup: [1x2 struct]
                   NumReads: []
           ScannedDictionary: {0x1 cell}
      ScannedDictionaryCount: [0x1 uint64]
```

Return information about the `ex1.sam` file including the number of sequence reads:

```
info = saminfo('ex1.sam','numofreads', true)

info =

                   Filename: 'ex1.sam'
                   FilePath: [1x89 char]
                   FileSize: 254270
                FileModDate: '12-May-2011 14:23:25'
                     Header: [1x1 struct]
         SequenceDictionary: [1x1 struct]
                  ReadGroup: [1x2 struct]
                   NumReads: 1501
           ScannedDictionary: {0x1 cell}
      ScannedDictionaryCount: [0x1 uint64]
```

More About

Tips
Use `saminfo` to investigate the size and content of a SAM file before using the `samread` function to read the file contents into a MATLAB structure.

- "Manage Short-Read Sequence Data in Objects"

- Sequence Read Archive

- SAM format specification

References

[1] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Goncalo, A., and Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics *25, 16*, 2078–2079.

See Also

`BioIndexedFile` | `BioMap` | `samread` | `fastqread` | `fastqwrite` | `fastqinfo` | `fastainfo` | `fastaread` | `fastawrite` | `sffinfo` | `sffread`

samplealign

Align two data sets containing sequential observations by introducing gaps

Syntax

```
[I, J] = samplealign(X, Y)
[I, J] = samplealign(X, Y, ...'Band', BandValue, ...)
[I, J] = samplealign(X, Y, ...'Width', WidthValue, ...)
[I, J] = samplealign(X, Y, ...'Gap', GapValue, ...)
[I, J] = samplealign(X, Y, ...'Quantile', QuantileValue, ...)
[I, J] = samplealign(X, Y, ...'Distance', DistanceValue, ...)
[I, J] = samplealign(X, Y, ...'Weights', WeightsValue, ...)
[I, J] = samplealign(X, Y, ...'ShowConstraints',
ShowConstraintsValue, ...)
[I, J] = samplealign(X, Y, ...'ShowNetwork', ShowNetworkValue, ...)
[I, J] = samplealign(X, Y, ...'ShowAlignment',
ShowAlignmentValue, ...)
```

Input Arguments

X, *Y*	Matrices of data where rows correspond to observations or samples, and columns correspond to features or dimensions. *X* and *Y* can have a different number of rows, but they must have the same number of columns. The first column is the reference dimension and must contain unique values in ascending order. The reference dimension could contain sample indices of the observations or a measurable value, such as time.
BandValue	Either of the following: • Scalar. • Function specified using @(*z*), where *z* is the mid-point between a given observation in one data set and a given observation in the other data set.

	BandValue specifies a maximum allowable distance between observations (along the reference dimension only) in the two data sets, thus limiting the number of potential matches between observations in two data sets. If *S* is the value in the reference dimension for a given observation (row) in one data set, then that observation is matched only with observations in the other data set whose values in the reference dimension fall within *S* ± *BandValue*. Then, only these potential matches are passed to the algorithm for further scoring. Default *BandValue* is Inf.
WidthValue	Either of the following: • Two-element vector, [*U*, *V*] • Scalar that is used for both *U* and *V* *WidthValue* limits the number of potential matches between observations in two data sets; that is, each observation in *X* is scored to the closest *U* observations in *Y*, and each observation in *Y* is scored to the closest *V* observations in *X*. Then, only these potential matches are passed to the algorithm for further scoring. Closeness is measured using only the first column (reference dimension) in each data set. Default is Inf if 'Band' is specified; otherwise default is 10.

GapValue	Any of the following:
	• Cell array, {*G*, *H*}, where *G* is either a scalar or a function handle specified using @(*X*), and *H* is either a scalar or a function handle specified using @(*Y*). The functions @(*X*) and @(*Y*) must calculate the penalty for each observation (row) when it is matched to a gap in the other data set. The functions @(*X*) and @(*Y*) must return a column vector with the same number of rows as *X* or *Y*, containing the gap penalty for each observation (row).
	• Single function handle specified using @(*Z*), which is used for both *G* and *H*. The function @(*Z*) must calculate the penalty for each observation (row) in both *X* and *Y* when it is matched to a gap in the other data set. The function @(*Z*) must take as arguments *X* and *Y*. The function @(*Z*) must return a column vector with the same number of rows as *X* or *Y*, containing the gap penalty for each observation (row).
	• Scalar that is used for both *G* and *H*.
	GapValue specifies the position-dependent terms for assigning gap penalties. The calculated value, *GPX*, is the gap penalty for matching observations from the first data set *X* to gaps inserted in the second data set *Y*, and is the product of two terms: *GPX = G * QMS*. The term *G* takes its value as a function of the observations in *X*. Similarly, *GPY* is the gap penalty for matching observations from *Y* to gaps inserted in *X*, and is the product of two terms: *GPY = H * QMS*. The term *H* takes its value as a function of the observations in *Y*. By default, the term *QMS* is the 0.75 quantile of the score for the pairs of observations that are potential matches (that is, pairs that comply with the 'Band' and 'Width' constraints). Default *GapValue* is 1.

QuantileValue	Scalar between 0 and 1 that specifies the quantile value used to calculate the term *QMS*, which is used by the 'Gap' property to calculate gap penalties. Default is 0.75.
DistanceValue	Function handle specified using @(*R*,*S*). The function @(*R*,*S*) must: • Calculate the distance between pairs of observations that are potential matches. • Take as arguments, *R* and *S*, matrices that have the same number of rows and columns, and whose paired rows represent all potential matches of observations in *X* and *Y* respectively. • Return a column vector of positive values with the same number of elements as rows in *R* and *S*. Default is the Euclidean distance between the pairs. --- **Caution** All columns in *X* and *Y*, including the reference dimension, are considered when calculating distances. If you do not want to include the reference dimension in the distance calculations, use the 'Weight' property to exclude it.

WeightsValue	Either of the following:
	• Logical row vector with the same number of elements as columns in *X* and *Y*, that specifies columns in *X* and *Y*.
	• Numeric row vector with the same number of elements as columns in *X* and *Y*, that specifies the relative weights of the columns (features).
	This property controls the inclusion/exclusion of columns (features) or the emphasis of columns (features) when calculating the distance score between observations that are potential matches, that is, when using the `'Distance'` property. Default is a logical row vector with all elements set to `true`.
	Tip Using a numeric row vector for *WeightsValue* and setting some values to `0` can simplify the distance calculation when the data sets have many columns (features).
	Note: The weight values are not considered when using the `'Band'`, `'Width'`, or `'Gap'` property.
ShowConstraintsValue	Controls the display of the search space constrained by the specified `'Band'` and `'Width'` input parameters, thereby giving an indication of the memory required to run the algorithm with the specific `'Band'` and `'Width'` parameters on your data sets. Choices are `true` or `false` (default).
ShowNetworkValue	Controls the display of the dynamic programming network, the match scores, the gap penalties, and the winning path. Choices are `true` or `false` (default).

| *ShowAlignmentValue* | Controls the display of the first and second columns of the *X* and *Y* data sets in the abscissa and the ordinate respectively, of a two-dimensional plot. Choices are `true`, `false` (default), or an integer specifying a column of the *X* and *Y* data sets to plot as the ordinate. |

Output Arguments

| *I* | Column vector containing indices of rows (observations) in *X* that match to a row (observation) in *Y*. Missing indices indicate that row (observation) is matched to a gap. |
| *J* | Column vector containing indices of rows (observations) in *Y* that match to a row (observation) in *X*. Missing indices indicate that row (observation) is matched to a gap. |

Description

`[I, J] = samplealign(X, Y)` aligns the observations in two matrices of data, *X* and *Y*, by introducing gaps. *X* and *Y* are matrices of data where rows correspond to observations or samples, and columns correspond to features or dimensions. *X* and *Y* can have different number of rows, but must have the same number of columns. The first column is the reference dimension and must contain unique values in ascending order. The reference dimension could contain sample indices of the observations or a measurable value, such as time. The `samplealign` function uses a dynamic programming algorithm to minimize the sum of positive scores resulting from pairs of observations that are potential matches and the penalties resulting from the insertion of gaps. Return values *I* and *J* are column vectors containing indices that indicate the matches for each row (observation) in *X* and *Y* respectively.

Tip If you do not specify return values, `samplealign` does not run the dynamic programming algorithm. Running `samplealign` without return values, but setting the `'ShowConstraints'`, `'ShowNetwork'`, or `'ShowAlignment'` property to `true`, lets you explore the constrained search space, the dynamic programming network, or the aligned observations, without running into potential memory problems.

[*I*, *J*] = samplealign(*X*, *Y*, ...'*PropertyName*', *PropertyValue*, ...) calls samplealign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

[*I*, *J*] = samplealign(*X*, *Y*, ...'Band', *BandValue*, ...) specifies a maximum allowable distance between observations (along the reference dimension only) in the two data sets, thus limiting the number of potential matches between observations in the two data sets. If *S* is the value in the reference dimension for a given observation (row) in one data set, then that observation is matched only with observations in the other data set whose values in the reference dimension fall within *S* ± *BandValue*. Then, only these potential matches are passed to the algorithm for further scoring. *BandValue* can be a scalar or a function specified using @(*z*), where *z* is the mid-point between a given observation in one data set and a given observation in the other data set. Default *BandValue* is Inf.

This constraint reduces the time and memory complexity of the algorithm from O(*MN*) to O(sqrt(*MN*)*K), where *M* and *N* are the number of observations in *X* and *Y* respectively, and *K* is a small constant such that *K*<<*M* and *K*<<*N*. Adjust *BandValue* to the maximum expected shift between the reference dimensions in the two data sets, that is, between *X*(:,1) and *Y*(:,1).

[*I*, *J*] = samplealign(*X*, *Y*, ...'Width', *WidthValue*, ...) limits the number of potential matches between observations in two data sets; that is, each observation in *X* is scored to the closest *U* observations in *Y*, and each observation in *Y* is scored to the closest *V* observations in *X*. Then, only these potential matches are passed to the algorithm for further scoring. *WidthValue* is either a two-element vector, [*U*, *V*] or a scalar that is used for both *U* and *V*. Closeness is measured using only the first column (reference dimension) in each data set. Default is Inf if 'Band' is specified; otherwise default is 10.

This constraint reduces the time and memory complexity of the algorithm from O(*MN*) to O(sqrt(*MN*)*sqrt(*UV*)), where *M* and *N* are the number of observations in *X* and *Y* respectively, and *U* and *V* are small such that *U*<<*M* and *V*<<*N*.

Note: If you specify both 'Band' and 'Width', only pairs of observations that meet both constraints are considered potential matches and passed to the algorithm for scoring.

Tip Specify `'Width'` when you do not have a good estimate for the `'Band'` property. To get an indication of the memory required to run the algorithm with specific `'Band'` and `'Width'` parameters on your data sets, run `samplealign`, but do not specify return values and set `'ShowConstraints'` to `true`.

[*I*, *J*] = samplealign(*X*, *Y*, ...'Gap', *GapValue*, ...) specifies the position-dependent terms for assigning gap penalties.

GapValue is any of the following:

- Cell array, {*G*, *H*}, where *G* is either a scalar or a function handle specified using @(*X*), and *H* is either a scalar or a function handle specified using @(*Y*). The functions @(*X*) and @(*Y*) must calculate the penalty for each observation (row) when it is matched to a gap in the other data set. The functions @(*X*) and @(*Y*) must return a column vector with the same number of rows as *X* or *Y*, containing the gap penalty for each observation (row).

- Single function handle specified using @(*Z*), that is used for both *G* and *H*. The function @(*Z*) must calculate the penalty for each observation (row) in both *X* and *Y* when it is matched to a gap in the other data set. The function @(*Z*) must take as arguments *X* and *Y*. The function @(*Z*) must return a column vector with the same number of rows as *X* or *Y*, containing the gap penalty for each observation (row).

- Scalar that is used for both *G* and *H*.

The calculated value, *GPX*, is the gap penalty for matching observations from the first data set *X* to gaps inserted in the second data set *Y*, and is the product of two terms: *GPX* = *G* * *QMS*. The term *G* takes its value as a function of the observations in *X*. Similarly, *GPY* is the gap penalty for matching observations from *Y* to gaps inserted in *X*, and is the product of two terms: *GPY* = *H* * *QMS*. The term *H* takes its value as a function of the observations in *Y*. By default, the term *QMS* is the 0.75 quantile of the score for the pairs of observations that are potential matches (that is, pairs that comply with the `'Band'` and `'Width'` constraints).

If *G* and *H* are positive scalars, then *GPX* and *GPY* are independent of the observation where the gap is being inserted.

Default *GapValue* is 1, that is, both *G* and *H* are 1, which indicates that the default penalty for gap insertions in both sequences is equivalent to the quantile (set by the `'Quantile'` property, default = 0.75) of the score for the pairs of observations that are potential matches.

Note: *GapValue* defaults to a relatively safe value. However, the success of the algorithm depends on the fine tuning of the gap penalties, which is application dependent. When the gap penalties are large relative to the score of the correct matches, `samplealign` returns alignments with fewer gaps, but with more incorrectly aligned regions. When the gap penalties are smaller, the output alignment contains longer regions with gaps and fewer matched observations. Set `'ShowNetwork'` to `true` to compare the gap penalties to the score of matched observations in different regions of the alignment.

[*I*, *J*] = samplealign(*X*, *Y*, ...'Quantile', *QuantileValue*, ...) specifies the quantile value used to calculate the term *QMS*, which is used by the `'Gap'` property to calculate gap penalties. *QuantileValue* is a scalar between 0 and 1. Default is 0.75.

Tip Set *QuantileValue* to an empty array ([ ]) to make the gap penalties independent of *QMS*, that is, *GPX* and *GPY* are functions of only the *G* and *H* input parameters respectively.

[*I*, *J*] = samplealign(*X*, *Y*, ...'Distance', *DistanceValue*, ...) specifies a function to calculate the distance between pairs of observations that are potential matches. *DistanceValue* is a function handle specified using @(*R*,*S*). The function @(*R*,*S*) must take as arguments, *R* and *S*, matrices that have the same number of rows and columns, and whose paired rows represent all potential matches of observations in *X* and *Y* respectively. The function @(*R*,*S*) must return a column vector of positive values with the same number of elements as rows in *R* and *S*. Default is the Euclidean distance between the pairs.

Caution All columns in *X* and *Y*, including the reference dimension, are considered when calculating distances. If you do not want to include the reference dimension in the distance calculations, use the `'Weight'` property to exclude it.

[*I*, *J*] = samplealign(*X*, *Y*, ...'Weights', *WeightsValue*, ...) controls the inclusion/exclusion of columns (features) or the emphasis of columns (features) when calculating the distance score between observations that are potential matches, that is when using the `'Distance'` property. *WeightsValue* can be a logical row vector that specifies columns in *X* and *Y*. *WeightsValue* can also be a numeric row vector with the

same number of elements as columns in *X* and *Y*, that specifies the relative weights of the columns (features). Default is a logical row vector with all elements set to `true`.

Tip Using a numeric row vector for *WeightsValue* and setting some values to `0` can simplify the distance calculation when the data sets have many columns (features).

Note: The weight values are not considered when computing the constrained alignment space, that is when using the `'Band'` or `'Width'` properties, or when calculating the gap penalties, that is when using the `'Gap'` property.

`[I, J] = samplealign(X, Y, ...'ShowConstraints',` *ShowConstraintsValue*, ...) controls the display of the search space constrained by the input parameters `'Band'` and `'Width'`, giving an indication of the memory required to run the algorithm with specific `'Band'` and `'Width'` on your data sets. Choices are `true` or `false` (default).

`[I, J] = samplealign(X, Y, ...'ShowNetwork',` *ShowNetworkValue*, ...) controls the display of the dynamic programming network, the match scores, the gap penalties, and the winning path. Choices are `true` or `false` (default).

`[I, J] = samplealign(X, Y, ...'ShowAlignment',` *ShowAlignmentValue*, ...) controls the display of the first and second columns of the *X* and *Y* data sets in the abscissa and the ordinate respectively, of a two-dimensional plot. Links between all the potential matches that meet the constraints are displayed, and the matches belonging to the output alignment are highlighted. Choices are `true`, `false` (default), or an integer specifying a column of the *X* and *Y* data sets to plot as the ordinate.

Examples

Warping a sine wave with a smooth function to more closely follow cyclical sunspot activity

1 Load `sunspot.dat`, a data file included with the MATLAB software, that contains the variable `sunspot`, which is a two-column matrix containing variations in sunspot activity over the last 300 years. The first column is the reference dimension (years), and the second column contains sunspot activity values. Sunspot activity is cyclical, reaching a maximum about every 11 years.

```
load sunspot.dat
```

2 Create a sine wave with a known period of sunspot activity.

```
years = (1700:1990)';
T = 11.038;
f = @(y) 60 + 60 * sin(y*(2*pi/T));
```

3 Align the observations between the sine wave and the sunspot activity by introducing gaps.

```
[i,j] = samplealign([years f(years)],sunspot,'weights',...
                    [0 1],'showalignment',true);
```
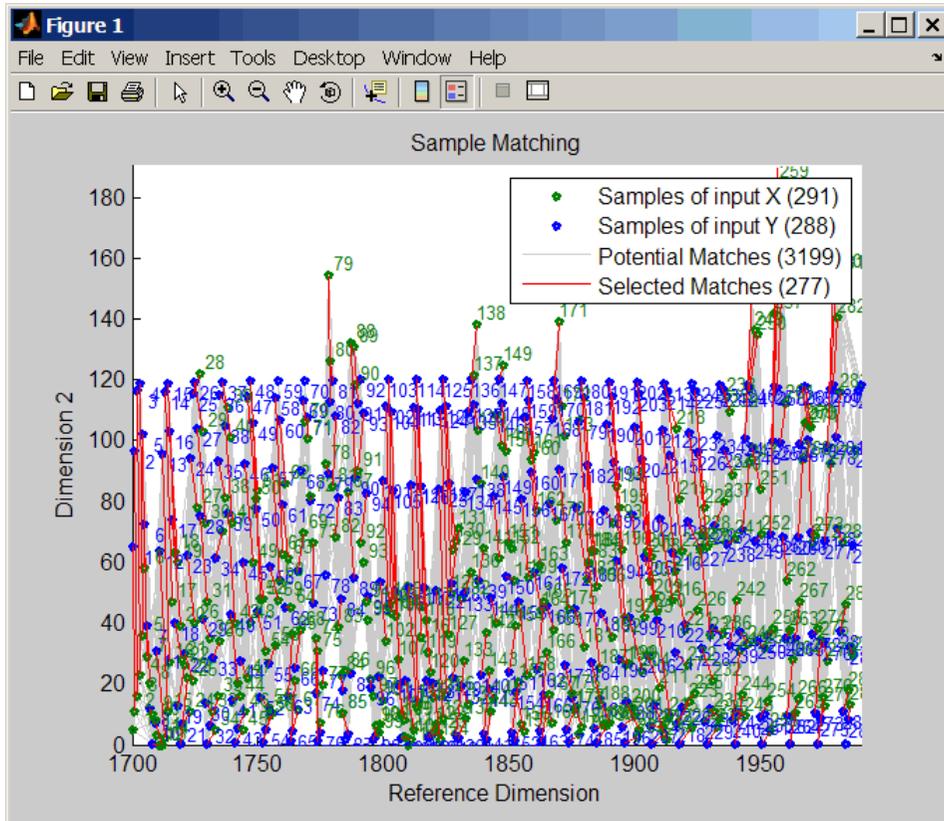

4 Estimate a smooth function to warp the sine wave.

```
[p,s,mu] = polyfit(years(i),years(j),15);
wy = @(y) polyval(p,(y-mu(1))./mu(2));
```

5 Plot the sunspot cycles, unwarped sine wave, and warped sine wave.

```
years = (1700:1/12:1990)';
figure
plot(sunspot(:,1),sunspot(:,2),years,f(years),wy(years),...
    f(years))
legend('Sunspots','Unwarped Sine Wave','Warped Sine Wave')
title('Smooth Warping Example')
```
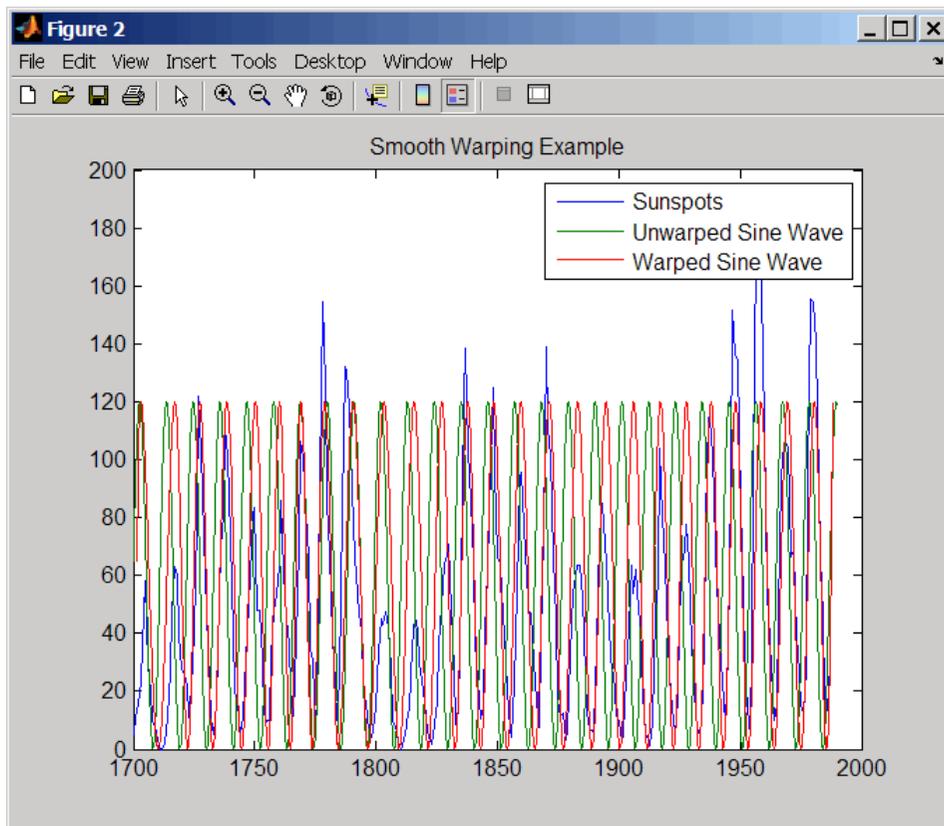

Recovering a nonlinear warping between two signals containing noisy Gaussian peaks

1 Create two signals with noisy Gaussian peaks.

```
rand('twister',5489)
peakLoc = [30 60 90 130 150 200 230 300 380 430];
peakInt = [7 1 3 10 3 6 1 8 3 10];
time = 1:450;
comp = exp(-(bsxfun(@minus,time,peakLoc')./5).^2);
sig_1 = (peakInt + rand(1,10)) * comp + rand(1,450);
sig_2 = (peakInt + rand(1,10)) * comp + rand(1,450);
```

2 Define a nonlinear warping function.

```
wf = @(t) 1 + (t<=100).*0.01.*(t.^2) + (t>100).*...
      (310+150*tanh(t./100-3));
```

3 Warp the second signal to distort it.

```
sig_2 = interp1(time,sig_2,wf(time),'pchip');
```

4 Align the observations between the two signals by introducing gaps.

```
[i,j] = samplealign([time;sig_1]',[time;sig_2]',...
                    'weights',[0,1],'band',35,'quantile',.5);
```

5 Plot the reference signal, distorted signal, and warped (corrected) signal.

```
figure
sig_3 = interp1(time,sig_2,interp1(i,j,time,'pchip'),'pchip');
plot(time,sig_1,time,sig_2,time,sig_3)
legend('Reference','Distorted Signal','Corrected Signal')
title('Non-linear Warping Example')
```
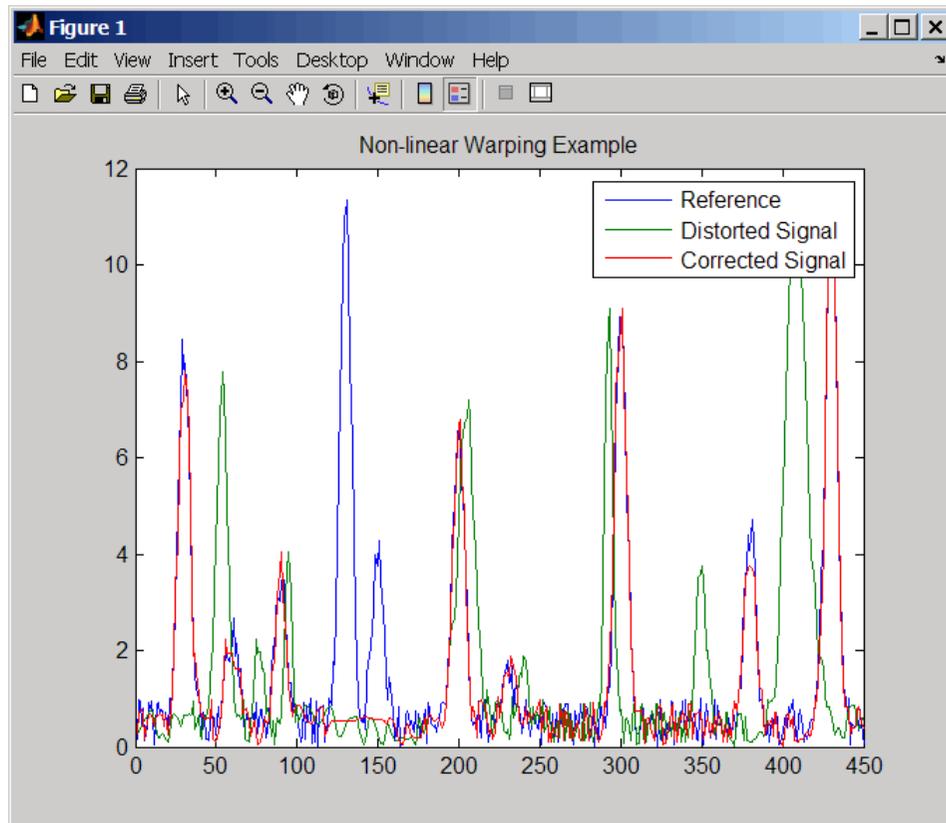
6 Plot the real and the estimated warping functions.

```
figure
plot(time,wf(time),time,interp1(j,i,time,'pchip'))
legend('Distorting Function','Estimated Warping')
```
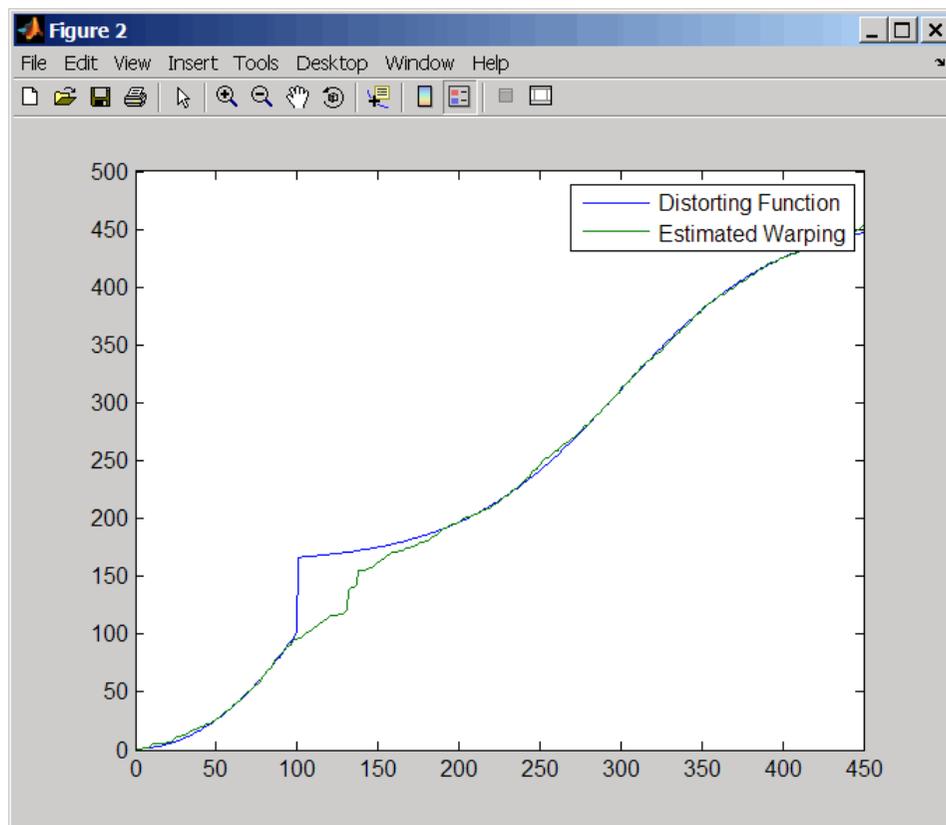
References

[1] Myers, C.S. and Rabiner, L.R. (1981). A comparative study of several dynamic time-warping algorithms for connected word recognition. The Bell System Technical Journal *60:7*, 1389–1409.

[2] Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans. Acoustics, Speech and Signal Processing *ASSP-26(1)*, 43–49.

See Also

msalign | msheatmap | mspalign | msppresample | msresample

sampleData

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set sample metadata in ExpressionSet object

Syntax

MetaDataObj = sampleData(*ESObj*)
NewESObj = sampleData(*ESObj*, *NewMetaDataObj*)

Description

MetaDataObj = sampleData(*ESObj*) returns a MetaData object containing the sample metadata from an ExpressionSet object.

NewESObj = sampleData(*ESObj*, *NewMetaDataObj*) replaces the sample metadata in *ESObj*, an ExpressionSet object, with *NewMetaDataObj*, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

NewMetaDataObj

Object of the `bioma.data.MetaData` class, containing sample metadata, stored in two dataset arrays. The sample names and variable names in *NewMetaDataObj* must match the sample names and variable names in the *MetaDataObj* being replaced in the ExpressionSet object, *ESObj*.

Default:

Output Arguments

MetaDataObj

Object of the `bioma.data.MetaData` class, containing the sample metadata, stored in two dataset arrays.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing the MetaData object containing the sample metadata.

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1-   section of the `bioma.ExpressionSet` class reference page. Retrieve the MetaData object that contains sample metadata, stored in the ExpressionSet object:

```
% Retrieve the sample data
NewMDObj = sampleData(ESObj);
```

See Also

bioma.ExpressionSet | bioma.data.ExptData | sampleNames | featureData

How To

- "Managing Gene Expression Data in Objects"

sampleNames

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set sample names in ExpressionSet object

Syntax

```
SamNames = sampleNames(ESObj)
SamNames = sampleNames(ESObj, Subset)
NewESObj = sampleNames(ESObj, Subset, NewSamNames)
```

Description

SamNames = sampleNames(*ESObj*) returns a cell array of strings specifying all sample names in an ExpressionSet object.

SamNames = sampleNames(*ESObj*, *Subset*) returns a cell array of strings specifying a subset the sample names in an ExpressionSet object.

NewESObj = sampleNames(*ESObj*, *Subset*, *NewSamNames*) replaces the sample names specified by *Subset* in *ESObj*, an ExpressionSet object, with *NewSamNames*, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

Subset

One of the following to specify a subset of the sample names in an ExpressionSet object:

- String specifying a sample name
- Cell array of strings specifying sample names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewSamNames

New sample names for specific sample names within an ExpressionSet object, specified by one of the following:

- Numeric vector
- String or cell array of strings
- String, which `sampleNames` uses as a prefix for the sample names, with sample numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `sampleNames` assigns unique sample names using the format `Sample1`, `Sample2`, etc.

The number of sample names in *NewSamNames* must equal the number of samples specified by *Subset*.

Default:

Output Arguments

SamNames

Cell array of strings specifying all or some of the sample names in an ExpressionSet object. The sample names are the column names in the DataMatrix objects in the ExpressionSet object. The sample names are also the row names of the *VarValues* dataset array in the MetaData object in the ExpressionSet object.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing specific sample names.

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page
1-   section of the `bioma.ExpressionSet` class reference page. Retrieve the sample
names from it:

```
% Retrieve the sample names
SNames = sampleNames(ESObj);
```

See Also

`bioma.ExpressionSet` | `bioma.data.ExptData` | `bioma.data.MetaData` |
`featureNames` | `DataMatrix`

How To

· "Managing Gene Expression Data in Objects"

sampleNames

Class: bioma.data.ExptData
Package: bioma.data

Retrieve or set sample names in ExptData object

Syntax

```
SamNames = sampleNames(EDObj)
SamNames = sampleNames(EDObj, Subset)
NewEDObj = sampleNames(EDObj, Subset, NewSamNames)
```

Description

SamNames = sampleNames(*EDObj*) returns a cell array of strings specifying all sample names in an ExptData object.

SamNames = sampleNames(*EDObj*, *Subset*) returns a cell array of strings specifying a subset the sample names in an ExptData object.

NewEDObj = sampleNames(*EDObj*, *Subset*, *NewSamNames*) replaces the sample names specified by *Subset* in *EDObj*, an ExptData object, with *NewSamNames*, and returns *NewEDObj*, a new ExptData object.

Input Arguments

EDObj

Object of the `bioma.data.ExptData` class.

Default:

Subset

One of the following to specify a subset of the sample names in an ExptData object:

- String specifying a sample name
- Cell array of strings specifying sample names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewSamNames

New sample names for specific sample names within an ExptData object, specified by one of the following:

- Numeric vector
- String or cell array of strings
- String, which `sampleNames` uses as a prefix for the sample names, with sample numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `sampleNames` assigns unique sample names using the format `Sample1`, `Sample2`, etc.

The number of sample names in *NewSamNames* must equal the number of samples specified by *Subset*.

Default:

Output Arguments

SamNames

Cell array of strings specifying all or some of the sample names in an ExptData object. The sample names are the column names in the DataMatrix objects in the ExptData object.

NewEDObj

Object of the `bioma.data.ExptData` class, returned after replacing specific sample names.

Examples

Construct an ExptData object, and then retrieve the sample names from it:

```
% Import bioma.data package to make constructor functions
% available
import bioma.data.*
% Create DataMatrix object from .txt file containing
% expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Retrieve sample names
SNames = sampleNames(EDObj);
```

See Also
bioma.data.ExptData | dmNames | DataMatrix | elementNames | featureNames

How To

· "Representing Expression Data Values in ExptData Objects"

sampleNames

Class: bioma.data.MetaData
Package: bioma.data

Retrieve or set sample names in MetaData object

Syntax

SamFeatNames = sampleNames(*MDObj*)
SamFeatNames = sampleNames(*MDObj*, *Subset*)
NewMDObj = sampleNames(*MDObj*, *Subset*, *NewSamFeatNames*)

Description

SamFeatNames = sampleNames(*MDObj*) returns a cell array of strings specifying all sample names in a MetaData object.

SamFeatNames = sampleNames(*MDObj*, *Subset*) returns a cell array of strings specifying a subset the sample names in a MetaData object.

NewMDObj = sampleNames(*MDObj*, *Subset*, *NewSamFeatNames*) replaces the sample names specified by *Subset* in *MDObj*, a MetaData object, with *NewSamFeatNames*, and returns *NewMDObj*, a new MetaData object.

Input Arguments

MDObj

Object of the bioma.data.MetaData class.

Default:

Subset

One of the following to specify a subset of the sample names in a MetaData object:

- String specifying a sample name
- Cell array of strings specifying sample names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewSamFeatNames

New sample names for specific names within a MetaData object, specified by one of the following:

- Numeric vector
- String or cell array of strings
- String, which `sampleNames` uses as a prefix for the sample or feature names, with numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `sampleNames` assigns unique names using the format `Sample1`, `Sample2`, etc.

The number of names in *NewSamFeatNames* must equal the number of samples specified by *Subset*.

Default:

Output Arguments

SamFeatNames

Cell array of strings specifying all or some of the sample names in a MetaData object. The sample names are also the row names of the *VarValues* dataset array in the MetaData object.

NewMDObj

Object of the `bioma.data.MetaData` class, returned after replacing specific sample names.

Examples

Construct a MetaData object, and then retrieve the sample names from it:

```
% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the sample names
SNames = sampleNames(MDObj2)
```

See Also
bioma.data.MetaData | variableDesc | variableValues | variableNames

How To
·   "Representing Sample and Feature Metadata in MetaData Objects"

sampleVarDesc

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set sample variable descriptions in ExpressionSet object

Syntax

DSVarDescriptions = sampleVarDesc(*ESObj*)
NewESObj = sampleVarDesc(*ESObj*, *NewDSVarDescriptions*)

Description

DSVarDescriptions = sampleVarDesc(*ESObj*) returns a dataset array
containing the sample variable names and descriptions from the MetaData object in an
ExpressionSet object.

NewESObj = sampleVarDesc(*ESObj*, *NewDSVarDescriptions*) replaces the sample
variable descriptions in *ESObj*, an ExpressionSet object, with *NewDSVarDescriptions*,
and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

NewDSVarDescriptions

Descriptions of the sample variable names, specified by either of the following:

· A new dataset array containing the sample variable names and descriptions. In
  this dataset array, each row corresponds to a variable. The first column contains
  the variable name, and the second column (`VariableDescription`) contains a

description of the variable. The row names (variable names) must match the row names (variable names) in *DSVarDescriptions*, the dataset array being replaced in the MetaData object in the ExpressionSet object, *ESObj*.

- Cell array of strings containing descriptions of the sample variables. The number of elements in *VarDesc* must equal the number of row names (variable names) in *DSVarDescriptions*, the dataset array being replaced in the MetaData object in the ExpressionSet object, *ESObj*.

Default:

Output Arguments

DSVarDescriptions

A dataset array containing the sample variable names and descriptions from the MetaData object of an ExpressionSet object. In this dataset array, each row corresponds to a variable. The first column contains the variable name, and the second column (`VariableDescription`) contains a description of the variable.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing the dataset array containing the sample variable descriptions.

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1- section of the `bioma.ExpressionSet` class reference page. Retrieve the sample variable descriptions in the ExpressionSet object:

```
% Retrieve the sample variable descriptions
SVarDescriptions = sampleVarDesc(ESObj)
```

See Also
bioma.ExpressionSet | bioma.data.MetaData | variableDesc

How To
- "Managing Gene Expression Data in Objects"

sampleVarNames

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set sample variable names in ExpressionSet object

Syntax

SamVarNames = sampleVarNames(*ESObj*)
SamVarNames = sampleVarNames(*ESObj*, *Subset*)
NewESObj = sampleVarNames(*ESObj*, *Subset*, *NewSamVarNames*)

Description

SamVarNames = sampleVarNames(*ESObj*) returns a cell array of strings specifying all sample variable names in an ExpressionSet object.

SamVarNames = sampleVarNames(*ESObj*, *Subset*) returns a cell array of strings specifying a subset the sample variable names in an ExpressionSet object.

NewESObj = sampleVarNames(*ESObj*, *Subset*, *NewSamVarNames*) replaces the sample variable names specified by *Subset* in *ESObj*, an ExpressionSet object, with *NewSamVarNames*, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

Subset

One of the following to specify a subset of the sample variable names in an ExpressionSet object:

- String specifying a sample variable name
- Cell array of strings specifying sample variable names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewSamVarNames

New sample variable names for specific sample variable names within an ExpressionSet object, specified by one of the following:

- Numeric vector
- String or cell array of strings
- String, which `sampleVarNames` uses as a prefix for the sample variable names, with sample variable numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `sampleVarNames` assigns unique sample variable names using the format `Var1`, `Var2`, etc.

The number of sample variable names in *NewSamVarNames* must equal the number of sample variable names specified by *Subset*.

Default:

Output Arguments

SamVarNames

Cell array of strings specifying all or some of the sample variable names in an ExpressionSet object. The sample variable names are the column names of the *VarValues* dataset array. The sample variable names are also the row names of the *VarDescriptions* dataset array. Both dataset arrays are in the MetaData object in the ExpressionSet object.

NewESObj

Object of the `bioma.ExpressionSet` class, returned after replacing specific sample names.

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1-   section of the `bioma.ExpressionSet` class reference page. Retrieve the sample variable names from the ExpressionSet object:

```
% Retrieve the sample variable names
VNames = sampleVarNames(ESObj)
```

See Also

`bioma.ExpressionSet` | `bioma.data.MetaData` | `sampleNames` | `featureNames` | `featureVarNames`

How To

• "Managing Gene Expression Data in Objects"

sampleVarValues

Class: bioma.ExpressionSet
Package: bioma

Retrieve or set sample variable values in ExpressionSet object

Syntax

```
DSVarValues = sampleVarValues(ESObj)
NewESObj = sampleVarValues(ESObj, NewDSVarValues)
```

Description

DSVarValues = sampleVarValues(*ESObj*) returns a dataset array containing the measured value of each variable per sample from the MetaData object of an ExpressionSet object.

NewESObj = sampleVarValues(*ESObj*, *NewDSVarValues*) replaces the sample variable values in *ESObj*, an ExpressionSet object, with *NewDSVarValues*, and returns *NewESObj*, a new ExpressionSet object.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

NewDSVarValues

A new dataset array containing a value for each variable per sample. In this dataset array, the columns correspond to variables and rows correspond to samples. The row names (sample names) must match the row names (sample names) in *DSVarValues*, the dataset array being replaced in the MetaData object in the ExpressionSet object, *ESObj*.

Default:

Output Arguments

`DSVarValues`

A dataset array containing the measured value of each variable per sample from the MetaData object of an ExpressionSet object. In this dataset array, the columns correspond to variables and rows correspond to samples.

`NewESObj`

Object of the `bioma.ExpressionSet` class, returned after replacing the dataset array containing the sample variable values.

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1-   section of the `bioma.ExpressionSet` class reference page. Retrieve the sample variable values in ExpressionSet object:

```
% Retrieve the sample variable values
SVarValues = sampleVarValues(ESObj);
```

See Also
`bioma.ExpressionSet` | `bioma.data.MetaData` | `variableValues`

How To
• "Managing Gene Expression Data in Objects"

samread

Read data from Sequence Alignment/Map (SAM) file

Syntax

```
SAMStruct = samread(File)
[SAMStruct, HeaderStruct]= samread(File)
... = samread(File,'ParameterName',ParameterValue)
```

Description

SAMStruct = samread(*File*) reads a SAM-formatted file and returns the data in a MATLAB array of structures.

[*SAMStruct*, *HeaderStruct*]= samread(*File*) returns the alignment and header data in two separate variables.

... = samread(*File*,'*ParameterName*',*ParameterValue*) accepts one or more comma-separated parameter name/value pairs. Specify *ParameterName* inside single quotes.

Input Arguments

File

Either of the following:

- String specifying a file name or path and file name of a SAM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.
- MATLAB string containing the text of a SAM-formatted file.

Default:

Parameter Name/Value Pairs

`'Tags'`

Controls the reading of the optional tags in addition to the first 11 fields for each alignment in the SAM-formatted file. Choices are `true` (default) or `false`.

Default:

`'ReadGroup'`

String specifying the read group ID for which to read alignment records from. Default is to read records from all groups.

Tip For a list of the read groups (if present), return the header information in a separate *Header* structure and view the `ReadGroup` field in this structure.

Default:

`'BlockRead'`

Scalar or vector that controls the reading of a single sequence entry or block of sequence entries from a SAM-formatted file containing multiple sequences. Enter a scalar *N*, to read the *N*th entry in the file. Enter a 1-by-2 vector [*M1, M2*], to read a block of entries starting at the *M1* entry and ending at the *M2* entry. To read all remaining entries in the file starting at the *M1* entry, enter a positive value for *M1* and enter `Inf` for *M2*.

Default:

Output Arguments

`SAMStruct`

An *N*-by-1 array of structures containing sequence alignment and mapping information from a SAM-formatted file, where *N* is the number of alignment records stored in the SAM-formatted file. Each structure contains the following fields.

Field	Description
QueryName	Name of read sequence (if unpaired) or name of sequence pair (if paired).

Field	Description
	Tip You can use this information to populate the `Header` property of the BioMap object.
`Flag`	Integer indicating the bit-wise information that specifies the status of each of 11 flags described by the SAM format specification. **Tip** You can use the `bitget` function to determine the status of a specific SAM flag.
`ReferenceName`	Name of the reference sequence.
`Position`	Position (one-based offset) of the forward reference sequence where the left-most base of the alignment of the read sequence starts.
`MappingQuality`	Integer specifying the mapping quality score for the read sequence.
`CigarString`	CIGAR-formatted string representing how the read sequence aligns with the reference sequence.
`MateReferenceName`	Name of the reference sequence associated with the mate. If this name is the same as `ReferenceName`, then this value is `=`. If there is no mate, then this value is `*`.
`MatePosition`	Position (one-based offset) of the forward reference sequence where the left-most base of the alignment of the mate of the read sequence starts.
`InsertSize`	The number of base positions between the read sequence and its mate, when both are mapped to the same reference sequence. Otherwise, this value is `0`.
`Sequence`	String containing the letter representations of the read sequence. It is the reverse-complement if the read sequence aligns to the reverse strand of the reference sequence.

Field	Description
Quality	String containing the ASCII representation of the per-base quality score for the read sequence. The quality score is reversed if the read sequence aligns to the reverse strand of the reference sequence.
Tags	List of applicable SAM tags and their values.

HeaderStruct

Structure containing header information for the SAM-formatted file in the following fields.

Field	Description
Header*	Structure containing the file format version, sort order, and group order.
SequenceDictionary*	Structure containing the: • Sequence name • Sequence length • Genome assembly identifier • MD5 checksum of sequence • URI of sequence • Species
ReadGroup*	Structure containing the: • Read group identifier • Sample • Library • Description • Platform unit • Predicted median insert size • Sequencing center • Date

Field	Description
	• Platform
`Program*`	Structure containing the: • Program name • Version • Command line

* — These structures and their fields appear in the output structure only if they are present in the SAM file. The information in these structures depends on the information present in the SAM file.

Examples

Read the header information and the alignment data from the `ex1.sam` file included with Bioinformatics Toolbox, and then return the information in two separate variables:

```
[data header] = samread('ex1.sam');
```

Read a block of entries, excluding the tags, from the `ex1.sam` file, and then return the information in an array of structures:

```
% Read entries 5 through 10 and do not include the tags
data = samread('ex1.sam','blockread', [5 10], 'tags', false);
```

More About

Tips

- Use the `saminfo` function to investigate the size and content of a SAM-formatted file before using the `samread` function to read the file contents into a MATLAB array of structures.
- If your SAM-formatted file is too large to read using available memory, try one of the following:
  - Use the `BlockRead` parameter with the `samread` function to read a subset of entries.

- Create a BioIndexedFile object from the SAM-formatted file, then access the entries using methods of the `BioIndexedFile` class.

- Use the *SAMStruct* output argument that `samread` returns to create a `BioMap` object, which lets you explore, access, filter, and manipulate all or a subset of the data, before doing subsequent analyses or viewing the data.

- "Manage Short-Read Sequence Data in Objects"
- "Work with Large Multi-Entry Text Files"
- Sequence Read Archive
- SAM format specification

References

[1] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Goncalo, A., and Durbin, R. (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics *25, 16*, 2078–2079.

See Also
`BioIndexedFile` | `BioMap` | `saminfo` | `soapread` | `fastqread` | `bamread` | `bamindexread` | `baminfo` | `fastqwrite` | `fastqinfo` | `fastainfo` | `fastaread` | `fastawrite` | `sffinfo` | `sffread`

scfread

Read trace data from SCF file

Syntax

```
Sample = scfread(File)
[Sample, Probability] = scfread(File)
[Sample, Probability, Comments] = scfread(File)
[A, C, G, T] = scfread (File)
[A, C, G, T, ProbA, ProbC, ProbG, ProbT] = scfread (File)
[A, C, G, T, ProbA, ProbC, ProbG, ProbT, Comments, PkIndex, Base] =
scfread (File)
```

Arguments

File	String specifying the file name or a path and file name of an SCF formatted file.

Description

scfread reads data from an SCF formatted file into MATLAB structures.

Sample = scfread(*File*) reads an SCF formatted file and returns the sample data in the structure Sample, which contains the following fields:

Field	Description
A	Column vector containing intensity of A fluorescence tag
C	Column vector containing intensity of C fluorescence tag
G	Column vector containing intensity of G fluorescence tag
T	Column vector containing intensity of T fluorescence tag

[*Sample, Probability*] = scfread(*File*) also returns the probability data in the structure Probability, which contains the following fields:

Field	Description
peak_index	Column vector containing the position in the SCF file for the start of the data for each peak
prob_A	Column vector containing the probability of each base in the sequence being an A
prob_C	Column vector containing the probability of each base in the sequence being a C
prob_G	Column vector containing the probability of each base in the sequence being a G
prob_T	Column vector containing the probability of each base in the sequence being a T
base	Column vector containing the called bases for the sequence

[*Sample, Probability, Comments*] = scfread(*File*) also returns the comment information from the SCF file in a character array `Comments`.

[*A, C, G, T*] = scfread (*File*) returns the sample data for the four bases in separate variables.

[*A, C, G, T, ProbA, ProbC, ProbG, ProbT*] = scfread (*File*) also returns the probabilities data for the four bases in separate variables.

[*A, C, G, T, ProbA, ProbC, ProbG, ProbT, Comments, PkIndex, Base*] = scfread (*File*) also returns the peak indices and called bases in separate variables.

SCF files store data from DNA sequencing instruments. Each file includes sample data, sequence information, and the relative probabilities of each of the four bases.

Examples

```
[sampleStruct, probStruct, Comments] = scfread('sample.scf')
sampleStruct =

    A: [10827x1 double]
    C: [10827x1 double]
    G: [10827x1 double]
    T: [10827x1 double]
```

```
probStruct =

    peak_index: [742x1 double]
        prob_A: [742x1 double]
        prob_C: [742x1 double]
        prob_G: [742x1 double]
        prob_T: [742x1 double]
          base: [742x1 char]


Comments =

SIGN=A=121,C=103,G=119,T=82
SPAC= 16.25
PRIM=0
MACH=Arkansas_SN312
DYEP=DT3700POP5{BD}v2.mob
NAME=HCIUP1D61207
LANE=6
GELN=
PROC=
RTRK=
CONV=phred version=0.990722.h
COMM=
SRCE=ABI 373A or 377
```

See Also
genbankread | traceplot

select (phytree)

Select tree branches and leaves in phytree object

Syntax

```
S = select(Tree, N)
[S, Selleaves, Selbranches] = select(...)
select(..., 'Reference', ReferenceValue, ...)
select(..., 'Criteria', CriteriaValue, ...)
select(..., 'Threshold', ThresholdValue, ...)
select(..., 'Exclude', ExcludeValue, ...)
select(..., 'Propagate', PropagateValue, ...)
```

Arguments

Tree	Phylogenetic tree (`phytree` object) created with the function `phytree`.
N	Number of closest nodes to the root node.
ReferenceValue	Property to select a reference point for measuring distance.
CriteriaValue	Property to select a criteria for measuring distance.
ThresholdValue	Property to select a distance value. Nodes with distances below this value are selected.
ExcludeValue	Property to remove (exclude) branch or leaf nodes from the output. Enter `'none'`, `'branchs'`, or `'leaves'`. The default value is `'none'`.
PropagateValue	Property to select propagating nodes toward the leaves or the root.

Description

S = select(*Tree*, *N*) returns a logical vector (*S*) of size [NumNodes x 1] indicating the *N* closest nodes to the root node of a `phytree` object (`Tree`) where NumNodes =

NumLeaves + NumBranches. The first criterion `select` uses is branch levels, then patristic distance (also known as tree distance). By default, `select` uses `inf` as the value of N, and `select(`*Tree*`)` returns a vector with values of `true`.

[*S*, *Selleaves*, *Selbranches*] = `select(...)` returns two additional logical vectors, one for the selected leaves and one for the selected branches.

`select(..., '`*PropertyName*`', `*PropertyValue*`, ...)` calls `select` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`select(..., 'Reference', `*ReferenceValue*`, ...)` changes the reference point(s) to measure the closeness. `Reference` can be the root (default) or leaves. When using leaves, a node can have multiple distances to its descendant leaves (nonultrametric tree). If this the case, `select` considers the minimum distance to any descendant leaf.

`select(..., 'Criteria', `*CriteriaValue*`, ...)` changes the criteria `select` uses to measure closeness. If `C` = `'levels'` (default), the first criterion is branch levels and then patristic distance. If `C` = `'distance'`, the first criterion is patristic distance and then branch levels.

`select(..., 'Threshold', `*ThresholdValue*`, ...)` selects all the nodes where closeness is less than or equal to the threshold value (*ThresholdValue*). Notice, you can also use either of the properties `'criteria'` or `'reference'`, if N is not specified, then `N` = `infF`; otherwise you can limit the number of selected nodes by N.

`select(..., 'Exclude', `*ExcludeValue*`, ...)` when *ExcludeValue* = `'branches'`, sets a postfilter that excludes all the branch nodes from *S*, or when *ExcludeValue* = `'leaves'`, all the leaf nodes. The default is `'none'`.

`select(..., 'Propagate', `*PropagateValue*`, ...)` activates a postfunctionality that propagates the selected nodes to the leaves when P==`'toleaves'` or toward the root finding a common ancestor when P == `'toroot'`. The default value is `'none'`. P may also be `'both'`. The `'Propagate'` property acts after the `'Exclude'` property.

Examples

```
% Load a phylogenetic tree created from a protein family:
tr = phytreeread('pf00002.tree');
```

```
% To find close products for a given protein (e.g. vipr2_human):
ind = getbyname(tr,'vipr2_human');
[sel,sel_leaves] = select(tr,'criteria','distance',...
                          'threshold',0.6,'reference',ind);
view(tr,sel_leaves)

% To find potential outliers in the tree, use
[sel,sel_leaves] = select(tr,'criteria','distance',...
                          'threshold',.3,...
                          'reference','leaves',...
                          'exclude','leaves',...
                          'propagate','toleaves');
view(tr,~sel_leaves)
```

More About

· phytree object

See Also

phytree | pdist | prune | phytreeviewer | get

seq2regexp

Convert sequence with ambiguous characters to regular expression

Syntax

```
RegExp = seq2regexp(Seq)
RegExp = seq2regexp(Seq, ...'Alphabet', AlphabetValue, ...)
RegExp = seq2regexp(Seq, ...'Ambiguous', AmbiguousValue, ...)
```

Input Arguments

Seq	Either of the following:
	• Character string of codes specifying an amino acid or nucleotide sequence.
	• Structure containing a `Sequence` field that contains an amino acid or nucleotide sequence, such as returned by `fastaread`, `fastqread`, `getembl`, `getgenbank`, `getgenpept`, or `getpdb`.
AlphabetValue	String specifying the sequence alphabet. Choices are:
	• `'NT'` (default) — Nucleotide
	• `'AA'` — Amino acid
AmbiguousValue	Controls whether ambiguous characters are included in *RegExp*, the regular expression return value. Choices are:
	• `true` (default) — Include ambiguous characters in the return value
	• `false` — Return only unambiguous characters

Output Arguments

RegExp	Character string of codes specifying an amino acid or nucleotide sequence in regular expression format using IUB/IUPAC codes.

Description

RegExp = seq2regexp(*Seq*) converts ambiguous amino acid or nucleotide symbols in a sequence to a regular expression format using IUB/IUPAC codes.

RegExp = seq2regexp(*Seq*, ...'*PropertyName*', *PropertyValue*, ...) calls seq2regexp with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

RegExp = seq2regexp(*Seq*, ...'Alphabet', *AlphabetValue*, ...) specifies the sequence alphabet. *AlphabetValue* can be either 'NT' for nucleotide sequences or 'AA' for amino acid sequences. Default is 'NT'.

RegExp = seq2regexp(*Seq*, ...'Ambiguous', *AmbiguousValue*, ...) controls whether ambiguous characters are included in *RegExp*, the regular expression return value. Choices are true (default) or false. For example:

- If *Seq* = 'ACGTK', and *AmbiguousValue* is true , the MATLAB software returns ACGT[GTK] with the unambiguous characters G and T and the ambiguous character K.
- If *Seq* = 'ACGTK', and *AmbiguousValue* is false, the MATLAB software returns ACGT[GT] with only the unambiguous characters.

Nucleotide Conversions

Nucleotide Code	Nucleotide	Conversion
A	Adenosine	A
C	Cytosine	C
G	Guanine	G
T	Thymidine	T
U	Uridine	U
R	Purine	[AG]
Y	Pyrimidine	[TC]
K	Keto	[GT]
M	Amino	[AC]

Nucleotide Code	Nucleotide	Conversion
S	Strong interaction (3 H bonds)	[GC]
W	Weak interaction (2 H bonds)	[AT]
B	Not A	[CGT]
D	Not C	[AGT]
H	Not G	[ACT]
V	Not T or U	[ACG]
N	Any nucleotide	[ACGT]
-	Gap of indeterminate length	-
?	Unknown	?

Amino Acid Conversion

Amino Acid Code	Amino Acid	Conversion
B	Asparagine or Aspartic acid (Aspartate)	[DN]
Z	Glutamine or Glutamic acid (Glutamate)	[EQ]
X	Any amino acid	[A R N D C Q E G H I L K M F P S T W Y V]

Examples

1   Convert a nucleotide sequence to a regular expression.

```
seq2regexp('ACWTMAN')

ans =
AC[ATW]T[ACM]A[ACGTRYKMSWBDHVN]
```

2   Convert the same nucleotide sequence, but remove ambiguous characters from the regular expression.

```
seq2regexp('ACWTMAN', 'ambiguous', false)
```

```
ans =
AC[AT]T[AC]A[ACGT]
```

See Also

restrict | regexp | seqwordcount | regexpi

seqalignviewer

Visualize and edit multiple sequence alignment

Syntax

```
seqalignviewer
seqalignviewer(Alignment)
seqalignviewer(Alignment,Name,Value)

seqalignviewer( ___ ,'R2012b',true)

seqalignviewer('close')
```

Description

`seqalignviewer` opens the Sequence Alignment app, where you can display and interactively adjust multiple sequence alignments.

`seqalignviewer(Alignment)` loads a group of previously multiply aligned sequences into the app, where you can view and interactively adjust the alignment.

`seqalignviewer(Alignment,Name,Value)` opens the app with additional options specified by one or more Name,Value pair arguments.

`seqalignviewer( ___ ,'R2012b',true)` runs the previous version of the Sequence Alignment app, using any of the input arguments in previous syntaxes.

`seqalignviewer('close')` closes the Sequence Alignment app.

Tip If gaps are available after you have selected a block from aligned sequences, then there are three regions that you can drag and move horizontally:

- Selected block

- Block on the left of the selection
- Block on the right of the selection

Examples

View a Multiple Sequence Alignment File

This example shows how to view a multiple sequence alignment file.

Load and view a multiple sequence alignment file.

```
seqalignviewer('aagag.aln')
```


Alternatively, you can click Sequence Alignment on the **Apps** tab to open the app, and view the alignment data.

- "View and Align Multiple Sequences"

•   "Investigating the Bird Flu Virus"

Input Arguments

`Alignment` — Multiple sequence alignment (MSA) data
structure | character array | string | 3-by-N character array

Multiple sequence alignment (MSA) data, specified as:

- MATLAB structure containing a `Sequence` field, such as returned by `fastaread`, `gethmmalignment`, `multialign`, or `multialignread`
- MATLAB character array containing MSA data, such as returned by `multialign`
- String specifying a file or URL containing MSA data
- 3-by-N character array showing the pairwise alignment of two sequences, such as returned by `nwalign` or `swalign`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Alphabet','AA'` specifies that the aligned sequences are amino acid sequences.

`'Alphabet'` — Type of aligned sequences
`'AA'` | `'NT'`

Type of aligned sequences, specified as `'AA'` for amino acid sequences or `'NT'` for nucleotide sequences. If you do not specify the type, `seqalignviewer` attempts to determine the correct type. If it cannot, it defaults to `'AA'`.

Example: `'Alphabet','AA'`

`'SeqHeaders'` — List of names to label sequences in alignment window
array of structures containing a `Header` or `Name` field | cell array of strings

List of names to label the sequences in the alignment window, specified as a MATLAB array of structures containing a `Header` or `Name` field or cell array of strings. The

number of elements in either array must be the same as the number of sequences in the alignment data Alignment.

Example: `'SeqHeaders',names`

Alternatives

You can also display a color-coded multiple or pairwise sequence alignment using the `showalignment` function. However, the alignment displays in a MATLAB Figure window, where you cannot interact with it.

See Also

`cigar2align` | `fastaread` | `gethmmalignment` | `multialign` | `multialignread` | `multialignwrite` | `nwalign` | `seqviewer` | `showalignment` | `swalign`

seqcomplement

Calculate complementary strand of nucleotide sequence

Syntax

SeqC = seqcomplement(*SeqNT*)

Arguments

SeqNT	Nucleotide sequence specified by any of the following: • Character string with the characters A, C, G, T, U, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N. • Row vector of integers from the table Mapping Nucleotide Integers to Letter Codes. • MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `emblread`, `fastaread`, `fastqread`, `genbankread`, `getembl`, or `getgenbank`.

Description

SeqC = seqcomplement(*SeqNT*) calculates the complementary strand of a DNA or RNA nucleotide sequence. The return sequence, *SeqC*, is in the same format as *SeqNT*. For example, if *SeqNT* is a vector of integers, then so is *SeqC*.

Nucleotide in *SeqNT*	Converts to This Nucleotide in *SeqC*
A	T or U
C	G
G	C
T or U	A

Examples

Return the complement of a DNA nucleotide sequence.

```
s = 'ATCG';
seqcomplement(s)

ans =
TAGC
```

See Also

codoncount | palindromes | seqrcomplement | seqreverse | seqviewer

seqconsensus

Calculate consensus sequence

Syntax

```
CSeq = seqconsensus(Seqs)
[CSeq, Score] = seqconsensus(Seqs)
CSeq = seqconsensus(Profile)
seqconsensus(..., 'PropertyName', PropertyValue,...)
seqconsensus(..., 'ScoringMatrix', ScoringMatrixValue)
```

Arguments

Seqs	Set of multiply aligned amino acid or nucleotide sequences. Enter an array of strings, a cell array of strings, or an array of structures with the field `Sequence`.
Profile	Sequence profile. Enter a profile from the function `seqprofile`. `Profile` is a matrix of size `[20 (or 4) x Sequence Length]` with the frequency or count of amino acids (or nucleotides) for every position. `Profile` can also have 21 (or 5) rows if gaps are included in the consensus.

ScoringMatrixValue	Either of the following:
	
	• String specifying the scoring matrix to use for the alignment. Choices for amino acid sequences are:
	
	  • `'BLOSUM62'`
	  • `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'`
	  • `'BLOSUM100'`
	  • `'PAM10'` increasing by 10 up to `'PAM500'`
	  • `'DAYHOFF'`
	  • `'GONNET'`
	
	Default is:

- 'BLOSUM50' — When *AlphabetValue* equals 'AA'

- 'NUC44' — When *AlphabetValue* equals 'NT'

Note: The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the 'Scale' property to specify an additional scale factor to convert the output score from bits to another unit.

- A 21x21, 5x5, 20x20, or 4x4 numeric array. For the gap-included cases, gap scores (last row/column) are set to mean(diag(ScoringMatrix)) for a gap matching with another gap, and set to mean(nodiag(ScoringMatrix)) for a gap matching with another symbol.

Note: If you use a scoring matrix that you created, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix.

Note: If you need to compile seqconsensus into a stand-alone application or software component using MATLAB Compiler, use a matrix instead of a string for *ScoringMatrixValue*.

Description

CSeq = seqconsensus(*Seqs*), for a multiply aligned set of sequences (*Seqs*), returns a string with the consensus sequence (*CSeq*). The frequency of symbols (20 amino acids, 4 nucleotides) in the set of sequences is determined with the function seqprofile. For ambiguous nucleotide or amino acid symbols, the frequency or count is added to the standard set of symbols.

[*CSeq*, *Score*] = seqconsensus(*Seqs*) returns the conservation score of the consensus sequence. Scores are computed with the scoring matrix BLOSUM50 for amino acids or NUC44 for nucleotides. Scores are the average euclidean distance between the scored symbol and the M-dimensional consensus value. M is the size of the alphabet. The consensus value is the profile weighted by the scoring matrix.

CSeq = seqconsensus(*Profile*) returns a string with the consensus sequence (*CSeq*) from a sequence profile (*Profile*).

seqconsensus(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

seqconsensus(..., 'ScoringMatrix', *ScoringMatrixValue*) specifies the scoring matrix.

The following input parameters are analogous to the function seqprofile when the alphabet is restricted to 'AA' or 'NT'.

seqconsensus(..., 'Alphabet', *AlphabetValue*)

seqconsensus(..., 'Gaps', *GapsValue*)

seqconsensus(..., 'Ambiguous', *AmbiguousValue*)

seqconsensus(..., 'Limits', *LimitsValue*)

Examples

```
seqs = fastaread('pf00002.fa');
[C,S] = seqconsensus(seqs,'limits',[50 60],'gaps','all')
```

See Also
fastaread | multialignread | multialignwrite | profalign | seqdisp | seqprofile

seqdisp

Format long sequence output for easy viewing

Syntax

```
seqdisp(Seq)
seqdisp(Seq, ...'Row', RowValue, ...)
seqdisp(Seq, ...'Column', ColumnValue, ...)
seqdisp(Seq, ...'ShowNumbers', ShowNumbersValue, ...)
```

Arguments

Seq	Nucleotide or amino acid sequence represented by any of the following: • Character array • FASTA file name • MATLAB structure with the field `Sequence` Multiply aligned sequences are allowed. FASTA files can have the file extension `fa`, `fasta`, `fas`, `fsa`, or `fst`.
RowValue	Integer that specifies the length of each row. Default is `60`.
ColumnValue	Integer that specifies the column width or number of symbols before displaying a space. Default is `10`.
ShowNumbersValue	Controls the display of numbers at the start of each row. Choices are `true` (default) to show numbers, or `false` to hide numbers.

Description

seqdisp(*Seq*) displays a sequence in rows, with a default row length of 60 and a default column width of 10.

seqdisp(*Seq*, ...'*PropertyName*', *PropertyValue*, ...) calls seqdisp with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

seqdisp(*Seq*, ...'Row', *RowValue*, ...) specifies the length of each row for the displayed sequence.

seqdisp(*Seq*, ...'Column', *ColumnValue*, ...) specifies the number of letters to display before adding a space. *RowValue* must be larger than and evenly divisible by *ColumnValue*.

seqdisp(*Seq*, ...'ShowNumbers', *ShowNumbersValue*, ...) controls the display of numbers at the start of each row. Choices are true (default) to show numbers, or false to hide numbers.

Examples

Read sequence information from the GenBank database. Display the sequence in rows with 50 letters, and within a row, separate every 10 letters with a space.

```
mouseHEXA = getgenbank('AK080777');
seqdisp(mouseHEXA, 'Row', 50, 'Column', 10)
```

Create and save a FASTA file with two sequences, and then display it.

```
hdr = ['Sequence A'; 'Sequence B'];
seq = ['TAGCTGRCCAAGGCCAAGCGAGCTTN';'ATCGACYGGTTCCGGTTCGCTCGAAN']
fastawrite('local.fa', hdr, seq);
seqdisp('local.fa', 'ShowNumbers', false')

ans =
>Sequence A
 1  TAGCTGRCCA AGGCCAAGCG AGCTTN
>Sequence B
 1  ATCGACYGGT TCCGGTTCGC TCGAAN
```

See Also

multialignread | multialignwrite | seqconsensus | seqlogo | seqprofile | seqshoworfs | seqshowwords | seqviewer | getgenbank

seqdotplot

Create dot plot of two sequences

Syntax

```
seqdotplot(Seq1, Seq2)
seqdotplot(Seq1,Seq2, Window, Number)
Matches = seqdotplot(...)
[Matches, Matrix] = seqdotplot(...)
```

Arguments

Seq1, *Seq2*	Nucleotide or amino acid sequences. Enter two character strings. Do not enter a vector of integers. You can also enter a structure with the field Sequence.
Window	Enter an integer for the size of a window.
Number	Enter an integer for the number of characters within the window that match.

Description

seqdotplot(*Seq1, Seq2*) plots a figure that visualizes the match between two sequences.

seqdotplot(*Seq1,Seq2, Window, Number*) plots sequence matches when there are at least *Number* matches in a window of size *Window*.

When plotting nucleotide sequences, start with a Window of 11 and Number of 7.

Matches = seqdotplot(...) returns the number of dots in the dot plot matrix.

[*Matches, Matrix*] = seqdotplot(...) returns the dot plot as a sparse matrix.

Examples

This example shows the similarities between the prion protein (PrP) nucleotide sequences of two ruminants, the moufflon and the golden takin.

```
moufflon = getgenbank('AB060288','Sequence',true);
takin = getgenbank('AB060290','Sequence',true);
seqdotplot(moufflon,takin,11,7)
```
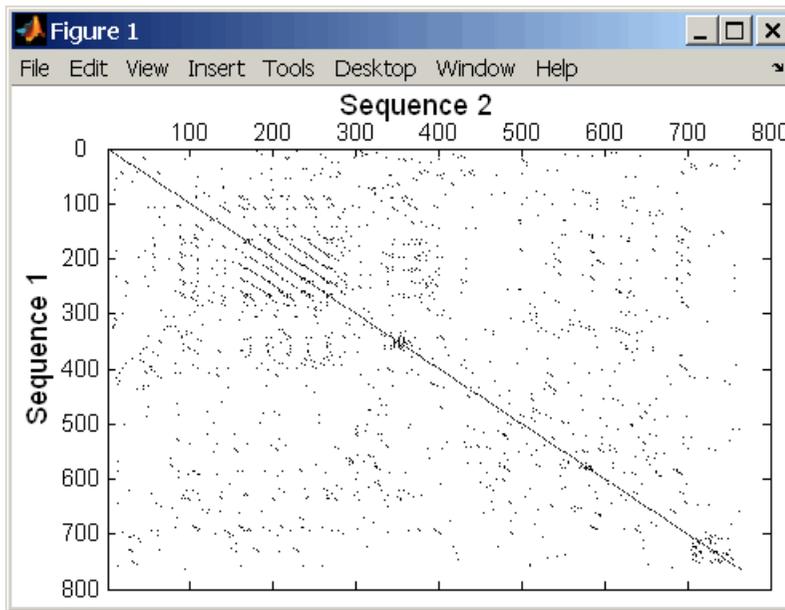

Note: For the correct interpretation of a dot plot, your monitor's display resolution must be able to contain the sequence lengths. If the resolution is not adequate, `seqdotplot` resizes the image and returns a warning.

```
Matches = seqdotplot(moufflon,takin,11,7)
Matches =
        5552

[Matches, Matrix] = seqdotplot(moufflon,takin,11,7)
```

See Also

nwalign | swalign

seqinsertgaps

Insert gaps into nucleotide or amino acid sequence

Syntax

NewSeq = seqinsertgaps(*Seq*, *Positions*)
NewSeq = seqinsertgaps(*Seq*, *GappedSeq*)
NewSeq = seqinsertgaps(*Seq*, *GappedSeq*, *Relationship*)

Input Arguments

Seq	Either of the following: • String specifying a nucleotide or amino acid sequence • MATLAB structure containing a Sequence field
Positions	Vector of integers to specify the positions in *Seq* before which to insert a gap.
GappedSeq	Either of the following: • String specifying a nucleotide or amino acid sequence • MATLAB structure containing a Sequence field
Relationship	Integer specifying the relationship between *Seq* and *GappedSeq*. Choices are: • 1 — Both sequences use the same alphabet, that is both are nucleotide sequences or both are amino acid sequences. • 3 — *Seq* contains nucleotides representing codons and *GappedSeq* contains amino acids (default).

Output Arguments

NewSeq	Sequence with gaps inserted, represented by a string specifying a nucleotide or amino acid sequence.

Description

NewSeq = seqinsertgaps(*Seq*, *Positions*) inserts gaps in the sequence *Seq* before the positions specified by the integers in the vector *Positions*.

NewSeq = seqinsertgaps(*Seq*, *GappedSeq*) finds the gap positions in the sequence *GappedSeq*, then inserts gaps in the corresponding positions in the sequence *Seq*.

NewSeq = seqinsertgaps(*Seq*, *GappedSeq*, *Relationship*) specifies the relationship between *Seq* and *GappedSeq*. Enter 1 for *Relationship* when both sequences use the same alphabet, that is both are nucleotide sequences or both are amino acid sequences. Enter 3 for *Relationship* when *Seq* contains nucleotides representing codons and *GappedSeq* contains amino acids. Default is 3.

Examples

1　Retrieve two nucleotide sequences from the GenBank database for the neuraminidase (NA) protein of two strains of the Influenza A virus (H5N1).

```
hk01 = getgenbank('AF509094');
vt04 = getgenbank('DQ094287');
```

2　Extract the coding region from the two nucleotide sequences.

```
hk01_cds = featuresparse(hk01,'feature','CDS','Sequence',true);
vt04_cds = featuresparse(vt04,'feature','CDS','Sequence',true);
```

3　Align the amino acids sequences converted from the nucleotide sequences.

```
[sc,al]=nwalign(nt2aa(hk01_cds),nt2aa(vt04_cds),'extendgap',1);
```

4　Use the seqinsertgaps function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

```
hk01_aligned = seqinsertgaps(hk01_cds,al(1,:))
vt04_aligned = seqinsertgaps(vt04_cds,al(3,:))
```

5　Once you have code aligned the two sequences, you can use them as input to other functions such as dnds, which calculates the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences. By setting Verbose to true, you can also display the codons considered in the computations and their amino acid translations.

```
[dn,ds] = dnds(hk01_aligned,vt04_aligned,'verbose',true)
```

See Also

dnds | dndsml | int2aa | int2nt

seqlinkage

Construct phylogenetic tree from pairwise distances

Syntax

```
PhyloTree = seqlinkage(Distances)
PhyloTree = seqlinkage(Distances, Method)
PhyloTree = seqlinkage(Distances, Method, Names)
```

Arguments

Distances	Matrix or vector of pairwise distances, such as returned by the `seqpdist` function.
Method	String that specifies a distance method. Choices are: • `'single'` • `'complete'` • `'average'` (default) • `'weighted'` • `'centroid'` • `'median'`
Names	Specifies alternative labels for leaf nodes. Choices are: • Vector of structures, each with a `Header` or `Name` field • Cell array of strings The elements must be unique. The number of elements must comply with the number of samples used to generate the pairwise distances in *Dist*.

Description

PhyloTree = seqlinkage(*Distances*) returns a phylogenetic tree object from the pairwise distances, *Distances*, between the species or products. *Distances* is a matrix or vector of pairwise distances, such as returned by the seqpdist function.

PhyloTree = seqlinkage(*Distances*, *Method*) creates a phylogenetic tree object using a specified patristic distance method. The available methods are:

'single'	Nearest distance (single linkage method)
'complete'	Furthest distance (complete linkage method)
'average' (default)	Unweighted Pair Group Method Average (UPGMA, group average).
'weighted'	Weighted Pair Group Method Average (WPGMA)
'centroid'	Unweighted Pair Group Method Centroid (UPGMC)
'median'	Weighted Pair Group Method Centroid (WPGMC)

PhyloTree = seqlinkage(*Distances*, *Method*, *Names*) passes a list of unique names to label the leaf nodes (for example, species or products) in a phylogenetic tree object.

Examples

Build Phylogenetic Tree from Pairwise Distances

Build a phylogenetic tree from pairwise distances, specifying both a distance-computing method and leaf names.

Create an array of structures representing a multiple alignment of amino acids:

```
seqs = fastaread('pf00002.fa');
```

Measure the Jukes-Cantor pairwise distances between sequences:

```
distances = seqpdist(seqs,'method','jukes-cantor','indels','pair');
```

You will use the output argument distances, a vector containing biological distances between each pair of sequences, as an input argument to seqlinkage.

Build the phylogenetic tree for the multiple sequence alignment from pairwise distances. Specify the method to compute the distances of the new nodes to all other nodes. Provide leaf names:

```
phylotree = seqlinkage(distances,'single',seqs)
```

```
Phylogenetic tree object with 32 leaves (31 branches)
```

View the phylogenetic tree:

```
view(phylotree)
```
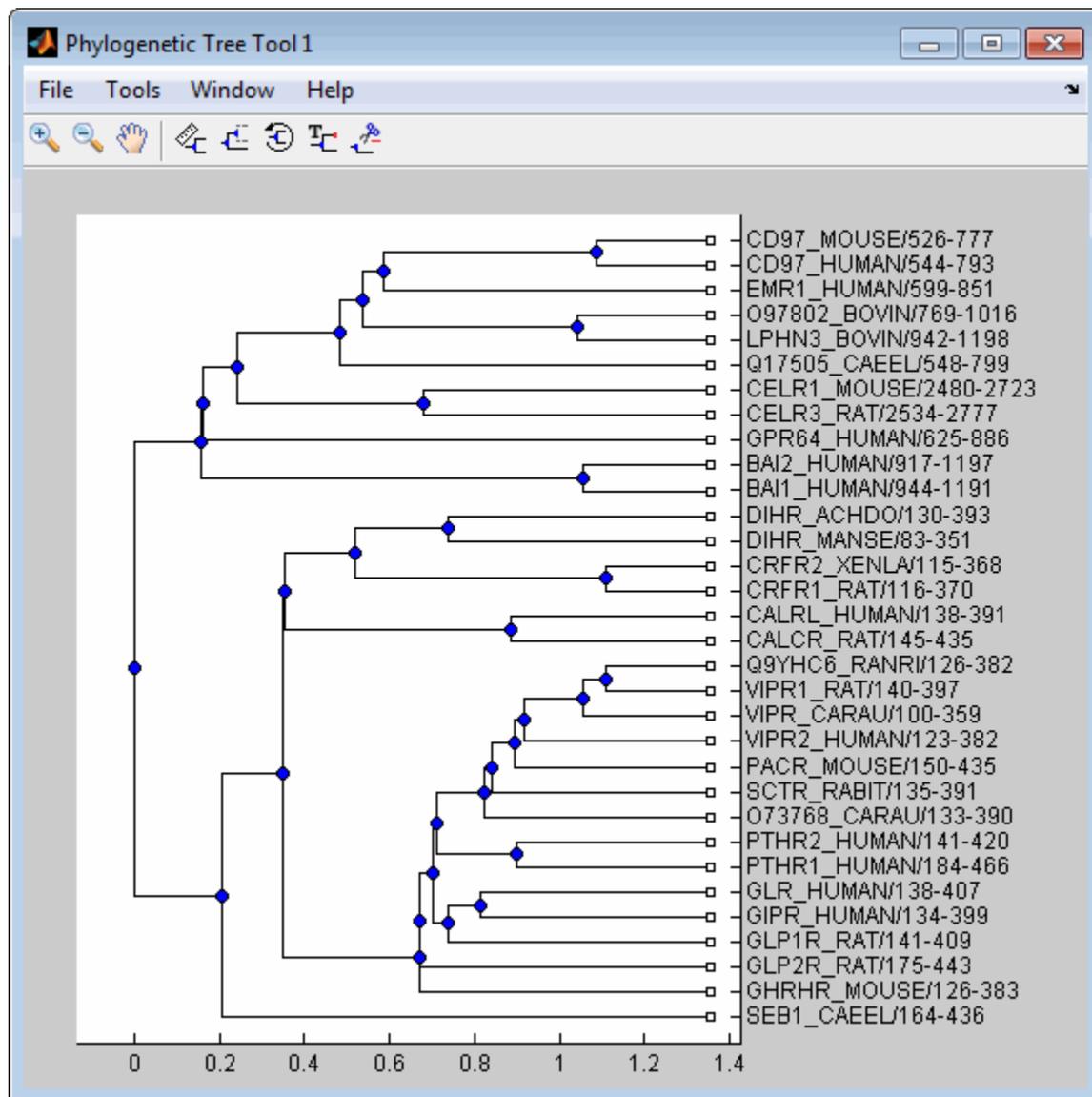
See Also

phytree | cluster | view | phytreewrite | seqpdist | seqneighjoin | plot

seqlogo

Display sequence logo for nucleotide or amino acid sequences

Syntax

```
seqlogo(Seqs)
seqlogo(Profile)
WgtMatrix = seqlogo(...)
[WgtMatrix, Handle] = seqlogo(...)
seqlogo(..., 'Displaylogo', DisplaylogoValue, ...)
seqlogo(..., 'Alphabet', AlphabetValue, ...)
seqlogo(..., 'Startat', StartatValue, ...)
seqlogo(..., 'Endat', EndatValue, ...)
seqlogo(..., 'SSCorrection', SSCorrectionValue, ...)
```

Input Arguments

Seqs	Set of pairwise or multiply aligned nucleotide or amino acid sequences, represented by any of the following:
	• Character array
	• Cell array of strings
	• Array of structures containing a `Sequence` field
Profile	Sequence profile distribution matrix with the frequency of nucleotides or amino acids for every column in the multiple alignment, such as returned by the `seqprofile` function.
	The size of the frequency distribution matrix is:
	• For nucleotides — `[4 x sequence length]`
	• For amino acids — `[20 x sequence length]`
	If gaps were included, *Profile* may have 5 rows (for nucleotides) or 21 rows (for amino acids), but `seqlogo` ignores gaps.

DisplaylogoValue	Controls the display of a sequence logo. Choices are `true` (default) or `false`.
AlphabetValue	String specifying the type of sequence (nucleotide or amino acid). Choices are `'NT'` (default) or `'AA'`.
StartatValue	Positive integer that specifies the starting position for the sequences in *Seqs*. Default starting position is `1`.
EndatValue	Positive integer that specifies the ending position for the sequences in *Seqs*. Default ending position is the maximum length of the sequences in *Seqs*.
SSCorrectionValue	Controls the use of small sample correction in the estimation of the number of bits. Choices are `true` (default) or `false`.

Output Arguments

WgtMatrix	Cell array containing the symbol list in *Seqs* or *Profile* and the weight matrix used to graphically display the sequence logo.
Handle	Handle to the sequence logo figure.

Description

seqlogo(*Seqs*) displays a sequence logo for *Seqs*, a set of aligned sequences. The logo graphically displays the sequence conservation at a particular position in the alignment of sequences, measured in bits. The maximum sequence conservation per site is `log2(4)` bits for nucleotide sequences and `log2(20)` bits for amino acid sequences. If the sequence conservation value is zero or negative, no logo is displayed in that position.

seqlogo(*Profile*) displays a sequence logo for *Profile*, a sequence profile distribution matrix with the frequency of nucleotides or amino acids for every column in the multiple alignment, such as returned by the seqprofile function.

Color Code for Nucleotides

Nucleotide	Color
A	Green

Nucleotide	Color
C	Blue
G	Yellow
T, U	Red
Other	Purple

Color Code for Amino Acids

Amino Acid	Chemical Property	Color
G S T Y C Q N	Polar	Green
A V L I P W F M	Hydrophobic	Orange
D E	Acidic	Red
K R H	Basic	Blue
Other	—	Tan

WgtMatrix = seqlogo(...) returns a cell array of unique symbols in the sequence *Seqs* or *Profile*, and the information weight matrix used to graphically display the logo.

[*WgtMatrix*, *Handle*] = seqlogo(...) returns a handle to the sequence logo figure.

seqlogo(*Seqs*, ...'*PropertyName*', *PropertyValue*, ...) calls seqpdist with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

seqlogo(..., 'Displaylogo', *DisplaylogoValue*, ...) controls the display of a sequence logo. Choices are true (default) or false.

seqlogo(..., 'Alphabet', *AlphabetValue*, ...) specifies the type of sequence (nucleotide or amino acid). Choices are 'NT' (default) or 'AA'.

Note: If you provide amino acid sequences to seqlogo, you must set Alphabet to 'AA'.

---

seqlogo(..., 'Startat', *StartatValue*, ...) specifies the starting position for the sequences in *Seqs*. Default starting position is 1.

seqlogo(..., 'Endat', *EndatValue*, ...) specifies the ending position for the sequences in *Seqs*. Default ending position is the maximum length of the sequences in *Seqs*.

seqlogo(..., 'SSCorrection', *SSCorrectionValue*, ...) controls the use of small sample correction in the estimation of the number of bits. Choices are `true` (default) or `false`.

Note: A simple calculation of bits tends to overestimate the conservation at a particular location. To compensate for this overestimation, when `SSCorrection` is set to `true`, a rough estimate is applied as an approximate correction. This correction works better when the number of sequences is greater than 50.

Examples

Display a Sequence Logo for Aligned Nucleotide Sequences

This example shows how to display a sequence logo for a set of aligned nucleotide sequences.

Create a series of aligned nucleotide sequences.

```
S = {'ATTATAGCAAACTA',...
     'AACATGCCAAAGTA',...
     'ATCATGCAAAAGGA'}
```

```
S =

    'ATTATAGCAAACTA'    'AACATGCCAAAGTA'    'ATCATGCAAAAGGA'
```

Display the sequence logo.

```
seqlogo(S)
```
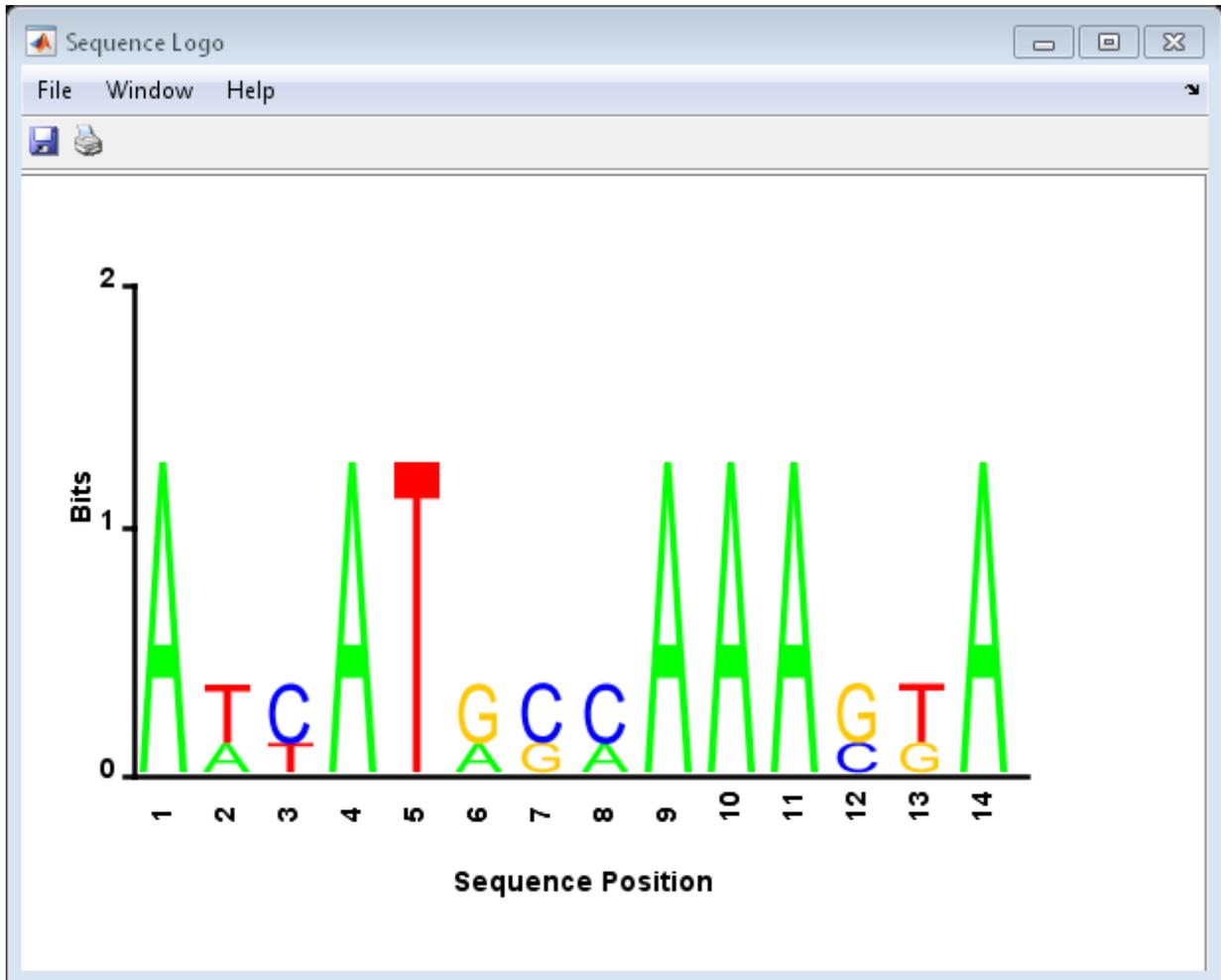
Display a Sequence Logo for Aligned Amino Acid Sequences

This example shows how to display a sequence logo for a set of aligned amino acid sequences.

Create a series of aligned amino acid sequences.

```
S2 = {'LSGGQRQRVAIARALAL',...
      'LSGGEKQRVAIARALMN',...
```

```
          'LSGGQIQRVLLARALAA',...
          'LSGGERRRLEIACVLAL',...
          'FSGGEKKKNELWQMLAL',...
          'LSGGERRRLEIACVLAL'};
```

Display the sequence logo, specifying an amino acid sequence and limiting the logo to sequence positions 2 through 10.

```
seqlogo(S2, 'alphabet', 'aa', 'startAt', 2, 'endAt', 10)
```
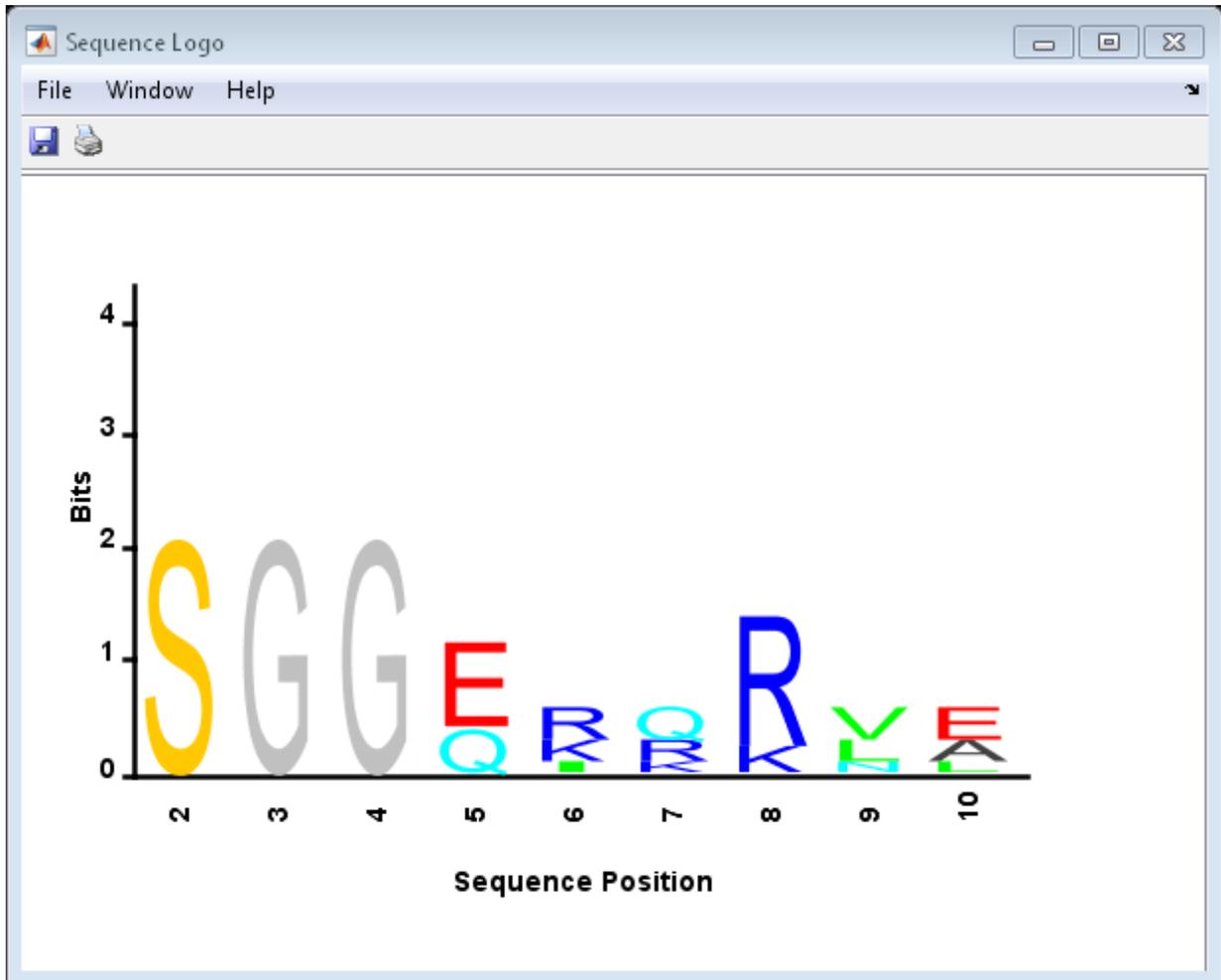
References

[1] Schneider, T.D., and Stephens, R.M. (1990). Sequence Logos: A new way to display consensus sequences. Nucleic Acids Research *18*, 6097–6100.

See Also
seqconsensus | seqdisp | seqprofile

seqmatch

Find matches for every string in library

Syntax

```
Index = seqmatch(Strings, Library)
```

Description

`Index = seqmatch(Strings, Library)` looks through the elements of `Library` to find strings that begin with every string in `Strings`. `Index` contains the index to the first occurrence for every string in the query. `Strings` and `Library` must be cell arrays of strings.

Examples

```
lib = {'VIPS_HUMAN', 'SCCR_RABIT', 'CALR_PIG' ,'VIPR_RAT', 'PACR_MOUSE'};
query = {'CALR','VIP'};
h = seqmatch(query,lib);
lib(h)

ans =

    'CALR_PIG'    'VIPS_HUMAN'
```

See Also
regexp | strncmp

seqneighjoin

Construct phylogenetic tree using neighbor-joining method

Syntax

```
PhyloTree = seqneighjoin(Distances)
PhyloTree = seqneighjoin(Distances, Method)
PhyloTree = seqneighjoin(Distances, Method, Names)
PhyloTree = seqneighjoin(..., 'Reroot', RerootValue)
```

Input Arguments

Distances	Matrix or vector containing biological distances between pairs of sequences, such as returned by the `seqpdist` function.
Method	String specifying a method to compute the distances between nodes. Choices are `'equivar'` (default) or `'firstorder'`.
Names	Either of the following:

- Vector of structures with the fields `Header` and `Name`
- Cell array of strings

The number of elements must equal the number of samples used to generate the pairwise distances in *Distances*.

Description

PhyloTree = seqneighjoin(*Distances*) computes *PhyloTree*, a phylogenetic tree object, from *Distances*, pairwise distances between the species or products, using the neighbor-joining method.

PhyloTree = seqneighjoin(*Distances*, *Method*) specifies *Method*, a method to compute the distances of the new nodes to all other nodes at every iteration. The general

expression to calculate the distances between the new node, n, after joining i and j and all other nodes (k), is given by

```
D(n,k) =  a*D(i,k) + (1-a)*D(j,k) - a*D(n,i) - (1-a)*D(n,j)
```

This expression is guaranteed to find the correct tree with additive data (minimum variance reduction).

Choices for *Method* are:

Method	Description
equivar (default)	Assumes equal variance and independence of evolutionary distance estimates (a = 1/2), such as in the original neighbor-joining algorithm by Saitou and Nei, JMBE (1987) or as in Studier and Keppler, JMBE (1988).
firstorder	Assumes a first-order model of the variances and covariances of evolutionary distance estimates, with 'a' being adjusted at every iteration to a value between 0 and 1, such as in Gascuel, JMBE (1997).

PhyloTree = seqneighjoin(*Distances*, *Method*, *Names*) passes *Names*, a list of names (such as species or products), to label the leaf nodes in the phylogenetic tree object.

PhyloTree = seqneighjoin(..., 'Reroot', *RerootValue*) specifies whether to reroot *PhyloTree*. Choices are true (default) or false. When *RerootValue* is false, seqneighjoin excludes rerooting the resulting tree, which is useful for observing the original linkage order followed by the algorithm. By default seqneighjoin reroots the resulting tree using the midpoint method.

Examples

Build Phylogenetic Tree using Neighbor Joining Method

Build a phylogenetic tree using the neighbor joining method and specifying both a distance-computing method and leaf names.

Create an array of structures representing a multiple alignment of amino acids:

```
seqs = fastaread('pf00002.fa');
```

Measure the Jukes-Cantor pairwise distances between sequences:

```
distances = seqpdist(seqs,'method','jukes-cantor','indels','pair');
```

You will use the output argument `distances`, a vector containing biological distances between each pair of sequences, as an input argument to `seqneighjoin`.

Build the phylogenetic tree for the multiple sequence alignment using the neighbor-joining algorithm. Specify the method to compute the distances of the new nodes to all other nodes. Provide leaf names:

```
phylotree = seqneighjoin(distances,'equivar',seqs)
```

```
Phylogenetic tree object with 32 leaves (31 branches)
```

View the phylogenetic tree:

```
view(phylotree)
```
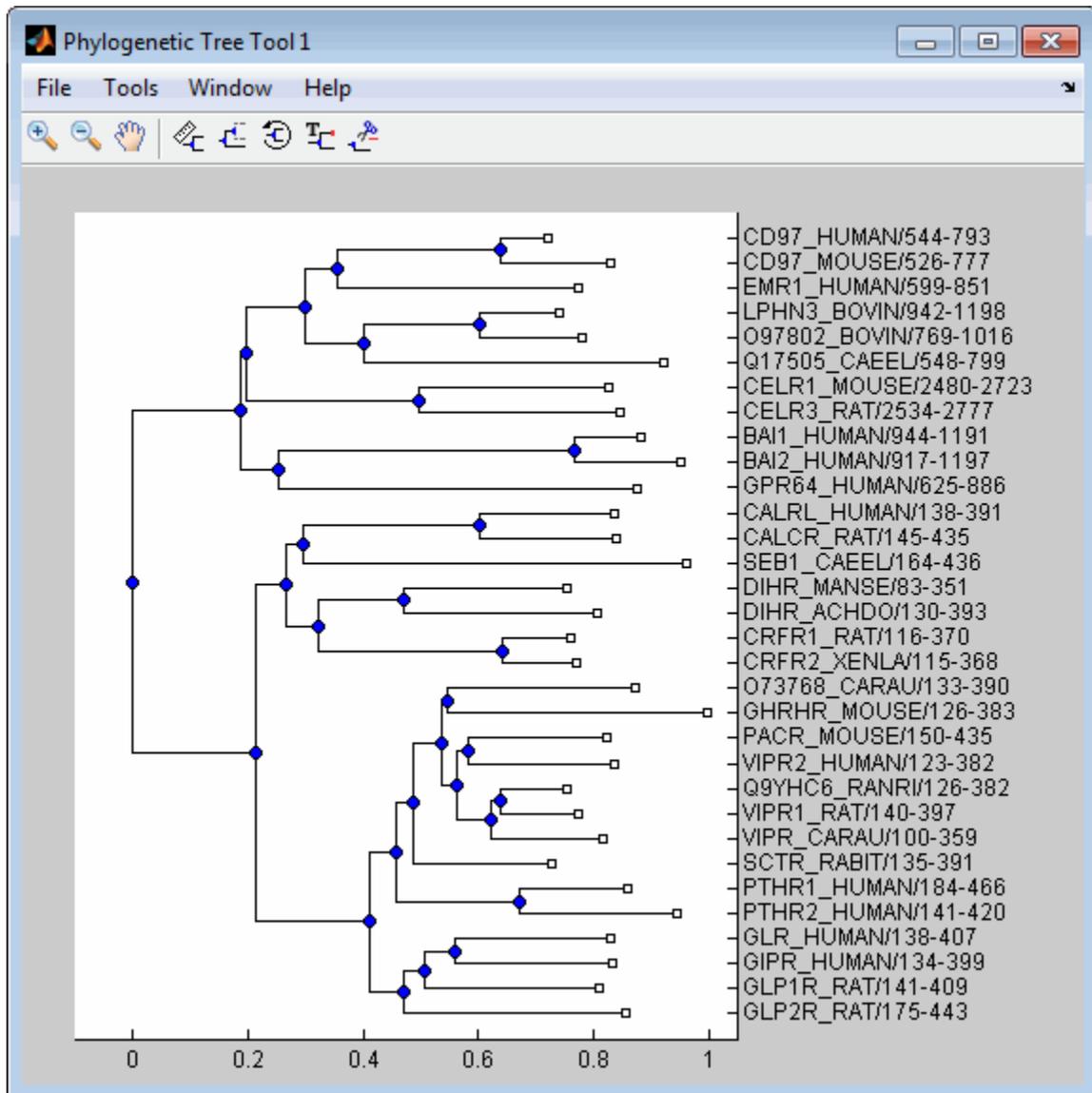
References [1] Saitou, N., and Nei, M. (1987). The neighbor-joining method: A new method for reconstructing phylogenetic trees. Molecular Biology and Evolution *4(4)*, 406–425.

[2] Gascuel, O. (1997). BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. Molecular Biology and Evolution *14* 685–695.

[3] Studier, J.A., Keppler, K.J. (1988). A note on the neighbor-joining algorithm of Saitou and Nei. Molecular Biology and Evolution *5(6)* 729–731.

See Also

```
multialign | cluster | reroot | phytree | seqlinkage | seqpdist | plot |
view
```

seqpdist

Calculate pairwise distance between sequences

Syntax

```
D = seqpdist(Seqs)
D = seqpdist(Seqs, ...'PropertyName', PropertyValue, ...)
D = seqpdist(Seqs, ...'Method', MethodValue, ...)
D = seqpdist(Seqs, ...'Indels', IndelsValue, ...)
D = seqpdist(Seqs, ...'OptArgs', OptArgsValue, ...)
D = seqpdist(Seqs, ...'PairwiseAlignment',
PairwiseAlignmentValue, ...)
D = seqpdist(Seqs, ...'UseParallel', UseParallelValue, ...)
D = seqpdist(Seqs, ...'SquareForm', SquareFormValue ...)
D = seqpdist(Seqs, ...'Alphabet', AlphabetValue, ...)
D = seqpdist(Seqs, ...'ScoringMatrix', ScoringMatrixValue, ...)
D = seqpdist(Seqs, ...'Scale', ScaleValue, ...)
D = seqpdist(Seqs, ...'GapOpen', GapOpenValue, ...)
D = seqpdist(Seqs, ...'ExtendGap', ExtendGapValue, ...)
```

Input Arguments

Seqs	Any of the following: • Cell array containing nucleotide or amino acid sequences • Vector of structures containing a `Sequence` field • Matrix of characters, in which each row corresponds to a nucleotide or amino acid sequence
MethodValue	String that specifies the method to calculate pairwise distances. Default is `'Jukes-Cantor'`.
IndelsValue	String that specifies how to treat sites with gaps. Default is `'score'`.

OptArgsValue	String or cell array that specifies one or more input arguments required or accepted by the distance method specified by the `Method` property.
PairwiseAlignmentValue	Controls the global pairwise alignment of input sequences (using the `nwalign` function), while ignoring the multiple alignment of the input sequences (if any). Choices are `true` or `false`. Default is: • `true` — When all input sequences do not have the same length. • `false` — When all input sequences have the same length. **Tip** If your input sequences are the same length, `seqpdist` assumes they are aligned. If they are not aligned, do one of the following: • Align the sequences before passing them to `seqpdist`, for example, using the `multialign` function. • Set `PairwiseAlignment` to `true` when using `seqpdist`.
UseParallelValue	Controls the calculation of the pairwise distances using `parfor`-loops. When `true`, and Parallel Computing Toolbox is installed and a `parpool` is open, computation occurs in parallel. If there are no open `parpool`, but automatic creation is enabled in the Parallel Preferences, the default pool will be automatically open and computation occurs in parallel. If Parallel Computing Toolbox is installed, but there are no open `parpool` and automatic creation is disabled, then computation uses `parfor`-loops in serial mode. If Parallel Computing Toolbox is not installed, then computation uses `parfor`-loops in serial mode. Default is `false`, which uses for-loops in serial mode.

SquareFormValue	Controls the conversion of the output into a square matrix. Choices are `true` or `false` (default).
AlphabetValue	String specifying the type of sequence (nucleotide or amino acid). Choices are `'NT'` or `'AA'` (default).

ScoringMatrixValue	Either of the following:
	• String specifying the scoring matrix to use for the alignment. Choices for amino acid sequences are: 　• `'BLOSUM62'` 　• `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'` 　• `'BLOSUM100'` 　• `'PAM10'` increasing by 10 up to `'PAM500'` 　• `'DAYHOFF'` 　• `'GONNET'` Default is: 　• `'BLOSUM50'` — When *AlphabetValue* equals `'AA'` 　• `'NUC44'` — When *AlphabetValue* equals `'NT'` --- **Note:** The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the `'Scale'` property to specify an additional scale factor to convert the output score from bits to another unit. • Matrix representing the scoring matrix to use for the alignment, such as returned by the `blosum`, `pam`, `dayhoff`, `gonnet`, or `nuc44` function. --- **Note:** If you use a scoring matrix that you created or was created by one of the above functions, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix. You can use the `'Scale'` property

	to specify a scale factor to convert the output score to another unit.
	Note: If you need to compile `seqpdist` into a stand-alone application or software component using MATLAB Compiler, use a matrix instead of a string for *ScoringMatrixValue*.
ScaleValue	Positive value that specifies the scale factor used to return the score in arbitrary units. If the scoring matrix information also provides a scale factor, then both are used.
GapOpenValue	Positive integer that specifies the penalty for opening a gap in the alignment. Default is 8.
ExtendedGapValue	Positive integer that specifies the penalty for extending a gap. Default is equal to *GapOpenValue*.

Output Arguments

D	Vector that contains biological distances between each pair of sequences stored in the M elements of *Seqs*.

Description

D = seqpdist(*Seqs*) returns *D*, a vector containing biological distances between each pair of sequences stored in the M sequences of *Seqs*, a cell array of sequences, a vector of structures, or a matrix or sequences.

D is a 1-by-(M*(M-1)/2) row vector corresponding to the M*(M-1)/2 pairs of sequences in *Seqs*. The output *D* is arranged in the order ((2,1),(3,1),..., (M,1), (3,2),...(M,2),...(M,M-1)). This is the lower-left triangle of the full M-by-M distance matrix. To get the distance between the *I*th and the *J*th sequences for I > J, use the formula D((J-1)*(M-J/2)+I-J).

D = seqpdist(*Seqs*, ...'*PropertyName*', *PropertyValue*, ...) calls seqpdist with optional properties that use property name/property value pairs. Specify one or more properties in any order. Enclose each *PropertyName* in single quotation

marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

D = seqpdist(*Seqs*, ...'Method', *MethodValue*, ...) specifies a method to compute distances between each sequence pair. Choices are shown in the following tables.

Methods for Nucleotides and Amino Acids

Method	Description
p-distance	Proportion of sites at which the two sequences are different. p is close to 1 for poorly related sequences, and p is close to 0 for similar sequences. d = p
Jukes-Cantor (default)	Maximum likelihood estimate of the number of substitutions between two sequences. p is described with the method p-distance. For nucleotides: d = -3/4 log(1-p * 4/3) For amino acids: d = -19/20 log(1-p * 20/19)
alignment-score	Distance (d) between two sequences (1, 2) is computed from the pairwise alignment score between the two sequences (score12), and the pairwise alignment score between each sequence and itself (score11, score22) as follows: d = (1-score12/score11)* (1-score12/score22) This option does not imply that prealigned input sequences will be realigned, it only scores them. Use with care; this distance method does not comply with the ultrametric condition. In the rare case where the score between sequences is greater than the score when aligning a sequence with itself, then d = 0

Methods with No Scoring of Gaps (Nucleotides Only)

Method	Description
Tajima-Nei	Maximum likelihood estimate considering the background nucleotide frequencies. It can be computed from the input sequences or given by

Method	Description
	setting `OptArgs` to [gA gC gG gT]. gA, gC, gG, gT are scalar values for the nucleotide frequencies.
Kimura	Considers separately the transitional nucleotide substitution and the transversional nucleotide substitution.
Tamura	Considers separately the transitional nucleotide substitution, the transversional nucleotide substitution, and the GC content. GC content can be computed from the input sequences or given by setting `OptArgs` to the proportion of GC content (scalar value from 0 to 1).
Hasegawa	Considers separately the transitional nucleotide substitution, the transversional nucleotide substitution, and the background nucleotide frequencies. Background frequencies can be computed from the input sequences or given by setting the `OptArgs` property to [gA gC gG gT].
Nei-Tamura	Considers separately the transitional nucleotide substitution between purines, the transitional nucleotide substitution between pyrimidines, the transversional nucleotide substitution, and the background nucleotide frequencies. Background frequencies can be computed from the input sequences or given by setting the `OptArgs` property to [gA gC gG gT].

Methods with No Scoring of Gaps (Amino Acids Only)

Method	Description
Poisson	Assumes that the number of amino acid substitutions at each site has a Poisson distribution.
Gamma	Assumes that the number of amino acid substitutions at each site has a Gamma distribution with parameter `a`. Set `a` using the `OptArgs` property. Default is 2.

You can also specify a user-defined distance function using @, for example, @`distfun`. The distance function must have the form:

```
function D = distfun(S1, S2, OptArgsValue)
```

The `distfun` function takes the following arguments:

- *S1* , *S2* — Two sequences of the same length (nucleotide or amino acid).

• *OptArgsValue* — Optional problem-dependent arguments.

The distfun function returns a scalar that represents the distance between *S1* and *S2*.

D = seqpdist(*Seqs*, ...'Indels', *IndelsValue*, ...) specifies how to treat sites with gaps. Choices are:

• score (default) — Scores these sites either as a point mutation or with the alignment parameters, depending on the method selected.

• pairwise-del — For every pairwise comparison, it ignores the sites with gaps.

• complete-del — Ignores all the columns in the multiple alignment that contain a gap. This option is available only if you provided a multiple alignment as the input *Seqs*.

D = seqpdist(*Seqs*, ...'OptArgs', *OptArgsValue*, ...) passes one or more arguments required or accepted by the distance method specified by the Method property. Use a string or cell array to pass one or more input arguments. For example, provide the nucleotide frequencies for the Tajima-Nei distance method, instead of computing them from the input sequences.

D = seqpdist(*Seqs*, ...'PairwiseAlignment', *PairwiseAlignmentValue*, ...) controls the global pairwise alignment of input sequences (using the nwalign function), while ignoring the multiple alignment of the input sequences (if any). Default is:

• true — When all input sequences do not have the same length.

• false — When all input sequences have the same length.

Tip If your input sequences have the same length, seqpdist assumes they are aligned. If they are not aligned, do one of the following:

• Align the sequences before passing them to seqpdist, for example, using the multialign function.

• Set PairwiseAlignment to true when using seqpdist.

D = seqpdist(*Seqs*, ...'UseParallel', *UseParallelValue*, ...) specifies whether to use parfor-loops when calculating the pairwise distances. When true, and Parallel Computing Toolbox is installed and a parpool is open, computation occurs in

parallel. If there are no open `parpool`, but automatic creation is enabled in the Parallel Preferences, the default pool will be automatically open and computation occurs in parallel. If Parallel Computing Toolbox is installed, but there are no open `parpool` and automatic creation is disabled, then computation uses `parfor`-loops in serial mode. If Parallel Computing Toolbox is not installed, then computation uses `parfor`-loops in serial mode. Default is `false`, which uses for-loops in serial mode.

D = seqpdist(*Seqs*, ...'SquareForm', *SquareFormValue* ...) controls the conversion of the output into a square matrix such that *D*(*I*,*J*) denotes the distance between the *I*th and *J*th sequences. The square matrix is symmetric and has a zero diagonal. Choices are `true` or `false` (default). Setting `Squareform` to `true` is the same as using the `squareform` function in Statistics and Machine Learning Toolbox .

D = seqpdist(*Seqs*, ...'Alphabet', *AlphabetValue*, ...) specifies the type of sequence (nucleotide or amino acid). Choices are `'NT'` or `'AA'` (default).

The remaining input properties are available when the `Method` property equals `'alignment-score'` or the `PairwiseAlignment` property equals `true`.

D = seqpdist(*Seqs*, ...'ScoringMatrix', *ScoringMatrixValue*, ...) specifies the scoring matrix to use for the global pairwise alignment. Default is:

- `'NUC44'` — When *AlphabetValue* equals `'NT'`.
- `'BLOSUM50'` — When *AlphabetValue* equals `'AA'`.

D = seqpdist(*Seqs*, ...'Scale', *ScaleValue*, ...) specifies the scale factor used to return the score in arbitrary units. Choices are any positive value. If the scoring matrix information also provides a scale factor, then both are used.

D = seqpdist(*Seqs*, ...'GapOpen', *GapOpenValue*, ...) specifies the penalty for opening a gap in the alignment. Choices are any positive integer. Default is 8.

D = seqpdist(*Seqs*, ...'ExtendGap', *ExtendGapValue*, ...) specifies the penalty for extending a gap in the alignment. Choices are any positive integer. Default is equal to *GapOpenValue*.

Examples

1 Read amino acid alignment data into a MATLAB structure.

```
        seqs = fastaread('pf00002.fa');
```

2 For every possible pair of sequences in the multiple alignment, ignore sites with gaps and score with the scoring matrix PAM250.

```
dist = seqpdist(seqs,'Method','alignment-score',...
                'Indels','pairwise-delete',...
                'ScoringMatrix','pam250');
```

3 Force the realignment of each sequence pair ignoring the provided multiple alignment.

```
dist = seqpdist(seqs,'Method','alignment-score',...
                'Indels','pairwise-delete',...
                'ScoringMatrix','pam250',...
                'PairwiseAlignment',true);
```

4 Measure the Jukes-Cantor pairwise distances after realigning each sequence pair, counting the gaps as point mutations.

```
dist = seqpdist(seqs,'Method','jukes-cantor',...
                'Indels','score',...
                'Scoringmatrix','pam250',...
                'PairwiseAlignment',true);
```

More About

• phytree object

See Also

fastaread | pdist | dnds | dndsml | multialign | nwalign | phytree |
seqlinkage

seqprofile

Calculate sequence profile from set of multiply aligned sequences

Syntax

```
Profile = seqprofile(Seqs)
[Profile, Symbols] = seqprofile(Seqs)
seqprofile(Seqs, ...'Alphabet', AlphabetValue, ...)
seqprofile(Seqs, ...'Counts', CountsValue, ...)
seqprofile(Seqs, ...'Gaps', GapsValue, ...)
seqprofile(Seqs, ...'Ambiguous', AmbiguousValue, ...)
seqprofile(Seqs, ...'Limits', LimitsValue, ...)
```

Arguments

Seqs	Set of multiply aligned sequences represented by any of the following:. • Array of strings • Cell array of strings • Array of structures containing the field `Sequence`
AlphabetValue	String specifying the sequence alphabet. Choices are: • `'NT'` — Nucleotides • `'AA'` — Amino acids (default) • `'none'` — No alphabet When `Alphabet` is `'none'`, the symbol list is based on the observed symbols. Each character can be any symbol, except for a hyphen (-) and a period (.), which are reserved for gaps.
CountsValue	Controls returning frequency (ratio of counts/total counts) or counts. Choices are `true` (counts) or `false` (frequency). Default is `false`.

GapsValue	String that controls the counting of gaps in a sequence. Choices are: • `'all'` — Counts all gaps • `'noflanks'` — Counts all gaps except those at the flanks of every sequence • `'none'` — Default. Counts no gaps.
AmbiguousValue	Controls counting ambiguous symbols. Enter `'Count'` to add partial counts to the standard symbols.
LimitsValue	Specifies whether to use part of the sequence. Enter a `[1x2]` vector with the first position and the last position to include in the profile. Default is `[1,SeqLength]`.

Description

Profile = seqprofile(*Seqs*) returns *Profile*, a matrix of size `[20 (or 4) x SequenceLength]` with the frequency of amino acids (or nucleotides) for every column in the multiple alignment. The order of the rows is given by

- 4 nucleotides — A C G T/U
- 20 amino acids — A R N D C Q E G H I L K M F P S T W Y V

[*Profile*, *Symbols*] = seqprofile(*Seqs*) returns *Symbols*, a unique symbol list where every symbol in the list corresponds to a row in *Profile*, the profile.

seqprofile(*Seqs*, ...'*PropertyName*', *PropertyValue*, ...) calls seqprofile with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

seqprofile(*Seqs*, ...'Alphabet', *AlphabetValue*, ...) selects a nucleotide alphabet, amino acid alphabet, or no alphabet.

seqprofile(*Seqs*, ...'Counts', *CountsValue*, ...) when Counts is `true`, returns the counts instead of the frequency.

seqprofile(*Seqs*, ...'Gaps', *GapsValue*, ...) appends a row to the bottom of a profile (`Profile`) with the count for gaps.

seqprofile(*Seqs*, ...'Ambiguous', *AmbiguousValue*, ...) when Ambiguous is 'count', counts the ambiguous amino acid symbols (B Z X) and nucleotide symbols (R Y K M S W B D H V N) with the standard symbols. For example, the amino acid X adds a 1/20 count to every row while the amino acid B counts as 1/2 at the D and N rows.

seqprofile(*Seqs*, ...'Limits', *LimitsValue*, ...) specifies the start and end positions for the profile relative to the indices of the multiple alignment.

Examples

Calculate Sequence Profile

Calculate the sequence profile from set of multiply aligned sequences.

Create an array of structures representing a multiple alignment of amino acids:

```
seqs = fastaread('pf00002.fa');
```

Return the sequence profile and symbol list from the set of multiply aligned sequences:

```
[Profile1,Symbols1] = seqprofile(seqs);
```

Calculate Sequence Profile from Part of Alignment, Counting All Gaps

Calculate the sequence profile from set of multiply aligned sequences. Specify only part of the alignment and to count all gaps.

Create an array of structures representing a multiple alignment of amino acids:

```
seqs = fastaread('pf00002.fa');
```

Return the sequence profile and symbol list from position 50 through 55 of the set of multiply aligned sequences, counting all gaps:

```
[Profile2,Symbols2] = seqprofile(seqs,'limits',[50 55],'gaps','all')

Profile2 =

    0.0313    0.0313    0.1563    0.4375    0.1250    0.2188
         0         0    0.3750         0         0         0
         0         0    0.0938    0.1563         0         0
```

```
       0          0          0     0.0313          0          0
       0     0.0625          0          0     0.0313          0
       0          0          0     0.0313          0          0
       0          0          0     0.1250          0          0
  0.0313          0     0.0625          0          0          0
       0          0          0          0          0          0
  0.4688     0.0625          0          0     0.3125     0.1563
  0.1250     0.6250     0.0313          0     0.2188     0.1875
       0          0     0.1250     0.0313          0          0
  0.1250     0.0625          0          0          0     0.0313
  0.1563     0.0313          0          0     0.0313          0
       0          0          0          0          0          0
       0     0.0313     0.1250     0.1250     0.0625     0.2500
       0          0          0          0     0.1563     0.0938
       0          0          0          0          0          0
       0          0          0          0          0          0
  0.0625     0.0938     0.0313     0.0625     0.0625     0.0625
       0          0          0          0          0          0


Symbols2 =

ARNDCQEGHILKMFPSTWYV-
```

See Also
fastaread | multialignread | multialignwrite | seqconsensus | seqdisp | seqlogo

seqrcomplement

Calculate reverse complementary strand of nucleotide sequence

Syntax

SeqRC = seqrcomplement(*SeqNT*)

Arguments

SeqNT	Nucleotide sequence specified by any of the following: • Character string with the characters A, C, G, T, U, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N. • Row vector of integers from the table Mapping Nucleotide Integers to Letter Codes. • MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by emblread, fastaread, fastqread, genbankread, getembl, or getgenbank.

Description

SeqRC = seqrcomplement(*SeqNT*) calculates the reverse complementary strand of a DNA or RNA nucleotide sequence. The return sequence, *SeqRC*, reads from 3' --> 5' and is in the same format as *SeqNT*. For example, if *SeqNT* is a vector of integers, then so is *SeqRC*.

Nucleotide in *SeqNT*	Converts to This Nucleotide in *SeqRC*
A	T or U
C	G
G	C
T or U	A

Examples

Return the reverse complement of a DNA nucleotide sequence.

```
s = 'ATCG'
seqrcomplement(s)

ans =
CGAT
```

See Also
codoncount | palindromes | seqcomplement | seqreverse | seqviewer

seqreverse

Calculate reverse strand of nucleotide sequence

Syntax

SeqR = seqreverse(*SeqNT*)

Arguments

SeqNT	Nucleotide sequence specified by any of the following:
	• Character string with the characters A, C, G, T, U, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N.
	• Row vector of integers from the table Mapping Nucleotide Integers to Letter Codes.
	• MATLAB structure containing a Sequence field that contains a nucleotide sequence, such as returned by emblread, fastaread, fastqread, genbankread, getembl, or getgenbank.

Description

SeqR = seqreverse(*SeqNT*) calculates the reverse strand of a DNA or RNA nucleotide sequence. The return sequence, *SeqR*, reads from 3' --> 5' and is in the same format as *SeqNT*. For example, if *SeqNT* is a vector of integers, then so is *SeqR*.

Examples

Return the reverse strand of a DNA nucleotide sequence.

```
s = 'ATCG'
seqreverse(s)
ans =
GCTA
```

See Also
codoncount | palindromes | seqcomplement | seqrcomplement | seqviewer |
fliplr

seqshoworfs

Display open reading frames in sequence

Syntax

```
seqshoworfs(SeqNT)
seqshoworfs(SeqNT, ...'Frames', FramesValue, ...)
seqshoworfs(SeqNT, ...'GeneticCode', GeneticCodeValue, ...)
seqshoworfs(SeqNT, ...'MinimumLength', MinimumLengthValue, ...)
seqshoworfs(SeqNT, ...'AlternativeStartCodons', AlternativeStartCodonsValue, .
seqshoworfs(SeqNT, ...'Color', ColorValue, ...)
seqshoworfs(SeqNT, ...'Columns', ColumnsValue, ...)
```

Arguments

SeqNT	Nucleotide sequence. Enter either a character string with the characters A, T (U), G, C, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N, or a vector of integers. You can also enter a structure with the field Sequence.
FramesValue	Property to select the frame. Enter 1, 2, 3, -1, -2, -3, enter a vector with integers, or 'all'. The default value is the vector [1 2 3]. Frames -1, -2, and -3 correspond to the first, second, and third reading frames for the reverse complement.
GeneticCodeValue	Genetic code name. Enter a code number or a code name from the table Genetic Code.
MinimumLengthValue	Property to set the minimum number of codons in an ORF.
AlternativeStartCodonsValue	Property to control using alternative start codons. Enter either true or false. The default value is false.

ColorValue	Color to highlight the reading frame. Specify one of the following:
	• Three-element numeric vector of RGB values
	• String containing a predefined single-letter color code
	• String containing a predefined color name
	For example, to use cyan, enter [0 1 1], 'c', or 'cyan'. For more information on specifying colors, see ColorSpec.
	To specify different colors for the three reading frames, use a 1-by-3 cell array of color values. If you are displaying reverse complement reading frames, then use a 1-by-6 cell array of color values.
	The default color scheme is blue for the first reading frame, red for the second, and green for the third.
ColumnsValue	Property to specify the number of columns in the output.

Genetic Code

Code Number	Code Name
1	Standard
2	Vertebrate Mitochondrial
3	Yeast Mitochondrial
4	Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma
5	Invertebrate Mitochondrial
6	Ciliate, Dasycladacean, and Hexamita Nuclear
9	Echinoderm Mitochondrial
10	Euplotid Nuclear

Code Number	Code Name
11	`Bacterial` and `Plant Plastid`
12	`Alternative Yeast Nuclear`
13	`Ascidian Mitochondrial`
14	`Flatworm Mitochondrial`
15	`Blepharisma Nuclear`
16	`Chlorophycean Mitochondrial`
21	`Trematode Mitochondrial`
22	`Scenedesmus Obliquus Mitochondrial`
23	`Thraustochytrium Mitochondrial`

Description

seqshoworfs identifies and highlights all open reading frames using the standard or an alternative genetic code.

seqshoworfs(*SeqNT*) displays the sequence with all open reading frames highlighted, and it returns a structure of start and stop positions for each ORF in each reading frame. The standard genetic code is used with start codon `'AUG'` and stop codons `'UAA'`, `'UAG'`, and `'UGA'`.

seqshoworfs(*SeqNT*, ...`'PropertyName'`, *PropertyValue*, ...) calls seqshoworfs with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

seqshoworfs(*SeqNT*, ...'Frames', *FramesValue*, ...) specifies the reading frames to display. The default is to display the first, second, and third reading frames with ORFs highlighted in each frame.

seqshoworfs(*SeqNT*, ...'GeneticCode', *GeneticCodeValue*, ...) specifies the genetic code to use for finding open reading frames.

seqshoworfs(*SeqNT*, ...'MinimumLength', *MinimumLengthValue*, ...) sets the minimum number of codons for an ORF to be considered valid. The default value is 10.

seqshoworfs(*SeqNT*, ...'AlternativeStartCodons', *AlternativeStartCodonsValue*, .
uses alternative start codons if AlternativeStartCodons is set to true. For example, in the human mitochondrial genetic code, AUA and AUU are known to be alternative start codons. For more details on alternative start codons, see

http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=t#SG1

seqshoworfs(*SeqNT*, ...'Color', *ColorValue*, ...) specifies the color used to highlight the open reading frames in the output display. The default color scheme is blue for the first reading frame, red for the second, and green for the third.

seqshoworfs(*SeqNT*, ...'Columns', *ColumnsValue*, ...) specifies how many columns per line to use in the output. The default value is 64.

Examples

Display the open reading frames in a random nucleotide sequence.

```
s = randseq(200, 'alphabet', 'dna');
seqshoworfs(s);
```

```
+-----------------------------------------------------------------+
| Open Reading Frames                                  _ [] X      |
+-----------------------------------------------------------------+
| [print]                                                      »  |
+-----------------------------------------------------------------+
| Frame 1                                                         |
|                                                                 |
| 000001   TCGGTTGAACTCTATCACGCCTGGTCTTCGAAGTTAGCACATCGAGCGGGCAAT ATGTACATAT |
| 000065   TTACCTCTACAATGGATGCGCAAAAACATTCCCTCATCACAATTGAACTAAAGGGCGCGAGACG |
| 000129   TATTCCCCGGTTGCTGCTTGGGACCATAAAACCTCATTCACCGCGGAACCCGACTATGCGACTG |
| 000193   GACGGCCT                                               |
|                                                                 |
| Frame 2                                                         |
|                                                                 |
| 000001   TCGGTTGAACTCTATCACGCCTGGTCTTCGAAGTTAGCACATCGAGCGGGCAATATGTACATAT |
| 000065   TTACCTCTACAATGGATGCGCAAAAACATTCCCTCATCACAATTGAACTAAAGGGCGCGAGACG |
| 000129   TATTCCCCGGTTGCTGCTTGGGACCATAAAACCTCATTCACCGCGGAACCCGACTATGCGACTG |
| 000193   GACGGCCT                                               |
|                                                                 |
| Frame 3                                                         |
|                                                                 |
| 000001   TCGGTTGAACTCTATCACGCCTGGTCTTCGAAGTTAGCACATCGAGCGGGCAATATGTACATAT |
| 000065   TTACCTCTACAATGGATGCGCAAAAACATTCCCTCATCACAATTGAACTAAAGGGCGCGAGACG |
| 000129   TATTCCCCGGTTGCTGCTTGGGACCATAAAACCTCATTCACCGCGGAACCCGACTATGCGACTG |
| 000193   GACGGCCT                                               |
|                                                                 |
+-----------------------------------------------------------------+
```

Display the open reading frames in a GenBank sequence.

```
HLA_DQB1 = getgenbank('NM_002123');
seqshoworfs(HLA_DQB1.Sequence);
```

Open Reading Frames

```
Frame 1

000001   gcttgggcagtgaaatg

Frame 2

000001   gcttgggcagtgaaatg

Frame 3

000001   gcttgggcagtgaaatg
```
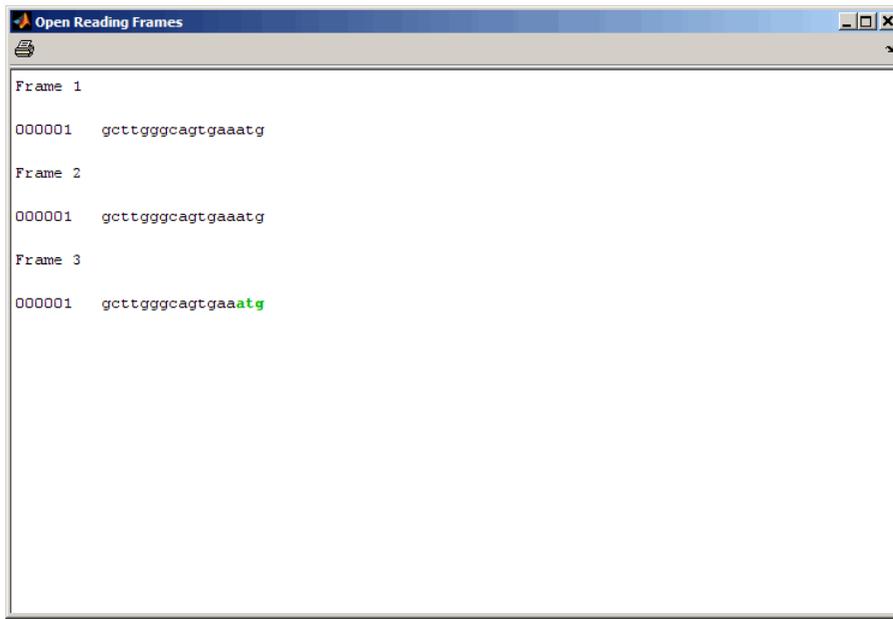
See Also

`codoncount` | `cpgisland` | `geneticcode` | `seqdisp` | `seqshowwords` | `seqviewer` | `seqwordcount` | `regexp`

seqshowwords

Graphically display words in sequence

Syntax

```
Struct = seqshowwords(Seq, Word)
seqshowwords(Seq, Word, ...'Color', ColorValue, ...)
seqshowwords(Seq, Word, ...'Columns', ColumnsValue, ...)
seqshowwords(Seq, Word, ...'Alphabet', AlphabetValue, ...)
```

Description

Struct = seqshowwords(*Seq*, *Word*) opens a separate window displaying a sequence with all occurrences of one or more words highlighted. It also returns a structure containing the start and stop positions for all occurrences of the words in the sequence.

seqshowwords(*Seq*, *Word*, ...'*PropertyName*', *PropertyValue*, ...) calls seqshowwords with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/ property value pairs are as follows:

seqshowwords(*Seq*, *Word*, ...'Color', *ColorValue*, ...) specifies the color to highlight the words in the output display of the sequence. Default is red.

seqshowwords(*Seq*, *Word*, ...'Columns', *ColumnsValue*, ...) specifies how many columns or characters per line in the output display of the sequence. Default is 64.

seqshowwords(*Seq*, *Word*, ...'Alphabet', *AlphabetValue*, ...) specifies the alphabet for the sequence and the word or words. Choices are 'AA' or 'NT' (default).

Input Arguments

Seq

Amino acid or nucleotide sequence specified by any of the following:

- Character string of letters representing amino acids or nucleotides, such as returned by `int2aa` or `int2nt`.
- MATLAB structure containing a `Sequence` field, such as returned by `fastaread`, `fastqread`, `emblread`, `getembl`, `genbankread`, `getgenbank`, `getgenpept`, `genpeptread`, `getpdb`, `pdbread`, or `sffread`.

Default:

Word

One or more short amino acid or nucleotide sequences specified by any of the following:

- Character string of letters
- Regular expression
- Cell array of strings or regular expressions

Note: If the search word or words contain amino acid or nucleotide symbols that represent multiple symbols, then `seqshowwords` shows all possible matches. For example, the symbol R represents either G or A (purines). If *Word* is `'ART'`, then `seqshowwords` shows occurrences of both `'AAT'` and `'AGT'`.

Tip If *Word* contains a repeating pattern, such as `'TATA'`, then `seqshowwords` does not highlight overlapping patterns of TA in the sequence. To highlight multiple repeats of TA in a sequence, use a regular expression, such as `'TA(TA)*TA'`, for *Word*. For more information, see "Examples" on page 1-1723.

Default:

ColorValue

Color to highlight all occurrences of one or more words in the sequence. Specify the color with one of the following:

- Three-element numeric vector of RGB values
- String containing a predefined single-letter color code
- String containing a predefined color name

For example, to use cyan, enter [0 1 1], 'c', or 'cyan'. For more information on specifying colors, see ColorSpec.

Default: Red, which is specified by [1 0 0], 'r', or 'red'

ColumnsValue

Positive integer specifying how many columns or characters per line in the output display of the sequence.

Default: 64

AlphabetValue

String specifying the type of sequences. Choices are 'AA' or 'NT' (default).

Default:

Output Arguments

Struct

MATLAB structure containing the start and stop positions of all occurrences or the word or words in the sequence. It includes two fields.

| Field	Description
Start	Row vector containing the start position of each occurrence of the search word or words.
Stop	Row vector containing the stop position of each occurrence of the search word or words.

Examples

Search for a word containing multiple symbols:

```
% Highlight the word 'BART' which represents 'TAGT' and 'TAAT'
seqshowwords('GCTAGTAACGTATATATAAT','BART')
```

```
ans =
    Start: [3 17]
     Stop: [6 20]
```
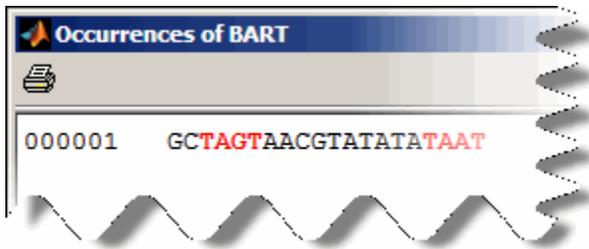

Search for a word that repeats, excluding overlaps:

```
% Highlight all occurrences of 'TATA', excluding those that are
% already part of another matched word.
seqshowwords('GCTATAACGTATATATATA','TATA')

ans =
    Start: [3 10 14]
     Stop: [6 13 17]
```
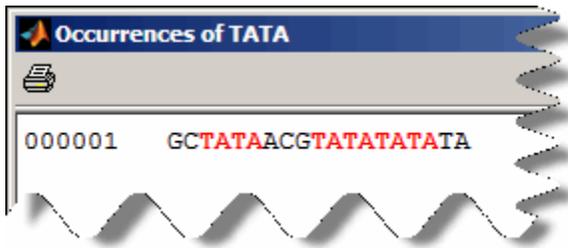

Search for a word that repeats, including overlaps:

```
% Use the regular expression 'TA(TA)*TA' to highlight all multiple
% repeats of 'TA'
seqshowwords('GCTATAACGTATATATATA','TA(TA)*TA')

ans =
    Start: [3 10]
     Stop: [6 19]
```
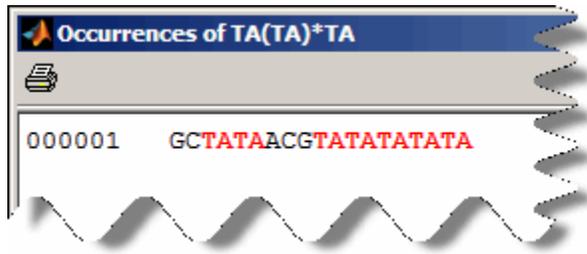
Search for multiple words:

```
% Use a cell array as input to highlight both the words
% 'CG' and 'GC'
seqshowwords('GCTATAACGTATATATATA',{'CG', 'GC'})


ans =

    Start: [1 8]
     Stop: [2 9]
```
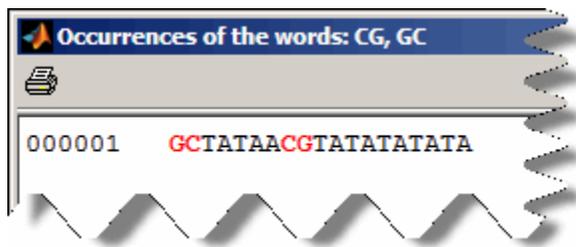

Alternatives

The `seqviewer` function opens the Biological Sequence Viewer, where you search for words in a sequence by selecting **Sequence > Find Word**. The Biological Sequence Viewer does not:

- Allow searching for multiple words in one step
- Return a structure containing the start and stop positions for all occurrences of the word in the sequence

More About

- Regular Expressions

See Also

palindromes | cleave | restrict | seqdisp | seqviewer | seqwordcount | strfind | regexp | ColorSpec

seqviewer

Visualize and interactively explore biological sequences

Syntax

```
seqviewer
seqviewer(Seq)
seqviewer(Seq,Name,Value)

seqviewer('close')
```

Description

`seqviewer` opens the Sequence Viewer app.

`seqviewer(Seq)` loads a sequence Seq into the app, where you can view and interactively explore the sequence.

`seqviewer(Seq,Name,Value)` opens the app with additional options specified by one or more Name,Value pair arguments.

`seqviewer('close')` closes the Sequence Viewer app.

Input Arguments

Seq — Amino acid or nucleotide sequence
string of single-letter codes | row vector of integers | structure | string specifying a file name

Amino acid or nucleotide sequence, specified as:

- String of single-letter codes
- Row vector of integers

- MATLAB structure containing a `Sequence` field that contains an amino acid or nucleotide sequence, such as returned by `fastaread`, `fastqread`, `getgenpept`, `genpeptread`, `getpdb`, `pdbread`, `emblread`, `getembl`, `genbankread`, or `getgenbank`

- String specifying a file name with an extension of .gbk, .gpt, .fasta, .fa, or .ebi.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'Alphabet','AA'` specifies that the aligned sequences are amino acid sequences.

`'Alphabet'` — Type of aligned sequences
`'AA'` | `'NT'`

Type of aligned sequences, specified as `'AA'` for amino acid sequences or `'NT'` for nucleotide sequences.

Example: `'Alphabet','AA'`

Examples

Open and View a Biological Sequence

Retrieve a sequence from the GenBank database.

```
S = getgenbank('M10051');
```

Load the sequence into the Sequence Viewer app.

```
seqviewer(S)
```

Alternatively, you can click Sequence Viewer on the **Apps** tab to open the app, and view the biological sequence `S`.
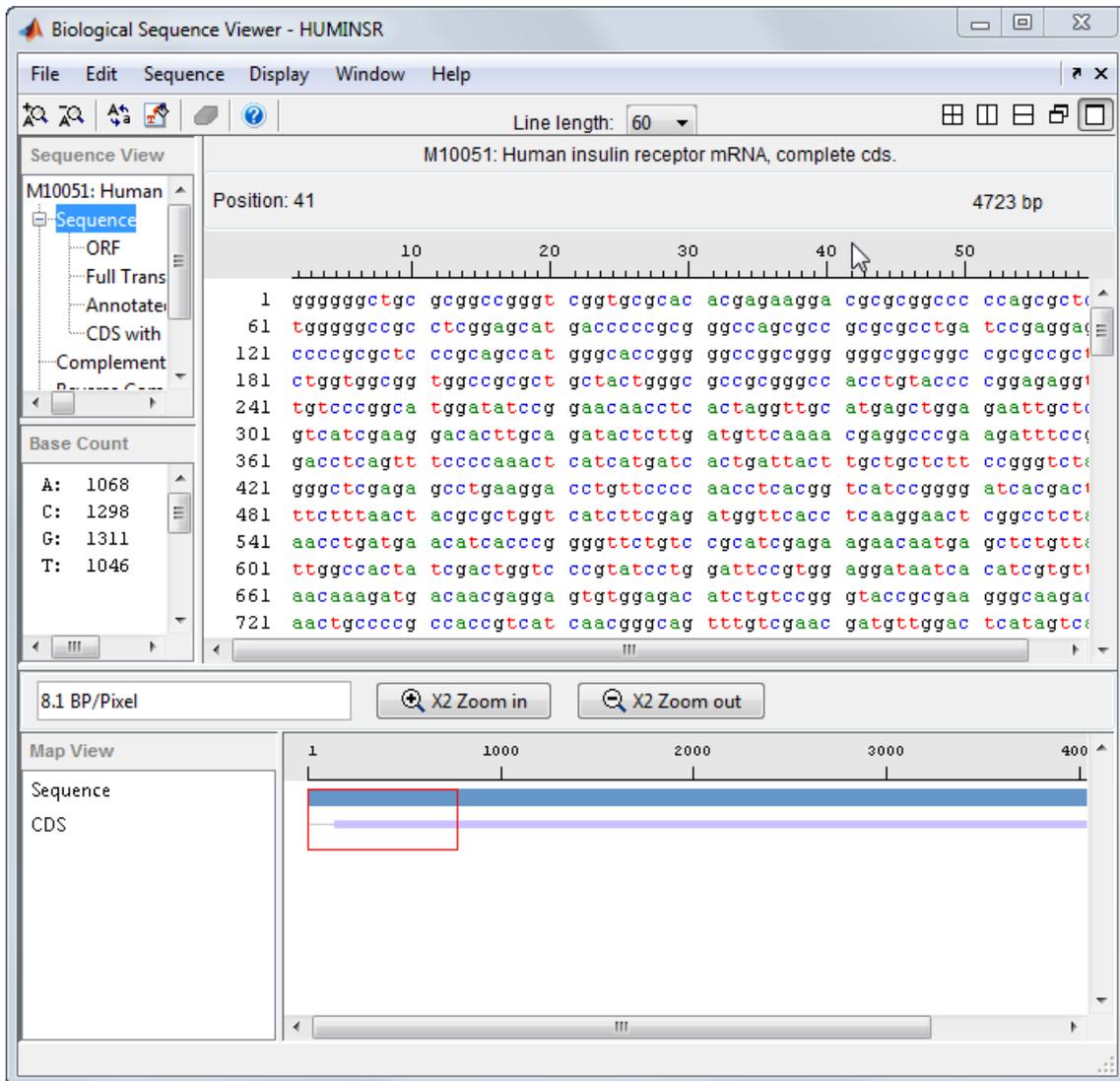
Close the app.

```
seqviewer('close')
```

- "Exploring a Nucleotide Sequence Using the Sequence Viewer App"

See Also

aa2nt | aacount | | aminolookup | basecount | baselookup | dimercount | emblread | fastaread | fastawrite | genbankread | geneticcode | genpeptread | getembl | getgenbank | getgenpept | nt2aa | proteinplot | seqcomplement | seqdisp | seqrcomplement | seqreverse | seqshoworfs | seqshowwords | seqwordcount

seqwordcount

Count number of occurrences of word in sequence

Syntax

seqwordcount(*Seq, Word*)

Arguments

Seq	Enter a nucleotide or amino acid sequence of characters. You can also enter a structure with the field `Sequence`.
Word	Enter a short sequence of characters.

Description

seqwordcount(*Seq, Word*) counts the number of times that a word appears in a sequence, and then returns the number of occurrences of that word.

If *Word* contains nucleotide or amino acid symbols that represent multiple possible symbols (ambiguous characters), then `seqwordcount` counts all matches. For example, the symbol R represents either G or A (purines). For another example, if word equals `'ART'`, then `seqwordcount` counts occurrences of both `'AAT'` and `'AGT'`.

Examples

`seqwordcount` does not count overlapping patterns multiple times. In the following example, `seqwordcount` reports three matches. TATATATA is counted as two distinct matches, not three overlapping occurrences.

seqwordcount('GCTATAACGTATATATAT','TATA')

ans =
     3

The following example reports two matches ('TAGT' and 'TAAT'). B is the ambiguous code for G, T, or C, while R is an ambiguous code for G and A.

```
seqwordcount('GCTAGTAACGTATATATAAT','BART')

ans =
     2
```

See Also
codoncount | seqshoworfs | seqshowwords | seqviewer | seq2regexp | strfind

set (biograph)

Set property of biograph object

Syntax

```
set(BGobj)
set(BGobj, 'PropertyName')
set(BGobj, 'PropertyName', PropertyValue)
set(BGobj, 'Property1Name', Property1Value, 'Property2Name',
Property2Value, ...)
```

Arguments

BGobj	Biograph object created with the function `biograph`.
PropertyName	Property name for a biograph object.
PropertyValue	Value of the property specified by *PropertyName*.

Description

`set(BGobj)` displays possible values for all properties that have a fixed set of property values in *BGobj*, a biograph object.

`set(BGobj, 'PropertyName')` displays possible values for a specific property that has a fixed set of property values in *BGobj*, a biograph object.

`set(BGobj, 'PropertyName', PropertyValue)` sets the specified property of *BGobj*, a biograph object.

`set(BGobj, 'Property1Name', Property1Value, 'Property2Name', Property2Value, ...)` sets the specified properties of *BGobj*, a biograph object.

Properties of a Biograph Object

Property	Description
`ID`	String to identify the biograph object. Default is `''`.

Property	Description
Label	String to label the biograph object. Default is `''`.
Description	String that describes the biograph object. Default is `''`.
LayoutType	String that specifies the algorithm for the layout engine. Choices are: • `'hierarchical'` (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges. • `'radial'` — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges. • `'equilibrium'` — Calculates layout by minimizing the energy in a dynamic spring system.
EdgeType	String that specifies how edges display. Choices are: • `'straight'` • `'curved'` (default) • `'segmented'` **Note:** Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with `LayoutType` equal to `'equilibrium'` or `'radial'` cannot produce curved or segmented edges.
Scale	Positive number that post-scales the node coordinates. Default is `1`.
LayoutScale	Positive number that scales the size of the nodes before calling the layout engine. Default is `1`.
EdgeTextColor	Three-element numeric vector of RGB values. Default is `[0, 0, 0]`, which defines black.
EdgeFontSize	Positive number that sets the size of the edge font in points. Default is `8`.
ShowArrows	Controls the display of arrows with the edges. Choices are `'on'` (default) or `'off'`.

Property	Description
ArrowSize	Positive number that sets the size of the arrows in points. Default is 8.
ShowWeights	Controls the display of text indicating the weight of the edges. Choices are 'on' (default) or 'off'.
ShowTextInNodes	String that specifies the node property used to label nodes when you display a biograph object using the view method. Choices are: • 'Label' — Uses the Label property of the node object (default). • 'ID' — Uses the ID property of the node object. • 'None'
NodeAutoSize	Controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.
NodeCallback	User-defined callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph object in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, @(node) inspect(node), which displays the Property Inspector dialog box.
EdgeCallback	User-defined callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph object in the Biograph Viewer, you can double-click an edge to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, @(edge) inspect(edge), which displays the Property Inspector dialog box.
CustomNodeDrawFcn	Function handle to a customized function to draw nodes. Default is [].
Nodes	Read-only column vector with handles to node objects of a biograph object. The size of the vector is the number of

Property	Description
	nodes. For properties of node objects, see Properties of a Node Object.
`Edges`	Read-only column vector with handles to edge objects of a biograph object. The size of the vector is the number of edges. For properties of edge objects, see Properties of an Edge Object.

Examples

1 Create a biograph object with default node IDs.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg = biograph(cm)
Biograph object with 5 nodes and 9 edges.
```

2 Use the `view` method to display the biograph object.

```
view(bg)
```
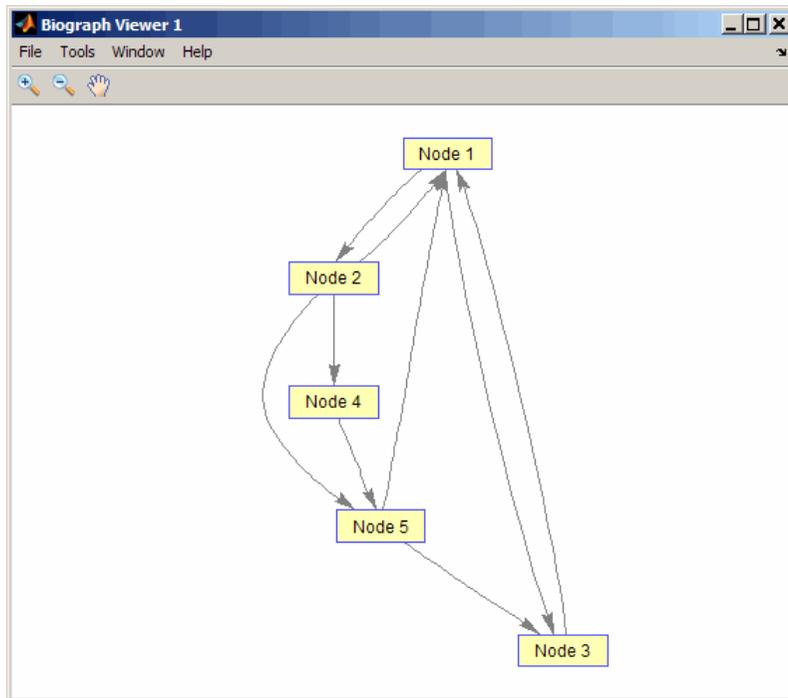
3 Use the `set` method to change the edge lines from curved to straight.

```
set(bg, 'EdgeType', 'straight')
```

4 Display the biograph object again.

```
view(bg)
```
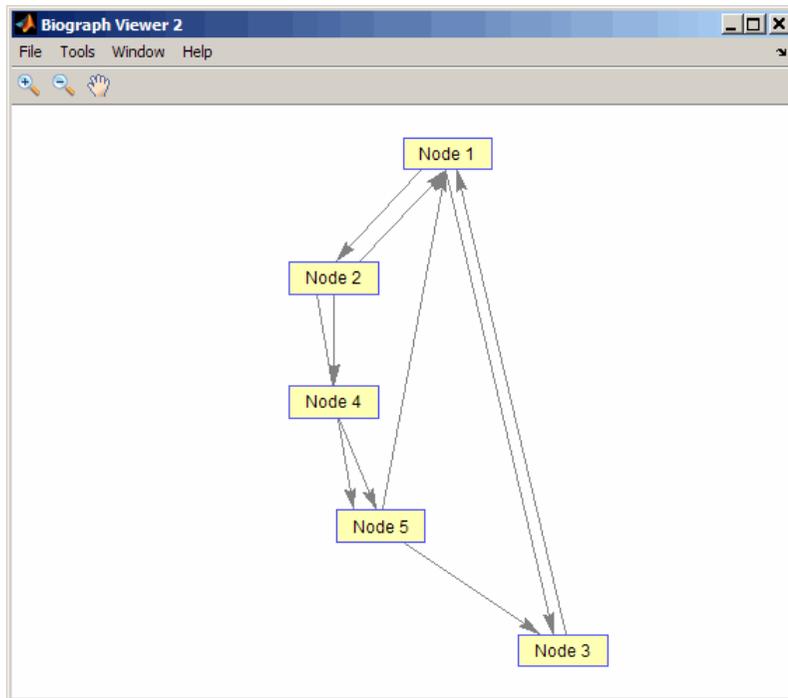
More About

- biograph object

See Also

biograph | get

set

Class: BioRead

Set property of object

Syntax

NewObj = set(*BioObj*, '*PropertyName*', *PropertyValue*)
NewObj = set(*BioObj*, '*Property1Name*', *Property1Value*,
'*Property2Name*', *Property2Value*, ...)
set(*BioObj*, '*PropertyName*')
PossVal = set(*BioObj*, '*PropertyName*')
set(*BioObj*)
PropNameVal = set(*BioObj*)

Description

NewObj = set(*BioObj*, '*PropertyName*', *PropertyValue*) returns *NewObj*, a new object that is a copy of *BioObj*, but with the specified property set to the specified value. set accepts a comma-separated property name/value pair for the BioRead or BioMap class. Specify *PropertyName* inside single quotes.

*NewObj* = set(*BioObj*, '*Property1Name*', *Property1Value*, '*Property2Name*', *Property2Value*, ...) sets multiple property values of a BioRead or BioMap object in a single statement.

set(*BioObj*, '*PropertyName*') displays all possible values for the specified property of *BioObj*, a BioRead or BioMap object.

PossVal = set(*BioObj*, '*PropertyName*') returns *PossVal*, a cell array containing all possible values for the specified property of *BioObj*, a BioRead or BioMap object. *PossVal* is a cell array of one or more strings or, if the property does not have a finite set of possible values, an empty cell array.

set(*BioObj*) displays all properties and their possible values for *BioObj*, a BioRead or BioMap object.

PropNameVal = set(*BioObj*) returns *PropNameVal*, a structure containing all properties and their possible values for *BioObj*, an object. *PropNameVal* is a structure whose field names are the property names, and whose values are cell arrays of one or more possible property values.

Tips

Use the set method to determine the property names and possible values before setting the properties with the set method or specific set methods such as setHeader, setSequence, and setQuality. Some of these specific set methods let you set all or a subset of a property.

Input Arguments

BioObj

Object of the BioRead or BioMap class.

Note: If *BioObj* was constructed from a BioIndexedFile object, you cannot set its properties (except for the Name and Reference properties).

Default:

PropertyName

Name of a property of the class.

Default:

PropertyValue

Value of a property of the class.

Default:

Output Arguments

NewObj

Object of the `BioRead` or `BioMap` class.

PossVal

Cell array containing all possible values for a specified property of *BioObj*. The cell array contains one or more strings or, if the property does not have a finite set of possible values, it is an empty cell array.

PropNameVal

Structure containing all properties and their possible values for *BioObj*. The structure has field names that are the property names, and whose values are cell arrays of one or more possible property values.

Examples

Construct a `BioRead` object and set the `Name` property:

```
% Construct a BioRead object from a FASTQ file
BRObj = BioRead('SRR005164_1_50.fastq');
% Set the Name property of the object
BRObj = set (BRObj, 'Name', 'MyObject')

BRObj =

  BioRead with properties:

     Quality: [50x1 File indexed property]
    Sequence: [50x1 File indexed property]
      Header: [50x1 File indexed property]
       NSeqs: 50
        Name: 'MyObject'
```

See Also

get | BioRead | BioMap

How To

- "Manage Short-Read Sequence Data in Objects"

Related Links

- Sequence Read Archive
- SAM format specification

set (clustergram)

Set property of clustergram object

Syntax

```
set(CGobj)
set(CGobj, 'PropertyName')
set(CGobj, 'PropertyName', PropertyValue)
set(CGobj, 'Property1Name', Property1Value, 'Property2Name',
Property2Value, ...)
```

Arguments

CGobj	Clustergram object created with the function `clustergram`.
PropertyName	Property name for a clustergram object.

Description

Note: You cannot set the properties of a clustergram object if you created it using the **Export Group to Workspace** command in the Clustergram window.

---

set(*CGobj*) displays possible values for all properties that have a fixed set of property values in *CGobj*, a clustergram object.

set(*CGobj*, '*PropertyName*') displays possible values for a specific property that has a fixed set of property values in *CGobj*, a clustergram object.

set(*CGobj*, '*PropertyName*', *PropertyValue*) sets the specified property of *CGobj*, a clustergram object.

set(*CGobj*, '*Property1Name*', *Property1Value*, '*Property2Name*', *Property2Value*, ...) sets the specified properties of *CGobj*, a clustergram object.

Properties of a Clustergram Object

Property	Description
RowLabels	Vector of numbers or cell array of text strings to label the rows in the dendrogram and heat map. Default is a vector of values 1 through *M*, where *M* is the number of rows in *Data*, the matrix of data used by the clustergram function to create the clustergram object.
ColumnLabels	Vector of numbers or cell array of text strings to label the columns in the dendrogram and heat map. Default is a vector of values 1 through *N*, where *N* is the number of columns in *Data*, the matrix of data used by the clustergram function to create the clustergram object.
Standardize	Numeric value that specifies the dimension for standardizing the values in *Data*, the matrix of data used to create the clustergram object. The standardized values are transformed so that the mean is 0 and the standard deviation is 1 in the specified dimension. Choices are: • 'column' or 1 — Standardize along the columns of data. • 'row' or 2 — Standardize along the rows of data. • 'none' or 3 (default) — Do not standardize.
Cluster	Numeric value that specifies the dimension for clustering the values in *Data*, the matrix of data used to create the clustergram object. Choices are: • 1 — Cluster rows of data only. • 2 — Cluster columns of data only. • 3 — Cluster rows of data, then cluster columns of row-clustered data.
RowPdist	String that specifies the distance metric to pass to the pdist function (Statistics and Machine Learning Toolbox software) to use to calculate the pairwise distances between rows. For information on choices, see the pdist function.

Property	Description
	Note: If the distance metric requires extra arguments, then *RowPdistValue* is a cell array. For example, to use the Minkowski distance with exponent P, you would use `{'minkowski', P}`.
ColumnPdist	String that specifies the distance metric to pass to the `pdist` function (Statistics and Machine Learning Toolbox software) to use to calculate the pairwise distances between columns. For information on choices, see the `pdist` function.
	Note: If the distance metric requires extra arguments, then *ColumnPdistValue* is a cell array. For example, to use the Minkowski distance with exponent P, you would use `{'minkowski', P}`.
Linkage	String or two-element cell array of strings that specifies the linkage method to pass to the `linkage` function (Statistics and Machine Learning Toolbox software) to use to create the hierarchical cluster tree for rows and columns. If a two-element cell array of strings, the first element is used for linkage between rows, and the second element is used for linkage between columns. For information on choices, see the `linkage` function.
Dendrogram	Scalar or two-element numeric vector or cell array that specifies the `'colorthreshold'` property to pass to the `dendrogram` function (Statistics and Machine Learning Toolbox software) to create the dendrogram plot. If a two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns. For more information, see the `dendrogram` function.
OptimalLeafOrder	Property to enable or disable the optimal leaf ordering calculation, which determines the leaf order that maximizes the similarity between neighboring leaves. Choices are `true` (enable) or `false` (disable).
	Tip Disabling the optimal leaf ordering calculation can be useful when working with large data sets because this

Property	Description
	calculation uses a large amount of memory and can be very time consuming.
LogTrans	Controls the $\log_2$ transform of *Data*, the matrix of data used to create the clustergram object, from natural scale. Choices are true or false.
Colormap	Either of the following: • *M*-by-3 matrix of RGB values • Name or function handle of a function that returns a colormap, such as redgreencmap or redbluecmap
DisplayRange	Positive scalar that specifies the display range of standardized values. For example, if you specify redgreencmap for the 'ColorMap' property, pure red represents values ≥ DisplayRange, and pure green represents values ≤ −DisplayRange.
Symmetric	Property to force the color scale of the heat map to be symmetric around zero. Choices are true or false.
Ratio	Either of the following: • Scalar • Two-element vector It specifies the ratio of space that the row and column dendrograms occupy relative to the heat map. If Ratio is a scalar, it is used as the ratio for both dendrograms. If Ratio is a two-element vector, the first element is used for the ratio of the row dendrogram width to the heat map width, and the second element is used for the ratio of the column dendrogram height to the heat map height. The second element is ignored for one-dimensional clustergrams.
Impute	Any of the following: • Name of a function that imputes missing data. • Handle to a function that imputes missing data.

Property	Description
	• Cell array where the first element is the name of or handle to a function that imputes missing data and the remaining elements are property name/property value pairs used as inputs to the function.
RowMarkers	Optional structure array for annotating the groups (clusters) of rows determined by the clustergram function. Each structure in the array represents a group of rows and contains the following fields: • GroupNumber — Number to annotate the row group. • Annotation — String specifying text to annotate the row group. • Color — String or three-element vector of RGB values specifying a color, which is used to label the row group. For more information on specifying colors, see colorspec. If this field is empty, default is 'blue'.
ColumnMarkers	Optional structure array for annotating groups (clusters) of columns determined by the clustergram function. Each structure in the array represents a group of rows and contains the following fields: • GroupNumber — Number to annotate the column group. • Annotation — String specifying text to annotate the column group. • Color — String or three-element vector of RGB values specifying a color, which is used to label the column group. For more information on specifying colors, see colorspec. If this field is empty, default is 'blue'.

Examples

1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains filtered yeast data. This MAT-file includes three variables: yeastvalues, a matrix of gene expression data, genes, a cell array of GenBank accession numbers for

labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`.

```
load filteredyeastdata
```

2 Create a clustergram object and display the dendrograms and heat map from the gene expression data in the first 30 rows of the `yeastvalues` matrix and standardize along the rows of data.

```
cgo = clustergram(yeastvalues(1:30,:),'Standardize','row')
```

Clustergram object with 30 rows of nodes and 7 column of nodes.
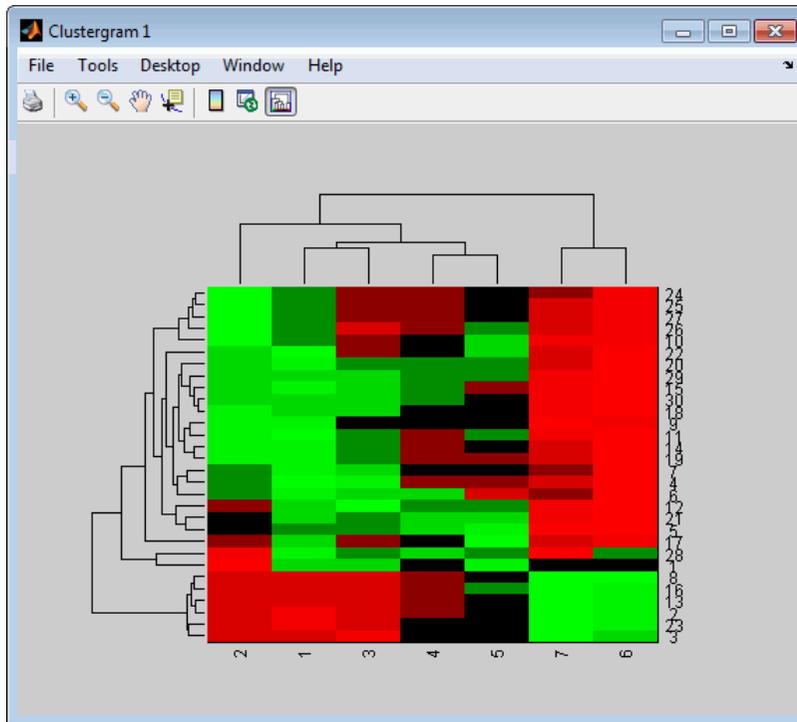
3 Use the `set` method and the `genes` and `times` vectors to add meaningful row and column labels to the clustergram.

```
set(cgo,'RowLabels',genes(1:30),'ColumnLabels',times)
```
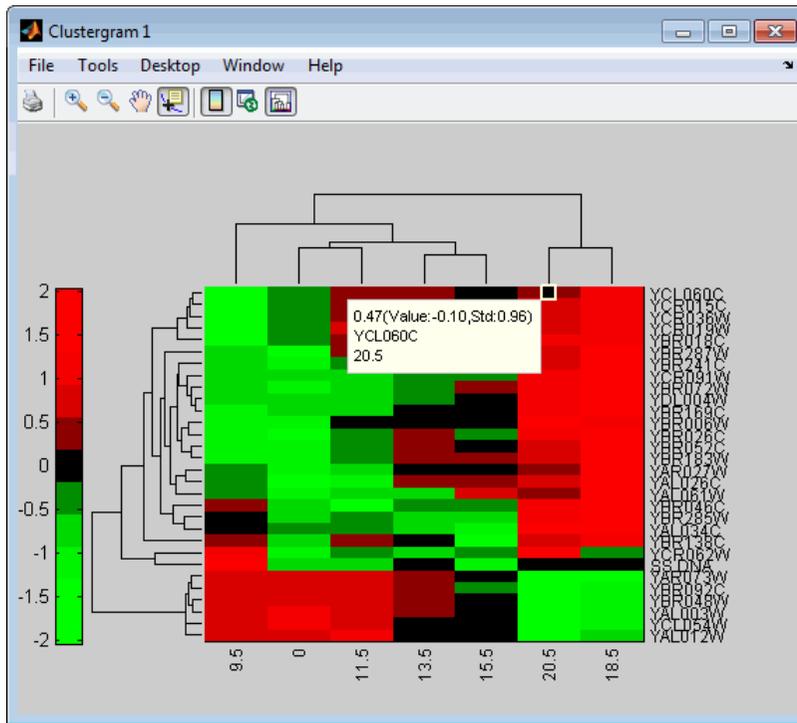
4 Reset the colormap of the heat map to `redbluecmap`.

```
set(cgo,'Colormap',redbluecmap);
```
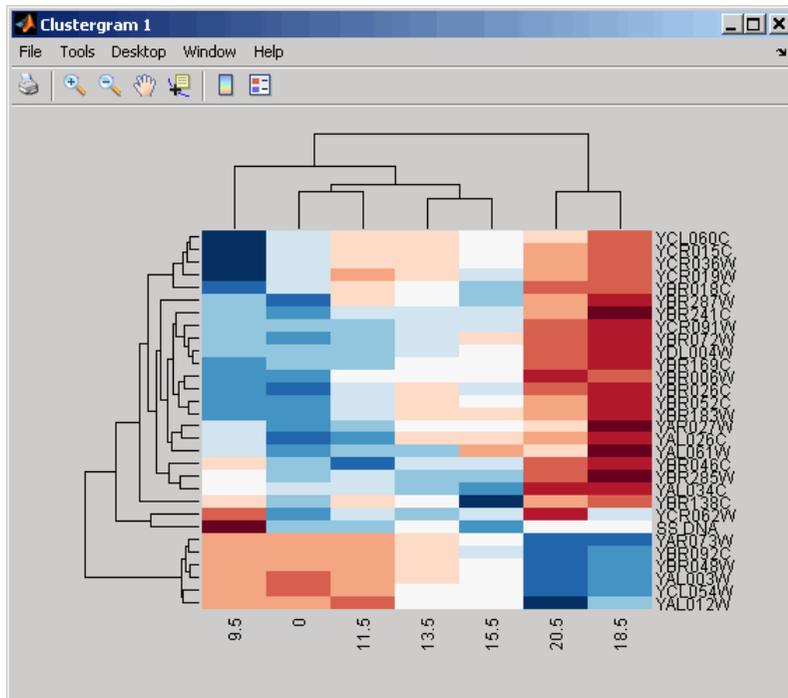
More About

- clustergram object

See Also

```
clustergram | plot | view | get
```

set (DataMatrix)

Set property of DataMatrix object

Syntax

```
set(DMObj)
set(DMObj, 'PropertyName')
DMObj = set(DMObj, 'PropertyName', PropertyValue)
DMObj = set(DMObj, 'Property1Name', Property1Value, 'Property2Name',
Property2Value, ...)
```

Arguments

DMObj	DataMatrix object, such as created by DataMatrix (object constructor).
PropertyName	Property name of a DataMatrix object.
PropertyValue	Value of the property specified by *PropertyName*.

Description

set(*DMObj*) displays possible values for all properties that have a fixed set of property values in *DMObj*, a DataMatrix object.

set(*DMObj*, '*PropertyName*') displays possible values for a specific property that has a fixed set of property values in *DMObj*, a DataMatrix object.

DMObj = set(*DMObj*, '*PropertyName*', *PropertyValue*) sets the specified property of *DMObj*, a DataMatrix object.

DMObj = set(*DMObj*, '*Property1Name*', *Property1Value*, '*Property2Name*', *Property2Value*, ...) sets the specified properties of *DMObj*, a DataMatrix object.

Properties of a DataMatrix Object

Property	Description
Name	String that describes the DataMatrix object. Default is `''`.
RowNames	Empty array or cell array of strings that specifies the names for the rows, typically gene names or probe identifiers. The number of elements in the cell array must equal the number of rows in the matrix. Default is an empty array.
ColNames	Empty array or cell array of strings that specifies the names for the columns, typically sample identifiers. The number of elements in the cell array must equal the number of columns in the matrix.
NRows	Positive number that specifies the number of rows in the matrix. **Note:** You cannot modify this property directly. You can access it using the `get` method.
NCols	Positive number that specifies the number of columns in the matrix. **Note:** You cannot modify this property directly. You can access it using the `get` method.
NDims	Positive number that specifies the number of dimensions in the matrix. **Note:** You cannot modify this property directly. You can access it using the `get` method.
ElementClass	String that specifies the class type, such as `single` or `double`. **Note:** You cannot modify this property directly. You can access it using the `get` method.

Examples

1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: `yeastvalues`, a matrix of gene expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`.

```
load filteredyeastdata
```

2 Import the microarray object package so that the `DataMatrix` constructor function will be available.

```
import bioma.data.*
```

3 Create a DataMatrix object from the gene expression data in the first 30 rows of the `yeastvalues` matrix.

```
dmo = DataMatrix(yeastvalues(1:30,:));
```

4 Use the `get` method to display the properties of the DataMatrix object, `dmo`.

```
get(dmo)

          Name: ''
      RowNames: []
      ColNames: []
         NRows: 30
         NCols: 7
         NDims: 2
    ElementClass: 'double'
```

Notice that the `RowNames` and `ColNames` fields are empty.

5 Use the `set` method and the `genes` and `times` variables to specify row names and column names for the DataMatrix object, `dmo`.

```
dmo = set(dmo,'RowNames',genes(1:30),'ColNames',times)
```

6 Use the `get` method to display the properties of the DataMatrix object, `dmo`.

```
get(dmo)

          Name: ''
      RowNames: {30x1 cell}
      ColNames: {'   0'  ' 9.5'  '11.5'  '13.5'  '15.5'  '18.5'  '20.5'}
         NRows: 30
         NCols: 7
```

```
         NDims: 2
   ElementClass: 'double'
```

More About

·   DataMatrix object

See Also
`DataMatrix` | `get`

setHeader

Class: BioRead

Set sequence headers for object

Syntax

```
NewObj = setHeader(BioObj, Headers)
NewObj = setHeader(BioObj, Headers, Subset)
```

Description

NewObj = setHeader(*BioObj*, *Headers*) returns *NewObj*, a new object, created from *BioObj*, an existing object, with the Header property set to *Headers*, a cell array of strings containing sequence headers.

NewObj = setHeader(*BioObj*, *Headers*, *Subset*) returns *NewObj*, a new object, created from *BioObj*, an existing object, with the Header property of a subset of the elements set to *Headers*, a cell array of strings containing sequence headers. setHeader sets the headers for only the object elements specified by *Subset*.

Tips

To update headers in an existing object, use the same object as the input *BioObj* and the output *NewObj*.

Input Arguments

BioObj

Object of the BioRead or BioMap class.

Note: If *BioObj* was constructed from a BioIndexedFile object, you cannot set its Header property.

Default:

Headers

Cell array of strings containing sequence headers.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: A one-to-one relationship must exist between the number and order of elements in *Headers* and *Subset*. If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

NewObj

Object of the `BioRead` or `BioMap` class.

Examples

Set the headers of some elements in a `BioRead` object:

```
% Construct a BioRead object from a FASTQ file
BRObj = BioRead('SRR005164_1_50.fastq');
% Set the Header property of the first five elements in the object
BRObj = setHeader(BRObj, {'H1', 'H2', 'H3', 'H4', 'H5'}, [1:5]);
```

Alternatives

An alternative to using the `setHeader` method to update an existing object is to use dot indexing with the `Header` property:

BioObj.Header(*Indices*) = *NewHeaders*

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers. *NewHeaders* is a cell array of strings containing headers. *Indices* and *NewHeaders* must have the same number and order of elements.

See Also
getHeader | BioRead | BioMap

How To
• "Manage Short-Read Sequence Data in Objects"

Related Links
• Sequence Read Archive
• SAM format specification

setQuality

Class: BioRead

Set sequence quality scores for object

Syntax

```
NewObj = setQuality(BioObj, Quality)
NewObj = setQuality(BioObj, Quality, Subset)
```

Description

NewObj = setQuality(*BioObj*, *Quality*) returns *NewObj*, a new object, created from *BioObj*, an existing object, with the Quality property set to *Quality*, a cell array of strings containing the ASCII representations of per-base quality scores for nucleotide sequences.

NewObj = setQuality(*BioObj*, *Quality*, *Subset*) returns *NewObj*, a new object, created from *BioObj*, an existing object, with the Quality property of a subset of the elements set to *Quality*, a cell array of strings containing the ASCII representations of per-base quality scores for nucleotide sequences. setQuality sets the quality scores for only the object elements specified by *Subset*.

Tips

To update quality scores in an existing object, use the same object as the input *BioObj* and the output *NewObj*.

Input Arguments

BioObj

Object of the BioRead or BioMap class.

Note: If *BioObj* was constructed from a `BioIndexedFile` object, you cannot set its `Quality` property.

Default:

`Quality`

Cell array of strings containing the ASCII representations of per-base quality scores for nucleotide sequences.

Default:

`Subset`

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: A one-to-one relationship must exist between the number and order of elements in *Quality* and *Subset*. If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

`NewObj`

Object of the `BioRead` or `BioMap` class.

Examples

Construct a `BioRead` object, and then set a subset of the quality scores:

```
% Construct a BioRead object from a FASTQ file
BRObj = BioRead('SRR005164_1_50.fastq');
% Create a new quality score
newValue = {repmat('N', 1, length(BRObj.Quality{2}))};
% Set the Quality property of the second element to the new
% quality score
BRObj = setQuality(BRObj, newValue, 2);
```

Alternatives

An alternative to using the `setQuality` method to update an existing object is to use dot indexing with the `Quality` property:

BioObj.Quality(*Indices*) = *NewQuality*

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequence headers. *NewQuality* is a cell array of strings containing ASCII representations of per-base quality scores. *Indices* and *NewQuality* must have the same number and order of elements.

See Also
getQuality | BioRead | BioMap

How To
- "Manage Short-Read Sequence Data in Objects"

Related Links
- Sequence Read Archive
- SAM format specification

setSequence

Class: BioRead

Set sequences for object

Syntax

```
NewObj = setSequence(BioObj, Sequences)
NewObj = setSequence(BioObj, Sequences, Subset)
```

Description

NewObj = setSequence(*BioObj*, *Sequences*) returns *NewObj*, a new object, created from *BioObj*, an existing object, with the Sequence property set to *Sequences*, a cell array of strings containing the letter representations of nucleotide sequences.

NewObj = setSequence(*BioObj*, *Sequences*, *Subset*) returns *NewObj*, a new object, created from *BioObj*, an existing object, sets the sequences of a subset of the elements in *BioObj* with the Sequence property of a subset of the elements set to *Sequences*, a cell array of strings containing the letter representations of nucleotide sequences. setSequence sets the sequences for only the object elements specified by *Subset*.

Tips

- To update sequences in an existing object, use the same object as the input *BioObj* and the output *NewObj*.

- If you use the setSequence method to modify the Sequence property, you also may need to modify the Start and Signature properties accordingly, which you can do using the setStart and setSignature methods.

Input Arguments

`BioObj`

Object of the `BioRead` or `BioMap` class.

Note: If *`BioObj`* was constructed from a `BioIndexedFile` object, you cannot set its `Sequence` property.

Default:

`Sequences`

Cell array of strings containing the letter representations of nucleotide sequences.

Default:

`Subset`

One of the following to specify a subset of the elements in *`BioObj`*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: A one-to-one relationship must exist between the number and order of elements in *`Sequences`* and *`Subset`*. If you use a cell array of header strings to specify *`Subset`*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

`NewObj`

Object of the `BioRead` or `BioMap` class.

Examples

Construct a `BioRead` object, and then set a subset of the sequences:

```
% Construct a BioRead object from a FASTQ file
BRObj = BioRead('SRR005164_1_50.fastq');
% Set the Sequence property of the second element to a new sequence
BRObj = setSequence(BRObj, {'NNNNNNN'}, 2);
```

Alternatives

An alternative to using the `setSequence` method to update an existing object is to use dot indexing with the `Sequence` property:

BioObj.Sequence(*Indices*) = *NewSequences*

In the previous syntax, *Indices* is a vector of positive integers or a logical vector. *Indices* cannot be a cell array of strings containing sequences. *NewSequences* is a cell array of strings containing sequences. *Indices* and *NewSequences* must have the same number and order of elements.

See Also
getSequence | setSignature | BioRead | BioMap | setStart

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive
· SAM format specification

setSubsequence

Class: BioRead

Set partial sequences for object

Syntax

NewObj = setSubsequence(*BioObj*, *Subsequences*, *Subset*, *Positions*)

Description

NewObj = setSubsequence(*BioObj*, *Subsequences*, *Subset*, *Positions*)
returns *NewObj*, a new object, created from *BioObj*, an existing object, with the partial
sequences, specified by *Positions*, of a subset of the elements in *BioObj*, set to
Subsequences, a cell array of strings containing the letter representations of partial
nucleotide sequences. setSubsequence sets the subsequences for only the object
elements specified by *Subset*.

Tips

To update subsequences in an existing object, use the same object as the input *BioObj*
and the output *NewObj*.

Input Arguments

BioObj

Object of the BioRead or BioMap class.

Note: If *BioObj* was constructed from a BioIndexedFile object, you cannot modify it.

Default:

Subsequences

Cell array of strings containing the letter representations of partial nucleotide sequences. Each string must be the same length.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: A one-to-one relationship must exist between the number and order of elements in *Subsequences* and *Subset*. If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Positions

Either of the following to indicate positions in the nucleotide sequences:

- Vector of positive integers
- Logical vector

The number of positions specified by *Positions* must equal the length of the strings in *Subsequences*.

Default:

Output Arguments

NewObj

Object of the `BioRead` or `BioMap` class.

Examples

Construct a `BioRead` object, and then set nucleotide positions in a subset of elements:

```
% Construct a BioRead object from a FASTQ file
BRObj = BioRead('SRR005164_1_50.fastq');
% Set the first five positions of the second sequence to NNNNN
BRObj = setSubsequence(BRObj, {'NNNNN'}, 2, [1:5]);
BRObj = setSubsequence(BRObj, {'NNNNN'}, 'SRR005164.2', [1:5]);
% Set the 10th position of the first three sequences to X
BRObj = setSubsequence(BRObj, {'X', 'X', 'X'}, 1:3, 10);
```

See Also
`BioRead` | `BioMap` | `getSubsequence`

How To
- "Manage Short-Read Sequence Data in Objects"

Related Links
- Sequence Read Archive
- SAM format specification

setSubset

Class: BioRead

Set elements for object

Syntax

NewObj = setSubset(*BioObj*, *Elements*, *Subset*)

Description

NewObj = setSubset(*BioObj*, *Elements*, *Subset*) returns *NewObj*, a new object, created from *BioObj*, an existing object, with a subset of the elements in *BioObj* set to *Elements*, an object containing the appropriate number of elements. setSubset sets the object elements specified by *Subset*.

Tips

To update an existing object, use the same object as the input *BioObj* and the output *NewObj*.

Input Arguments

BioObj

Object of the BioRead or BioMap class.

Note: If *BioObj* was constructed from a BioIndexedFile object, you cannot modify it.

Default:

Elements

`BioRead` or `BioMap` object containing a number of elements equal to the number of elements specified by *Subset*.

Default:

Subset

One of the following to specify a subset of the elements in *BioObj*:

- Vector of positive integers
- Logical vector
- Cell array of strings containing valid sequence headers

Note: A one-to-one relationship must exist between the number and order of elements in *Elements* and *Subset*. If you use a cell array of header strings to specify *Subset*, be aware that a repeated header specifies all elements with that header.

Default:

Output Arguments

NewObj

Object of the `BioRead` or `BioMap` class.

Examples

Construct a `BioRead` object, and then set a subset of the elements in the object:

```
% Construct two BioRead objects, one with 10 elements, and one
% with 2 elements
struct1 = fastqread('SRR005164_1_50.fastq',...
                    'blockread', [1 10], 'trimheaders', true);
struct2 = fastqread('SRR005164_1_50.fastq',...
                    'blockread', [11 12], 'trimheaders', true);
```

```
BRObj1 = BioRead(struct1);
BRObj2 = BioRead(struct2);
% Replace the first two elements in BRObj1 with the elements
% in BRObj2
BRObj1 = setSubset(BRObj1, BRObj2, 1:2);
```

See Also
getSubset | BioRead | BioMap

How To
· "Manage Short-Read Sequence Data in Objects"

Related Links
· Sequence Read Archive
· SAM format specification

sffinfo

Return information about SFF file

Syntax

InfoStruct = sffinfo(*File*)

Description

InfoStruct = sffinfo(*File*) returns a MATLAB structure containing summary information about a Standard Flowgram Format (SFF) file.

Input Arguments

File

String specifying a file name or path and file name of an SFF file produced by version 1.0 of the Genome Sequencer System data analysis software from 454 Life Sciences®. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

Default:

Output Arguments

InfoStruct

MATLAB structure containing summary information about an SFF file. The structure contains the following fields.

Field	Description
Filename	Name of the file.

Field	Description
FileModDate	Modification date of the file.
FileSize	Size of the file in bytes.
Version	Version number of the file.
FlowgramCode	Code of the format used to encode flowgram values.
NumberOfReads	Number of sequence reads in the file.
NumberOfFlowsPerRead	Number of flows for each read.
FlowChars	Bases used in each flow.
KeySequence	String of bases in the key sequence.

Examples

The SFF file, `SRR013472.sff`, used in this example is not provided with the Bioinformatics Toolbox software. You can download sample SFF files from:

`http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=show&f=main&m=main&s=main`

Return a summary of the contents of an SFF file:

```
info = sffinfo('SRR013472.sff')

info =

                Filename: 'SRR013472.sff'
             FileModDate: '23-Feb-2009 15:14:36'
                FileSize: 6632392
                 Version: [0 0 0 1]
            FlowgramCode: 1
           NumberOfReads: 3546
    NumberOfFlowsPerRead: 440
               FlowChars: [1x440 char]
             KeySequence: 'TCAG'
```

More About

• http://www.my454.com/

- http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?
  cmd=show&f=main&m=main&s=main

See Also
`fastqread` | `fastqwrite` | `fastqinfo` | `fastainfo` | `fastaread` | `fastawrite` |
`sffread` | `saminfo` | `samread`

sffread

Read data from SFF file

Syntax

```
SFFStruct = sffread(File)
sffread(..., 'Blockread', BlockreadValue, ...)
sffread(..., 'Feature', FeatureValue, ...)
```

Description

SFFStruct = sffread(*File*) reads a Standard Flowgram Format (SFF) file and returns the data in a MATLAB array of structures.

sffread(..., '*PropertyName*', *PropertyValue*, ...) calls sffread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

sffread(..., 'Blockread', *BlockreadValue*, ...) reads a single sequence entry or block of sequence entries from an SFF file containing multiple sequences.

sffread(..., 'Feature', *FeatureValue*, ...) specifies the information to include in the return structure.

Input Arguments

File

String specifying a file name or path and file name of an SFF file produced by version 1.0 of the Genome Sequencer System data analysis software from 454 Life Sciences. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

Default:

BlockreadValue

Scalar or vector that controls the reading of a single sequence entry or block of sequence entries from an SFF file containing multiple sequences. Enter a scalar *N*, to read the *N*th entry in the file. Enter a 1-by-2 vector [*M1, M2*], to read a block of entries starting at the *M1* entry and ending at the *M2* entry. To read all remaining entries in the file starting at the *M1* entry, enter a positive value for *M1* and enter Inf for *M2*.

Default:

FeatureValue

String specifying the information to include in the output structure. The string includes letters from the alphabet H, S, Q, C, F, and I, which represent the fields Header, Sequence, Quality, Clipping, FlowgramValue, and FlowgramIndex, respectively.

Default: 'HSQ'

Output Arguments

SFFStruct

Array of structures containing information from an SFF file. There is one structure for each read or entry in the file. Each structure contains one or more of the following fields.

Field	Description
Header	Universal accession number.
Sequence	Numeric representation of nucleotide sequence.
Quality	Per-base quality scores.
Clipping	Clipping boundary positions.
FlowgramValue	Sequence of flowgram intensity values.
FlowgramIndex	Sequence of flowgram intensity indices.

Examples

The SFF file, SRR013472.sff, used in these examples is not provided with the Bioinformatics Toolbox software. You can download sample SFF files from:

sffread

```
http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=show&f=main&m=main&s=main
```

Read an entire SFF file:

```
% Read the contents of an entire SFF file into an
% array of structures
reads = sffread('SRR013472.sff')

reads =

3546x1 struct array with fields:
    Header
    Sequence
    Quality
```

Read a block of entries from an SFF file:

```
% Read only the header and sequence information of the
% first five reads from an SFF file into an array of structures
reads5 = sffread('SRR013472.sff', 'block', [1 5], 'feature', 'hs')

reads5 =

5x1 struct array with fields:
    Header
    Sequence
```

More About

- http://www.my454.com/

- http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?
  cmd=show&f=main&m=main&s=main

See Also

```
fastqread | fastqwrite | fastqinfo | fastainfo | fastaread | fastawrite |
sffinfo | saminfo | samread
```

1-1775

shortestpath (biograph)

Solve shortest path problem in biograph object

Syntax

```
[dist, path, pred] = shortestpath(BGObj, S)
[dist, path, pred] = shortestpath(BGObj, S, T)
[...] = shortestpath(..., 'Directed', DirectedValue, ...)
[...] = shortestpath(..., 'Method', MethodValue, ...)
[...] = shortestpath(..., 'Weights', WeightsValue, ...)
```

Arguments

BGObj	Biograph object created by `biograph` (object constructor).
S	Node in graph represented by an N-by-N adjacency matrix extracted from a biograph object, *BGObj*.
T	Node in graph represented by an N-by-N adjacency matrix extracted from a biograph object, *BGObj*.
DirectedValue	Property that indicates whether the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj*, is directed or undirected. Enter `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`.
MethodValue	String that specifies the algorithm used to find the shortest path. Choices are: • `'Bellman-Ford'` — Assumes weights of the edges to be nonzero entries in the N-by-N adjacency matrix. Time complexity is `O(N*E)`, where `N` and `E` are the number of nodes and edges respectively. • `'BFS'` — Breadth-first search. Assumes all weights to be equal, and nonzero entries in the N-by-N adjacency matrix to represent edges. Time complexity is `O(N+E)`, where `N` and `E` are the number of nodes and edges respectively.

	• 'Acyclic' — Assumes the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj*, to be a directed acyclic graph and that weights of the edges are nonzero entries in the N-by-N adjacency matrix. Time complexity is O(N+E), where N and E are the number of nodes and edges respectively.
	• 'Dijkstra' — Default algorithm. Assumes weights of the edges to be positive values in the N-by-N adjacency matrix. Time complexity is O(log(N)*E), where N and E are the number of nodes and edges respectively.
WeightsValue	Column vector that specifies custom weights for the edges in the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. It must have one entry for every nonzero value (edge) in the N-by-N adjacency matrix. The order of the custom weights in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, shortestpaths gets weight information from the nonzero entries in the N-by-N adjacency matrix.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*dist*, *path*, *pred*] = shortestpath(*BGObj*, *S*) determines the single-source shortest paths from node *S* to all other nodes in the graph represented by an N-by-N adjacency matrix extracted from a biograph object, *BGObj*. Weights of the edges are all nonzero entries in the N-by-N adjacency matrix. *dist* are the N distances from the source to every node (using Infs for nonreachable nodes and 0 for the source node). *path* contains the winning paths to every node. *pred* contains the predecessor nodes of the winning paths.

[*dist*, *path*, *pred*] = shortestpath(*BGObj*, *S*, *T*) determines the single source-single destination shortest path from node *S* to node *T*.

[...] = shortestpath(..., '*PropertyName*', *PropertyValue*, ...) calls shortestpath with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[...] = shortestpath(..., 'Directed', *DirectedValue*, ...) indicates whether the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj*, is directed or undirected. Set *DirectedValue* to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

[...] = shortestpath(..., 'Method', *MethodValue*, ...) lets you specify the algorithm used to find the shortest path. Choices are:

- 'Bellman-Ford' — Assumes weights of the edges to be nonzero entries in the N-by-N adjacency matrix. Time complexity is O(N*E), where N and E are the number of nodes and edges respectively.
- 'BFS' — Breadth-first search. Assumes all weights to be equal, and nonzero entries in the N-by-N adjacency matrix to represent edges. Time complexity is O(N+E), where N and E are the number of nodes and edges respectively.
- 'Acyclic' — Assumes the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj*, to be a directed acyclic graph and that weights of the edges are nonzero entries in the N-by-N adjacency matrix. Time complexity is O(N+E), where N and E are the number of nodes and edges respectively.
- 'Dijkstra' — Default algorithm. Assumes weights of the edges to be positive values in the N-by-N adjacency matrix. Time complexity is O(log(N)*E), where N and E are the number of nodes and edges respectively.

[...] = shortestpath(..., 'Weights', *WeightsValue*, ...) lets you specify custom weights for the edges. *WeightsValue* is a column vector having one entry for every nonzero value (edge) in the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. The order of the custom weights in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, shortestpath gets weight information from the nonzero entries in the N-by-N adjacency matrix.

More About

- biograph object

References

[1] Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik *1*, 269–271.

[2] Bellman, R. (1958). On a Routing Problem. Quarterly of Applied Mathematics *16(1)*, 87–90.

[3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

biograph | allshortestpaths | isdag | isspantree | minspantree | traverse | graphshortestpath | conncomp | isomorphism | maxflow | topoorder

showalignment

Display color-coded sequence alignment

Syntax

```
showalignment(Alignment)
showalignment(..., 'MatchColor', MatchColorValue, ...)
showalignment(..., 'SimilarColor' SimilarColorValue, ...)
showalignment(..., 'StartPointers', StartPointersValue, ...)
showalignment(..., 'Columns', ColumnsValue, ...)
showalignment(..., 'TerminalGap', TerminalGapValue, ...)
```

Description

showalignment(*Alignment*) displays a color-coded sequence alignment in a MATLAB Figure window.

showalignment(..., '*PropertyName*', *PropertyValue*, ...) calls showalignment with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

showalignment(..., 'MatchColor', *MatchColorValue*, ...) specifies the color to highlight matching characters in the output display.

showalignment(..., 'SimilarColor' *SimilarColorValue*, ...) specifies the color to highlight similar characters in the output display.

showalignment(..., 'StartPointers', *StartPointersValue*, ...) specifies the starting indices in the original sequences of a local pairwise alignment.

showalignment(..., 'Columns', *ColumnsValue*, ...) specifies the number of characters to display in one row when displaying a pairwise alignment, and labels the start of each row with the sequence positions.

showalignment(..., 'TerminalGap', *TerminalGapValue*, ...) controls the inclusion or exclusion of terminal gaps from the count of matches and similar residues

when displaying a pairwise alignment. *TerminalGapValue* can be `true` (default) or `false`.

Input Arguments

Alignment

Pairwise or multiple sequence alignment specified by one of the following:

- 3-by-N character array showing the pairwise alignment of two sequences, such as returned by `nwalign` or `swalign`
- MATLAB structure containing a `Sequence` field, such as returned by `fastaread`, `gethmmalignment`, `multialign`, or `multialignread`
- MATLAB character array that contains a multiple sequence alignment, such as returned by `multialign`

Default:

MatchColorValue

Color to highlight matching characters in the output display. Specify the color with one of the following:

- Three-element numeric vector of RGB values
- String containing a predefined single-letter color code
- String containing a predefined color name

For example, to use cyan, enter `[0 1 1]`, `'c'`, or `'cyan'`. For more information on specifying colors, see `ColorSpec`.

Default: Red, which is specified by `[1 0 0]`, `'r'`, or `'red'`

SimilarColorValue

Color to highlight similar characters in the output display. Specify the color with one of the following:

- Three-element numeric vector of RGB values
- String containing a predefined single-letter color code

•   String containing a predefined color name

For example, to use cyan, enter [0 1 1], 'c', or 'cyan'. For more information on specifying colors, see ColorSpec.

Default: Magenta, which is specified by [1 0 1], 'm', or 'magenta'

StartPointersValue

Two-element vector that specifies the starting indices in the original sequences of a local pairwise alignment.

Tip You can use the third output returned by swalign as the *StartPointersValue*.

Default:

ColumnsValue

Scalar that specifies the number of characters to display in one row when displaying a pairwise alignment.

Default: 64

TerminalGapValue

Specifies whether to include or exclude terminal gaps from the count of matches and similar residues when displaying a pairwise alignment. Choices are true (default) or false.

Default:

Examples

Display a pairwise sequence alignment:

```
% Globally align two amino acid sequences
[Score, Alignment] = nwalign('VSPAGMASGYD','IPGKASYD');
% Display the color-coded alignment
showalignment(Alignment);
```
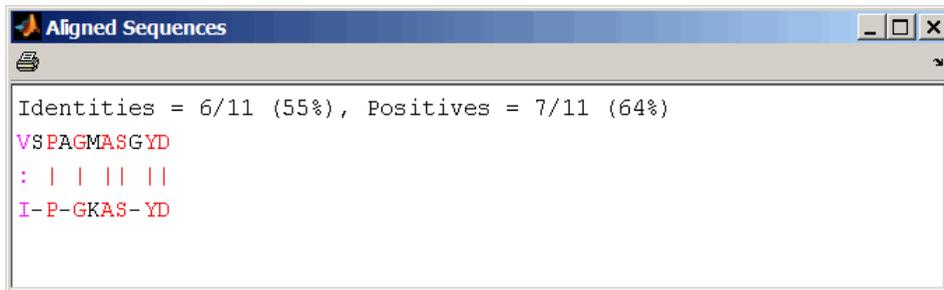
Notice that for pairwise sequence alignments, matching and similar characters appear in red and magenta respectively..

Display a multiple sequence alignment

```
% Read a multiple-sequence alignment file
gag = multialignread('aagag.aln');
% Display the color-coded alignment
showalignment(gag)
```
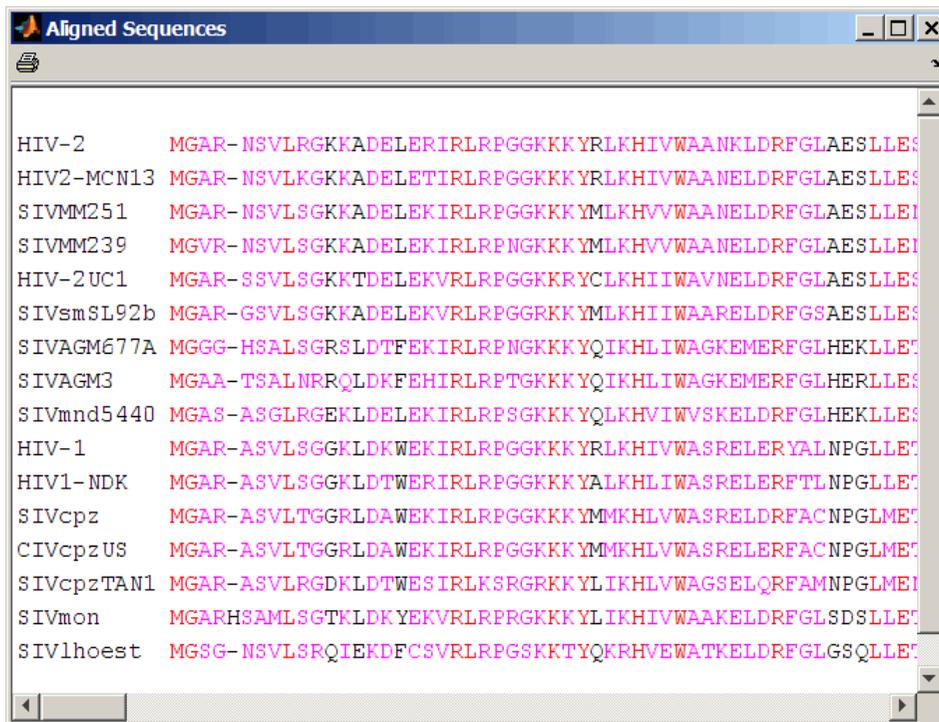
Notice that for multiple sequence alignments, highly conserved positions appear in red and conserved positions appear in magenta.

Tip To view a multiple-sequence alignment and interact with it, use the `seqalignviewer` function.

Alternatives

You can also display a multiple or pairwise sequence alignment using the `seqalignviewer` function. The alignment displays in the Biological Sequence Alignment window, where you can view and interactively adjust a sequence alignment.

See Also

multialign | seqalignviewer | nwalign | swalign | ColorSpec | gethmmalignment | fastaread | multialignread | localalign

showhmmprof

Plot hidden Markov model (HMM) profile

Syntax

```
showhmmprof(Model)
showhmmprof(Model, ...'Scale', ScaleValue, ...)
showhmmprof(Model, ...'Order', OrderValue, ...)
```

Arguments

Model	Hidden Markov model created by the function `gethmmprof` or `pfamhmmread`.
ScaleValue	Property to select a probability scale. Enter one of the following values: • `'logprob'` — Log probabilities • `'prob'` — Probabilities • `'logodds'` — Log-odd ratios
OrderValue	Property to specify the order of the amino acid alphabet. Enter a character string with the 20 standard amino acids characters `A R N D C Q E G H I L K M F P S T W Y V`. The ambiguous characters `B Z X` are not allowed.

Description

`showhmmprof(Model)` plots a profile hidden Markov model described by the structure *Model*.

`showhmmprof(..., 'PropertyName', PropertyValue, ...)` calls `showhmmprof` with optional properties that use property name/property value pairs. You can specify

one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

showhmmprof(*Model*, ...'Scale', *ScaleValue*, ...) specifies the scale to use. If log probabilities (*ScaleValue*='logprob'), probabilities (*ScaleValue*='prob'), or log-odd ratios (*ScaleValue*='logodds'). To compute the log-odd ratios, the null model probabilities are used for symbol emission and equally distributed transitions are used for the null transition probabilities. The default *ScaleValue* is 'logprob'.

showhmmprof(*Model*, ...'Order', *OrderValue*, ...) specifies the order in which the symbols are arranged along the vertical axis. This option allows you reorder the alphabet and group the symbols according to their properties.

Examples

1   Load a model example.

```
model = pfamhmmread('pf00002.ls');
```

2   Plot the profile.

```
showhmmprof(model, 'Scale', 'logodds')
```
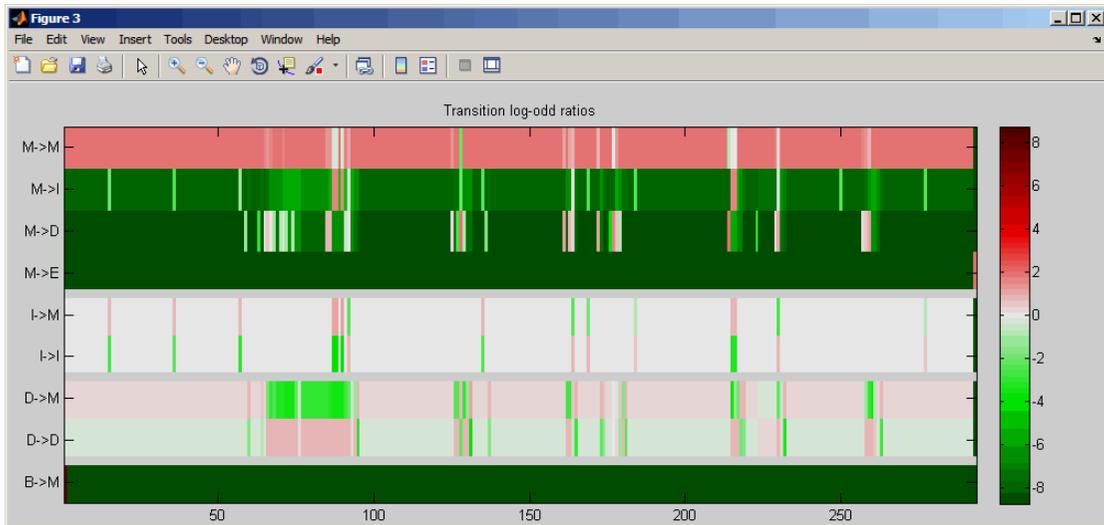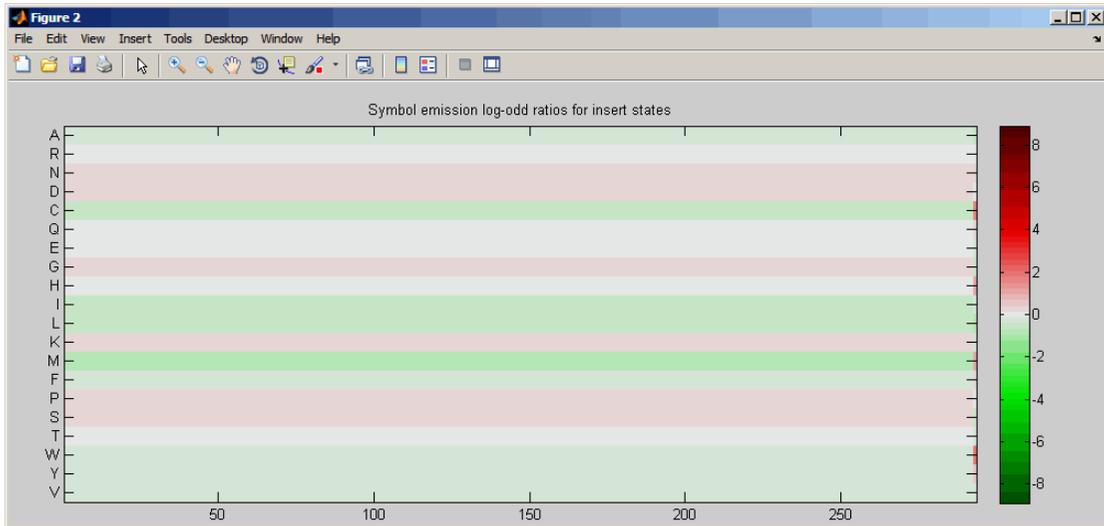
3   Order the alphabet by hydrophobicity.

```
hydrophobic = 'IVLFCMAGTSWYPHNDQEKR';
```

4   Plot the profile.

```
showhmmprof(model, 'Order', hydrophobic)
```
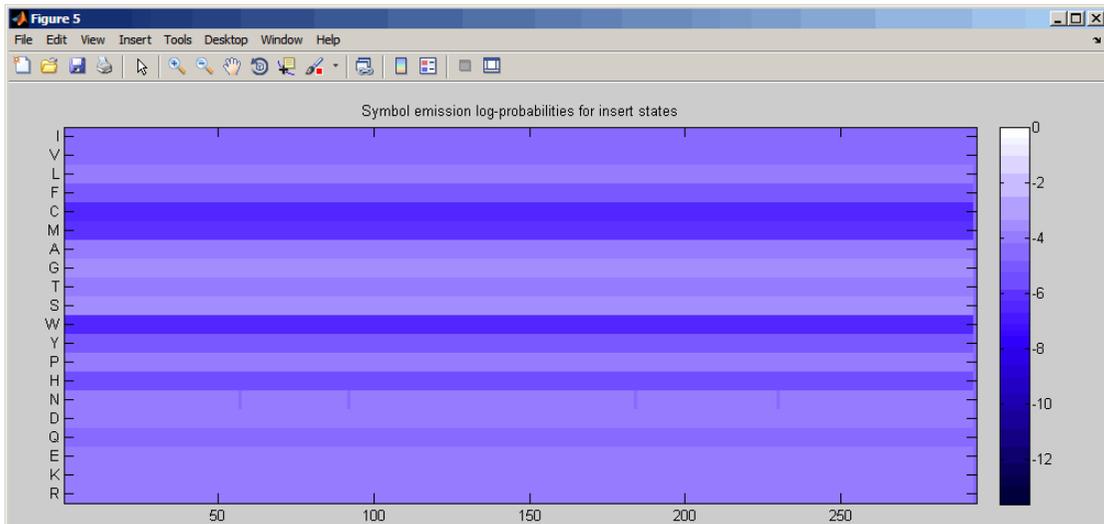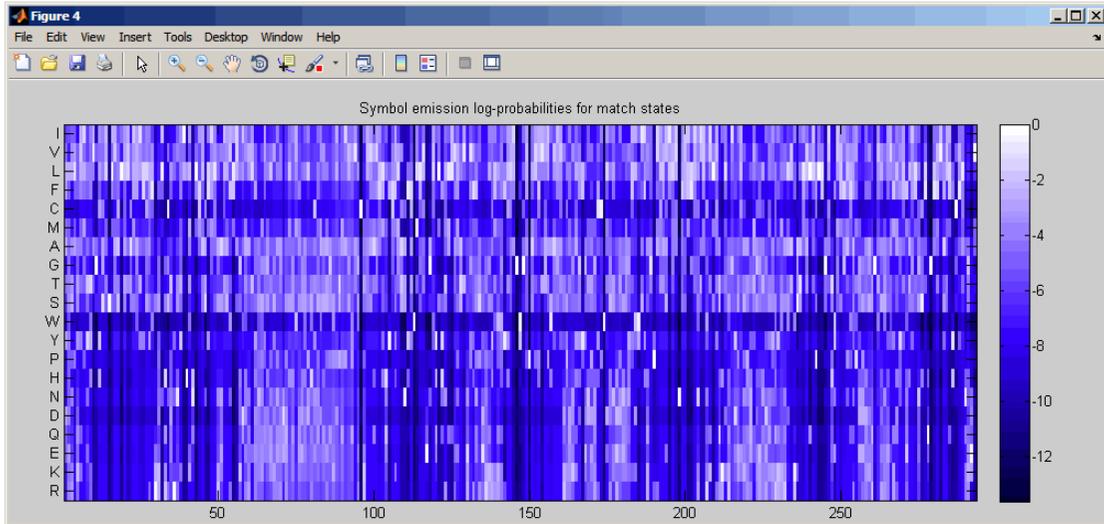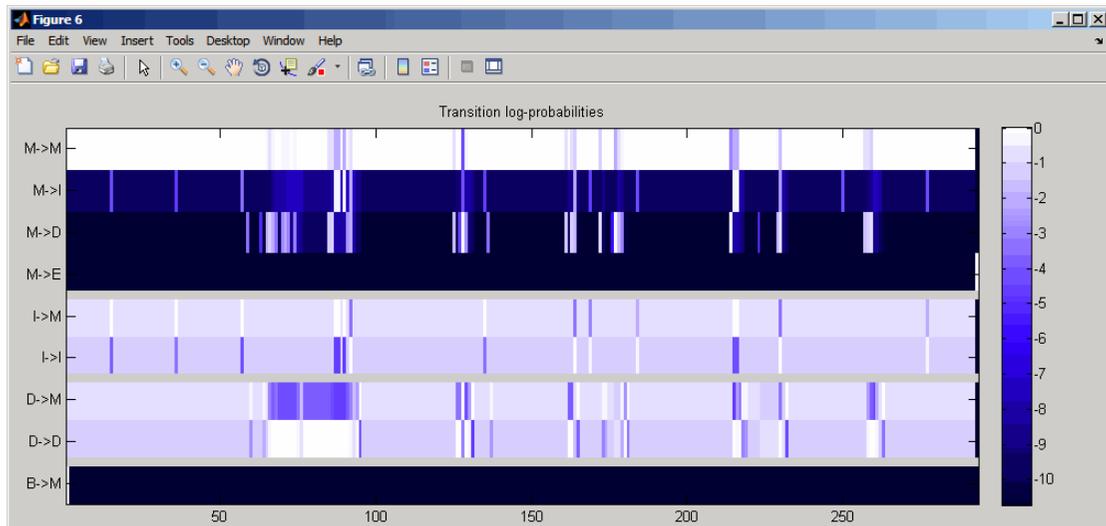
See Also
gethmmprof | hmmprofalign | hmmprofestimate | hmmprofgenerate | hmmprofstruct | pfamhmmread

single (DataMatrix)

Convert DataMatrix object to single-precision array

Syntax

```
B = single(DMObj)
B = single(DMObj, Rows)
B = single(DMObj, Rows, Cols)
```

Input Arguments

DMObj	DataMatrix object, such as created by `DataMatrix` (object constructor).
Rows, *Cols*	Row(s) or column(s) in *DMObj*, specified by one of the following: • Scalar • Vector of positive integers • String specifying a row or column name • Cell array of row or column names • Logical vector

Output Arguments

B	MATLAB numeric array.

Description

B = `single(`*DMObj*`)` converts *DMObj*, a DataMatrix object, to a single-precision array, which it returns in *B*.

B = `single(`*DMObj*`, `*Rows*`)` converts a subset of *DMObj*, a DataMatrix object, specified by *Rows*, to a single-precision array, which it returns in *B*. *Rows* can be a positive integer,

vector of positive integers, string specifying a row name, cell array of row names, or a logical vector.

B = single(*DMObj, Rows, Cols*) converts a subset of *DMObj*, a DataMatrix object, specified by *Rows* and *Cols*, to a single-precision array, which it returns in *B*. *Cols* can be a positive integer, vector of positive integers, string specifying a column name, cell array of column names, or a logical vector.

More About

- DataMatrix object

See Also
DataMatrix | double

size

Class: bioma.ExpressionSet
Package: bioma

Return size of ExpressionSet object

Syntax

```
NFeatSam = size(ESObj)
[NFeatures, NSamples] = size(ESObj)
DimLength = size(ESObj, Dim)
```

Description

NFeatSam = size(*ESObj*) returns a two-element row vector containing the number of features and number of samples in an ExpressionSet object.

[*NFeatures*, *NSamples*] = size(*ESObj*) returns the number of features and number of samples in an ExpressionSet object as separate variables.

DimLength = size(*ESObj*, *Dim*) returns the length of the dimension specified by *Dim*.

Input Arguments

ESObj

Object of the `bioma.ExpressionSet` class.

Default:

Dim

Scalar specifying the dimension of the ExpressionSet object. Choices are:

- 1 — Features

- 2 — Samples

Default:

Examples

Construct an ExpressionSet object, `ESObj`, as described in the "Examples" on page 1-   section of the `bioma.ExpressionSet` class reference page. Determine the number of features and samples in the ExpressionSet object:

```
% Retrieve the number of features and samples
NumFeatSam = size(ESObj)
```

See Also

bioma.ExpressionSet | bioma.data.ExptData | DataMatrix

How To

- "Managing Gene Expression Data in Objects"

size

Class: bioma.data.ExptData
Package: bioma.data

Return size of ExptData object

Syntax

```
NFeatSam = size(EDObj)
[NFeatures, NSamples] = size(EDObj)
DimLength = size(EDObj, Dim)
```

Description

NFeatSam = size(*EDObj*) returns a two-element row vector containing the number of features and number of samples in an ExptData object.

[*NFeatures*, *NSamples*] = size(*EDObj*) returns the number of features and number of samples in an ExptData object as separate variables.

DimLength = size(*EDObj*, *Dim*) returns the length of the dimension specified by *Dim*.

Input Arguments

EDObj

Object of the `bioma.data.ExptData` class.

Default:

Dim

Scalar specifying the dimension of the ExptData object. Choices are:

- 1 — Features

- 2 — Samples

Default:

Examples

Construct an ExptData object, and then determine the number of features and samples in it:

```
% Import bioma.data package to make constructor functions
% available
import bioma.data.*
% Create DataMatrix object from .txt file containing
% expression values from microarray experiment
dmObj = DataMatrix('File', 'mouseExprsData.txt');
% Construct ExptData object
EDObj = ExptData(dmObj);
% Retrieve the number of features and samples
NumFeatSam = size(EDObj)
```

See Also
```
bioma.data.ExptData
```

How To

- "Representing Expression Data Values in ExptData Objects"

size

Class: bioma.data.MetaData
Package: bioma.data

Return size of MetaData object

Syntax

```
NSamVar = size(MDObj)
[NSamples, NVariables] = size(MDObj)
DimLength = size(MDObj, Dim)
```

Description

NSamVar = size(*MDObj*) returns a two-element row vector containing the number of samples or features and number of variables in a MetaData object.

[*NSamples*, *NVariables*] = size(*MDObj*) returns the number of samples or features and the number of variables in a MetaData object as separate variables.

DimLength = size(*MDObj*, *Dim*) returns the length of the dimension specified by *Dim*.

Input Arguments

MDObj

Object of the `bioma.data.MetaData` class.

Default:

Dim

Scalar specifying the dimension of the MetaData object. Choices are:

- 1 — Samples

- 2 — Variables

Default:

Examples

Construct a MetaData object, and then determine the number of samples and variables in it:

```
% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the number of samples and variables
NumSamVar = size(MDObj2)
```

See Also
bioma.data.MetaData

How To
- "Representing Sample and Feature Metadata in MetaData Objects"

soapread

Read data from Short Oligonucleotide Analysis Package (SOAP) file

Syntax

```
SOAPStruct = soapread(File)
SOAPStruct = soapread(File,Name,Value)
```

Description

`SOAPStruct = soapread(File)` reads File, a SOAP-formatted file (version 2.15) and returns the data in SOAPStruct, a MATLAB array of structures.

`SOAPStruct = soapread(File,Name,Value)` reads a SOAP-formatted file with additional options specified by one or more Name,Value pair arguments.

Input Arguments

File

Either of the following:

- String specifying a file name or path and file name of a SOAP-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the Current Folder.
- MATLAB string containing the text of a SOAP-formatted file.

The `soapread` function reads SOAP-formatted files (version 2.15).

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single

quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'BlockRead'

Scalar or vector that controls the reading of a single sequence entry or block of sequence entries from a SOAP-formatted file containing multiple sequences. Enter a scalar *N*, to read the *N*th entry in the file. Enter a 1-by-2 vector [*M1, M2*], to read a block of entries starting at the *M1* entry and ending at the *M2* entry. To read all remaining entries in the file starting at the *M1* entry, enter a positive value for *M1* and enter `Inf` for *M2*.

Default:

'AlignDetails'

Logical specifying whether or not to include the `AlignDetails` field in the SOAPStruct output argument. The `AlignDetails` field includes information on mismatches, insertions, and deletions in the alignment. Choices are `true` (default) or `false`.

Default: true

Output Arguments

SOAPStruct

An *N*-by-1 array of structures containing sequence alignment and mapping information from a SOAP-formatted file, where *N* is the number of alignment records stored in the SOAP-formatted file. Each structure contains the following fields.

Field	Description
QueryName	Name of aligned read sequence.
Sequence	String containing the letter representations of the read sequence. It is the reverse-complement if the read sequence aligns to the reverse strand of the reference sequence.
Quality	String containing the ASCII representation of the per-base quality score for the read sequence. The quality score is reversed if the

Field	Description
	read sequence aligns to the reverse strand of the reference sequence.
NumHits	The number of *total* instances where this read sequence aligned to an identical length of bases on another area of the reference sequence.
PairedEndSourceFile	Flag (a or b) specifying which source file to which the read sequence belongs. This field applies only to read sequences that are paired in the alignment.
Length	Scalar specifying the length of the read sequence.
Strand	+ or − specifying direction (forward or reverse) of reference sequence to which the read sequence aligns.
ReferenceName	Name or numeric ID of the reference sequence to which the read sequence aligns.
Position	Position (one-based offset) of the forward reference sequence where the left-most base of the alignment of the read sequence starts.
AlignDetails	Information on mismatches, insertions, and deletions in the alignment. For SOAP-formatted files v2.15, this field includes CIGAR strings.

Examples

Read the alignment records (entries) from the sample01.soap file into a MATLAB array of structures and access some of the data:

```
% Read the alignment records stored in the file sample01.soap
data = soapread('sample01.soap')

data =

17x1 struct array with fields:
```

```
     QueryName
     Sequence
     Quality
     NumHits
     PairedEndSourceFile
     Length
     Strand
     ReferenceName
     Position
     AlignDetails

% Access the quality score for the 6th entry
data(6).Quality

ans =

<>.>>>8>;:1>>>3>6>

% Determine the strand direction (forward or reverse) of the reference
% sequence to which the 12th entry aligns
data(12).Strand

ans =

-
```

Read a block of alignment records (entries) from the `sample01.soap` file into a MATLAB array of structures:

```
% Read a block of six entries from a SOAP file
data_5_10 = soapread('sample01.soap','blockread', [5 10])

data_5_10 =

6x1 struct array with fields:
     QueryName
     Sequence
     Quality
     NumHits
     PairedEndSourceFile
     Length
     Strand
     ReferenceName
     Position
     AlignDetails
```

More About

Tips

If your SOAP-formatted file is too large to read using available memory, try either of the following:

- Use the `BlockRead` name-value pair arguments to read a subset of entries.
- Create a `BioIndexedFile` object from the SOAP-formatted file (using `'TABLE'` for the *Format*), and then access the entries using methods of the `BioIndexedFile` class.
- "Work with Large Multi-Entry Text Files"
- Sequence Read Archive
- SOAP (Short Oligonucleotide Analysis Package)

References

[1] Li, R., Yu, C., Li, Y., Lam, T., Yiu, S., Kristiansen, K., and Wang, J. (2009). SOAP2: an improved ultrafast tool for short read alignment. Bioinformatics *25, 15*, 1966–1967.

[2] Li, R., Li, Y., Kristiansen, K., and Wang, J. (2008). SOAP: short oligonucleotide alignment program. Bioinformatics *24(5)*, 713–714.

See Also

samread | bamread | fastqread

sortcols (DataMatrix)

Sort columns of DataMatrix object in ascending or descending order

Syntax

```
DMObjNew = sortcols(DMObj1)
DMObjNew = sortcols(DMObj1, Row)
DMObjNew = sortcols(DMObj1, 'ColName')
DMObjNew = sortcols(DMObj1, ..., Mode)
[DMObjNew, Indices] = sortcols(DMObj1, ...)
```

Input Arguments

DMObj1	DataMatrix object, such as created by `DataMatrix` (object constructor).
Row	One or more rows in *DMObj1* by which to sort the columns. Choices are: • Positive integer • Vector of positive integers • String specifying a row name • Cell array of strings specifying multiple row names • Logical vector
`'ColName'`	String that specifies to sort the columns by the column names.
Mode	String specifying the order by which to sort the columns. Choices are `'ascend'` (default) or `'descend'`.

Output Arguments

DMObjNew	DataMatrix object created from sorting the columns of another DataMatrix object.
Indices	Index vector that links *DMObj1* to *DMObjNew*. In other words, *DMObjNew = DMObj1(:,idx)*.

Description

DMObjNew = sortcols(*DMObj1*) sorts the columns in *DMObj1* in ascending order based on the elements in the first row. For any columns that have equal elements in a row, sorting is based on the row immediately below.

DMObjNew = sortcols(*DMObj1*, *Row*) sorts the columns in *DMObj1* in ascending order based on the elements in the specified row. Any columns that have equal elements in the specified row are sorted based on the elements in the next specified row.

DMObjNew = sortcols(*DMObj1*, 'ColName') sorts the columns in *DMObj1* in ascending order according to the column names.

DMObjNew = sortcols(*DMObj1*, ..., *Mode*) specifies the order of the sort. *Mode* can be 'ascend' (default) or 'descend'.

[*DMObjNew*, *Indices*] = sortcols(*DMObj1*, ...) returns *Indices*, an index vector that links *DMObj1* to *DMObjNew*. In other words, *DMObjNew* = *DMObj1*(:,idx).

More About

- DataMatrix object

See Also

DataMatrix | sortrows

sortrows (DataMatrix)

Sort rows of DataMatrix object in ascending or descending order

Syntax

```
DMObjNew = sortrows(DMObj1)
DMObjNew = sortrows(DMObj1, Column)
DMObjNew = sortrows(DMObj1, 'RowName')
DMObjNew = sortrows(DMObj1, ..., Mode)
[DMObjNew, Indices] = sortrows(DMObj1, ...)
```

Input Arguments

DMObj1	DataMatrix object, such as created by `DataMatrix` (object constructor).
Column	One or more columns in *DMObj1* by which to sort the rows. Choices are: • Positive integer • Vector of positive integers • String specifying a column name • Cell array of strings specifying multiple column names • Logical vector
`'RowName'`	String that specifies to sort the rows by the row names.
Mode	String specifying the order by which to sort the rows. Choices are `'ascend'` (default) or `'descend'`.

Output Arguments

DMObjNew	DataMatrix object created from sorting the rows of another DataMatrix object.
Indices	Index vector that links *DMObj1* to *DMObjNew*. In other words, *DMObjNew = DMObj1*(idx,:).

Description

DMObjNew = sortrows(*DMObj1*) sorts the rows in *DMObj1* in ascending order based on the elements in the first column. For any rows that have equal elements in a column, sorting is based on the column immediately to the right.

DMObjNew = sortrows(*DMObj1, Column*) sorts the rows in *DMObj1* in ascending order based on the elements in the specified column. Any rows that have equal elements in the specified column are sorted based on the elements in the next specified column.

DMObjNew = sortrows(*DMObj1,* 'RowName') sorts the rows in *DMObj1* in ascending order according to the row names.

DMObjNew = sortrows(*DMObj1, ..., Mode*) specifies the order of the sort. *Mode* can be 'ascend' (default) or 'descend'.

[*DMObjNew, Indices*] = sortrows(*DMObj1, ...*) returns *Indices*, an index vector that links *DMObj1* to *DMObjNew*. In other words, *DMObjNew = DMObj1*(idx,:).

More About

· DataMatrix object

See Also

DataMatrix | sortcols

sptread

Read data from SPOT file

Syntax

```
SPOTData = sptread(File)
SPOTData = sptread(File, 'CleanColNames', CleanColNamesValue)
```

Arguments

File	Either of the following:
	• String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a SPOT-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Folder.
	• MATLAB character array that contains the text of a SPOT-formatted file.
CleanColNamesValue	Controls the use of valid MATLAB variable names.

Description

SPOTData = sptread(*File*) reads *File*, a SPOT-formatted file, and creates *SPOTData*, a MATLAB structure containing the following fields:

```
Header
Data
Blocks
Columns
Rows
IDs
ColumnNames
Indices
Shape
```

SPOTData = sptread(*File*, 'CleanColNames', *CleanColNamesValue*) controls the use of valid MATLAB variable names. The column names in the SPOT-formatted file contain periods and some characters that cannot be used in MATLAB variable names. If you plan to use the column names as variable names in a function, use this option with `CleanColNames` set to `true` and the function will return the field `ColumnNames` with valid variable names.

The `Indices` field of the structure includes the indices that you can use for plotting heat maps of the data.

Examples

1   Read in a sample SPOT file and plot the median foreground intensity for the 635 nm channel. Note that the example file `spotdata.txt` is not provided with the Bioinformatics Toolbox software.

```
spotStruct = sptread('spotdata.txt')
maimage(spotStruct,'Rmedian');
```

2   Alternately, create a similar plot using more basic graphics commands.

```
Rmedian = magetfield(spotStruct,'Rmedian');
imagesc(Rmedian(spotStruct.Indices));
colormap bone
colorbar
```

See Also

affyread | agferead | celintensityread | geoseriesread | geosoftread | gprread | ilmnbsread | imageneread | maboxplot | magetfield

std (DataMatrix)

Return standard deviation values in DataMatrix object

Syntax

```
S = std(DMObj)
S = std(DMObj, Flag)
S = std(DMObj, Flag, Dim)
S = std(DMObj, Flag, Dim, IgnoreNaN)
```

Input Arguments

DMObj	DataMatrix object, such as created by `DataMatrix` (object constructor).
Flag	Scalar specifying how to normalize the data. Choices are: • `0` — Default. Normalizes using a sample size of $N - 1$, unless $N = 1$, in which case, normalizes using a sample size of 1. • `1` — Normalizes using a sample size of N. N = the number of elements in each column or row, as specified by *Dim*. For more information on the normalization equations, see the MATLAB function `std`.
Dim	Scalar specifying the dimension of *DMObj* to calculate the standard deviations. Choices are: • `1` — Default. Returns standard deviation values for elements in each column. • `2` — Returns standard deviation values for elements in each row.
IgnoreNaN	Specifies if NaNs should be ignored. Choices are `true` (default) or `false`.

Output Arguments

S	Either of the following:
	• Row vector containing the standard deviation values from elements in each column in *DMObj* (when *Dim* = 1)
	• Column vector containing the standard deviation values from elements in each row in *DMObj* (when *Dim* = 2)

Description

S = std(*DMObj*) returns the standard deviation values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. The data is normalized using a sample size of $N - 1$, where N = the number of elements in each column. *S* is a row vector containing the standard deviation values for elements in each column in *DMObj*.

S = std(*DMObj*, *Flag*) specifies how to normalize the data. If *Flag* = 0, normalizes using a sample size of $N - 1$. If *Flag* = 1, normalizes using a sample size of N. N = the number of elements in each column or row, as specified by *Dim*. For more information on the normalization equations, see the MATLAB function std. Default *Flag* = 0.

S = std(*DMObj*, *Flag*, *Dim*) returns the standard deviation values of the elements in the columns or rows of a DataMatrix object, as specified by *Dim*. If *Dim* = 1, returns *S*, a row vector containing the standard deviation values for elements in each column in *DMObj*. If *Dim* = 2, returns *S*, a column vector containing the standard deviation values for elements in each row in *DMObj*. Default *Dim* = 1.

S = std(*DMObj*, *Flag*, *Dim*, *IgnoreNaN*) specifies if NaNs should be ignored. *IgnoreNaN* can be true (default) or false.

More About

• DataMatrix object

See Also
DataMatrix | median | mean | var

subtree (phytree)

Extract phylogenetic subtree

Syntax

Tree2 = subtree(*Tree1*, *Nodes*)

Description

Tree2 = subtree(*Tree1*, *Nodes*) extracts a new subtree (*Tree2*) where the new root is the first common ancestor of the *Nodes* vector from *Tree1*. Nodes in the tree are indexed as [1:NUMLEAVES] for the leaves and as [NUMLEAVES+1:NUMLEAVES +NUMBRANCHES] for the branches. Nodes can also be a logical array of following sizes [NUMLEAVES+NUMBRANCHES x 1], [NUMLEAVES x 1] or [NUMBRANCHES x 1].

Examples

1   Load a phylogenetic tree created from a protein family.

```
tr = phytreeread('pf00002.tree');
```

2   Get the subtree that contains the VIPR2 and GLR human proteins.

```
sel = getbyname(tr,{'vipr2_human','glr_human'});
sel = any(sel,2);
tr =  subtree(tr,sel);
view(tr)
```
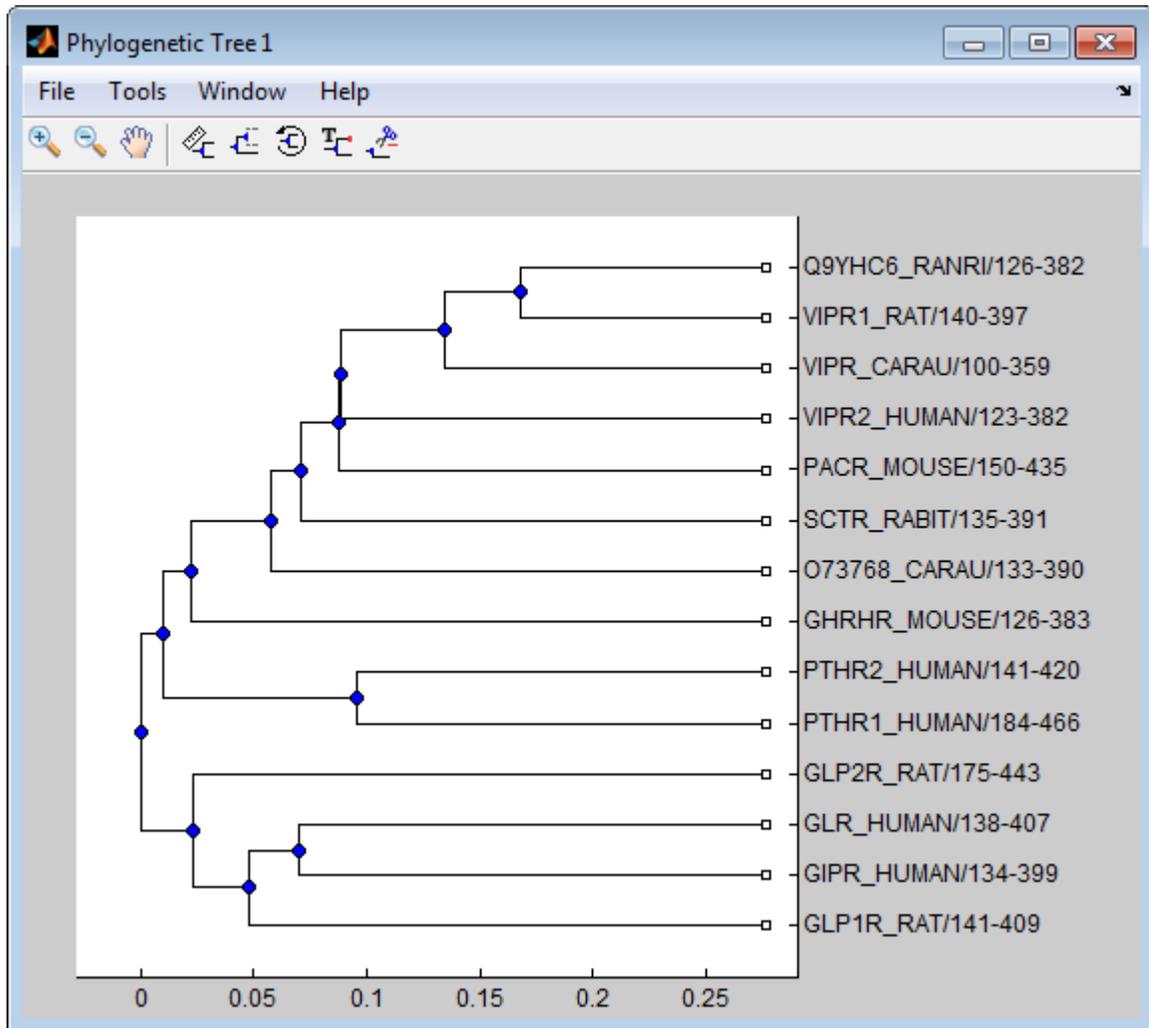
More About

· phytree object

See Also
phytree | getbyname | select | get | prune

sum (DataMatrix)

Return sum of elements in DataMatrix object

Syntax

S = sum(*DMObj*)
S = sum(*DMObj*, *Dim*)
S = sum(*DMObj*, *Dim*, *IgnoreNaN*)

Input Arguments

DMObj	DataMatrix object, such as created by `DataMatrix` (object constructor).
Dim	Scalar specifying the dimension of *DMObj* to calculate the sums. Choices are: • `1` — Default. Returns sum of elements in each column. • `2` — Returns sum of elements in each row.
IgnoreNaN	Specifies if NaNs should be ignored. Choices are `true` (default) or `false`.

Output Arguments

S	Either of the following: • Row vector containing the sums of the elements in each column in *DMObj* (when *Dim* = 1) • Column vector containing the sums of the elements in each row in *DMObj* (when *Dim* = 2)

Description

S = sum(*DMObj*) returns the sum of the elements in the columns of a DataMatrix object, treating NaNs as missing values. *S* is a row vector containing the sums of the elements in each column in *DMObj*. If the values in *DMObj* are singles, then *S* is a single; otherwise, *S* is a double.

S = sum(*DMObj*, *Dim*) returns the sum of the elements in the columns or rows of a DataMatrix object, as specified by *Dim*. If *Dim* = 1, returns *S*, a row vector containing the sums of the elements in each column in *DMObj*. If *Dim* = 2, returns *S*, a column vector containing the sums of the elements in each row in *DMObj*. Default *Dim* = 1.

S = sum(*DMObj*, *Dim*, *IgnoreNaN*) specifies if NaNs should be ignored. *IgnoreNaN* can be true (default) or false.

More About

- DataMatrix object

See Also

DataMatrix | min | max

swalign

Locally align two sequences using Smith-Waterman algorithm

Syntax

```
Score = swalign(Seq1, Seq2)
[Score, Alignment] = swalign(Seq1, Seq2)
[Score, Alignment, Start] = swalign(Seq1, Seq2)
... = swalign(Seq1,Seq2, ...'Alphabet', AlphabetValue)
... = swalign(Seq1,Seq2, ...'ScoringMatrix',
ScoringMatrixValue, ...)
... = swalign(Seq1,Seq2, ...'Scale', ScaleValue, ...)
... = swalign(Seq1,Seq2, ...'GapOpen', GapOpenValue, ...)
... = swalign(Seq1,Seq2, ...'ExtendGap', ExtendGapValue, ...)
... = swalign(Seq1,Seq2, ...'Showscore', ShowscoreValue, ...)
```

Input Arguments

Seq1, *Seq2*	Amino acid or nucleotide sequences. Enter any of the following: • Character string of letters representing amino acids or nucleotides, such as returned by `int2aa` or `int2nt` • Vector of integers representing amino acids or nucleotides, such as returned by `aa2int` or `nt2int` • Structure containing a `Sequence` field **Tip** For help with letter and integer representations of amino acids and nucleotides, see Amino Acid Lookup or Nucleotide Lookup.
AlphabetValue	String specifying the type of sequence. Choices are `'AA'` (default) or `'NT'`.
ScoringMatrixValue	Either of the following:

- String specifying the scoring matrix to use for the local alignment. Choices for amino acid sequences are:

  - `'BLOSUM62'`
  - `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'`
  - `'BLOSUM100'`
  - `'PAM10'` increasing by 10 up to `'PAM500'`
  - `'DAYHOFF'`
  - `'GONNET'`

  Default is:

  - `'BLOSUM50'` — When *AlphabetValue* equals `'AA'`
  - `'NUC44'` — When *AlphabetValue* equals `'NT'`

  Note: The above scoring matrices, provided with the software, also include a structure containing a scale factor that converts the units of the output score to bits. You can also use the `'Scale'` property to specify an additional scale factor to convert the output score from bits to another unit.

- Matrix representing the scoring matrix to use for the local alignment, such as returned by the `blosum`, `pam`, `dayhoff`, `gonnet`, or `nuc44` function.

  Note: If you use a scoring matrix that you created or was created by one of the above functions, the matrix does not include a scale factor. The output score will be returned in the same units as the scoring matrix. You can use the `'Scale'` property to specify a scale factor to convert the output score to another unit.

Note: If you need to compile `swalign` into a stand-alone application or software component using

	MATLAB Compiler, use a matrix instead of a string for *ScoringMatrixValue*.
ScaleValue	Positive value that specifies a scale factor that is applied to the output score. For example, if the output score is initially determined in bits, and you enter `log(2)` for *ScaleValue*, then swalign returns *Score* in nats. Default is 1, which does not change the units of the output score. **Note:** If the `'ScoringMatrix'` property also specifies a scale factor, then swalign uses it first to scale the output score, then applies the scale factor specified by *ScaleValue* to rescale the output score. **Tip** Before comparing alignment scores from multiple alignments, ensure the scores are in the same units. You can use the `'Scale'` property to control the units of the output scores.
GapOpenValue	Positive value specifying the penalty for opening a gap in the alignment. Default is 8.
ExtendGapValue	Positive value specifying the penalty for extending a gap using the affine gap penalty scheme. **Note:** If you specify this value, swalign uses the affine gap penalty scheme, that is, it scores the first gap using the *GapOpenValue* and scores subsequent gaps using the *ExtendGapValue*. If you do not specify this value, swalign scores all gaps equally, using the *GapOpenValue* penalty.
ShowscoreValue	Controls the display of the scoring space and the winning path of the alignment. Choices are `true` or `false` (default).

Output Arguments

Score	Optimal local alignment score in bits.
Alignment	3-by-N character array showing the two sequences, *Seq1* and *Seq2*, in the first and third rows, and symbols representing the optimal local alignment between them in the second row.
Start	2-by-1 vector of indices indicating the starting point in each sequence for the alignment.

Description

Score = swalign(*Seq1, Seq2*) returns the optimal local alignment score in bits. The scale factor used to calculate the score is provided by the scoring matrix.

[*Score, Alignment*] = swalign(*Seq1, Seq2*) returns a 3-by-N character array showing the two sequences, *Seq1* and *Seq2*, in the first and third rows, and symbols representing the optimal local alignment between them in the second row. The symbol | indicates amino acids or nucleotides that match exactly. The symbol : indicates amino acids or nucleotides that are related as defined by the scoring matrix (nonmatches with a zero or positive scoring matrix value).

[*Score, Alignment, Start*] = swalign(*Seq1, Seq2*) returns a 2-by-1 vector of indices indicating the starting point in each sequence for the alignment.

... = swalign(*Seq1,Seq2*, ...'*PropertyName*', *PropertyValue*, ...) calls swalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = swalign(*Seq1,Seq2*, ...'Alphabet', *AlphabetValue*) specifies the type of sequences. Choices are 'AA' (default) or 'NT'.

... = swalign(*Seq1,Seq2*, ...'ScoringMatrix', *ScoringMatrixValue*, ...) specifies the scoring matrix to use for the local alignment. Default is:

- `'BLOSUM50'` — When *AlphabetValue* equals `'AA'`
- `'NUC44'` — When *AlphabetValue* equals `'NT'`

`... = swalign(`*Seq1*`,`*Seq2*`, ...'Scale',` *ScaleValue*`, ...)` specifies a scale factor that is applied to the output score, thereby controlling the units of the output score. Choices are any positive value.

`... = swalign(`*Seq1*`,`*Seq2*`, ...'GapOpen',` *GapOpenValue*`, ...)` specifies the penalty for opening a gap in the alignment. Choices are any positive value. Default is `8`.

`... = swalign(`*Seq1*`,`*Seq2*`, ...'ExtendGap',` *ExtendGapValue*`, ...)` specifies the penalty for extending a gap using the affine gap penalty scheme. Choices are any positive value.

`... = swalign(`*Seq1*`,`*Seq2*`, ...'Showscore',` *ShowscoreValue*`, ...)` controls the display of the scoring space and winning path of the alignment. Choices are `true` or `false` (default).
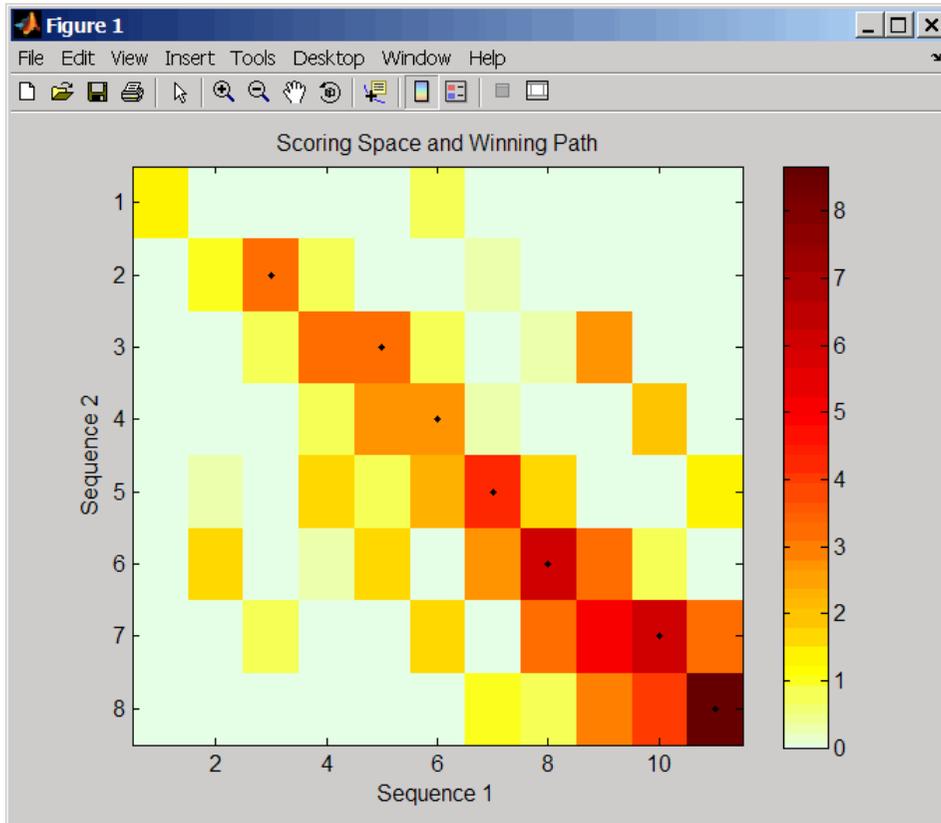
The scoring space is a heat map displaying the best scores for all the partial alignments of two sequences. The color of each (`n1`,`n2`) coordinate in the scoring space represents the best score for the pairing of subsequences `Seq1(s1:n1)` and `Seq2(s2:n2)`, where `n1` is a position in `Seq1`, `n2` is a position in `Seq2`, `s1` is any position in `Seq1` between `1:n1`, and `s2` is any position in `Seq2` between `1:n2`. The best score for a pairing of specific subsequences is determined by scoring all possible alignments of the subsequences by summing matches and gap penalties.

The winning path is represented by black dots in the scoring space, and it illustrates the pairing of positions in the optimal local alignment. The color of the last point (lower right) of the winning path represents the optimal local alignment score for the two sequences and is the *Score* output returned by `swalign`.

> **Note:** The scoring space visually shows tandem repeats, small segments that potentially align, and partial alignments of domains from rearranged sequences.

Examples

1 Locally align two amino acid sequences using the BLOSUM50 (default) scoring matrix and the default values for the GapOpen and ExtendGap properties. Return the optimal local alignment score in bits and the alignment character array.

```
[Score, Alignment] = swalign('VSPAGMASGYD','IPGKASYD')

Score =

    8.6667

Alignment =

PAGMASGYD
| |  || ||
P-GKAS-YD
```

2 Locally align two amino acid sequences specifying the PAM250 scoring matrix and a gap open penalty of 5.

```
[Score, Alignment] = swalign('HEAGAWGHEE','PAWHEAE',...
                             'ScoringMatrix', 'pam250',...
                             'GapOpen',5)

Score =

    8
Alignment =

GAWGHE
:|| ||
PAW-HE
```

3 Locally align two amino acid sequences returning the *Score* in nat units (nats) by specifying a scale factor of log(2).

```
[Score, Alignment] = swalign('HEAGAWGHEE','PAWHEAE','Scale',log(2))

Score =
```

```
    6.4694

Alignment =

AWGHE
|| ||
AW-HE
```

References

[1] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). Biological Sequence Analysis (Cambridge University Press).

[2] Smith, T., and Waterman, M. (1981). Identification of common molecular subsequences. Journal of Molecular Biology *147*, 195–197.

See Also

aa2int | aminolookup | baselookup | blosum | dayhoff | gonnet | int2aa | int2nt | localalign | multialign | nt2aa | nt2int | nuc44 | nwalign | pam | pdbsuperpose | seqdotplot | showalignment

term class

Data structure containing information about Gene Ontology (GO) term

Description

A term object is a data structure containing information about a Gene Ontology (GO) term. You can explore and traverse Gene Ontology terms using "is_a" and "part_of" relationships.

Construction

.geneont

Create geneont object and term objects

Properties

definition

Read-only string that defines GO term

id

Read-only numeric value that corresponds to GO identifier of GO term

is_a

Read-only numeric array containing GO identifiers of GO terms that have an "is a" relationship with this GO term

name

Read-only string representing name of GO term

obsolete

Read-only Boolean value that indicates whether a GO term is obsolete

ontology	Read-only string describing the ontology of GO term
part_of	Read-only numeric array containing GO identifiers of GO terms that have a "part of" relationship with this GO term
synonym	Read-only array containing GO terms that are synonyms of this GO term

Instance Hierarchy

A geneont object contains term objects.

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Indexing

You can use parenthesis () indexing to access the terms in an array of handles to term objects. See "Examples" on page 1-1826 below.

Examples

1   Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27827 Terms.
```

2 Use the `terms` property to create a variable containing an array of handles to the term objects of the geneont object.

```
array_of_terms = GeneontObj.terms

27827x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

Note: Although the `terms` property is an array of handles to term objects, in the MATLAB Command Window, it displays as a structure array, with one structure for each GO term in the geneont object.

3 Return the fifth term (term object) of the geneont object.

```
fifth_term = array_of_terms(5)

          id: 6
        name: [1x60 char]
    ontology: 'molecular function'
  definition: [1x321 char]
     comment: ''
     synonym: []
        is_a: 5385
     part_of: [0x1 double]
     obsolete: 0
```

See Also
geneont

tgspcinfo

Return information about SPC file

Syntax

InfoStruct = tgspcinfo(*File*)

Description

InfoStruct = tgspcinfo(*File*) returns a MATLAB structure containing summary information about a Galactic SPC file from Thermo Scientific®.

Input Arguments

File

String specifying a file name or path and file name of an SPC file. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

Default:

Output Arguments

InfoStruct

MATLAB structure containing the following fields:

Field	Description
Filename	Name of the SPC file.
FileSize	Size of the SPC file in bytes.

Field	Description
ExperimentType	Experimental technique used to create the data.
NumDataPoints	Number of data points (*y* data values) in the SPC file.
XFirst	First *x* data value in the SPC file.
XLast	Last *x* data value in the SPC file.
NumScans	Number of scans or subfiles in the SPC file.
XLabel	Label for the *x* data values.
YLabel	Label for the *y* data values.
ZLabel	Label for the *z* data values.
CollectionTime	Date and time the scans were collected.
CollectionTimeDatenum	Date and time the scans were collected in serial date number format. For more information, see datenum.
Resolution	Instrument resolution.
SourceInstrument	Name or model of the instrument used to collect data.
InterferogramPeakPointNumber	Peak point number for interferograms. It is 0 for scans that are not interferograms.
Comment	User-provided comments.
CustomAxisUnitLabel	User-provided labels for the axis units.
SubScanHeaders	Header information for subfiles or scans, including scan index, next scan index, and *w* data value.
ZValues	Vector containing the *z* data values of all scans in the SPC file.

Examples

The SPC file, sample.spc, used in the following example is not provided with the Bioinformatics Toolbox software. You can download sample files from:

https://ftirsearch.com/default3.htm

Return information about an SPC file:

```
% Return information about an SPC file named sample.spc
info = tgspcinfo('sample.spc')

Reading header for file: SAMPLE.SPC
File contains 1 scans

info =

                          Filename: 'SAMPLE.SPC'
                          FileSize: 48380
                    ExperimentType: 'General SPC'
                     NumDataPoints: 12031
                            XFirst: 6.2998e+003
                             XLast: 499.9531
                          NumScans: 1
                            XLabel: 'Wavenumber (cm-1)'
                            YLabel: 'Absorbance'
                            ZLabel: 'Arbitrary'
                     CollectionTime: '08-Mar-1993 15:13:00'
              CollectionTimeDatenum: 7.2800e+005
                        Resolution: '  .00   '
                   SourceInstrument: ''
      InterferogramPeakPointNumber: 0
                           Comment: [1x74 char]
               CustomAxisUnitLabel: ''
                     SubScanHeaders: [1x1 struct]
                            ZValues: 0
```

More About

- https://ftirsearch.com/default3.htm
- https://ftirsearch.com/features/converters/SPCFileFormat.htm
- http://www.thermofisher.com/

See Also

`tgspcread | datenum`

tgspcread

Read data from SPC file

Syntax

SPCStruct = tgspcread(*File*)
tgspcread(..., 'ZRange', *ZRangeValue*, ...)
tgspcread(..., 'ScanIndices', *ScanIndicesValue*, ...)
tgspcread(..., 'Verbose', *VerboseValue*, ...)

Description

SPCStruct = tgspcread(*File*) reads a Galactic SPC file from Thermo Scientific, and returns the data in a MATLAB structure.

tgspcread(..., '*PropertyName*', *PropertyValue*, ...) calls tgspcread with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Enclose each *PropertyName* in single quotation marks. Each *PropertyName* is case insensitive. These property name/property value pairs are as follows:

tgspcread(..., 'ZRange', *ZRangeValue*, ...) specifies a range of *z* data values in the SPC file from which to extract scans.

tgspcread(..., 'ScanIndices', *ScanIndicesValue*, ...) specifies a scan, multiple scans, or range of scans in the SPC file to read.

tgspcread(..., 'Verbose', *VerboseValue*, ...) controls the display of the progress of the reading of the SPC file. Choices are true or false (default).

Input Arguments

File

String specifying a file name or path and file name of an SPC file that conforms to the Thermo Scientific Universal Data Format Specification. If you specify only a file name, that file must be on the MATLAB search path or in the current folder.

Default:

ZRangeValue

Two-element numeric array [*Start End*] that specifies the range of *z* data values in *File* to read. *Start* and *End* must be positive scalars, and *Start* must be less than *End*. Default is to extract all scans.

Tip For summary information about the *z* data values in an SPC file, use the tgspcinfo function.

Note: If you specify a *ZRangeValue*, you cannot specify a *ScanIndicesValue*.

Default:

ScanIndicesValue

Positive integer, vector of integers, or a two-element numeric array [*Start_Ind*: *End_Ind*] that specifies a scan, multiple scans, or a range of scans in *File* to read. *Start_Ind* and *End_Ind* are each positive integers indicating a scan index. *Start_Ind* must be less than *End_Ind*. Default is to read all scans.

Tip For summary information about the scan indices in an SPC file, check the NumScans field in the structure returned by the tgspcinfo function.

Note: If you specify a *ScanIndicesValue*, you cannot specify a *ZRangeValue*.

Default:

VerboseValue

Controls the display of the progress of the reading of *File*. Choices are true or false (default).

Default:

Output Arguments

SPCStruct

Structure containing information from an SPC file. The structure contains the following fields.

Field	Description
Header	Structure containing the following fields: • Filename — Name of the SPC file. • FileSize — Size of the SPC file in bytes. • ExperimentType — Experimental technique used to create the data. • NumDataPoints — Number of data points (*y* data values) in the SPC file. • XFirst — First *x* data value in the SPC file. • XLast — Last *x* data value in the SPC file. • NumScans — Number of scans or subfiles in the SPC file. • XLabel — Label for the *x* data values. • YLabel — Label for the *y* data values. • ZLabel — Label for the *z* data values. • CollectionTime — Date and time the scan data were collected. • CollectionTimeDatenum — Date and time the scan data were collected in serial date number format. For more information, see datenum. • Resolution — Instrument resolution. • SourceInstrument — Name or model of instrument used to collect data. • InterferogramPeakPointNumber — Peak point number for interferograms. It is 0 for scans that are not interferograms. • Comment — User-provided comments.

Field	Description
	• `CustomAxisUnitLabel` — User-provided labels for the axis units.
	• `SubScanHeaders` — Header information for subfiles or scans, including scan index, next scan index, and w data value.
	• `ZValues` — Vector containing the z data values of all scans in the SPC file.
X	Vector or cell array containing the x data values.
	If all scans share the same x data values, then X is a vector. If scans have different x data values, then X is a cell array.
Y	Vector, matrix, or cell array containing the y data values.
	If there is only one scan, then Y is a vector. If there are multiple scans that share the same x data values, then Y is a matrix. If there are multiple scans having different x data values, then Y is a cell array.
Z	Vector containing the z data values of scans read from the SPC file

Examples

The SPC file, `results.spc`, used in the following example is not provided with the Bioinformatics Toolbox software. You can download sample files from:

`https://ftirsearch.com/default3.htm`

Read an SPC file:

```
% Read the contents of an SPC file into a MATLAB structure
out = tgspcread('results.spc')

File contains 1 scans

out =

    Header: [1x1 struct]
         X: [12031x1 single]
         Y: [12031x1 double]
```

```
          Z: 0
```

Plot an SPC file:

```
% Plot the first scan in the SPC file:
plot(out.X,out.Y(:,1));
```

More About

- https://ftirsearch.com/default3.htm
- https://ftirsearch.com/features/converters/SPCFileFormat.htm
- http://www.thermofisher.com/

See Also

tgspcinfo | jcampread | mzcdfinfo | mzcdf2peaks | mzcdfread | mzxmlread | mzxml2peaks | mzxmlinfo | datenum

times (DataMatrix)

Multiply DataMatrix objects

Syntax

DMObjNew = times(*DMObj1*, *DMObj2*)
DMObjNew = *DMObj1* .* *DMObj2*
DMObjNew = times(*DMObj1*, *B*)
DMObjNew = *DMObj1* .* *B*
DMObjNew = times(*B*, *DMObj1*)
DMObjNew = *B* .* *DMObj1*

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by DataMatrix (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

DMObjNew	DataMatrix object created by multiplication.

Description

DMObjNew = times(*DMObj1*, *DMObj2*) or the equivalent *DMObjNew* = *DMObj1* .* *DMObj2* performs an element-by-element multiplication of the DataMatrix objects *DMObj1* and *DMObj2* and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = times(*DMObj1*, *B*) or the equivalent *DMObjNew* = *DMObj1* .* *B* performs an element-by-element multiplication of the DataMatrix object *DMObj1* and *B*, a numeric or logical array, and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

DMObjNew = times(*B*, *DMObj1*) or the equivalent *DMObjNew* = *B* .* *DMObj1* performs an element-by-element multiplication of *B*, a numeric or logical array, and the DataMatrix object *DMObj1*, and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

Note: Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls *DMObjNew* = times(*X*, *Y*) for the syntax *DMObjNew* = *X* .* *Y* when *X* or *Y* is a DataMatrix object.

More About

- DataMatrix object

See Also
DataMatrix | plus | minus | "Arithmetic"

topoorder (biograph)

Perform topological sort of directed acyclic graph extracted from biograph object

Syntax

order = topoorder(*BGObj*)

Arguments

BGObj	Biograph object created by `biograph` (object constructor).

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

---

order = topoorder(*BGObj*) returns an index vector with the order of the nodes sorted topologically. In topological order, an edge can exist between a source node u and a destination node v, if and only if u appears before v in the vector *order*. *BGObj* is a biograph object from which an N-by-N adjacency matrix is extracted and represents a directed acyclic graph (DAG). In the N-by-N sparse matrix, all nonzero entries indicate the presence of an edge.

More About

- biograph object

References

[1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

biograph | allshortestpaths | isdag | isspantree | minspantree | traverse
| graphtopoorder | conncomp | isomorphism | maxflow | shortestpath

traceplot

Draw nucleotide trace plots

Syntax

```
traceplot(TraceStructure)
traceplot(A, C, G, T)
h = traceplot(...)
```

Description

traceplot(*TraceStructure*) creates a trace plot from data in a structure with fields A, C, G, and T.

traceplot(*A, C, G, T*) creates a trace plot from data in vectors *A*, *C*, *G*, and *T*.

h = traceplot(...) returns a structure with the handles of the lines corresponding to A, C, G, T.

Examples

1   Read trace data from an SCF-formatted file into a MATLAB structure.

```
tstruct = scfread('sample.scf')

tstruct =

    A: [10827x1 double]
    C: [10827x1 double]
    G: [10827x1 double]
    T: [10827x1 double]
```

2   Draw a nucleotide trace plot of the data.

```
traceplot(tstruct)
```
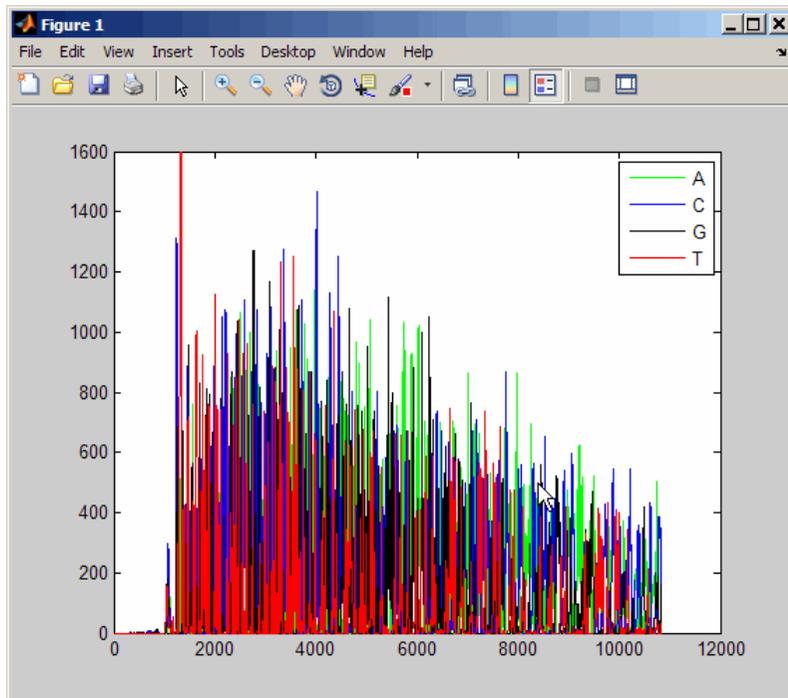
See Also

scfread

traverse (biograph)

Traverse biograph object by following adjacent nodes

Syntax

```
[disc, pred, closed] = traverse(BGObj, S)
[...] = traverse(BGObj, S, ...'Depth', DepthValue, ...)
[...] = traverse(BGObj, S, ...'Directed', DirectedValue, ...)
[...] = traverse(BGObj, S, ...'Method', MethodValue, ...)
```

Arguments

BGObj	Biograph object created by biograph (object constructor).
S	Integer that indicates the source node in *BGObj*.
DepthValue	Integer that indicates a node in *BGObj* that specifies the depth of the search. Default is Inf (infinity).
DirectedValue	Property that indicates whether graph represented by an N-by-N adjacency matrix extracted from a biograph object, *BGObj* is directed or undirected. Enter false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.
MethodValue	String that specifies the algorithm used to traverse the graph. Choices are: • 'BFS' — Breadth-first search. Time complexity is O(N+E), where N and E are number of nodes and edges respectively. • 'DFS' — Default algorithm. Depth-first search. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.

Description

Tip For introductory information on graph theory functions, see "Graph Theory Functions".

[*disc*, *pred*, *closed*] = traverse(*BGObj*, *S*) traverses the directed graph represented by an N-by-N adjacency matrix extracted from a biograph object, *BGObj*, starting from the node indicated by integer S. In the N-by-N sparse matrix, all nonzero entries indicate the presence of an edge. *disc* is a vector of node indices in the order in which they are discovered. *pred* is a vector of predecessor node indices (listed in the order of the node indices) of the resulting spanning tree. *closed* is a vector of node indices in the order in which they are closed.

[...] = traverse(*BGObj*, *S*, ...'*PropertyName*', *PropertyValue*, ...) calls traverse with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[...] = traverse(*BGObj*, *S*, ...'Depth', *DepthValue*, ...) specifies the depth of the search. *DepthValue* is an integer indicating a node in the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. Default is Inf (infinity).

[...] = traverse(*BGObj*, *S*, ...'Directed', *DirectedValue*, ...) indicates whether the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj* is directed or undirected. Set *DirectedValue* to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

[...] = traverse(*BGObj*, *S*, ...'Method', *MethodValue*, ...) lets you specify the algorithm used to traverse the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. Choices are:

- 'BFS' — Breadth-first search. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.

- 'DFS' — Default algorithm. Depth-first search. Time complexity is O(N+E), where N and E are number of nodes and edges respectively.

More About

- biograph object

References

[1] Sedgewick, R., (2002). Algorithms in C++, Part 5 Graph Algorithms (Addison-Wesley).

[2] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

biograph | allshortestpaths | isdag | isspantree | minspantree | topoorder | graphtraverse | conncomp | isomorphism | maxflow | shortestpath

var (DataMatrix)

Return variance values in DataMatrix object

Syntax

```
V = var(DMObj)
V = var(DMObj, Flag)
V = var(DMObj, Wgt)
V = var(..., Dim)
V = var(..., Dim, IgnoreNaN)
```

Input Arguments

DMObj	DataMatrix object, such as created by DataMatrix (object constructor).
Flag	Scalar specifying how to normalize the data. Choices are: • 0 — Default. Normalizes using a sample size of $N - 1$, unless $N = 1$, in which case, normalizes using a sample size of 1. • 1 — Normalizes using a sample size of N. N = the number of elements in each column or row, as specified by *Dim*. For more information on the normalization equations, see the MATLAB function std.
Wgt	Weight vector equal in length to the dimension over which var operates (specified by *Dim*. It is used to compute the variance.
Dim	Scalar specifying the dimension of *DMObj* to calculate the variances. Choices are: • 1 — Default. Returns variance values for elements in each column. • 2 — Returns variance values for elements in each row.
IgnoreNaN	Specifies if NaNs should be ignored. Choices are true (default) or false.

Output Arguments

V	An unbiased estimator of the variance within the columns or rows of a DataMatrix object. It can be either of the following: • Row vector containing the variance values from elements in each column in *DMObj* (when *Dim* = 1) • Column vector containing the variance values from elements in each row in *DMObj* (when *Dim* = 2)

Description

V = var(*DMObj*) returns the variance values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. The data is normalized using a sample size of $N-1$, where N = the number of elements in each column. *V* is a row vector containing the variance values for elements in each column in *DMObj*. The variance is the square of the standard deviation.

V = var(*DMObj*, *Flag*) specifies how to normalize the data. If *Flag* = 0, normalizes using a sample size of $N-1$. If *Flag* = 1, normalizes using a sample size of N. N = the number of elements in each column or row, as specified by *Dim*. For more information on the normalization equations, see the MATLAB function std. Default *Flag* = 0.

V = var(*DMObj*, *Wgt*) computes the variance using *Wgt*, a weight vector whose length must equal the length of the dimension over which var operates (specified by *Dim*). All elements in *Wgt* must be nonnegative. The var function normalizes *Wgt* to sum of 1.

V = var(..., *Dim*) returns the variance values of the elements in the columns or rows of a DataMatrix object, as specified by *Dim*. If *Dim* = 1, returns *V*, a row vector containing the variance values for elements in each column in *DMObj*. If *Dim* = 2, returns *V*, a column vector containing the variance values for elements in each row in *DMObj*. Default *Dim* = 1.

V = var(..., *Dim*, *IgnoreNaN*) specifies if NaNs should be ignored. *IgnoreNaN* can be true (default) or false.

More About

• DataMatrix object

See Also
DataMatrix | median | mean | std

variableDesc

Class: bioma.data.MetaData
Package: bioma.data

Retrieve or set variable descriptions for samples in MetaData object

Syntax

DSVarDescriptions = variableDesc(*MDObj*)
NewMDObj = variableDesc(*MDObj*, *NewDSVarDescriptions*)

Description

DSVarDescriptions = variableDesc(*MDObj*) returns a dataset array containing the variable names and descriptions for samples from a MetaData object.

NewMDObj = variableDesc(*MDObj*, *NewDSVarDescriptions*) replaces the sample variable descriptions in *MDObj*, a MetaData object, with *NewDSVarDescriptions*, and returns *NewMDObj*, a new MetaData object.

Input Arguments

MDObj

Object of the `bioma.data.MetaData` class.

Default:

NewDSVarDescriptions

Descriptions of the sample variable names, specified by one of the following:

- A new dataset array containing the variable names and descriptions for samples. In this dataset array, each row corresponds to a variable. The first column contains the variable name, and the second column (`VariableDescription`) contains a description of the variable. The row names (variable names) must match the row

names (variable names) in *DSVarDescriptions*, the dataset array being replaced in the MetaData object, *MDObj*.

- Cell array of strings containing descriptions of the variables. The number of elements in *VarDesc* must equal the number of row names (variable names) in *DSVarDescriptions*, the dataset array being replaced in the MetaData object, *MDObj*.

Default:

Output Arguments

DSVarDescriptions

A dataset array containing the variable names and descriptions from a MetaData object. In this dataset array, each row corresponds to a sample variable. The first column contains the variable name, and the second column (VariableDescription) contains a description of the variable.

NewMDObj

Object of the bioma.data.MetaData class, returned after replacing the dataset array containing the sample variable descriptions.

Examples

Construct a MetaData object, and then retrieve the sample variable descriptions from it:

```
% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the sample variable descriptions
VarDescriptions = variableDesc(MDObj2)
```

See Also

bioma.data.MetaData | sampleNames | variableValues | variableNames

How To

- "Representing Sample and Feature Metadata in MetaData Objects"

variableNames

Class: bioma.data.MetaData
Package: bioma.data

Retrieve or set variable names for samples in MetaData object

Syntax

```
VarNames = variableNames(MDObj)
VarNames = variableNames(MDObj, Subset)
NewMDObj = variableNames(MDObj, Subset, NewVarNames)
```

Description

VarNames = variableNames(*MDObj*) returns a cell array of strings specifying all variable names in a MetaData object.

VarNames = variableNames(*MDObj*, *Subset*) returns a cell array of strings specifying a subset the variable names in a MetaData object.

NewMDObj = variableNames(*MDObj*, *Subset*, *NewVarNames*) replaces the variable names specified by *Subset* in *MDObj*, a MetaData object, with *NewVarNames*, and returns *NewMDObj*, a new MetaData object.

Input Arguments

MDObj

Object of the bioma.data.MetaData class.

Default:

Subset

One of the following to specify a subset of the variable names in a MetaData object:

- String specifying a variable name

- Cell array of strings specifying variable names
- Positive integer
- Vector of positive integers
- Logical vector

Default:

NewVarNames

New variable names for specific sample or feature variable names within a MetaData object, specified by one of the following:

- Numeric vector
- String or cell array of strings
- String, which `variableNames` uses as a prefix for the variable names, with variable numbers appended to the prefix
- Logical `true` or `false` (default). If `true`, `variableNames` assigns unique variable names using the format `Var1`, `Var2`, etc.

The number of variable names in *NewVarNames* must equal the number of variable names specified by *Subset*.

Default:

Output Arguments

VarNames

Cell array of strings specifying all variable names in a MetaData object.

NewMDObj

Object of the `bioma.data.MetaData` class, returned after replacing the variable names.

Examples

Construct a MetaData object, and then retrieve the sample variable names from it:

```
% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the sample variable names
VNames = variableNames(MDObj2)
```

See Also
bioma.data.MetaData | sampleNames | variableValues | variableDesc

How To
- "Representing Sample and Feature Metadata in MetaData Objects"

variableValues

Class: bioma.data.MetaData
Package: bioma.data

Retrieve or set variable values for samples in MetaData object

Syntax

DSVarValues = variableValues(*MDObj*)
NewMDObj = variableValues(*MDObj, NewDSVarValues*)

Description

DSVarValues = variableValues(*MDObj*) returns a dataset array containing the measured value of each variable per sample from a MetaData object.

NewMDObj = variableValues(*MDObj, NewDSVarValues*) replaces the sample variable values in *MDObj*, a MetaData object, with *NewDSVarValues*, and returns *NewMDObj*, a new MetaData object.

Input Arguments

MDObj

Object of the `bioma.data.MetaData` class.

Default:

NewDSVarValues

A new dataset array containing a value for each variable per sample. In this dataset array, the columns correspond to variables and rows correspond to samples. The row names (sample names) must match the row names (sample names) in *DSVarValues*, the dataset array being replaced in the MetaData object, *MDObj*.

Default:

Output Arguments

`DSVarValues`

A dataset array containing the measured value of each variable per sample from a MetaData object. In this dataset array, the columns correspond to variables and rows correspond to samples.

`NewMDObj`

Object of the `bioma.data.MetaData` class, returned after replacing the dataset array containing the sample variable values.

Examples

Construct a MetaData object, and then retrieve the sample variable values from it:

```
% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the sample variable values
VarValues = variableValues(MDObj2)
```

See Also

bioma.data.MetaData | sampleNames | variableNames | variableDesc

How To

• "Representing Sample and Feature Metadata in MetaData Objects"

varValuesTable

Class: bioma.data.MetaData
Package: bioma.data

Create 2-D graphic table GUI of variable values in MetaData object

Syntax

Handle = varValuesTable(*MDObj*)
Handle = varValuesTable(*MDObj, ParentHandle*)

Description

Handle = varValuesTable(*MDObj*) creates a 2-D graphic table containing variable data from a MetaData object and returns a uitable handle to the table.

Handle = varValuesTable(*MDObj, ParentHandle*) specifies the parent handle to the table. The parent can be a figure or uipanel handle.

Input Arguments

MDObj

Object of the `bioma.data.MetaData` class.

Default:

ParentHandle

Figure or uipanel handle to be the parent handle to the table.

Default:

Examples

Construct a MetaData object, and then create a 2-D table of the variable values from it:

```
% Import bioma.data package to make constructor function
% available
import bioma.data.*
% Construct MetaData object from .txt file
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#');
% Retrieve the sample variable values in a table
handle = varValuesTable(MDObj2)
```
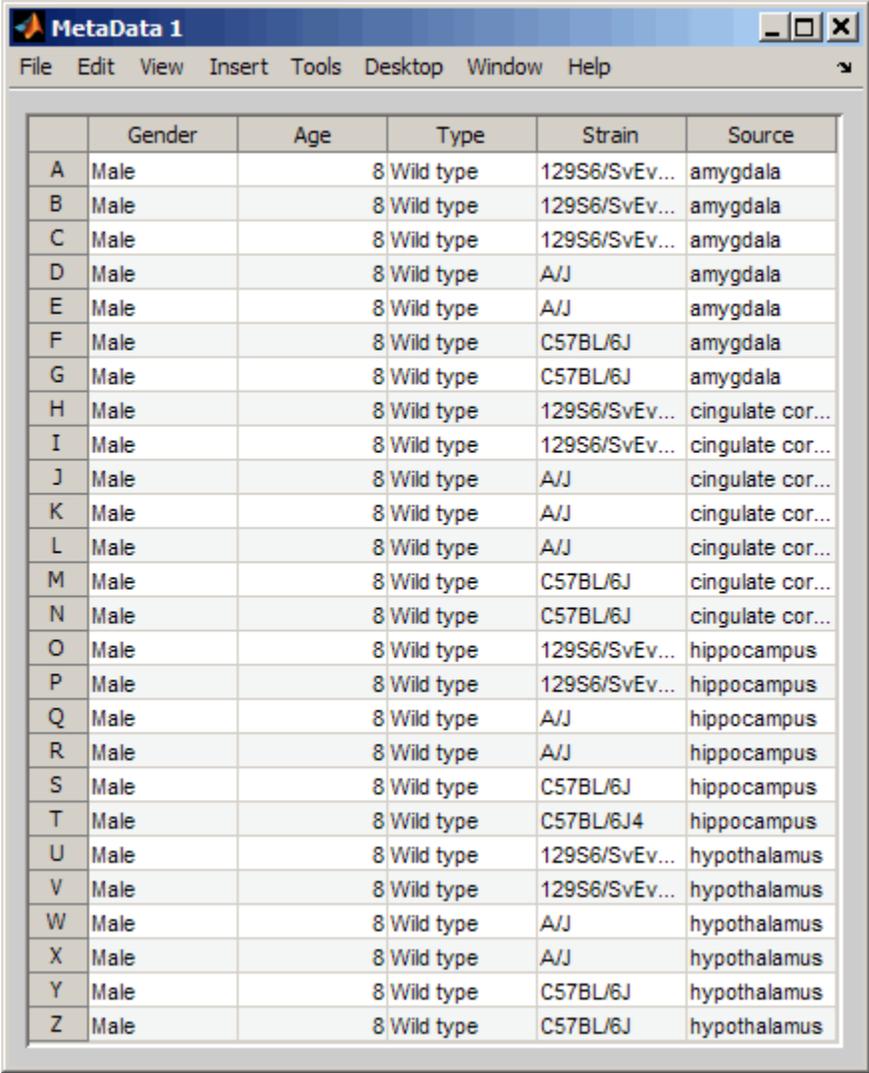

	Gender	Age	Type	Strain	Source
A	Male	8	Wild type	129S6/SvEv...	amygdala
B	Male	8	Wild type	129S6/SvEv...	amygdala
C	Male	8	Wild type	129S6/SvEv...	amygdala
D	Male	8	Wild type	A/J	amygdala
E	Male	8	Wild type	A/J	amygdala
F	Male	8	Wild type	C57BL/6J	amygdala
G	Male	8	Wild type	C57BL/6J	amygdala
H	Male	8	Wild type	129S6/SvEv...	cingulate cor...
I	Male	8	Wild type	129S6/SvEv...	cingulate cor...
J	Male	8	Wild type	A/J	cingulate cor...
K	Male	8	Wild type	A/J	cingulate cor...
L	Male	8	Wild type	A/J	cingulate cor...
M	Male	8	Wild type	C57BL/6J	cingulate cor...
N	Male	8	Wild type	C57BL/6J	cingulate cor...
O	Male	8	Wild type	129S6/SvEv...	hippocampus
P	Male	8	Wild type	129S6/SvEv...	hippocampus
Q	Male	8	Wild type	A/J	hippocampus
R	Male	8	Wild type	A/J	hippocampus
S	Male	8	Wild type	C57BL/6J	hippocampus
T	Male	8	Wild type	C57BL/6J4	hippocampus
U	Male	8	Wild type	129S6/SvEv...	hypothalamus
V	Male	8	Wild type	129S6/SvEv...	hypothalamus
W	Male	8	Wild type	A/J	hypothalamus
X	Male	8	Wild type	A/J	hypothalamus
Y	Male	8	Wild type	C57BL/6J	hypothalamus
Z	Male	8	Wild type	C57BL/6J	hypothalamus

See Also

`bioma.data.MetaData`

How To

- "Representing Sample and Feature Metadata in MetaData Objects"

vertcat (DataMatrix)

Concatenate DataMatrix objects vertically

Syntax

```
DMObjNew = vertcat(DMObj1, DMObj2, ...)
DMObjNew = (DMObj1; DMObj2; ...)
DMObjNew = vertcat(DMObj1, B, ...)
DMObjNew = (DMObj1, B, ...)
```

Input Arguments

DMObj1, *DMObj2*	DataMatrix objects, such as created by `DataMatrix` (object constructor).
B	MATLAB numeric or logical array.

Output Arguments

DMObjNew	DataMatrix object created by vertical concatenation.

Description

DMObjNew = vertcat(*DMObj1*, *DMObj2*, ...) or the equivalent *DMObjNew* = (*DMObj1*; *DMObj2*; ...) vertically concatenates the DataMatrix objects *DMObj1* and *DMObj2* into *DMObjNew*, another DataMatrix object. *DMObj1* and *DMObj2* must have the same number of columns. The column names and the order of columns for *DMObjNew* are the same as *DMObj1*. The column names of *DMObj2* and any other DataMatrix object input arguments are not preserved. The row names for *DMObjNew* are the row names of *DMObj1*, *DMObj2*, and other DataMatrix object input arguments.

DMObjNew = vertcat(*DMObj1*, *B*, ...) or the equivalent *DMObjNew* = (*DMObj1*, *B*, ...) vertically concatenates the DataMatrix object *DMObj1* and a numeric or logical

array *B* into *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same number of columns. The column names for *DMObjNew* are the same as *DMObj1*. The column names of *DMObj2* and any other DataMatrix object input arguments are not preserved. The row names for *DMObjNew* are the row names of *DMObj1* and empty for the rows from *B*.

MATLAB calls *DMObjNew* = vertcat(*X1*, *X2*, *X3*, ...) for the syntax *DMObjNew* = [*X1*; *X2*; *X3*; ...] when any one of *X1*, *X2*, *X3*, etc. is a DataMatrix object.

More About

*   DataMatrix object

See Also
DataMatrix | horzcat

view (biograph)

Draw figure from biograph object

Syntax

```
view(BGobj)
BGobjHandle = view(BGobj)
```

Arguments

BGobj	Biograph object created with the function `biograph`.

Description

`view(BGobj)` opens a Figure window and draws a graph represented by a biograph object (*BGobj*). When the biograph object is already drawn in the Figure window, this function only updates the graph properties.

BGobjHandle = `view(BGobj)` returns a handle to a deep copy of the biograph object (BGobj) in the Figure window. When updating an existing figure, you can use the returned handle to change object properties programmatically or from the command line. When you close the Figure window, the handle is no longer valid. The original biograph object (*BGobj*) is left unchanged.

Examples

1 Create a biograph object.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg = biograph(cm)
```

2 Render the biograph object into a Handles Graphic figure and get back a handle.

```
h = view(bg)
```

3 Change the color of all nodes and edges.

```
set(h.Nodes,'Color',[.5 .7 1])
set(h.Edges,'LineColor',[0 0 0])
```

More About

- biograph object

See Also

biograph | get | getdescendants | getnodesbyid | set | view | dolayout |
getancestors | getedgesbynodeid | getrelatives

view (clustergram)

View clustergram and dendrograms of clustergram object

Syntax

```
view(CGObject)
```

Arguments

CGObject	Clustergram object created with the function `clustergram`.

Description

`view(CGObject)` opens a Clustergram window and draws a clustergram and dendrograms representing a clustergram object, *CGObject*. The clustergram shows hierarchical clustering using a heat map and dendrograms.

Note: You can further explore the heat map and dendrograms using the mouse, toolbar buttons, and menu items in the Clustergram window. For more information, see the Examples section of the `clustergram` function.

Examples

View the clustergram object created in the first two steps of the "Examples" on page 1-447 section of the `clustergram` function reference page.

```
view(cgo)
```

More About

- clustergram object

See Also
clustergram | plot | set | get

view (HeatMap)

View heat map of HeatMap object

Syntax

```
view(HMObject)
```

Arguments

HMObject	HeatMap object created with the function `HeatMap`.

Description

`view(HMObject)` opens a HeatMap window and draws a heat map representing a HeatMap object, *HMObject*.

Examples

View the HeatMap object created in the "Examples" on page 1-   section of the `HeatMap` function reference page.

```
view(hmo)
```

More About

· HeatMap object

See Also

HeatMap | plot

view (phytree)

View phylogenetic tree

Syntax

```
view(Tree)
view(Tree, IntNodes)
```

Arguments

Tree	Phylogenetic tree (`phytree` object) created with the function `phytree`.
IntNodes	Nodes from the `phytree` object to initially display in the *Tree*.

Description

`view(Tree)` opens the Phylogenetic Tree window and draws a tree from data in a `phytree` object (*Tree*). The significant distances between branches and nodes are in the horizontal direction. Vertical distances have no significance and are selected only for display purposes. You can access tools to edit and analyze the tree from the Phylogenetic Tree menu bar or by using the left and right mouse buttons.

`view(Tree, IntNodes)` opens the Phylogenetic Tree window with an initial selection of nodes specified by *IntNodes*. *IntNodes* can be a logical array of any of the following sizes: `NumLeaves + NumBranches x 1`, `NumLeaves x 1`, or `NumBranches x 1`. *IntNodes* can also be a list of indices.

Examples

View a Phylogenetic Tree

Load and view a sample phylogenetic tree.
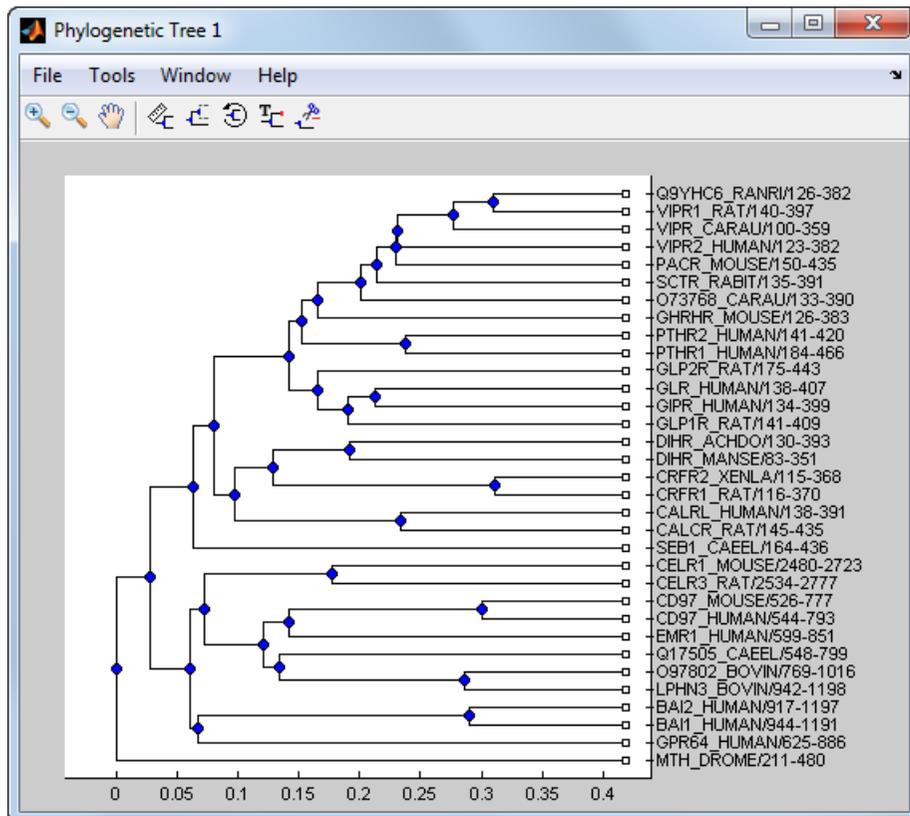
```
tr = phytreeread('pf00002.tree');
```

view(tr)

More About

- phytree object

See Also

```
phytree | cluster | phytreeread | phytreeviewer | seqlinkage |
seqneighjoin | plot
```

weights (phytree)

Calculate weights for phylogenetic tree

Syntax

W = weights(*Tree*)

Arguments

Tree	Phylogenetic tree (phytree object) created with the function phytree.

Description

W = weights(*Tree*) calculates branch proportional weights for every leaf in a tree (*Tree*) using the Thompson-Higgins-Gibson method. The distance of every segment of the tree is adjusted by dividing it by the number of leaves it contains. The sequence weights are the result of normalizing to unity the new patristic distances between every leaf and the root.

Examples

1 Create an ultrametric tree with specified branch distances.

```
bd = [1 2 3]';
tr_1 = phytree([1 2;3 4;5 6],bd)
```

2 View the tree.

```
view(tr_1)
```
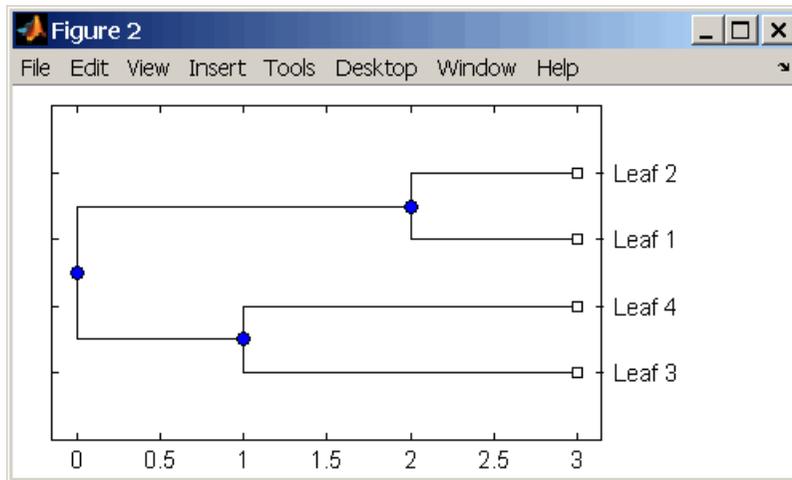
3 Display the calculated weights.

```
weights(tr_1)

ans =

     1.0000
     1.0000
     0.8000
     0.8000
```

More About

- phytree object

References

[1] Thompson JD, Higgins DG, Gibson TJ (1994), "CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," Nucleic Acids Research, 22(22):4673-4680.

[2] Henikoff S, Henikoff JG (1994), "Position-based sequence weights," Journal Molecular Biology, 243(4):574-578.

See Also

`multialign` | `phytree` | `profalign` | `seqlinkage`

write

Class: BioRead

Write contents of BioRead or BioMap object to file

Syntax

```
write(BioObj,FileName)
write(BioObj,FileName,Name,Value)
```

Description

`write(BioObj,FileName)` writes the contents of a `BioRead` or `BioMap` object BioObj to a file called FileName.

`write(BioObj,FileName,Name,Value)` writes the contents of `BioRead` or `BioMap` object to a file with additional options specified by Name,Value pair arguments.

Input Arguments

BioObj

Object of the `BioRead` or `BioMap` class.

Default:

FileName

String containing the name of a file to be written. The file extension is automatically added depending on the type of data the object contains. If you provide the extension, the function checks the consistency between the provided extension and the data format of the object. The file name can be prefixed by a file path. If the path is missing, the file is written to the same folder where the source file is located or to the current folder if the data is in memory.

Default:

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`'  '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'Format'

String specifying the type of file format. Available formats are `'FASTA'`, `'FASTQ'`, `'SAM'`, and `'BAM'` for `BioMap` objects, and `'FASTA'` and `'FASTQ'` for `BioRead` objects. Default format for `BioRead` objects is `'FASTA'` when the objects do not contain qualities, i.e., the `Quality` property of the object is empty. Otherwise, the default is `'FASTQ'`. Default format for `BioMap` objects is `'BAM'`.

Default:

'Overwrite'

Boolean indicator specifying whether an existing file can be overwritten. Default is `false`. If `true`, the method overwrites the file and deletes any respective index file (`*.idx`,`*.bai`,`*.linearindex`) or ordered file (`*.ordered.bam`, `*.ordered.sam`) that has become stale.

Default: `false`

Tip

Note: When you have an object that contains a subset of information from the source file, use this method to save the object in a separate file. Having a compact file only for the object can be more efficient and helpful in processing data or sharing among collaborators.

Examples

Write the Contents of BioRead Object to a File

Create a `BioRead` object from a FASTQ file.

```
BRObj = BioRead('SRR005164_1_50.fastq');
```

Extract the first 10 elements from `BRObj` and store them in a new `BioRead` object.

```
subsetBRObj = getSubset(BRObj, [1:10]);
```

Write the contents of the subset object `subsetBRObj` to a file named `subsetBRObj.fastq` in a `project` folder (or any other folder) on a local `C` (or any other) drive.

```
write(subsetBRObj, 'C:\project\subsetBRObj');
```

See Also
BioRead | BioMap

How To
· "Manage Short-Read Sequence Data in Objects"

zonebackadj

Perform background adjustment on Affymetrix microarray probe-level data using zone-based method

Syntax

BackAdjustedData = zonebackadj(*Data*)
[*BackAdjustedData*, *ZoneStruct*] = zonebackadj(*Data*)
[*BackAdjustedData*, *ZoneStruct*, *Background*] = zonebackadj(*Data*)
... = zonebackadj(*Data*, ...'NumZones', *NumZonesValue*, ...)
... = zonebackadj(*Data*, ...'Percent', *PercentValue*, ...)
... = zonebackadj(*Data*, ...'SmoothFactor', *SmoothFactorValue*, ...)
... = zonebackadj(*Data*, ...'NoiseFrac', *NoiseFracValue*, ...)
... = zonebackadj(*Data*, ...'CDF', *CDFValue*, ...)
... = zonebackadj(*Data*, ...'Mask', *MaskValue*, ...)
... = zonebackadj(*Data*, ...'Showplot', *ShowplotValue*, ...)

Input Arguments

Data	Either of the following:
	• MATLAB structure containing probe intensities from an Affymetrix CEL file, such as returned by affyread when used to read a CEL file.
	• Array of MATLAB structures containing probe intensities from multiple Affymetrix CEL files.
NumZonesValue	Scalar or two-element vector that specifies the number of zones to use in the background adjustment. If a scalar, it must be a square number. If a two-element vector, the first element specifies the number of rows and the second element specifies the number of columns in a nonsquare grid. Default is 16.
PercentValue	Value that specifies a percentage, *P*, such that the lowest *P* percent of ranked intensity values from each zone is used to estimate the background for that zone. Default is 2.

SmoothFactorValue	Value that specifies the smoothing factor used in the calculation of the weighted average of the contributions of each zone to the background of a point. Default is `100`.
NoiseFracValue	Value that specifies the noise fraction, `NF`, such that the background-adjusted value is given by `max((Intensity - WeightedBackground), NF*LocalNoiseEstimate)`. Default is `0.5`.
CDFValue	Either of the following: • String specifying a file name or path and file name of an Affymetrix CDF library file. If you specify only a file name, the file must be on the MATLAB search path or in the current folder. • MATLAB structure containing information from an Affymetrix CDF library file, such as returned by `affyread` when used to read a CDF file. The CDF library file or structure specifies control cells, which are not used in the background estimates.
MaskValue	Logical vector that specifies which cells to mask and not use in the background estimates. In the vector, `0 = not masked` and `1 = masked`. Defaults are the values in the `Masked` column of the `Probes` field of the CEL file.
ShowplotValue	Controls the plotting of an image of the background estimates. Choices are `true` or `false` (default).

Output Arguments

BackAdjustedData	Matrix or cell array of vectors containing background-adjusted probe intensity values.
ZoneStruct	MATLAB structure containing the centers of the zones used to perform the background adjustment and the estimates of the background values at the center of each zone.
Background	Matrix or cell array of vectors containing the estimated background values for each probe.

Description

BackAdjustedData = zonebackadj(*Data*) returns the background-adjusted probe intensities from *Data*, which contains probe intensities from Affymetrix CEL files. Details of the background adjustment are described in Statistical Algorithms Description Document.

[*BackAdjustedData*, *ZoneStruct*] = zonebackadj(*Data*) also returns a structure containing the centers of the zones used to perform the background adjustment and the estimates of the background values at the center of each zone.

[*BackAdjustedData*, *ZoneStruct*, *Background*] = zonebackadj(*Data*) also returns a matrix or cell array of vectors containing the estimated background values for each probe.

... = zonebackadj(*Data*, ...'*PropertyName*', *PropertyValue*, ...) calls zonebackadj with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = zonebackadj(*Data*, ...'NumZones', *NumZonesValue*, ...) specifies the number of zones to use in the background adjustment. *NumZonesValue* can be either a scalar that is a square number or a two-element array in which the first element specifies the number of rows and the second element specifies the number of columns in a nonsquare grid. Default is 16.

... = zonebackadj(*Data*, ...'Percent', *PercentValue*, ...) specifies a percentage, *P*, such that the lowest *P* percent of ranked intensity values from each zone is used to estimate the background for that zone. Default is 2.

... = zonebackadj(*Data*, ...'SmoothFactor', *SmoothFactorValue*, ...) specifies the smoothing factor used in the calculation of the weighted average of the contributions of each zone to the background of a point, thus providing a smooth transition between zones. Default is 100.

... = zonebackadj(*Data*, ...'NoiseFrac', *NoiseFracValue*, ...) specifies the noise fraction, such that the background-adjusted value is given by max((Intensity - WeightedBackground), NF*LocalNoiseEstimate), where NF is *NoiseFracValue*. Default is 0.5.

... = zonebackadj(*Data*, ...'CDF', *CDFValue*, ...) specifies an Affymetrix CDF library file or structure, which specifies control cells, which are not used in the background estimates.

... = zonebackadj(*Data*, ...'Mask', *MaskValue*, ...) specifies a logical vector of that specifies which cells to mask and not use in the background estimates. In the vector, 0 = not masked and 1 = masked. Defaults are the values in the Masked column of the Probes field of the CEL file.

... = zonebackadj(*Data*, ...'Showplot', *ShowplotValue*, ...) plots an image of the background estimates. Choices are true or false (default).
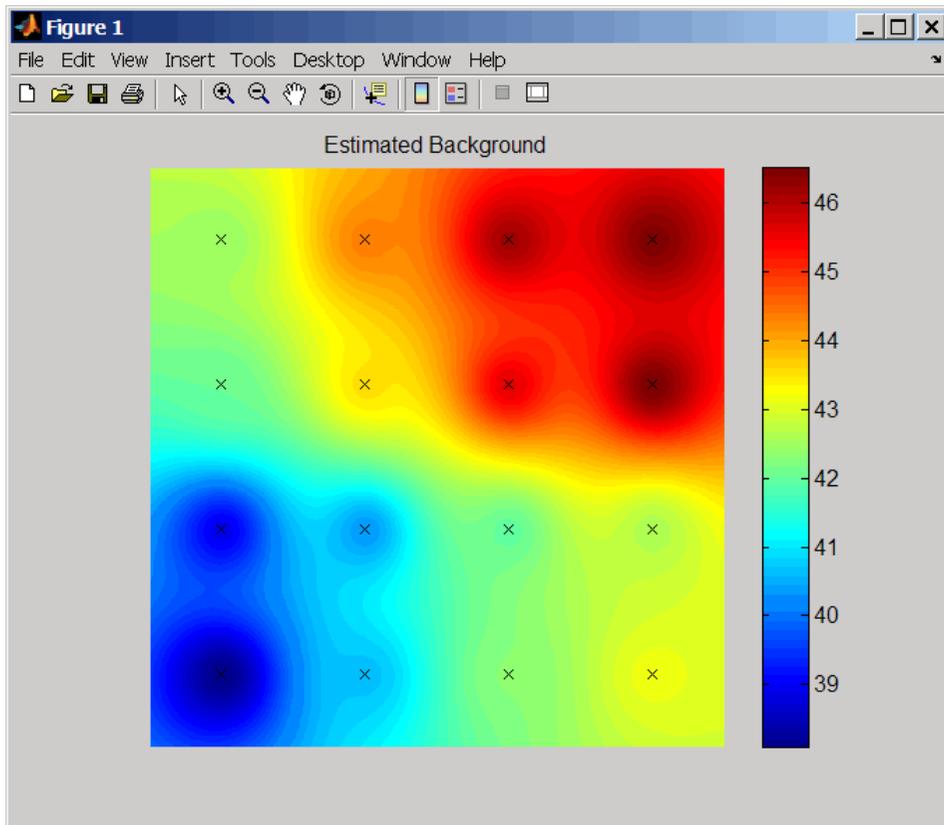
Examples

The following example uses a sample CEL file and CDF library file from the *E. coli* Antisense Genome array, which you can download from:

```
http://www.affymetrix.com/support/technical/sample_data/demo_data.affx
```

After you download the sample data, you will need the Affymetrix Data Transfer Tool to extract the CEL file from a DTT file. You can download the Affymetrix Data Transfer Tool from:

```
http://www.affymetrix.com/browse/products.jsp?productId=131431&navMode=34000&navAction=
```

The following example assumes that the `Ecoli-antisense-121502.CEL` file is stored on the MATLAB search path or in the current folder. It also assumes that the associated CDF library file, `Ecoli_ASv2.CDF`, is stored at `D:\Affymetrix\LibFiles\Ecoli`.

1 Use the `affyread` function to read an Affymetrix CEL file and create `celStruct`, a MATLAB structure containing probe intensities for a single Affymetrix GeneChip.

```
celStruct = affyread('Ecoli-antisense-121502.CEL');
```

2 Perform background adjustment on the probe intensities in the structure, excluding the probe intensities from the control cells on the chip.

```
 BackAdjMatrix = zonebackadj(celStruct, 'cdf',...
                 'D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

References

[1] Statistical Algorithms Description Document, http://www.affymetrix.com/support/technical/whitepapers/ sadd_whitepaper.pdf

See Also

affyinvarsetnorm | affyread | celintensityread | gcrma | gcrmabackadj | probelibraryinfo | probesetlink | probesetlookup | probesetvalues | quantilenorm | rmabackadj | rmasummary

date property

Class: geneont

Read-only string containing date and time OBO file was last updated

Description

date is a read-only property of the geneont class. date is a string containing the date and time the OBO file was last updated. The OBO file is the Open Biomedical Ontology file from which the geneont object was created.

Values

Possible values are any date and time the OBO file was updated. Use this date information to compare the dates associated with ontologies used to create various geneont objects.

Examples

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Display the date and time associated with the OBO file used to create the geneont object.

```
GeneontObj.date

ans =

02:12:2008 19:30
```

See Also
geneont.format_version

default_namespace property

Class: geneont

Read-only string containing namespace to which GO terms are assigned

Description

`default_namespace` is a read-only property of the geneont class. `default_namespace` is a string containing the ontology namespace to which the GO terms are assigned.

Values

Currently, `gene_ontology` is the only possible namespace. However, other namespaces may be used in the future. Use this namespace information to determine the ontology namespace to which the GO terms in a geneont object are assigned.

Examples

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Display the namespace associated with the GO terms of the geneont object.

```
GeneontObj.default_namespace

ans =

gene_ontology
```

format_version property

Class: geneont

Read-only string containing version of encoding of OBO file

Description

`format_version` is a read-only property of the geneont class. `format_version` is a string containing the version of the encoding of the OBO file. The OBO file is the Open Biomedical Ontology file from which the geneont object was created.

Values

Possible values are the current or previous versions of the OBO file. Use this version information to compare the version associated with OBO file used to create various geneont objects.

Examples

1   Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2   Display the version of the OBO file used to create the geneont object.

```
GeneontObj.format_version

ans =

1.0
```

terms property

Class: geneont

Read-only column vector with handles to term objects of geneont object

Description

terms is a read-only property of the geneont class. terms is a column vector with handles to the term objects of a geneont object.

Note: Although terms is a column vector with handles to term objects, in the MATLAB Command Window, it displays as a structure array, with one structure for each GO term in the geneont object.

Values

Use the information in this structure to access (by GO ID) the terms of a geneont object and to view the properties of individual terms.

Examples

1   Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

    ```
    GeneontObj = geneont('LIVE', true)
    ```

    The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

    ```
    Gene Ontology object with 27786 Terms.
    ```

2   Use the terms property to display the MATLAB structure array containing 27,786 term objects associated with the geneont object.

    ```
    GeneontObj.terms
    ```

```
27786x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

Note: Although the `terms` property is an array of handles to term objects, in the MATLAB Command Window, it displays as a structure array, with one structure for each GO term in the geneont object.

---

3 Use the `terms` property to view the properties of the term object in the 14,723rd position in the geneont object.

```
GeneontObj.terms(14723)

          id: 31655
        name: 'negative regulation of heat dissipation'
    ontology: 'biological process'
  definition: [1x113 char]
     comment: ''
     synonym: {4x2 cell}
        is_a: [3x1 double]
     part_of: 31653
    obsolete: 0
```

4 Create a cell array of strings that list the `ontology` property for each term in the geneont object.

```
ontologies = get(GeneontObj.terms,'ontology');
```

5 Create a logical mask that identifies all the terms with an `ontology` property of `cellular component`.

```
mask = strcmp(ontologies,'cellular component');
```

6 Apply the logical mask to all the terms in the `GeneontObj` geneont object to return a MATLAB structure array of term objects, containing only terms with an `ontology` property of `cellular component`.

```
cell_comp_terms = GeneontObj.terms(mask)
```

```
2362x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

There are 2,362 terms with an `ontology` property of `cellular component`.

7 Create a subontology of all the cellular component terms by indexing into the `GeneontObj` geneont object with the masked term objects.

```
subontology = GeneontObj(cell_comp_terms)
```

```
Gene Ontology object with 2367 Terms.
```

See Also
`term`

definition property

Class: term

Read-only string that defines GO term

Description

`definition` is a read-only property of the term class. `definition` is a string that defines the GO term.

Values

Possible values are any definition used for a term in the Gene Ontology database. Use the `definition` property to determine definitions of term objects, or to access or filter term objects by definition.

Examples

Using the definition Property to Determine the Definition of a term Object

1   Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2   Display the definition of the term object in the 287th position in the geneont object, `GeneontObj`.

```
GeneontObj.terms(287).name

ans =

"The smaller of the two subunits of an organellar ribosome." [GOC:mcc]
```

Tip If you know the GO identifier (for example, 314) of a term object, instead of its index or position number (for example, 287), you can use the following syntax to display the definition of a term object:

```
GeneontObj(314).terms.definition
```

For help converting the index or position number of a term object to its GO identifier, see the `term.id` property.

Filtering term Objects by Text in Their Definitions

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Display the structure array containing 27,786 term objects associated with the geneont object.

```
GeneontObj.terms

27786x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

3 Find term objects whose definitions include the phrase "ceramide oligosaccharides" by first creating a cell array of strings that list the `definition` property for each term in the geneont object.

```
definitions = get(GeneontObj.terms,'definition');
```

4 Use the `regexpi` function to search these strings for `'ceramide oligosaccharides'`.

```
matches = regexpi(definitions,'ceramide oligosaccharides','once');
```

5 Create a logical mask that identifies all the terms with a `definition` property that includes the phrase "ceramide oligosaccharides."

```
mask = ~cellfun(@isempty,matches);
```

6 Apply the logical mask to all the terms in the `GeneontObj` geneont object to return a structure containing the GO identifiers of terms with a definition that includes the phrase "ceramide oligosaccharides."

```
get(GO.terms(mask),'id')

ans =

    [1573]
    [1574]
```

7 Apply the logical mask to all the terms in the `GeneontObj` geneont object to return a structure containing the full definitions of terms with a definition that includes the phrase "ceramide oligosaccharides."

```
char(get(GO.terms(mask),'definition'))
```

id property

Class: term

Read-only numeric value that corresponds to GO identifier of GO term

Description

id is a property of the term class. id is a read-only numeric value that corresponds to the GO identifier of the GO term.

Tip You can use the num2goid function to convert id to a GO ID string formatted as a 7-digit number preceded by the prefix GO:, which is the standard used by the Gene Ontology database.

Values

Any value from 1 to *N*, where *N* is the largest value for an identifier of a term object in a geneont object. Use the id property to determine GO identifiers of term objects, or to access term objects by their GO identifier.

Tip You can use the id property for a GO term as input to methods of a geneont object, such as geneont.getancestors, geneont.getdescendants, and geneont.getrelatives.

Examples

Displaying and Formatting the GO Identifier of a term Object

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Display the GO identifier of the term object in the 183rd position in the geneont object, `GeneontObj`.

```
GeneontObj.terms(183).id

ans =

    207
```

Note: The index or position (183 in this example) of the term object in the geneont object is not the same as the GO identifier (207 in this example) for the term object. This is because there are many terms that are obsolete and are not included as term objects in the geneont object.

---

3 Format the GO identifier into a character array.

```
num2goid(GeneontObj.terms(183).id)

ans =

    'GO:0000207'
```

Using the GO Identifier with Methods of a geneont Object

1 Find the index or position number of the term object whose `name` property is `'membrane'`.

```
membrane_index = find(strcmp(get(GeneontObj.terms,'name'),'membrane'))

membrane_index =

       9556
```

2 Use this index or position number and the `id` property to determine the GO identifier of the term object.

```
membrane_goid = GeneontObj.terms(membrane_index).id

membrane_goid =
```

```
         16020
```

3 Use this GO identifier as input to the `getrelatives` method to find the GO
identifiers of other term objects that are immediate relatives of the term object
whose `name` property is `'membrane'`.

```
relative_ids = getrelatives(GeneontObj,membrane_goid)

relative_ids =

          5628
          5886
         16020
         19867
         30673
         31090
         34045
         34357
         42175
         42622
         42734
         44464
         45211
         48475
         60342
```

4 List the `name` properties of these term objects.

```
get(GeneontObj(relative_ids).terms,'name')

ans =

    'prospore membrane'
    'plasma membrane'
    'membrane'
    'outer membrane'
    'axolemma'
    'organelle membrane'
    'pre-autophagosomal structure membrane'
    'photosynthetic membrane'
    'nuclear envelope-endoplasmic reticulum network'
    'photoreceptor outer segment membrane'
    'presynaptic membrane'
    'cell part'
    'postsynaptic membrane'
```

```
'coated membrane'
'photoreceptor inner segment membrane'
```

See Also
num2goid | geneont.getdescendants | geneont.getrelatives |
geneont.getancestors

is_a property

Class: term

Read-only numeric array containing GO identifiers of GO terms that have an "is a" relationship with this GO term

Description

`is_a` is a read-only property of the term class. `is_a` is a column vector containing GO identifiers. These GO identifiers specify other term objects to which the term object has an "is a" relationship. The term object is an example of the term objects specified by its `is_a` property.

Values

Possible values are identifiers of GO terms from the Gene Ontology database. Use the `is_a` property to determine GO identifiers of GO terms that have an "is a" relationship with a specific GO term.

Examples

Using the is_a Property to Determine term Objects with an "is a" Relationship

1   Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

    ```
    GeneontObj = geneont('LIVE', true)
    ```

    The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

    ```
    Gene Ontology object with 27769 Terms.
    ```

2   Display the term objects to which the term object in the 18,703rd position has an "is a" relationship.

    ```
    GeneontObj.terms(18703).is_a
    ```

```
ans =

        42754
        45187
        48521
        51241
```

Tip You can also use the `getancestors` method of a geneont object with the `'Relationtype'` property set to `'is_a'` to determine term objects with an "is a" relationship.

Tip If you know the GO identifier (for example, 42321) of a term object, instead of its index or position number (for example, 18703), you can use the following syntax to display the `is_a` property of a term object:

`GeneontObj(42321).terms.is_a`

For help converting the index or position number of a term object to its GO identifier, see the `term.id` property.

See Also

geneont.getdescendants | geneont.getrelatives | geneont.getancestors

name property

Class: term

Read-only string representing name of GO term

Description

`name` is a read-only property of the term class. `name` is a string representing the name of the GO term.

Values

Possible values are any name used for a term in the Gene Ontology database. Use the `name` property to determine names of term objects, or to access or filter term objects by name.

Examples

Using the name Property to Determine the Name of a term Object

1   Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2   Display the name of the term object in the 157th position in the geneont object, `GeneontObj`.

```
GeneontObj.terms(157).name

ans =

  cytosolic small ribosomal subunit
```

Tip If you know the GO identifier (for example, 181) of a term object, instead of its index or position number (for example, 157), you can use the following syntax to display the name of a term object:

```
GeneontObj(181).terms.name
```

For help converting the index or position number of a term object to its GO identifier, see the `term.id` property.

Using the name Property to Find and Display Specific term Objects

1 Find the index or position number of the term object whose `name` property is `'membrane'`.

```
membrane_index = find(strcmp(get(GeneontObj.terms,'name'),'membrane'))

membrane_index =

        9556
```

2 Use this index or position number and the `id` property to determine the GO identifier of the term object.

```
membrane_goid = GeneontObj.terms(membrane_index).id

membrane_goid =

        16020
```

3 Use this GO identifier as input to the `getrelatives` method to find the GO identifiers of other term objects that are immediate relatives of the term object whose `name` property is `'membrane'`.

```
relative_ids = getrelatives(GeneontObj,membrane_goid)

relative_ids =

         5628
         5886
        16020
        19867
        30673
        31090
        34045
```

```
        34357
        42175
        42622
        42734
        44464
        45211
        48475
        60342
```

4 List the name properties of these term objects.

```
get(GeneontObj(relative_ids).terms,'name')

ans =

    'prospore membrane'
    'plasma membrane'
    'membrane'
    'outer membrane'
    'axolemma'
    'organelle membrane'
    'pre-autophagosomal structure membrane'
    'photosynthetic membrane'
    'nuclear envelope-endoplasmic reticulum network'
    'photoreceptor outer segment membrane'
    'presynaptic membrane'
    'cell part'
    'postsynaptic membrane'
    'coated membrane'
    'photoreceptor inner segment membrane'
```

obsolete property

Class: term

Read-only Boolean value that indicates whether a GO term is obsolete

Description

`obsolete` is a read-only property of the term class. `obsolete` is a Boolean value that indicates if the GO term is obsolete (`1`) or not obsolete (`0`).

Values

1 — Obsolete
0 — Not obsolete

Use the `obsolete` property to determine whether a term object is obsolete, or to access or filter term objects by obsolete value.

Examples

Using the obsolete Property to Determine the Obsolete Status of a term Object

1   Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2   Display the obsolete status of the term object in the third and seventh positions in the geneont object, `GO`

```
GeneontObj.terms(3).obsolete

ans =
```

```
         0

GeneontObj.terms(7).obsolete

ans =

         1
```

Tip If you know the GO identifier (for example, 8) of a term object, instead of its index or position number (for example, 7), you can use the following syntax to display the obsolete status of a term object:

```
GeneontObj(8).terms.obsolete
```

For help converting the index or position number of a term object to its GO identifier, see the `term.id` property.

Filtering term Objects by Obsolete Status

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Display the structure array containing 27,786 term objects associated with the geneont object.

```
GeneontObj.terms

27786x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
```

```
        part_of
        obsolete
```

3   Create a cell array of logicals that list the `obsolete` property for each term in the geneont object.

```
obsolescence = get(GeneontObj.terms,'obsolete');
```

4   Create a logical mask from the cell array that identifies all the nonobsolete terms.

```
mask = ~cell2mat(obsolescence);
```

5   Apply the logical mask to all the terms in the `GeneontObj` geneont object to return a structure containing only terms that are not obsolete.

```
nonobsolete_terms = GeneontObj.terms(mask)

26424x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

There are 26,424 terms that are not obsolete.

ontology property

Class: term

Read-only string describing the ontology of GO term

Description

ontology is a read-only property of the term class. ontology is a string describing the ontology of the GO term.

Values

```
'molecular function'
'biological process'
'cellular component'
```

Use the ontology property to determine the ontology of term objects, or to access or filter term objects by ontology.

Examples

Using the ontology Property to Determine the Ontology of a term Object

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Display the ontology of the term object in the 155th position in the geneont object, GeneontObj.

```
GeneontObj.terms(155).ontology
```

```
ans =

molecular function
```

Tip If you know the GO identifier (for example, 179) of a term object, instead of its index or position number (for example, 155), you can use the following syntax to display the ontology of a term object:

```
GeneontObj(179).terms.ontology
```

For help converting the index or position number of a term object to its GO identifier, see the `term.id` property.

Filtering term Objects by Cellular Component Ontology

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Display the structure array containing 27,786 term objects associated with the geneont object.

```
GeneontObj.terms

27786x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

3 View the properties of the term object in the 14,723rd position in the geneont object.

```
GeneontObj.terms(14723)
```

```
              id: 31655
            name: 'negative regulation of heat dissipation'
        ontology: 'biological process'
      definition: [1x113 char]
         comment: ''
         synonym: {4x2 cell}
            is_a: [3x1 double]
         part_of: 31653
        obsolete: 0
```

4 Create a cell array of strings that list the `ontology` property for each term in the geneont object.

```
ontologies = get(GeneontObj.terms,'ontology');
```

5 Create a logical mask that identifies all the terms with an `ontology` property of `cellular component`.

```
mask = strcmp(ontologies,'cellular component');
```

6 Apply the logical mask to all the terms in the `GeneontObj` geneont object to return a structure containing only terms with an `ontology` property of `cellular component`.

```
cell_comp_terms = GeneontObj.terms(mask)

2362x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

There are 2,362 terms with an `ontology` property of `cellular component`.

part_of property

Class: term

Read-only numeric array containing GO identifiers of GO terms that have a "part of" relationship with this GO term

Description

part_of is a read-only property of the term class. part_of is a column vector containing GO identifiers. These GO identifiers specify other term objects to which the term object has a "part_of" relationship. The term object is a subset of the term objects specified by its part_of property.

Values

Possible values are identifiers of GO terms from the Gene Ontology database. Use the part_of property to determine GO identifiers of GO terms that have a "part of" relationship with a specific GO term.

Examples

Using the part_of Property to Determine term Objects with a "part of" Relationship

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Display the term objects to which the term object in the 18,703rd position has a "part of" relationship.

```
GeneontObj.terms(18703).part_of
```

```
ans =

     50802
```

Tip You can also use the `getancestors` method of a geneont object with the `'Relationtype'` property set to `'part_of'` to determine term objects with a "part of" relationship.

Tip If you know the GO identifier (for example, 42321) of a term object, instead of its index or position number (for example, 18703), you can use the following syntax to display the `part_of` property of a term object:

`GeneontObj(42321).terms.part_of`

For help converting the index or position number of a term object to its GO identifier, see the `term.id` property.

See Also
geneont.getdescendants | geneont.getrelatives | geneont.getancestors

synonym property

Class: term

Read-only array containing GO terms that are synonyms of this GO term

Description

synonym is a read-only property of the term class. synonym is a two-column cell array containing GO terms that are synonyms of this GO term. The first column contains a string specifying the type of synonym, such as `'exact_synonym'`, `'related_synonym'`, `'broad_synonym'`, `'narrow_synonym'`, or `'alt_id'`. The second column contains the GO identifier of the synonymous term or a string describing the synonymous term.

Values

Possible values are identifiers of GO terms from the Gene Ontology database. Use the synonym property to determine GO identifiers of synonymous term objects.

Examples

Using the synonym Property to Determine Synonymous term Objects

1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GeneontObj = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of term objects associated with the geneont object.

```
Gene Ontology object with 27769 Terms.
```

2 Display the term objects that are synonymous to the term object in the third position in the geneont object, `GeneontObj`.

```
synonyms = GeneontObj.terms(3).synonym
```

```
synonyms =

    'alt_id'            'GO:0019952'
    'alt_id'            'GO:0050876'
    'exact_synonym'     [1x39 char]
```

3   Because the exact synonym does not have a GO identifier listed, display the text of this synonym.

```
synonyms(3,2)

ans =

    '"reproductive physiological process" []'
```

4   Display the term objects that are synonymous to the term object in the 352nd position in the geneont object, `GeneontObj`.

```
GeneontObj.terms(352).synonym

ans =

    'alt_id'            'GO:0006374'
    'alt_id'            'GO:0006375'
    'related_synonym'          [1x26 char]
    'related_synonym'          [1x26 char]
    'narrow_synonym'           [1x51 char]
    'narrow_synonym'           [1x50 char]
    'broad_synonym'     '"mRNA splicing" []'
    'broad_synonym'            [1x22 char]
```

Tip If you know the GO identifier (for example, 398) of a term object, instead of its index or position number (for example, 352), you can use the following syntax to display the synonym of a term object:

```
GeneontObj(398).terms.synonym
```

For help converting the index or position number of a term object to its GO identifier, see the `term.id` property.