Bioinformatics Toolbox 3
Reference

MATLAB®

The MathWorks™
Accelerating the pace of engineering and science
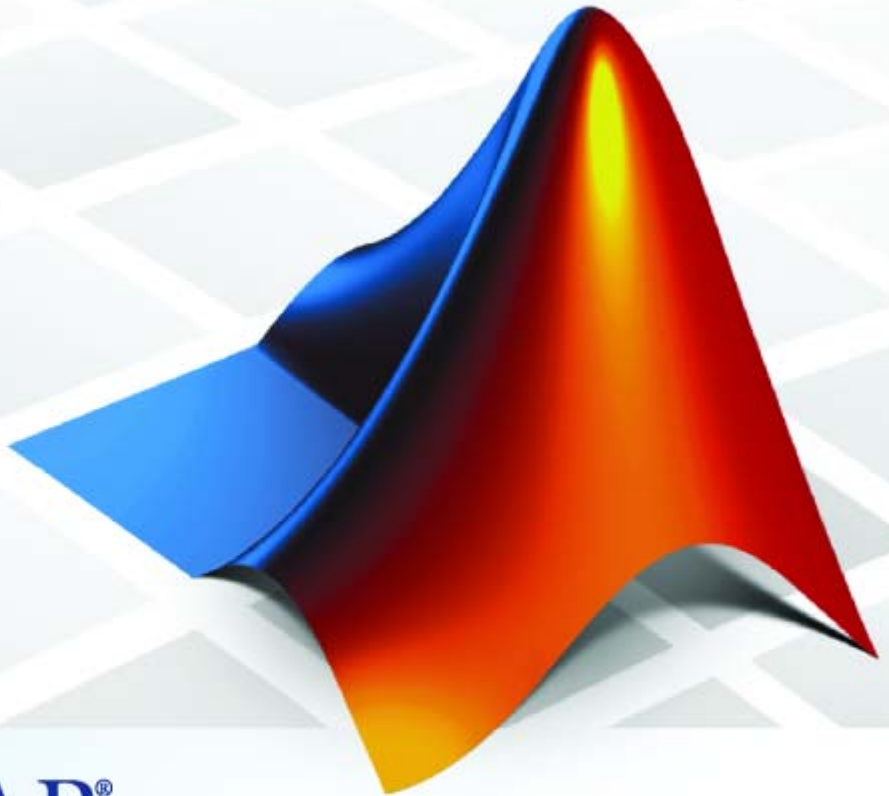
How to Contact The MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Bioinformatics Toolbox Reference

Revision History

Contents

Functions — By Category

1

Functions — Alphabetical List

2

Methods — By Category

3

1

Functions — By Category

Microarray Data Analysis and Visualization (p. 1-16)	Analyze and visualize microarray data with t tests, spatial plots, box plots, loglog plots, and intensity-ratio plots.
Microarray Normalization and Filtering (p. 1-17)	Normalize microarray data with lowess and mean normalization functions; filter raw data for cleanup before analysis.
Statistical Learning (p. 1-19)	Classify and identify features in data sets, set up cross-validation experiments, and compare different classification methods.
Mass Spectrometry File Formats, Preprocessing, and Visualization (p. 1-19)	Read data from common mass spectrometry file formats, preprocess raw mass spectrometry data from instruments, and analyze spectra to identify patterns and compounds.

Constructor

`biograph`	Create `biograph` object
`geneont`	Create `geneont` object
`phytree`	Create `phytree` object

Data Formats and Databases

`affyprobeseqread`	Read data file containing probe sequence information for Affymetrix GeneChip array
`affyread`	Read microarray data from Affymetrix GeneChip file

`agferead`	Read Agilent Feature Extraction Software file
`blastformat`	Create local BLAST database
`blastread`	Read data from NCBI BLAST report file
`blastreadlocal`	Read data from local BLAST report
`celintensityread`	Read probe intensities from Affymetrix CEL files
`cytobandread`	Read cytogenetic banding information
`emblread`	Read data from EMBL file
`fastaread`	Read data from FASTA file
`fastawrite`	Write to file using FASTA format
`galread`	Read microarray data from GenePix array list file
`genbankread`	Read data from GenBank file
`genpeptread`	Read data from GenPept file
`geosoftread`	Read Gene Expression Omnibus (GEO) SOFT format data
`getblast`	Retrieve BLAST report from NCBI Web site
`getembl`	Retrieve sequence information from EMBL database
`getgenbank`	Retrieve sequence information from GenBank database
`getgenpept`	Retrieve sequence information from GenPept database
`getgeodata`	Retrieve Gene Expression Omnibus (GEO) Sample (GSM) data

gethmmalignment	Retrieve multiple sequence alignment associated with hidden Markov model (HMM) profile from PFAM database
gethmmprof	Retrieve hidden Markov model (HMM) profile from PFAM database
gethmmtree	Retrieve phylogenetic tree data from PFAM database
getpdb	Retrieve protein structure data from Protein Data Bank (PDB) database
gprread	Read microarray data from GenePix Results (GPR) file
imageneread	Read microarray data from ImaGene Results file
jcampread	Read JCAMP-DX formatted files
multialignread	Read multiple-sequence alignment file
mzxmlread	Read mzXML file into MATLAB as structure
pdbread	Read data from Protein Data Bank (PDB) file
pdbwrite	Write to file using Protein Data Bank (PDB) format
pfamhmmread	Read data from PFAM-HMM file
phytreeread	Read phylogenetic tree file
phytreewrite	Write phylogenetic tree object to Newick-formatted file
scfread	Read trace data from SCF file
sptread	Read data from SPOT file

Trace Tools

scfread	Read trace data from SCF file
traceplot	Draw nucleotide trace plots

Sequence Conversion

aa2int	Convert amino acid sequence from letter to integer representation
aa2nt	Convert amino acid sequence to nucleotide sequence
aminolookup	Find amino acid codes, integers, abbreviations, names, and codons
baselookup	Nucleotide codes, abbreviations, and names
dna2rna	Convert DNA sequence to RNA sequence
int2aa	Convert amino acid sequence from integer to letter representation
int2nt	Convert nucleotide sequence from integer to letter representation
nt2aa	Convert nucleotide sequence to amino acid sequence
nt2int	Convert nucleotide sequence from letter to integer representation
rna2dna	Convert RNA sequence of nucleotides to DNA sequence
rnaconvert	Convert secondary structure of RNA sequence between bracket and matrix notations

seq2regexp	Convert sequence with ambiguous characters to regular expression
seqcomplement	Calculate complementary strand of nucleotide sequence
seqrcomplement	Calculate reverse complement of nucleotide sequence
seqreverse	Reverse letters or numbers in nucleotide sequence

Sequence Utilities

aminolookup	Find amino acid codes, integers, abbreviations, names, and codons
baselookup	Nucleotide codes, abbreviations, and names
blastlocal	Perform search on local BLAST database to create BLAST report
blastncbi	Create remote NCBI BLAST report request ID or link to NCBI BLAST report
cleave	Cleave amino acid sequence with enzyme
evalrasmolscript	Send RasMol script commands to Molecule Viewer window
featuresparse	Parse features from GenBank, GenPept, or EMBL data
geneticcode	Nucleotide codon to amino acid mapping
joinseq	Join two sequences to produce shortest supersequence

molviewer	Display and manipulate 3-D molecule structure
oligoprop	Calculate sequence properties of DNA oligonucleotide
palindromes	Find palindromes in sequence
pdbdistplot	Visualize intermolecular distances in Protein Data Bank (PDB) file
proteinplot	Characteristics for amino acid sequences
proteinpropplot	Plot properties of amino acid sequence
ramachandran	Draw Ramachandran plot for Protein Data Bank (PDB) data
randseq	Generate random sequence from finite alphabet
rebasecuts	Find restriction enzymes that cut protein sequence
restrict	Split nucleotide sequence at restriction site
revgeneticcode	Reverse mapping for genetic code
rnafold	Predict minimum free-energy secondary structure of RNA sequence
seqconsensus	Calculate consensus sequence
seqdisp	Format long sequence output for easy viewing
seqinsertgaps	Insert gaps into nucleotide or amino acid sequence
seqlogo	Display sequence logo for nucleotide or amino acid sequences
seqmatch	Find matches for every string in library

seqprofile	Calculate sequence profile from set of multiply aligned sequences
seqshoworfs	Display open reading frames in sequence

Sequence Statistics

aacount	Count amino acids in sequence
aminolookup	Find amino acid codes, integers, abbreviations, names, and codons
basecount	Count nucleotides in sequence
baselookup	Nucleotide codes, abbreviations, and names
codonbias	Calculate codon frequency for each amino acid in DNA sequence
codoncount	Count codons in nucleotide sequence
cpgisland	Locate CpG islands in DNA sequence
dimercount	Count dimers in sequence
isoelectric	Estimate isoelectric point for amino acid sequence
molweight	Calculate molecular weight of amino acid sequence
nmercount	Count number of n-mers in nucleotide or amino acid sequence
ntdensity	Plot density of nucleotides along sequence
seqshowwords	Graphically display words in sequence
seqwordcount	Count number of occurrences of word in sequence

Sequence Visualization

featuresmap	Draw linear or circular map of features from GenBank structure
rnaplot	Draw secondary structure of RNA sequence
seqtool	Open tool to interactively explore biological sequences

Pairwise Sequence Alignment

fastaread	Read data from FASTA file
nwalign	Globally align two sequences using Needleman-Wunsch algorithm
seqdotplot	Create dot plot of two sequences
showalignment	Display color-coded sequence alignment
swalign	Locally align two sequences using Smith-Waterman algorithm

Multiple Sequence Alignment

fastaread	Read data from FASTA file
multialign	Align multiple sequences using progressive method
multialignread	Read multiple-sequence alignment file
multialignviewer	Open viewer for multiple sequence alignments

profalign	Align two profiles using Needleman-Wunsch global alignment
seqpdist	Calculate pairwise distance between sequences
showalignment	Display color-coded sequence alignment

Scoring Matrices

blosum	Return BLOSUM scoring matrix
dayhoff	Return Dayhoff scoring matrix
gonnet	Return Gonnet scoring matrix
nuc44	Return NUC44 scoring matrix for nucleotide sequences
pam	Return PAM scoring matrix

Phylogenetic Tree Tools

dnds	Estimate synonymous and nonsynonymous substitution rates
dndsml	Estimate synonymous and nonsynonymous substitution rates using maximum likelihood method
gethmmtree	Retrieve phylogenetic tree data from PFAM database
phytreeread	Read phylogenetic tree file

phytreetool	View, edit, and explore phylogenetic tree data
phytreewrite	Write phylogenetic tree object to Newick-formatted file
seqinsertgaps	Insert gaps into nucleotide or amino acid sequence
seqlinkage	Construct phylogenetic tree from pairwise distances
seqneighjoin	Neighbor-joining method for phylogenetic tree reconstruction
seqpdist	Calculate pairwise distance between sequences

Graph Theory

graphallshortestpaths	Find all shortest paths in graph
graphcluster	Find strongly or weakly connected components in graph
graphconncomp	Find strongly or weakly connected components in graph
graphisdag	Test for cycles in directed graph
graphisomorphism	Find isomorphism between two graphs
graphisspantree	Determine if tree is spanning tree
graphmaxflow	Calculate maximum flow and minimum cut in directed graph
graphminspantree	Find minimal spanning tree in graph
graphpred2path	Convert predecessor indices to paths
graphshortestpath	Solve shortest path problem in graph

`graphtopoorder`	Perform topological sort of directed acyclic graph
`graphtraverse`	Traverse graph by following adjacent nodes

Gene Ontology

`goannotread`	Annotations from Gene Ontology annotated file
`num2goid`	Convert numbers to Gene Ontology IDs

Protein Analysis

`aacount`	Count amino acids in sequence
`aminolookup`	Find amino acid codes, integers, abbreviations, names, and codons
`atomiccomp`	Calculate atomic composition of protein
`cleave`	Cleave amino acid sequence with enzyme
`evalrasmolscript`	Send RasMol script commands to Molecule Viewer window
`isoelectric`	Estimate isoelectric point for amino acid sequence
`molviewer`	Display and manipulate 3-D molecule structure
`molweight`	Calculate molecular weight of amino acid sequence

pdbdistplot	Visualize intermolecular distances in Protein Data Bank (PDB) file
proteinplot	Characteristics for amino acid sequences
proteinpropplot	Plot properties of amino acid sequence
ramachandran	Draw Ramachandran plot for Protein Data Bank (PDB) data
rebasecuts	Find restriction enzymes that cut protein sequence

Profile Hidden Markov Models

gethmmalignment	Retrieve multiple sequence alignment associated with hidden Markov model (HMM) profile from PFAM database
gethmmprof	Retrieve hidden Markov model (HMM) profile from PFAM database
gethmmtree	Retrieve phylogenetic tree data from PFAM database
hmmprofalign	Align query sequence to profile using hidden Markov model alignment
hmmprofestimate	Estimate profile hidden Markov model (HMM) parameters using pseudocounts
hmmprofgenerate	Generate random sequence drawn from profile hidden Markov model (HMM)
hmmprofmerge	Concatenate prealigned strings of several sequences to profile hidden Markov model (HMM)

hmmprofstruct	Create profile Hidden Markov Model (HMM) structure
pfamhmmread	Read data from PFAM-HMM file
showhmmprof	Plot hidden Markov model (HMM) profile

Microarray File Formats

affyprobeseqread	Read data file containing probe sequence information for Affymetrix GeneChip array
affyread	Read microarray data from Affymetrix GeneChip file
agferead	Read Agilent Feature Extraction Software file
celintensityread	Read probe intensities from Affymetrix CEL files
galread	Read microarray data from GenePix array list file
geosoftread	Read Gene Expression Omnibus (GEO) SOFT format data
getgeodata	Retrieve Gene Expression Omnibus (GEO) Sample (GSM) data
gprread	Read microarray data from GenePix Results (GPR) file
imageneread	Read microarray data from ImaGene Results file
sptread	Read data from SPOT file

Microarray Utility

`magetfield`	Extract data from microarray structure
`probelibraryinfo`	Create table of probe set library information
`probesetlink`	Display probe set information on NetAffx Web site
`probesetlookup`	Look up information for probe set
`probesetplot`	Plot Affymetrix probe set intensity values
`probesetvalues`	Create table of Affymetrix probe set intensity values

Microarray Data Analysis and Visualization

`cghcbs`	Perform circular binary segmentation (CBS) on array-based comparative genomic hybridization (aCGH) data
`chromosomeplot`	Plot chromosome ideogram with G-banding pattern
`clustergram`	Create dendrogram and heat map
`maboxplot`	Box plot for microarray data
`mafdr`	Estimate false discovery rate (FDR) of differentially expressed genes from two experimental conditions or phenotypes
`maimage`	Spatial image for microarray data
`mairplot`	Create intensity versus ratio scatter plot of microarray data

maloglog	Create loglog plot of microarray data
mapcaplot	Create Principal Component Analysis (PCA) plot of microarray data
mattest	Perform two-tailed t-test to evaluate differential expression of genes from two experimental conditions or phenotypes
mavolcanoplot	Create significance versus gene expression ratio (fold change) scatter plot of microarray data
redgreencmap	Create red and green color map

Microarray Normalization and Filtering

affyinvarsetnorm	Perform rank invariant set normalization on probe intensities from multiple Affymetrix CEL or DAT files
affyprobeaffinities	Compute Affymetrix probe affinities from their sequences and MM probe intensities
exprprofrange	Calculate range of gene expression profiles
exprprofvar	Calculate variance of gene expression profiles
gcrma	Perform GC Robust Multi-array Average (GCRMA) background adjustment, quantile normalization, and median-polish summarization on Affymetrix microarray probe-level data

gcrmabackadj	Perform GC Robust Multi-array Average (GCRMA) background adjustment on Affymetrix microarray probe-level data using sequence information
geneentropyfilter	Remove genes with low entropy expression values
genelowvalfilter	Remove gene profiles with low absolute values
generangefilter	Remove gene profiles with small profile ranges
genevarfilter	Filter genes with small profile variance
mainvarsetnorm	Perform rank invariant set normalization on gene expression values from two experimental conditions or phenotypes
malowess	Smooth microarray data using Lowess method
manorm	Normalize microarray data
quantilenorm	Quantile normalization over multiple arrays
rmabackadj	Perform background adjustment on Affymetrix microarray probe-level data using Robust Multi-array Average (RMA) procedure
rmasummary	Calculate gene (probe set) expression values from Affymetrix microarray probe-level data using Robust Multi-array Average (RMA) procedure
zonebackadj	Perform background adjustment on Affymetrix microarray probe-level data using zone-based method

Statistical Learning

classperf	Evaluate performance of classifier
crossvalind	Generate cross-validation indices
knnclassify	Classify data using nearest neighbor method
knnimpute	Impute missing data using nearest-neighbor method
optimalleaforder	Determine optimal leaf ordering for hierarchical binary cluster tree
randfeatures	Generate randomized subset of features
rankfeatures	Rank key features by class separability criteria
samplealign	Align two data sets containing sequential observations by introducing gaps
svmclassify	Classify data using support vector machine
svmsmoset	Create or edit Sequential Minimal Optimization (SMO) options structure
svmtrain	Train support vector machine classifier

Mass Spectrometry File Formats, Preprocessing, and Visualization

jcampread	Read JCAMP-DX formatted files
msalign	Align peaks in mass spectrum to reference peaks

`msbackadj`	Correct baseline of mass spectrum
`msdotplot`	Plot set of peak lists from LC/MS or GC/MS data set
`msheatmap`	Create pseudocolor image of set of mass spectra
`mslowess`	Smooth mass spectrum using nonparametric method
`msnorm`	Normalize set of mass spectra
`mspalign`	Align mass spectra from multiple peak lists from LC/MS or GC/MS data set
`mspeaks`	Convert raw mass spectrometry data to peak list (centroided data)
`msppresample`	Resample mass spectrometry signal while preserving peaks
`msresample`	Resample mass spectrometry signal
`mssgolay`	Smooth mass spectrum with least-squares polynomial
`msviewer`	Explore mass spectrum or set of mass spectra
`mzxml2peaks`	Convert mzXML structure to peak list
`mzxmlread`	Read mzXML file into MATLAB as structure
`samplealign`	Align two data sets containing sequential observations by introducing gaps

Functions — Alphabetical List

aa2int

Purpose   Convert amino acid sequence from letter to integer representation

Syntax   *SeqInt* = aa2int(*SeqChar*)

Arguments   *SeqChar*   Either of the following:

- Character string of single-letter codes specifying an amino acid sequence. See the table Mapping Amino Acid Letters to Integers on page 2-2 for valid codes. Unknown characters are mapped to 0. Integers are arbitrarily assigned to IUB/IUPAC letters.

- Structure containing a Sequence field that contains an amino acid sequence, such as returned by fastaread, getembl, getgenpept, or getpdb.

Return Values   *SeqInt*   Row vector of integers specifying an amino acid sequence.

Mapping Amino Acid Letters 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

Mapping Amino Acid Letters to Integers (Continued)

Amino Acid	Code	Integer
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
Any amino acid	X	23
Translation stop	*	24
Gap of indeterminate length	-	25
Unknown or any character or symbol not in table	?	0

Description   *SeqInt* = aa2int(*SeqChar*) converts *SeqChar*, a string of single-letter codes specifying an amino acid sequence, to *SeqInt*, a 1-by-N array of integers specifying the same amino acid sequence. See the table Mapping Amino Acid Letters to Integers on page 2-2 for valid codes.

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

Convert a random amino acid sequence of letters to integers.

1 Create a random character 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

Bioinformatics Toolbox functions: aminolookup, int2aa, int2nt, nt2int

Purpose	Convert amino acid sequence to nucleotide sequence
Syntax	*SeqNT* = aa2nt(*SeqAA*) *SeqNT* = aa2nt(*SeqAA*, ...'GeneticCode', *GeneticCodeValue*, ...) *SeqNT* = aa2nt(*SeqAA*, ...'Alphabet' *AlphabetValue*, ...)

Arguments

SeqAA	Amino acid sequence. Enter a character string or a vector of integers from the table Mapping Amino Acid Letters to Integers on page 2-2. Examples: 'ARN' or [1 2 3]
GeneticCodeValue	Integer or string specifying a genetic code. Enter a code number or code name from the table Genetic Code on page 2-6. If you use a code name, you can truncate the name to the first two characters of the name. Default is 1 or 'Standard'.
AlphabetValue	String specifying a nucleotide alphabet. Choices are: • 'DNA' (default) — Uses the symbols A, C, G, T • 'RNA' — Uses the symbols A, C, G, U

Return Values

SeqNT	Nucleotide sequence specified by a string of letters.

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

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 more than one possible nucleotide codon, 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. Choices for *GeneticCodeValue* are an integer or string specifying a code number or code name from the table Genetic Code on page 2-6. Default is 1 or 'Standard'.

SeqNT = aa2nt(*SeqAA*, ...'Alphabet' *AlphabetValue*, ...) specifies a nucleotide alphabet. Choices for *AlphabetValue* are 'DNA' (default) or 'RNA'.

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

Standard Genetic Code (Continued)

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 or any amino acid	X	Random codon
Translation stop	*	TAA TAG TGA

Standard Genetic Code (Continued)

Amino Acid Name	Amino Acid Code	Nucleotide Codon
Gap of indeterminate length	-	- - -
Any character or any symbol not in table	?	???

Examples

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

```
aa2nt('MATLAB')

Warning: The sequence contains ambiguous characters.
ans =
ATGGCAACCCTGGCGAAT
```

2 Use the Vertebrate Mitochondrial genetic code.

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

ans =
ATGGCAACTCTAGCGCCT
```

3 Use the genetic code for the Echinoderm Mitochondrial RNA alphabet.

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

Warning: The sequence contains ambiguous characters.
ans =
AUGGCUACAUUGGCUGAU
```

4 Convert a sequence with the ambiguous amino acid character B.

```
aa2nt('abcd')

Warning: The sequence contains ambiguous characters.
ans =
GCCACATGCGAC
```

See Also   Bioinformatics Toolbox functions: `geneticcode`, `nt2aa`, `revgeneticcode`, `seqtool`

MATLAB function: `rand`

Purpose   Count amino acids in sequence

Syntax
```
Amino = aacount(SeqAA)
aacount(..., 'PropertyName', PropertyValue,...)
aacount(..., 'Chart', ChartValue)
aacount(..., 'Others', OthersValue)
aacount(..., 'Structure', StructureValue)
```

Arguments

SeqAA	Amino acid sequence. Enter a character string or vector of integers from the Mapping Amino Acid Letters to Integers on page 2-2 table. Examples: `'ARN'` or `[1 2 3]`. You can also enter a structure with the field Sequence.
ChartValue	Property to select a type of plot. Enter either `'pie'` or `'bar'`.
OthersValue	Property to control the counting of ambiguous characters individually. Enter either `'full'` or `'bundle'`(default).
StructureValue	Property to control blocking the unknown characters warning and to not count unknown characters.

Description   *Amino* = aacount(*SeqAA*) counts the type and number of amino acids in an amino acid sequence (*SeqAA*) and returns the counts in a 1-by-1 structure (*Amino*) with 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 V).

- If a sequence contains amino acids with ambiguous characters (B, Z, X), the stop character (*), or gaps indicated with a hyphen (-), the field Others is added to the structure and a warning message is displayed.

  ```
  Warning: Symbols other than the standard 20 amino acids
  appear in the sequence.
  ```

- If a sequence contains any characters other than the 20 standard amino acids, ambiguous characters, stop, and gap characters, the characters are counted in the field Others and a warning message is displayed.

      Warning: Sequence contains unknown characters. These will
      be ignored.

- If the property Others = 'full', this function lists the ambiguous characters separately, asterisks are counted in a new field (Stop), and hyphens are counted in a new field (Gap).

aacount(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs:

aacount(..., 'Chart', *ChartValue*) creates a chart showing the relative proportions of the amino acids.

aacount(..., 'Others', *OthersValue*), when *OthersValue* is 'full'', counts the ambiguous amino acid characters individually instead of adding them together in the field Others.

aacount(..., 'Structure', *StructureValue*), when *StructureValue* is 'full', blocks the unknown characters warning and ignores counting unknown characters.

- aacount(*SeqAA*) — Display 20 amino acids, and only if there are ambiguous and unknown characters, add an Others field with the counts.

- aacount(*SeqAA*, 'Others', 'full') — Display 20 amino acids, 3 ambiguous amino acids, stops, gaps, and only if there are unknown characters, add an Others field with the unknown counts.

- aacount(*SeqAA*, 'Structure', 'full') — Display 20 amino acids and always display an Others field. If there are ambiguous and unknown characters, add counts to the Others field; otherwise display 0.

- aacount(*SeqAA*, 'Others', 'full', 'Structure', 'full') —
  Display 20 amino acids, 3 ambiguous amino acids, stops, gaps, and
  Others field. If there are unknown characters, add counts to the
  Others field otherwise display 0.

Examples

1 Create a sequence.

```
Seq = aacount('MATLAB')
```

2 Count the amino acids in the sequence.

```
AA = aacount(Seq)

Warning: Symbols other than the standard 20 amino acids appear
in the sequence.
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
Others: 1
```

3 Get the count for alanine (A) residues.

```
AA.A
ans =
        2
```

See Also   Bioinformatics Toolbox functions: aminolookup, atomiccomp,
basecount, codoncount, dimercount, isoelectric, molweight,
proteinplot, seqtool

Purpose   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(..., 'StopPrctile', *StopPrctileValue*, ...)
... affyinvarsetnorm(..., 'RayPrctile', *RayPrctileValue*, ...)
... 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.

affyinvarsetnorm

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].
StopPrctileValue	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.

RayPrctileValue	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.

affyinvarsetnorm

- 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 = 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(..., 'StopPrctile',` *StopPrctileValue*`, ...)` 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(..., 'RayPrctile',` *RayPrctileValue*`, ...)` 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*.
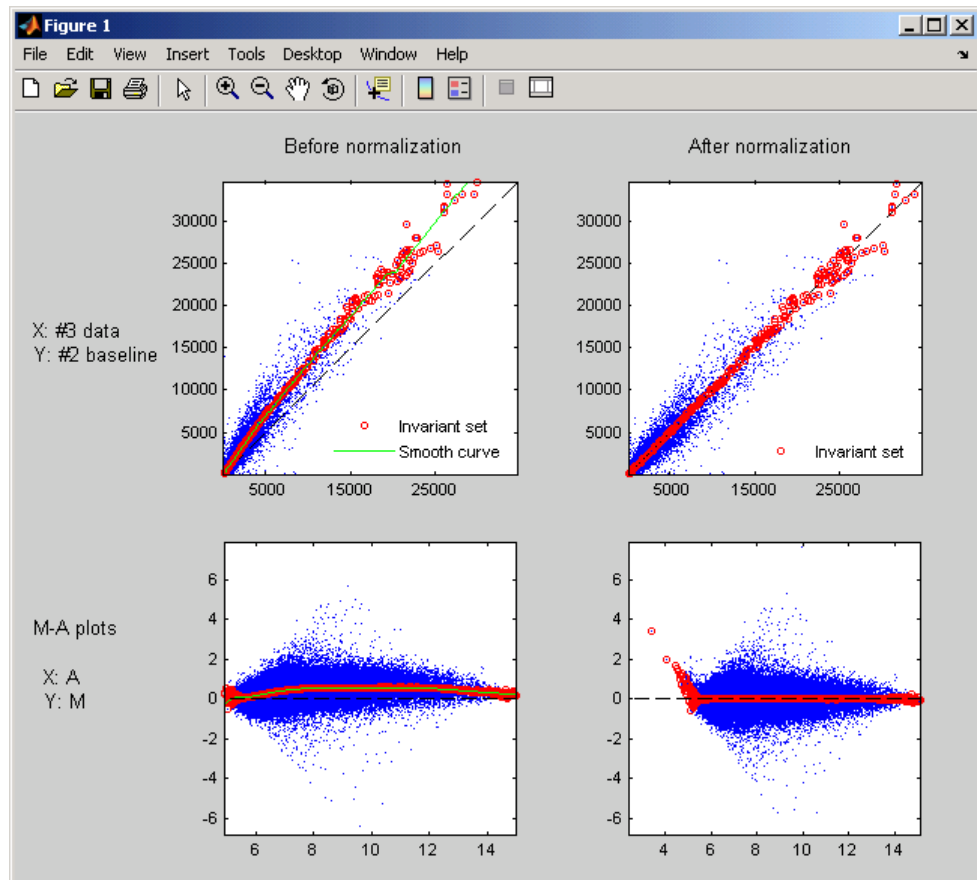
Examples

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

```
load prostatecancerrawdata
```

2 Normalize the data in pmMatrix, using the affyinvarsetnorm function.

```
NormMatrix = affyinvarsetnorm(pmMatrix);
```

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

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://biosun1.harvard.edu/complab/dchip/-normalizing%20arrays.htm#isn`

[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   Bioinformatics Toolbox functions: `affyread`, `celintensityread`, `mainvarsetnorm`, `malowess`, `manorm`, `quantilenorm`, `rmabackadj`, `rmasummary`

Purpose 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*, ...)

affyprobeaffinities

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).

Return Values

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.

affyprobeaffinities

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

1 Load the MAT file, included with Bioinformatics Toolbox, 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
```

2 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);
```
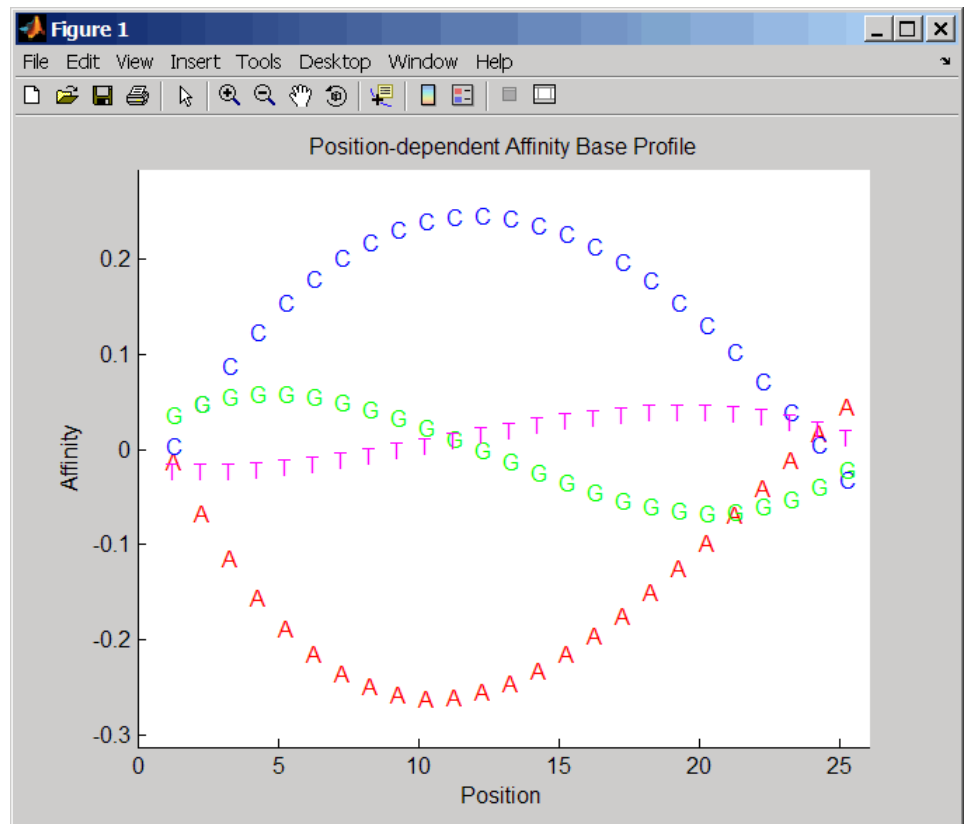
The `prostatecancerrawdata.mat` file used in this example contains data from Best et al., 2005.

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   Bioinformatics Toolbox functions: `affyprobeseqread`, `affyread`, `celintensityread`, `probelibraryinfo`

affyprobeseqread

Purpose   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, ...)

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
                                 Directory (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 Directory (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 directory or path and directory where *SeqFile* is stored.	
CDFPathValue	String specifying a directory or path and directory where *CDFFile* is stored.	
SeqOnlyValue	Controls the return of a structure, *Struct*, with only one field, SequenceMatrix. Choices are true or false (default).	

Return Values

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.

affyprobeseqread

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 directory where *SeqFile* is stored.

Struct = affyprobeseqread(*SeqFile*, *CDFFile*, ...'CDFPath', *CDFPathValue*, ...) lets you specify a path directory 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 Directory.

```
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 directory 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

Bioinformatics Toolbox functions: affyinvarsetnorm, affyread, celintensityread, int2nt, probelibraryinfo, probesetlink, probesetlookup, probesetplot, probesetvalues

Purpose   Read microarray data from Affymetrix GeneChip file

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

affyread

Arguments

File   String specifying a file name or a path and file name of one of the following Affymetrix file types:

- **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.

- **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 with which the probe set is associated.

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

LibraryPath String specifying the path and directory 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 directory for the CDF file. If it does not find the CDF file, it still reads the CHP file, but the probe set names and types will be omitted from the return value, *AffyStruct*.

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

Description

> **Note** This function does not work on the Solaris platform.

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

AffyStruct = affyread(*File*, *LibraryPath*) specifies the path and directory 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 directory for the CDF file. If it does not find the CDF file, it still reads the CHP file, but the probe set names and types will be omitted from the return value, *AffyStruct*.

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. 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/products/software/specific/dtt.affx
> ```
>
> You will have to register and log in at the Affymetrix Web site to
> download the Affymetrix Data Transfer Tool.

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

File Types EXP, DAT, CEL, CHP, CDF, and GIN

Field	Description
Name	File name.
DataPath	Path and directory of the file.
LibPath	Path and directory of the CDF and GIN library files associated with the file being read.
FullPathName	Path and directory of the file.

File Types EXP, DAT, CEL, CHP, CDF, and GIN (Continued)

Field	Description
ChipType	Name of the Affymetrix GeneChip array (for example, DrosGenome1 or HG-Focus).
Date	Date the file was created.

EXP File

Field	Description
ChipLot Operator SampleType SampleDesc Project Comments Reagents ReagentLot Protocol Station Module HydridizeDate ScanPixelSize ScanFilter ScanDate ScannerID NumberOfScans ScannerType NumProtocolSteps ProtocolSteps	Information about experimental conditions and protocols captured by the Affymetrix software.

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 LowerRightX LowerRightY	Pixel coordinates of the scanned image.

DAT File (Continued)

Field	Description
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 probes that are masked and not used in subsequent processing.
NumOutliers	Number of cells identified as outliers (very high or very low intensity) by the image processing step.
NumProbes	Number of probes (Rows * Cols) on the GeneChip array.

CEL File (Continued)

Field	Description
UpperLeftX UpperLeftY UpperRightX UpperRightY LowerLeftX LowerLeftY LowerRightX LowerRightY	Pixel coordinates of the scanned image.
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 columns of this array are contained in the ProbeColumnNames field.

CHP File

Field	Description
AssayType	Type of assay that the GeneChip array contained (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 needed 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.
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.

CHP File (Continued)

Field	Description
ProbeSets (Expression GeneChip array)	A 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 outliers and masked probes have been removed. • 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. • Change — When CompDataExists is true, describes how the probe is changed 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.

CHP File (Continued)

Field	Description
ProbeSets (Genotyping GeneChip array)	A 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 — A 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)	A NumProbeSets-by-1 structure array containing information for each resequencing probe set, including the following fields: • CalledBases — A 1-by-NumProbeSets character array containing the bases called by the resequencing algorithm. Possible values are a, c, g, t, and n. • Scores — A 1-by-NumProbeSets array containing the score associated with each base call.

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.

CDF File (Continued)

Field	Description
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 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). • 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.
ProbeSets	A 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 is the name of the probe set. For genotyping arrays, this is 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(s) of the probe sets.
SourceURL	Source URL(s) 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 demo 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 you download the demo data, you will need the Affymetrix Data Transfer Tool to extract the CEL, DAT, and CHP files from a DTT file. You can download the Affymetrix Data Transfer Tool from:

> http://www.affymetrix.com/products/software/specific/dtt.affx

The following example assumes that files
Ecoli-antisense-121502.CEL, Ecoli-antisense-121502.dat,
and Ecoli-antisense-121502.chp are stored on the MATLAB
search path or in the current directory. 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 Display a spatial plot of the probe intensities.

```
maimage(celStruct, 'Intensity')
```
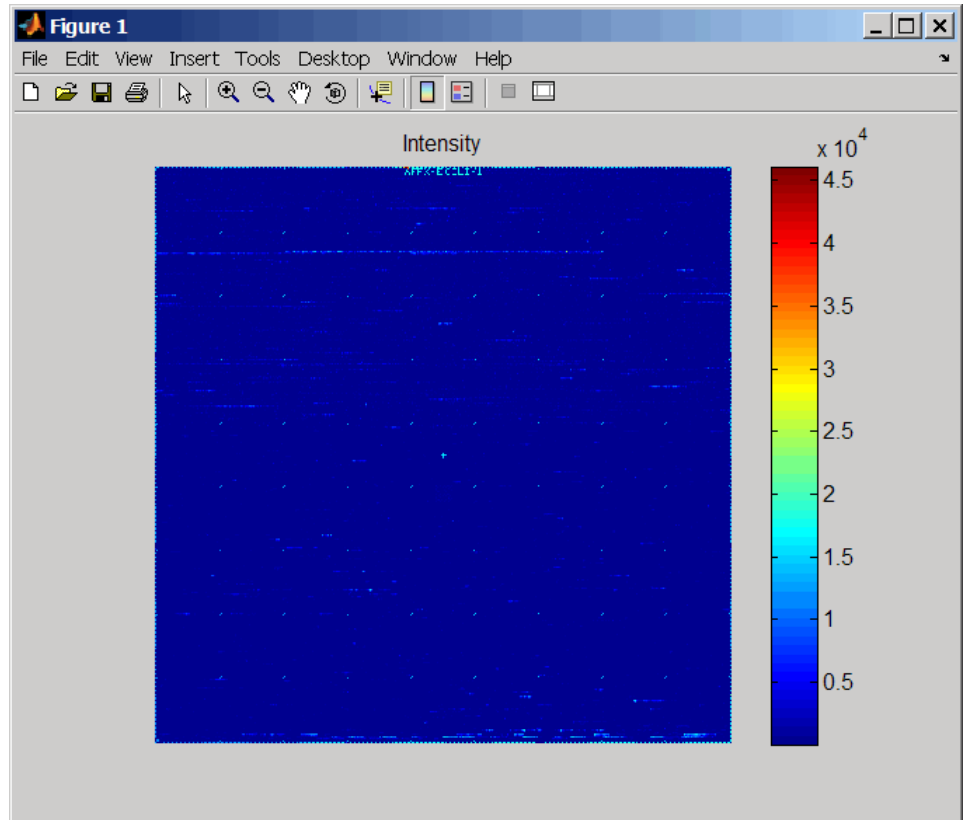

3 Zoom in on a specific region of the plot.

```
axis([200 340 0 70])
```
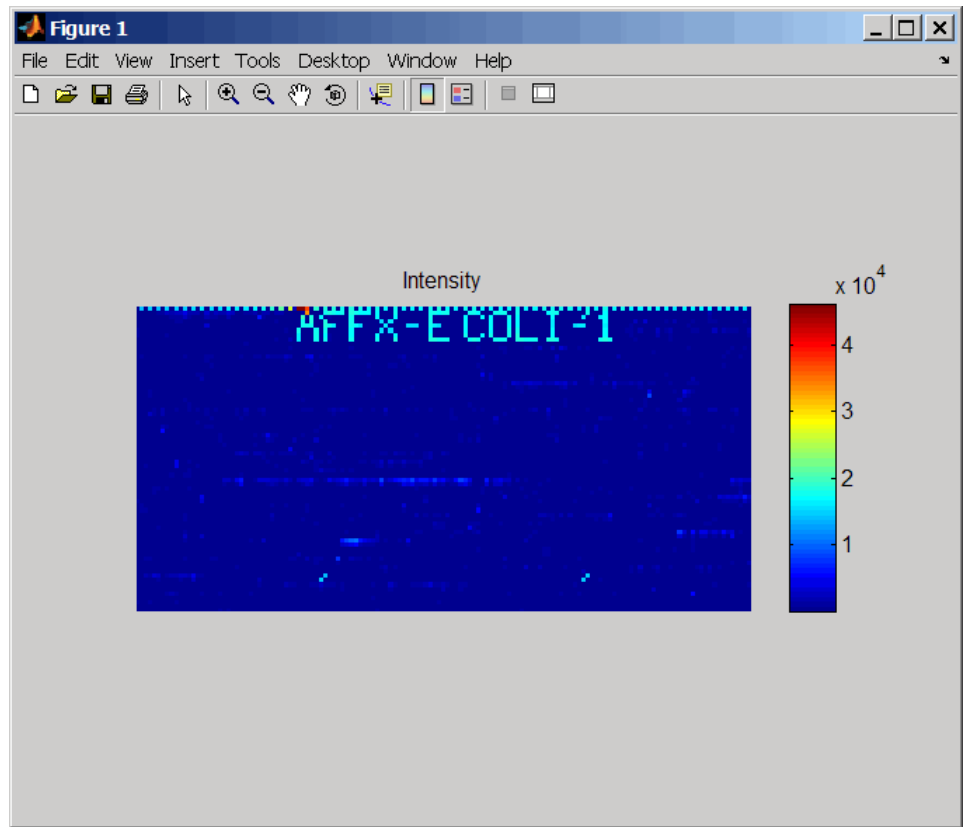
affyread

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
```
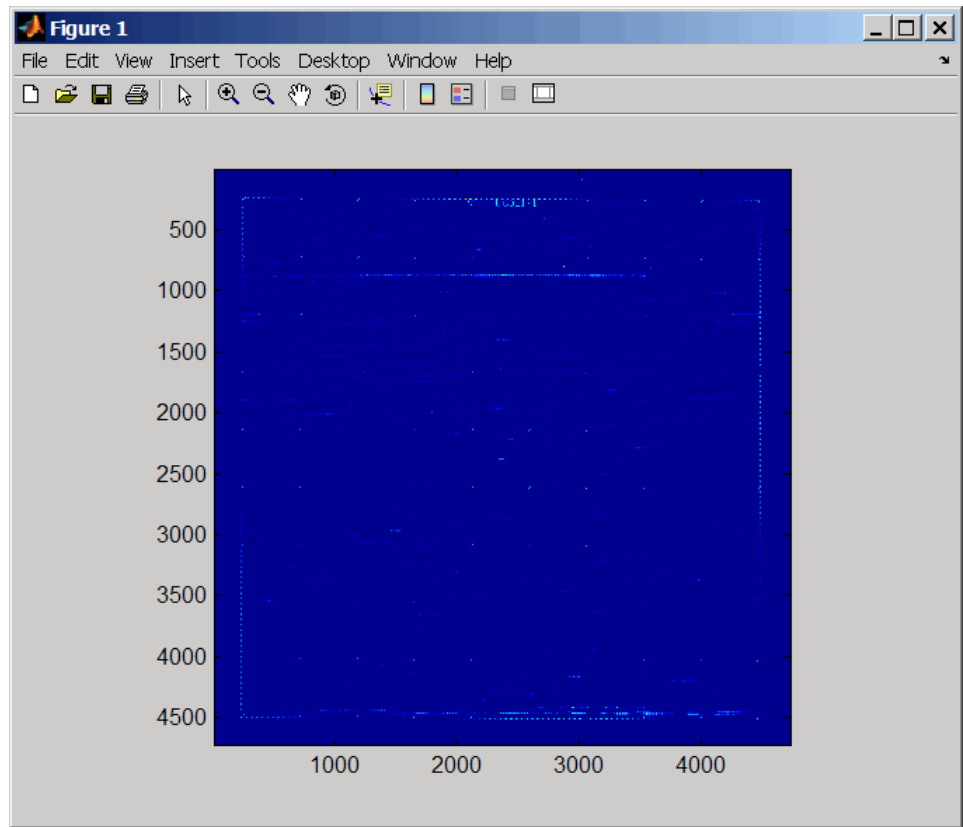
5 Zoom in on a specific region of the plot.

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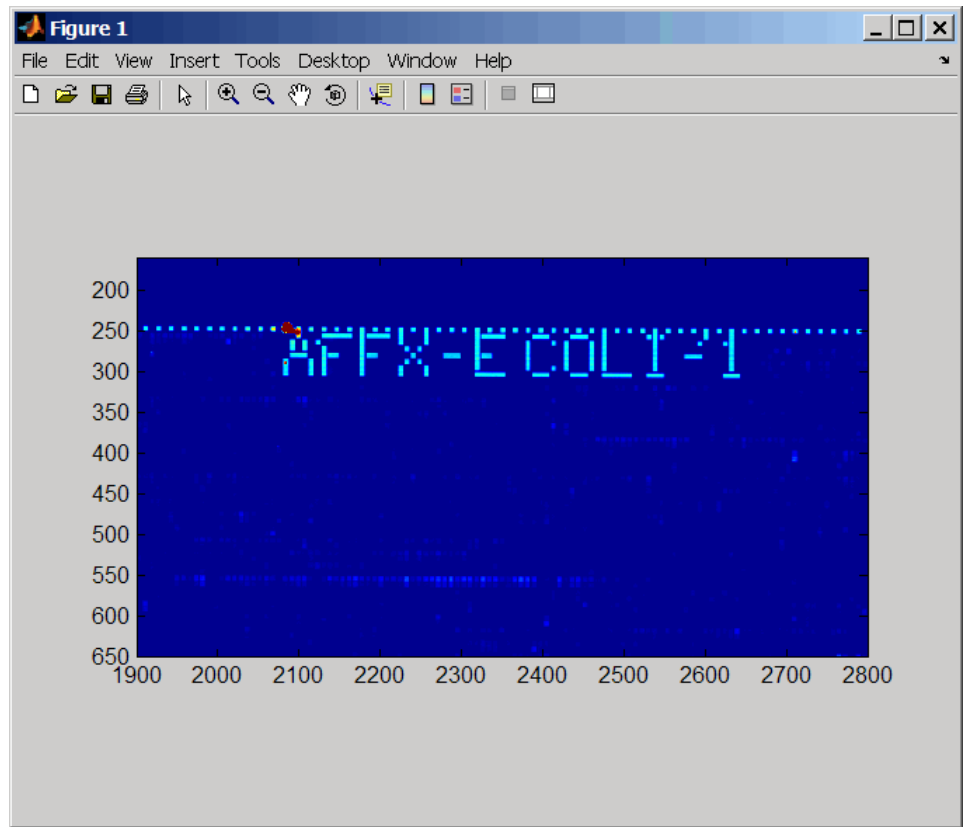
6 Read the contents of a CHP file into a MATLAB structure, specifying the location of the associated CDF library file. Then extract information for probe set 3315278.

```
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]
```

See Also   Bioinformatics Toolbox functions: agferead, celintensityread, gprread, probelibraryinfo, probesetlink, probesetlookup, probesetplot, probesetvalues, sptread

agferead

Purpose	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 micoararry 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. For more information about this software, see:

    http://www.chem.agilent.com/scripts/pds.asp?lpage=2547

Examples   **1** Read in a sample Agilent Feature Extraction Software file. Note that the file fe_sample.txt is not provided with Bioinformatics Toolbox.

```
agfeStruct = agferead('fe_sample.txt')
```

2 Plot the median foreground.

```
maimage(agfeStruct,'gMedianSignal');
maboxplot(agfeStruct,'gMedianSignal');
```

See Also   Bioinformatics Toolbox functions: `affyread`, `celintensityread`, `galread`, `geosoftread`, `gprread`, `imageneread`, `magetfield`, `sptread`

aminolookup

Purpose   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	Character 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 on page 2-57.
CodeValue	String specifying a single-letter code representing an amino acid. For valid single-letter codes, see the table Amino Acid Lookup on page 2-57.
IntegerValue	Single integer representing an amino acid. For valid integers, see the table Amino Acid Lookup on page 2-57.
AbbreviationValue	String specifying a three-letter abbreviation representing an amino acid. For valid three-letter abbreviations, see the table Amino Acid Lookup on page 2-57.
NameValue	String specifying an amino acid name. For valid amino acid names, see the table Amino Acid Lookup on page 2-57.

Amino Acid Lookup

Code	Integer	Abbreviation	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

Amino Acid Lookup (Continued)

Code	Integer	Abbreviation	Name	Codons
V	20	Val	Valine	GUU GUC GUA GUG
B	21	Asx	Asparagine or Aspartic acid (Aspartate)	AAU AAC GAU GAC
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

Description   aminolookup displays a table of amino acid codes, integers, abbreviations, names, and codons.

aminolookup(*SeqAA*) converts between single-letter codes and three-letter abbreviations for an amino acid sequence. If the input is a character string of single-letter codes, then the output is a character string of three-letter abbreviations. If the input is a character string of three-letter abbreviations, then the output is a character 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   Bioinformatics Toolbox functions: aa2int, aacount, geneticcode, int2aa, nt2aa, revgeneticcode

Purpose   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 Letters to Integers on page 2-2. 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** Get an amino acid sequence from the NCBI GenPept database.

```
rhodopsin = getgenpept('NP_000530');
```

2 Count the atoms in a sequence.

```
rhodopsinAC = atomiccomp(rhodopsin)

rhodopsinAC =

    C: 1814
    H: 2725
    N: 423
    O: 477
    S: 25
```

3 Retrieve the number of carbon atoms in the sequence.

```
rhodopsinAC.C

ans =

      1814
```

atomiccomp

See Also   Bioinformatics Toolbox functions: `aacount`, `molweight`, `proteinplot`

Purpose   Count nucleotides in sequence

Syntax

NumberBases = basecount(*SeqNT*)
basecount(..., '*PropertyName*', *PropertyValue*,...)
basecount(..., 'Chart', *ChartValue*)
basecount(..., 'Others', *OthersValue*)
basecount(..., 'Structure', *StructureValue*)

Arguments

SeqNT	Nucleotide sequence. Enter a character string with the letters A, T, U, C, and G. The count for U characters is included with the count for T characters. You can also enter a structure with the field Sequence.
ChartValue	String specifying the type of chart to plot. Choices are 'pie' or 'bar'.
OthersValue	String specifying whether to count ambiguous characters individually. Choices are 'full' or 'bundle' (default).

Description   *NumberBases* = basecount(*SeqNT*) counts the number of bases in a nucleotide sequence (SeqNT) and returns the base counts in a 1-by-1 structure (Bases) with the fields A, C, G, T.

- For sequences with the character U, the number of U characters is added to the number of T characters.

- If a sequence contains ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, N), or gaps indicated with a hyphen (-), this function creates a field Others and displays a warning message:

  ```
  Warning: Ambiguous symbols 'symbol list' appear
  in the sequence.
  These will be in Others.
  ```

- If a sequence contains undefined nucleotide characters (E F H I J L O P Q X Z), the characters are counted in the field Others and a warning message is displayed.

  ```
  Warning: Unknown symbols 'symbol list' appear
  in the sequence.
  These will be ignored.
  ```

- If the property Others = 'full', ambiguous characters are listed separately and hyphens are counted in a new field (Gaps).

basecount(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs:

basecount(..., 'Chart', *ChartValue*) creates a chart showing the relative proportions of the nucleotides.

basecount(..., 'Others', *OthersValue*), when *OthersValue* is 'full', counts all the ambiguous nucleotide symbols individually instead of bundling them together into the Others field of the output structure.

basecount(..., 'Structure', *StructureValue*), when *StructureValue* is 'full' , blocks the unknown characters warning and ignores counting unknown characters.

- basecount(SeqNT) — Display four nucleotides, and only if there are ambiguous and unknown characters, add an Others field with the counts.

- basecount(SeqNT, 'Others', 'full') — Display four nucleotides, 11 ambiguous nucleotides, gaps, and only if there are unknown characters, add an Others field with the unknown counts.

- basecount(SeqNT, 'Structure', 'full') — Display four nucleotides and always display an Others field. If there are ambiguous and unknown characters, add counts to the Others field; otherwise display 0.

- `basecount(SeqNT, 'Others', 'full', 'Structure', 'full')`
  — Display 4 nucleotides, 11 ambiguous nucleotides, gaps, and the
  `Others` field. If there are unknown characters, add counts to the
  `Others` field; otherwise display `0`.

Examples

1 Count the number of bases in a DNA sequence.

```
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 with ambiguous characters.

```
basecount('ABCDGGCCAAGCGAGCTTG','Others','full')

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
     Gaps: 0
```

See Also   Bioinformatics Toolbox functions: aacount, baselookup, codoncount, cpgisland, dimercount, nmercount, ntdensity, seqtool

Purpose   Nucleotide codes, abbreviations, and names

Syntax   baselookup('Complement', *SeqNT*)
                baselookup('Code', *CodeValue*)
                baselookup('Integer', *IntegerValue*)
                baselookup('Name', *NameValue*)

Arguments

SeqNT	Nucleotide sequence. Enter a character string of single-letter codes from the table Nucleotide Lookup on page 2-67.
	In addition to a single nucleotide sequence, *SeqNT* can be a cell array of sequences, or a two-dimensional character array of sequences. The complement for each sequence is determined independently.
CodeValue	Nucleotide letter code. Enter a single character from the table Nucleotide Lookup on page 2-67. *CodeValue* can also be a cell array or a two-dimensional character array.
IntegerValue	Nucleotide integer. Enter an integer from the table Nucleotide Lookup on page 2-67. Integers are arbitrarily assigned to IUB/IUPAC letters.
NameValue	Nucleotide name. Enter a nucleotide name from the table Nucleotide Lookup on page 2-67. *NameValue* can also be a single name, a cell array, or a two-dimensional character array.

Nucleotide Lookup

Code	Integer	Base Name	Meaning	Complement
A	1	Adenine	A	T
C	2	Cytosine	C	G

Nucleotide Lookup (Continued)

Code	Integer	Base Name	Meaning	Complement
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	N
-	16	Gap of indeterminate length	Gap	-

Description   baselookup('Complement', *SeqNT*) displays the complementary nucleotide sequence.

baselookup

baselookup('Code', *CodeValue*) displays the corresponding letter code, meaning, and name. For ambiguous nucleotide letters (R Y K M S W B D H V N X), the name is replace by a descriptive name.

baselookup('Integer', *IntegerValue*) displays the corresponding letter code, meaning, and nucleotide name.

baselookup('Name', *NameValue*) displays the corresponding letter code and meaning.

Examples

    baselookup('Complement', 'TAGCTGRCCAAGGCCAAGCGAGCTTN')

    baselookup('Name','cytosine')

See Also   Bioinformatics Toolbox functions: basecount, codoncount, dimercount, geneticcode, nt2aa, nt2int, revgeneticcode, seqtool

biograph

Purpose   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 ' '. (This information is for bookkeeping purposes only.)

biograph

LabelValue	String to label the biograph object. Default is ''. (This information is for bookkeeping purposes only.)
DescriptionValue	String that describes the biograph object. Default is ''. (This information is for bookkeeping purposes only.)
LayoutTypeValue	String that specifies the algorithm for the layout engine. Choices are:
	• 'hierarchical' (default)
	• 'equilibrium'
	• 'radial'
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' (default) or 'off'.
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'`.
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 ''. (This information is for bookkeeping purposes only.)

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'Label', *LabelValue*, ...) specifies a label for the biograph object. Default is ''. (This information is for bookkeeping purposes only.)

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'Description', *DescriptionValue*, ...) specifies a description of the biograph object. Default is ''. (This information is for bookkeeping purposes only.)

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

1 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'
```

2 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'
```

3 Use the view method to display the biograph object.

biograph

```
view(bg2)
```
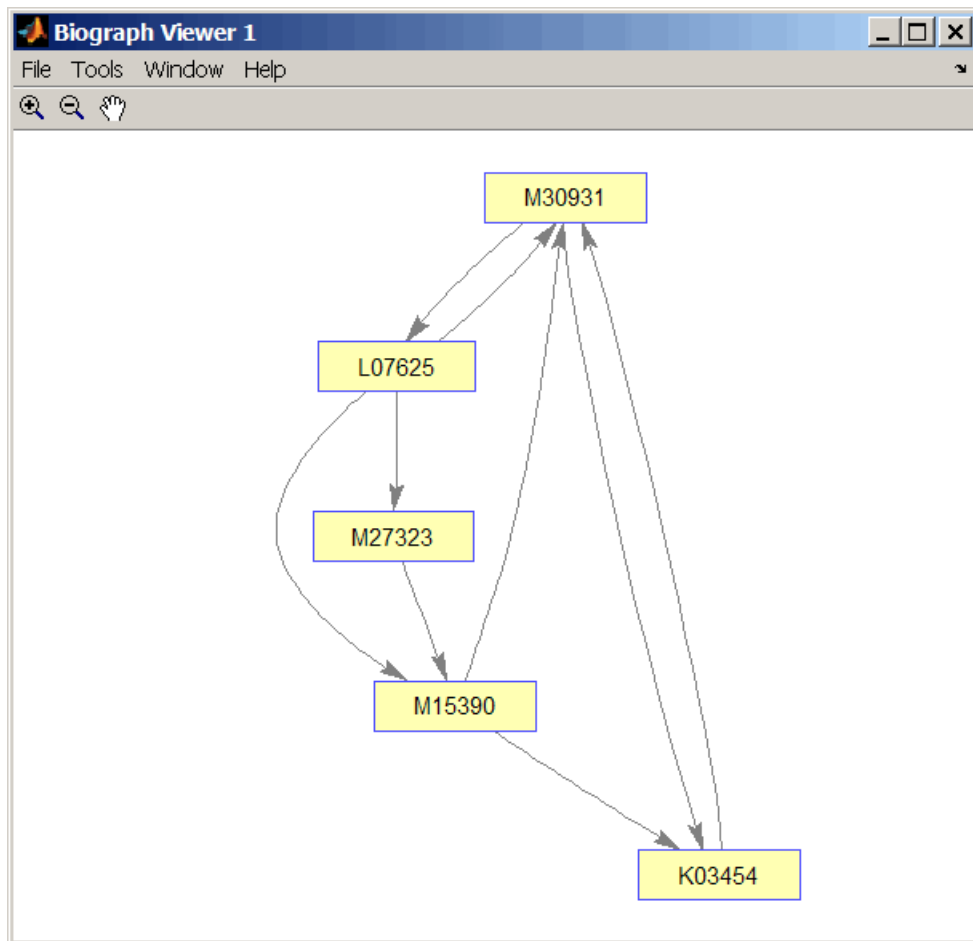

See Also   Bioinformatics Toolbox object: `biograph object`

Bioinformatics Toolbox methods of a biograph object:
`allshortestpaths, conncomp, dolayout, getancestors,
getdescendants, getedgesbynodeid, getmatrix, getnodesbyid,`

getrelatives, isdag, isomorphism, isspantree, maxflow, minspantree, shortestpath, topoorder, traverse, view

MATLAB functions: get, set

blastformat

Purpose   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 directory. (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 on page 2-84.

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

   http://www.ncbi.nlm.nih.gov/blast/download.shtml

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 at:

   ftp://ftp.ncbi.nih.gov/blast/documents/blast.html

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 directory. (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 directory, 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')
```

blastformat

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 directory, then type the following
at your system's command prompt.

```
formatdb -
```

Examples   **Using blastformat with Property Name/Value Pairs**

1 Download the `ecoli.nt.gz` zip file from

   ```
   ftp://ftp.ncbi.nih.gov/blast/db/FASTA/
   ```

   and then extract the `ecoli.nt` FASTA file to your MATLAB current
   directory.

2 Create a local blastable database from the `ecoli.nt` FASTA file and
   give it a title using the `'title'` property.

   ```
   blastformat('inputdb', 'ecoli.nt', 'protein', 'false',...
               'title', 'myecoli_nt');
   ```

Using blastformat with formatdb Syntax and Input Arguments

1 Download the `ecoli.aa.gz` zip file from

   ```
   ftp://ftp.ncbi.nih.gov/blast/db/FASTA/
   ```

   and then extract the `ecoli.aa` FASTA file to your MATLAB current
   directory.

2 Create a local blastable database from the `ecoli.aa` FASTA file
   and rename the title and log file using `formatdb` syntax and input
   arguments.

   ```
   blastformat('inputdb', 'ecoli.aa',...
   ```

```
'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://www.ncbi.nlm.nih.gov/blast/docs/formatdb.html
```

See Also   Bioinformatics Toolbox functions: blastlocal, blastncbi, blastread, blastreadlocal, getblast

blastlocal

Purpose   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, ...)
```

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 directory. (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 7 in "Examples" on page 2-96.

Return Values

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 or local databases.

blastlocal

`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.

Field	Description
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.
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.

Field	Description
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).
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://www.ncbi.nlm.nih.gov/BLAST/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 directory. (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://www.ncbi.nlm.nih.gov/BLAST/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')
```

blastlocal

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 directory, 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 directory, then type the following
at your system's command prompt.

```
blastall -
```

Examples   **1** Download the `ecoli.nt.gz` zip file from

```
ftp://ftp.ncbi.nih.gov/blast/db/FASTA/
```

and then extract the `ecoli.nt` and `ecoli.aa` FASTA files to your
MATLAB current directory.

2 Use the `blastformat` function to create local blastable databases
from the `ecoli.nt` and `ecoli.aa` FASTA files.

```
blastformat('inputdb', 'ecoli.nt', 'protein', 'false');
blastformat('inputdb', 'ecoli.aa');
```

3 Use the `getgenbank` function to retrieve sequence information for the
E. coli threonine operon from the GenBank database.

```
S = getgenbank('M28570.1');
```

4 Use 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);
```

5 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 `ecoli.aa`. Specify the BLAST program `blastx`. Return the
BLAST search results in `results`, a MATLAB structure.

```
results = blastlocal('inputquery', 'query_nt.fa',...
                     'database', 'ecoli.aa',...
                     'program',  'blastx')
```

6 Use `blastall` syntax to submit the query sequence in the
`query_nt.fa` FASTA file for a BLAST search of the local nucleotide
database `ecoli.nt`. Specify the BLAST program `blastn` and an
expectation value of `0.0001`. Display the BLAST search results in
the MATLAB Command Window.

```
blastlocal('-i query_nt.fa -d ecoli.nt -p blastn -e 0.0001')
```

7 Submit the query sequence in the `query_nt.fa` FASTA file for a BLAST search of the local nucleotide database `ecoli.nt`. 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', 'ecoli.nt', '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://www.ncbi.nlm.nih.gov/blast/docs/blastall.html`

See Also

Bioinformatics Toolbox functions: `blastformat`, `blastncbi`, `blastread`, `blastreadlocal`, `getblast`

Purpose   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*, ...'Inclusion', *InclusionValue*,
    ...)
... blastncbi(*Seq*, *Program*, ...'Pct', *PctValue*, ...)

blastncbi

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 iterated 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.

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'

Choices for amino acid sequences are:

- 'nr' (default)
- 'refseq_protein'
- 'swissprot'
- 'pat'
- 'month'
- 'pdb'
- 'env_nr'

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 sequence 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 on page 2-109.
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	Either of the following:

- Integer that specifies the penalty for opening a gap in the alignment of amino acid sequences.
- 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 GapOpen Property by Matrix on page 2-110.

GapExtendValue	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 GapOpen Property by Matrix on page 2-110.

InclusionValue	Value specifying the statistical significance threshold for including a sequence in the Position-Specific Score 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'.

blastncbi

Return Values

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://www.ncbi.nlm.nih.gov/BLAST/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/staff/tao/URLAPI/blastcgihelp_new.html

... blastncbi(*Seq*, *Program*, ...'Database', *DatabaseValue*, ...) specifies a database for the alignment search. For help in selecting an appropriate database, visit:

    http://www.ncbi.nlm.nih.gov/BLAST/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://www.ncbi.nlm.nih.gov/BLAST/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 GapOpen Property by Matrix on page 2-110.

For more information about allowed gap penalties for various matrices, see:

```
http://www.ncbi.nlm.nih.gov/blast/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 GapOpen Property by Matrix on page 2-110.

... blastncbi(*Seq*, *Program*, ...'Inclusion', *InclusionValue*, ...) specifies the statistical significance threshold for including a sequence in the Position-Specific Score 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'.

Choices for Optional Properties by BLAST Program

When BLAST program is...	Then choices for the following properties are...					
	Database	Filter	Word	Matrix	GapOpen	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'	'L'	11 12 16 20 24 28 (default) 32 48 64			None 99 (default) 98 95 90 85 80 75 60
'tblastn'	'refseq_genomic' 'env_nt'	'L' (default) 'm' 'lcase'	2 3 (default)	'PAM30' 'PAM70' 'BLOSUM45' 'BLOSUM62' (default) 'BLOSUM80'	See the next table.	—
'tblastx'		'L' (default) 'R' 'm' 'lcase'				
'blastp' 'blastx' 'psiblast'	'nr' (default) 'swissprot' 'pat' 'pdb' 'month' 'refseq_protein' 'env_nr'	'L' (default) 'm' 'lcase'				

Choices for the GapOpen Property by Matrix

When Substitution Matrix is...	Then choices for GapOpen 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 try a search directly with an accession
% number and an alternative scoring matrix.
RID = blastncbi('AAA59174','blastp','matrix','PAM70,'...
                               'expect',1e-10)

% The results based on the RID are at
http://www.ncbi.nlm.nih.gov/BLAST/Blast.cgi

% or 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

Bioinformatics Toolbox functions: `blastformat`, `blastlocal`, `blastread`, `blastreadlocal`, `getblast`

blastread

Purpose   Read data from NCBI BLAST report file

Syntax   *Data* = blastread(*BLASTReport*)

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 current directory.

Return Values

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.

Field	Description
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.

Field	Description
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://www.ncbi.nlm.nih.gov/Education/BLASTinfo/Blast_output.html
```

See Also Bioinformatics Toolbox functions: `blastformat`, `blastlocal`, `blastncbi`, `blastreadlocal`, `getblast`

blastreadlocal

Purpose	Read data from local BLAST report

Syntax

Data = blastreadlocal(*BLASTReport*, *Format*)

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 directory.

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

Return Values	*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*.

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.

Field	Description
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.

Field	Description
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.

blastreadlocal

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.
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.

Examples

1 Download the ecoli.nt.gz zip file from

    ftp://ftp.ncbi.nih.gov/blast/db/FASTA/

and then extract the ecoli.nt FASTA file to your MATLAB current directory.

2 Create a local blastable database from the ecoli.nt FASTA file.

    blastformat('inputdb', 'ecoli.nt', 'protein', 'false');

3 Use the getgenbank function to retrieve two sequences from the GenBank database.

    S1 = getgenbank('M28570.1');

```
S2 = getgenbank('M12565');
```

4 Use 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);
```

5 Submit the query sequences in the `query_multi_nt.fa` FASTA
file for a BLAST search of the local nucleotide database `ecoli.nt`.
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, displaying
the results in the MATLAB Command Window.

```
blastlocal('inputquery', 'query_multi_nt.fa',...
           'database', 'ecoli.nt',...
           'tofile', 'myecoli_nt8.txt', 'program', 'blastn',...
           'format', 8);
blastreadlocal('myecoli_nt8.txt', 8);
```

6 Submit the query sequences in the `query_multi_nt.fa` FASTA
file for a BLAST search of the local nucleotide database `ecoli.nt`.
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', 'ecoli.nt',...
           'tofile', 'myecoli_nt1.txt', 'program', 'blastn',...
           'format', 1);
results = blastreadlocal('myecoli_nt1.txt', 1);
```

blastreadlocal

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://www.ncbi.nlm.nih.gov/Education/BLASTinfo/information3.html
```

See Also   Bioinformatics Toolbox functions: `blastformat`, `blastlocal`, `blastncbi`, `blastread`, `getblast`

Purpose   Return BLOSUM scoring matrix

Syntax   *Matrix* = blosum(*Identity*)
                [*Matrix*, *MatrixInfo*] = blosum(*Identity*)
                ... = blosum(*Identity*, ...'Extended', *ExtendedValue*, ...)
                ... = blosum(*Identity*, ...'Order', *OrderValue*, ...)

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.

blosum

Return Values	*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   Bioinformatics Toolbox functions: dayhoff, gonnet, nwalign, pam, swalign

celintensityread

Purpose
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, ...)

Arguments

CELFiles	Any of the following:
	• String specifying a single CEL file name.
	• '*', which reads all CEL files in the current directory.
	• ' ', 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 directory where the files specified in *CELFiles* are stored.

CDFPathValue	String specifying the path and directory 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.

Return Values

| *ProbeStructure* | MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs. |

Description

Note This function does not work on the Solaris platform.

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 directory. 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 directory where the files specified in *CELFiles* are stored.

ProbeStructure = celintensityread(..., 'CDFPath', *CDFPathValue*, ...) specifies a path and directory where the file specified in *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/products/software/specific/dtt.affx
> ```
>
> You will have to register and log in at the Affymetrix Web site to download the Affymetrix Data Transfer Tool.

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.
NumProbeSets	Number of probe sets in each CEL file.
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.

celintensityread

Field	Description
PMIntensities	Matrix containing 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	Matrix containing 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 directory 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 directory and a CDF file in a specified directory. 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   Bioinformatics Toolbox functions: `affyinvarsetnorm`, `affyprobeseqread`, `affyread`, `agferead`, `gcrma`, `gcrmabackadj`, `gprread`, `probelibraryinfo`, `probesetlink`, `probesetlookup`, `probesetplot`, `probesetvalues`, `rmabackadj`, `rmasummary`, `sptread`

cghcbs

Purpose   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*,... '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*, ...)

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 — Vector containing $\log_2$ ratio of test to reference signal intensity for each clone.

- 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.
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`.

cghcbs

Return Values

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 $\sim \mathrm{O}(n^2)$.

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*,... `'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).

cghcbs

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 Bioinformatics Toolbox, 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);
```
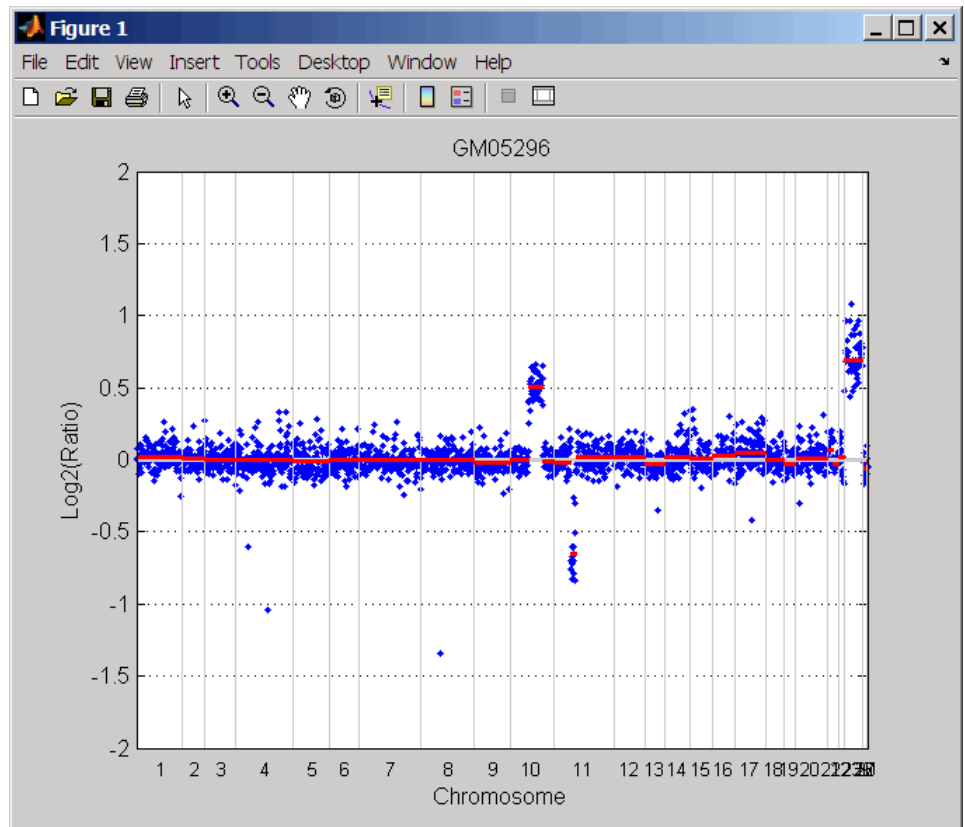
Chromosome 10 shows a gain, while chromosome 11 shows a loss.

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 Bioinformatics Toolbox, 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);
```
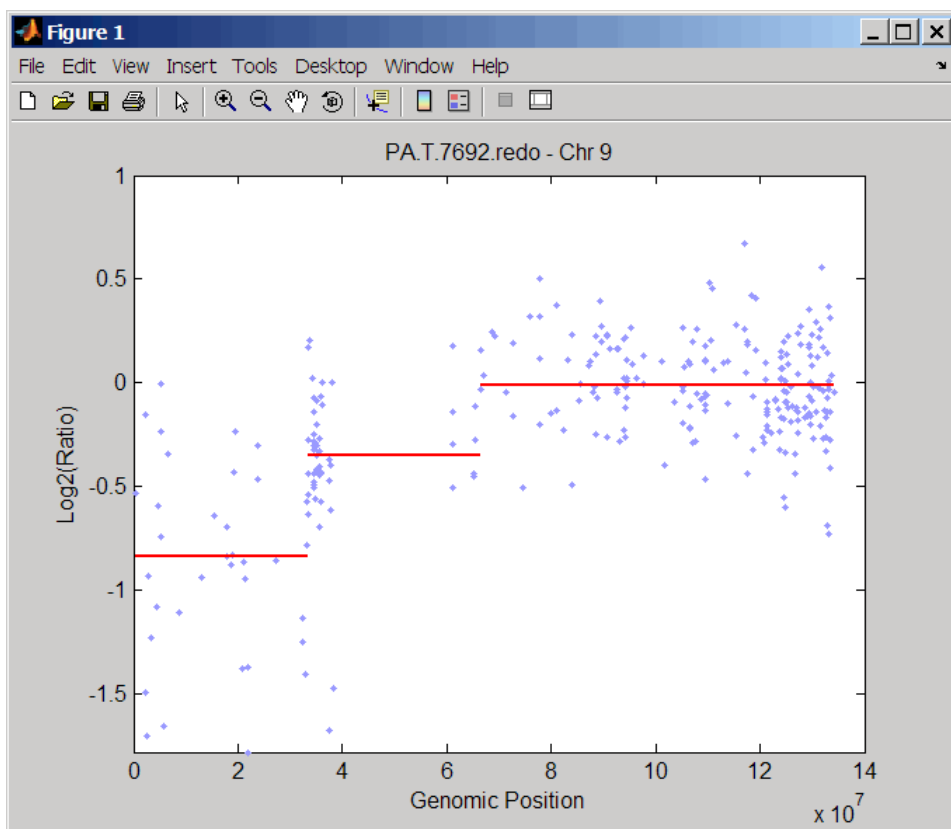
Chromosome 9 contains two segments that indicate losses. For more detailed information on interpreting the data, see Aguirre et al. (2004).

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)
```
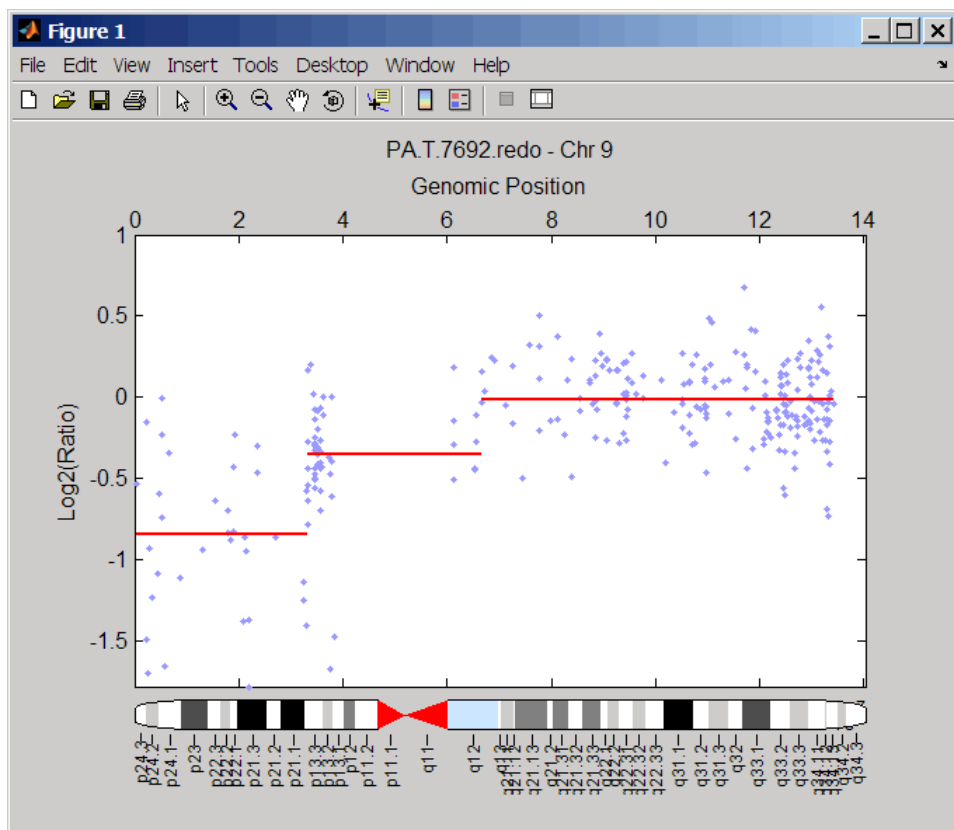
The pancrea_oligocgh.mat file used in this example contains data from Aguirre et al., 2004.

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 on page 2-141.) 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)
```
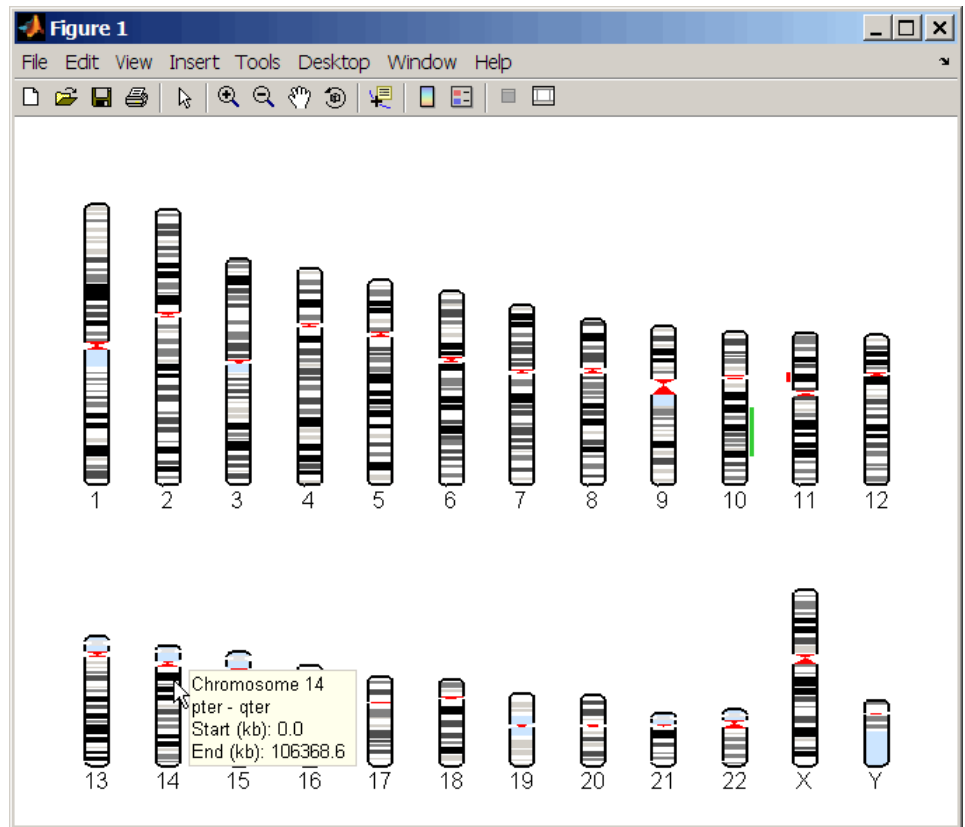
The coriell_baccgh.mat file used in this example contains data from Snijders et al., 2001.

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/repository/devel/vignette/DNAcopy.pdf`

[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   Bioinformatics Toolbox functions: `chromosomeplot`, `cytobandread`

chromosomeplot

Purpose   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 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.

chromosomeplot

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 following fields:

- Chromosome — Chromosome number

- CNVType — 1 = gain, 2 = loss

- Start — Starting genomic position

- End — Ending genomic position

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 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.

Field	Description
Chromosome	Vector containing the chromosome number on which each CNV is located.
CNVType	Vector containing the type of each CNV, either 1 (loss) or 2 (gain).
Start	Vector containing the starting genomic position of each CNV. Units must be in base pairs.
End	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')
```
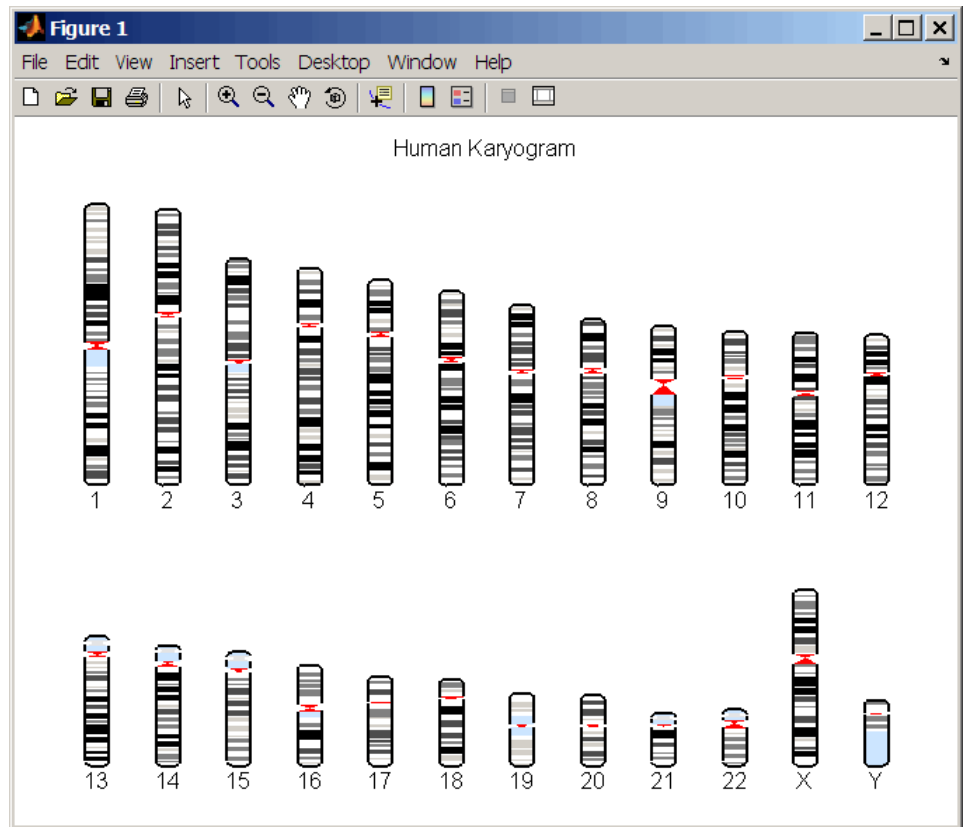
3 Display the ideogram of only chromosome 7 for *Homo sapiens* by right-clicking chromosome 7 in the plot, then selecting **Display in New Figure > Vertical**.

chromosomeplot
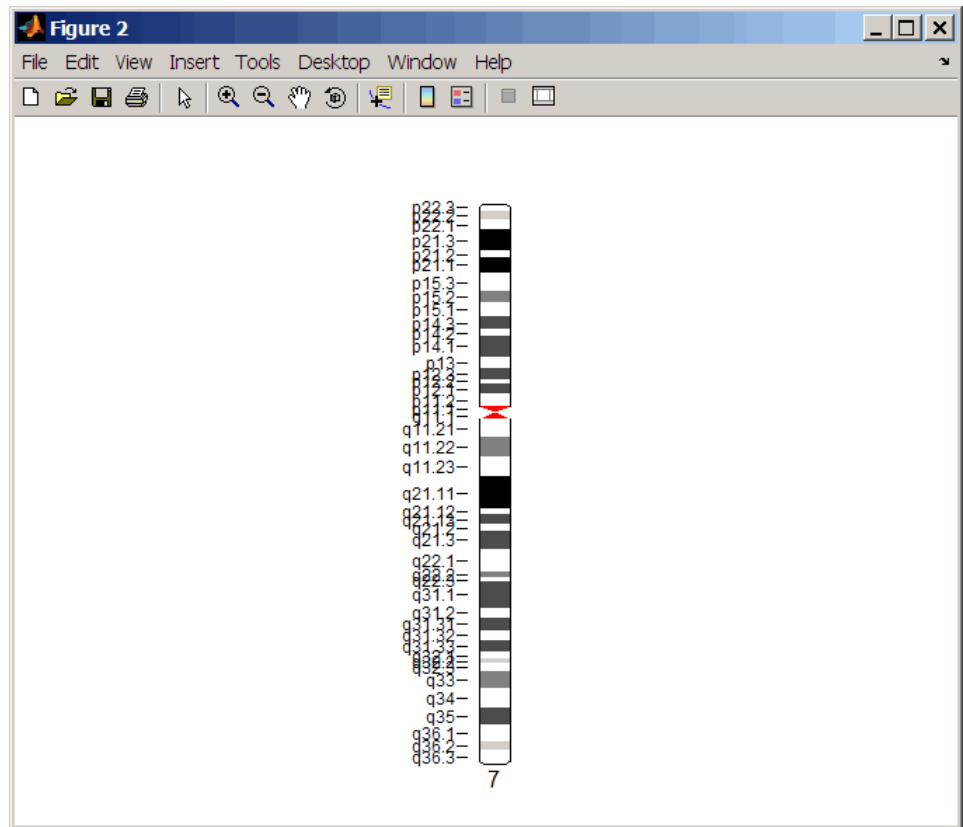
4 Plot the ideogram of only chromosome 15 for *Homo sapiens* in a
horizontal orientation. Set the units used in the data tip to kilo
base pairs.

```
chromosomeplot(hs_cytobands, 15, 'Orientation', 2, 'Unit', 2);
```
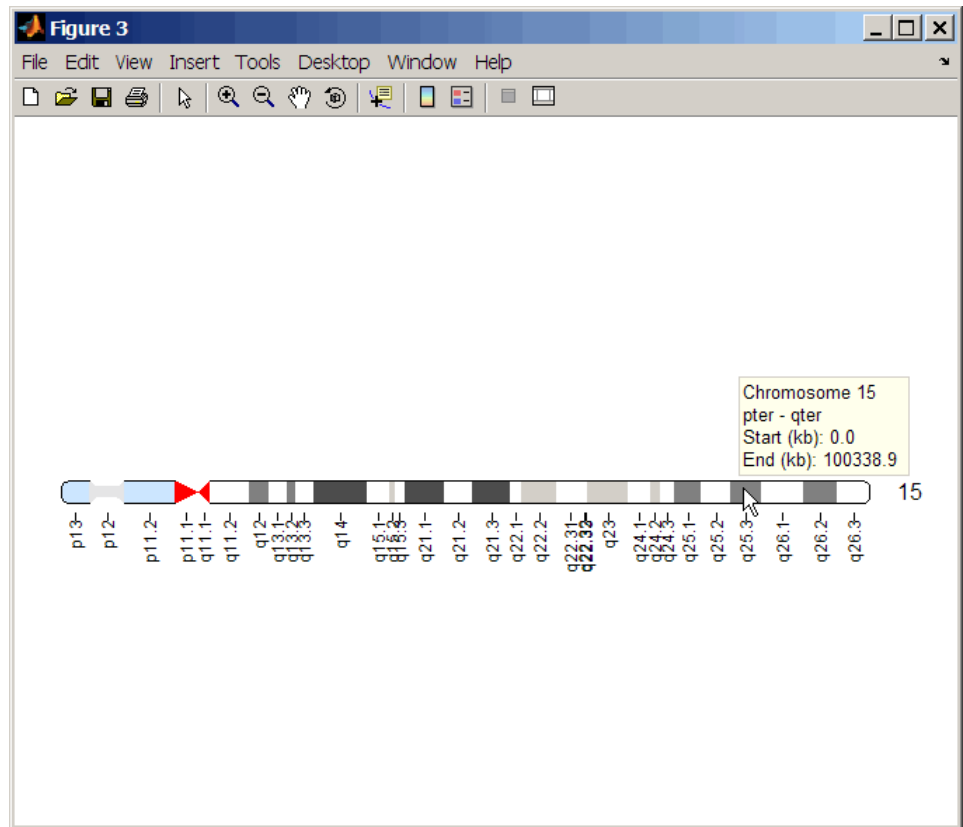
5 View a data tip with information about the chromosome by hovering
the cursor over the chromosome. View a data tip with detailed

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**.

chromosomeplot

> **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 Bioinformatics Toolbox, 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);
```
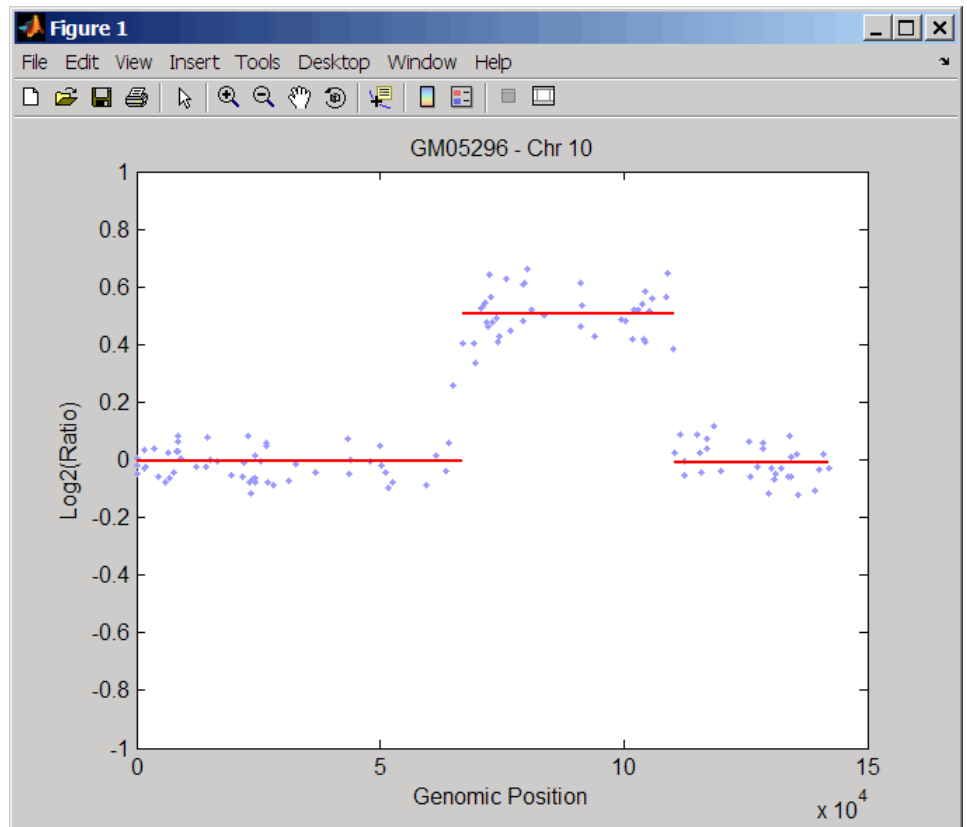
3 Use the chromosomeplot function with the 'addtoplot' property to
add the ideogram of chromosome 10 for *Homo sapiens* to the plot.
Because the plot of the CNV data from the Coriell cell line study is
in kb units, use the 'Unit' property to convert the ideogram data
to kb units.

```
chromosomeplot('hs_cytoBand.txt', 10, 'addtoplot', gca,...
               'Unit', 2)
```
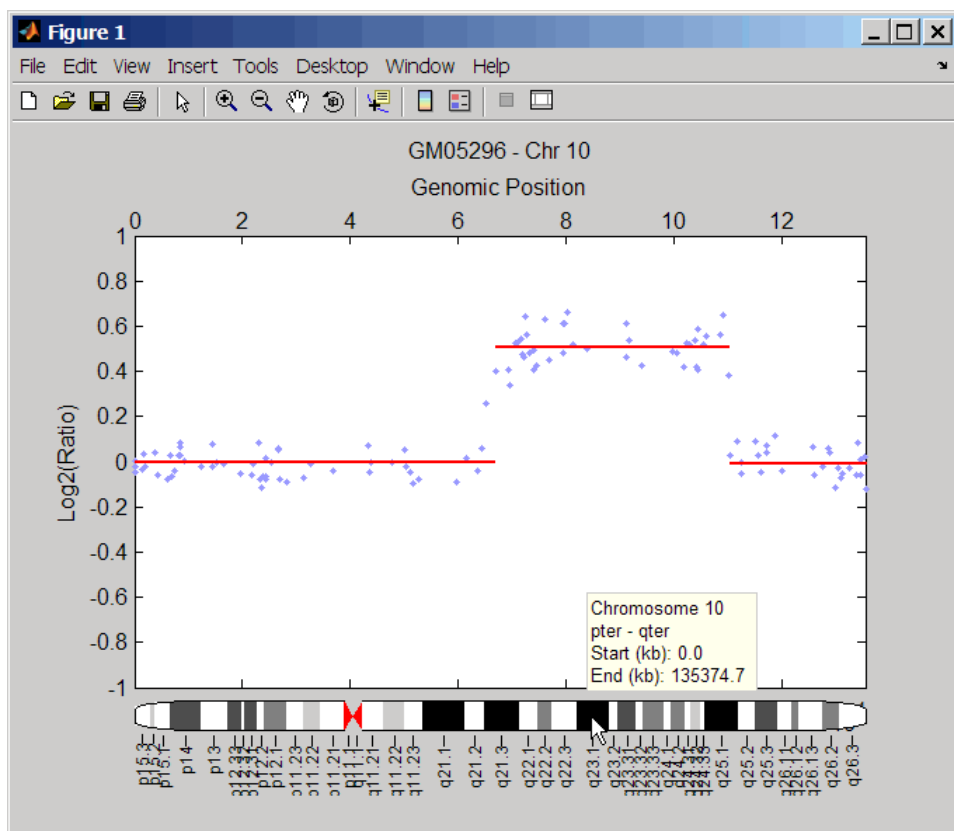
Tip Before printing the above figure containing an added chromosome ideogram, change the background to white by issuing the following command:

```
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 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 step 2 in Adding a Chromosome Ideogram to a Plot on page 2-158.) 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', [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.

```
chromosomeplot('hs_cytoBand.txt', 'cnv', cnvStruct);
```

chromosomeplot
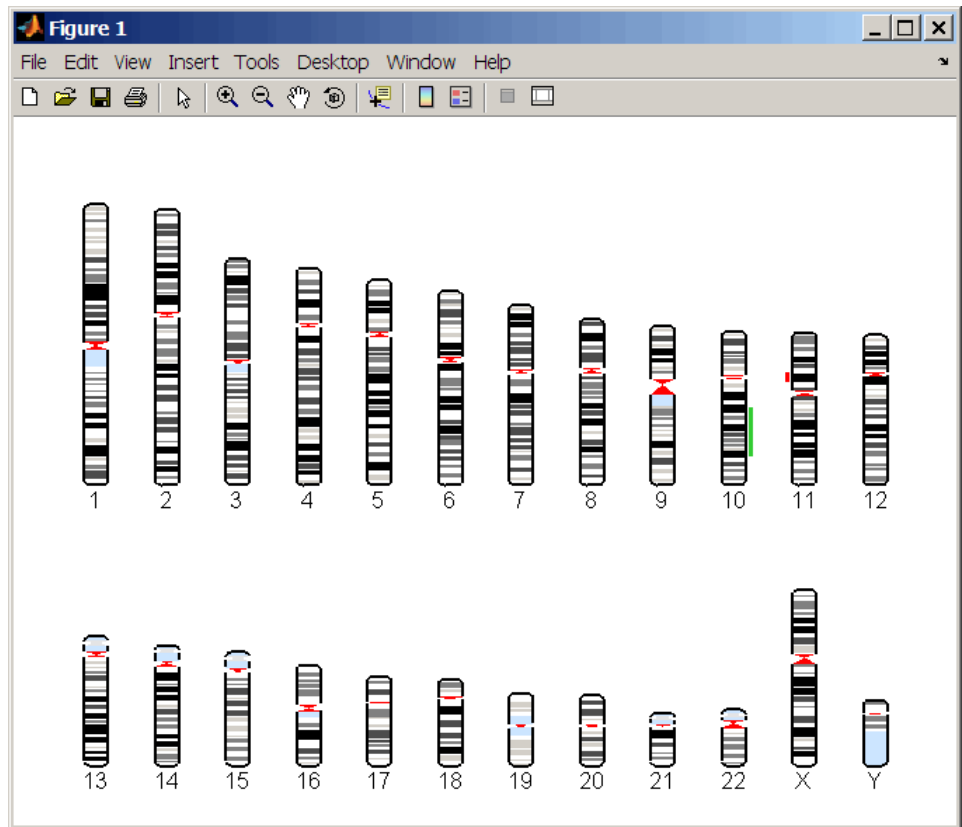
The coriell_baccgh.mat file used in this example contains data from Snijders et al., 2001.

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   Bioinformatics Toolbox functions: `cghcbs`, `cytobandread`

classperf

Purpose   Evaluate performance of classifier

Syntax
```
classperf
cp = classperf(groundtruth)
classperf(cp, classout)
classperf(cp, classout, testidx)
cp = classperf(groundtruth, classout,...)
cp = classperf(..., 'Positive', PositiveValue, 'Negative',
    NegativeValue)
```

Description   classperf provides an interface to keep track of the performance
during the validation of classifiers. classperf creates and updates
a classifier performance object (*CP*) that accumulates the results of
the classifier. Later, classification standard performance parameters
can be accessed using the function get or as fields in structures.
Some of these performance parameters are ErrorRate, CorrectRate,
ErrorDistributionByClass, Sensitivity and Specificity. classperf,
without input arguments, displays all the available performance
parameters.

cp = classperf(*groundtruth*) creates and initializes an empty object.
CP is the handle to the object. *groundtruth* is a vector containing the
true class labels for every observation. *groundtruth* can be a numeric
vector or a cell array of strings. When used in a cross-validation design
experiment, *groundtruth* should have the same size as the total
number of observations.

classperf(*cp*, *classout*) updates the *CP* object with the classifier
output *classout*. *classout* is the same size and type as *groundtruth*.
When *classout* is numeric and groundtruth is a cell array of strings,
the function grp2idx is used to create the index vector that links
classout to the class labels. 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 the *CP* object with
the classifier output *classout*. *classout* has smaller size than
groundtruth, and *testidx* is an index vector or a logical index vector of

the same size as `groundtruth`, which indicates the observations that were used in the current validation.

cp = classperf(*groundtruth*, *classout*,...) creates and updates the *CP* object with the first validation. This form is useful when you want to know the performance of a single validation.

cp = classperf(..., 'Positive', *PositiveValue,* 'Negative', *NegativeValue*) sets the 'positive' and 'negative' labels to identify the target disorder and the control classes. These labels are used to compute clinical diagnostic test performance. p and n must consist of disjoint sets of the labels used in *groundtruth*. For example, if

```
groundtruth = [1 2 2 1 3 4 4 1 3 3 3 2]
```

you could set

```
p = [1 2];
n = [3 4];
```

If *groundtruth* is a cell array of strings, p and n can either be cell arrays of strings or numeric vectors whose entries are subsets of grp2idx(*groundtruth*). *PositiveValue* defaults to the first class returned by grp2idx(*groundtruth*), while *NegativeValue* defaults to all the others. In clinical tests, inconclusive values ('' 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 true class 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 groundtruth are counted as inconclusive.

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}
                  GroundTruth: [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   Bioinformatics Toolbox functions: `crossvalind`, `knnclassify`, `svmclassify`,

Statistics Toolbox functions: `classify`, `grp2idx`

cleave

Purpose	Cleave amino acid sequence with enzyme

Syntax

Fragments = cleave(*SeqAA*, *PeptidePattern*, *Position*)
[*Fragments, CuttingSites*] = cleave(...)
[*Fragments, CuttingSites, Lengths*] = cleave(...)
cleave(..., '*PropertyName*', *PropertyValue*,...)
cleave(..., 'PartialDigest', *PartialDigestValue*)

Arguments

SeqAA	Amino acid sequence. Enter a character string or a vector of integers from the table Mapping Amino Acid Letters to Integers on page 2-2.
	Examples: 'ARN' or [1 2 3]. You can also enter a structure with the field Sequence.
PeptidePattern	Short amino acid sequence to search in a larger sequence. Enter a character string, vector of integers, or a regular expression.
Position	Position on the PeptidePattern where the sequence is cleaved. Enter a position within the PeptidePattern. Position 0 corresponds to the N terminal end of the PepetidePattern.
PartialDigestValue	Property to specify the probability that a cleavage site will be cleaved. Enter a value from 0 to 1 (default).

Description

Fragments = cleave(*SeqAA*, *PeptidePattern*, *Position*) cuts an amino acid sequence (*SeqAA*) into parts at the specified cleavage site specified by a peptide pattern and position.

[*Fragments, CuttingSites*] = cleave(...) returns a numeric vector with the indices representing the cleave sites. A 0 (zero) is added to the list, so numel(*Fragments*)==numel(*CuttingSites*). You can 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 with the lengths of every fragment.

cleave(..., *'PropertyName'*, *PropertyValue*,...) defines optional properties using property name/value pairs.

cleave(..., 'PartialDigest', *PartialDigestValue*) simulates a partial digestion where PartialDigest is the probability of a cleavage site being cut.

The following table lists some common proteases and their cleavage sites.

Protease	Peptide Pattern	Position
Trypsin	[KR](?!P)	1
Chymotrypsin	[WYF](?!P)	1
Glutamine C	[ED](?!P)	1
Lysine C	[K](?!P)	1
Aspartic acid N	D	1

Examples

1 Get a protein sequence from the GenPept database.

```
S = getgenpept('AAA59174')
```

2 Cleave the sequence using trypsin. Trypsin cleaves after K or R when the next residue is not P.

```
[parts, sites, lengths] = cleave(S.Sequence,'[KR](?!P)',1);
    for i=1:10
        fprintf('%5d%5d   %s\n',sites(i),lengths(i),parts{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
```

See Also Bioinformatics Toolbox functions: `rebasecuts`, `restrict`, `seqshowwords`

MATLAB function: `regexp`

Purpose   Create dendrogram and heat map

Syntax   clustergram(*Data*)
clustergram(*Data*, ...'RowLabels', *RowLabelsValue*, ...)
clustergram(*Data*, ...'ColumnLabels',
ColumnLabelsValue, ...)
clustergram(*Data*, ...'Pdist', *PdistValue*, ...)
clustergram(*Data*, ...'Linkage', *LinkageValue*, ...)
clustergram(*Data*, ...'Dendrogram', *DendrogramValue*, ...)
clustergram(*Data*, ...'OptimalLeafOrder',
   OptimalLeafOrderValue, ...)
clustergram(*Data*, ...'ColorMap', *ColorMapValue*, ...)
clustergram(*Data*, ...'SymmetricRange', *SymmetricRangeValue*,
   ...)
clustergram(*Data*, ...'Dimension', *DimensionValue*, ...)
clustergram(*Data*, ...'Ratio', *RatioValue*, ...)

Arguments

Data	Matrix in which each row corresponds to a gene and each column corresponds to a single experiment or microarray.
RowLabelsValue	Vector of numbers or cell array of text strings to label the rows in Data.
ColumnLabelsValue	Vector of numbers or cell array of text strings to label the columns in Data.

PdistValue	String to specify the distance metric to pass to the pdist function (Statistics Toolbox) to use to calculate the pairwise distances between observations. For information on choices, see the pdist function. Default is euclidean.

> **Note** If the distance metric requires extra arguments, then *PdistValue* is a cell array. For example, to use the Minkowski distance with exponent P, you would use {'minkowski', P}.

LinkageValue	String to specify the linkage method to pass to the linkage function (Statistics Toolbox) to use to create the hierarchical cluster tree. For information on choices, see the linkage function. Default is average.
DendrogramValue	Cell array of property name/property value pairs to pass to the dendrogram function (Statistics Toolbox) to create the dendrogram plot. For information on choices, see the dendrogram function.

OptimalLeafOrderValue	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). Default depends on the size of *Data*. If the number of rows or columns in *Data* is greater than 1000, 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 uses a large amount of memory and can be very time consuming.

ColorMapValue	Either of the following:

- M-by-3 matrix of RGB values
- Name or function handle of a function that returns a color map

Default is redgreencmap.

SymmetricRangeValue	Property to force the color range of the heat map to be symmetric around zero. Choices are true (default) or false.

clustergram

DimensionValue	Property to specify either a one-dimensional or two-dimensional clustergram. Choices are 1 (default) or 2.
RatioValue	Either of the following:

- Scalar

- Two-element vector

Default is 1/5.

Description clustergram(*Data*) creates a dendrogram and heat map from the gene expression data in the matrix *Data*. It uses hierarchical clustering with euclidean distance metric and average linkage to generate the hierarchical tree. The clustering is performed on the rows in matrix *Data*, in which the rows correspond to genes and the columns correspond to different microarrays. To cluster the columns instead of the rows, transpose the data using the transpose (') operator.

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. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

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 *Data*.

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 *Data*.

clustergram(*Data*, ...'Pdist', *PdistValue*, ...) specifies the distance metric to pass to the pdist function (Statistics Toolbox) to use to calculate the pairwise distances between observations. *PdistValue*

is a string. For information on choices, see the `pdist` function. Default is `euclidean`.

Note If the distance metric requires extra arguments, then *PdistValue* is a cell array. For example, to use the Minkowski distance with exponent P, you would use `{'minkowski', P}`.

---

clustergram(*Data*, ...`'Linkage'`, *LinkageValue*, ...) specifies the linkage method to pass to the `linkage` function (Statistics Toolbox) to use to create the hierarchical cluster tree. *LinkageValue* is a string. For information on choices, see the `linkage` function. Default is `average`.

clustergram(*Data*, ...`'Dendrogram'`, *DendrogramValue*, ...) specifies property name/property value pairs to pass to the `dendrogram` function (Statistics Toolbox) to create the dendrogram plot. *DendrogramValue* is a cell array of property name/property value pairs. For information on choices, see the `dendrogram` function.

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* is greater than 1000, 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 uses a large amount of memory and can be very time consuming.

---

clustergram(*Data*, ...`'ColorMap'`, *ColorMapValue*, ...) specifies the color map to use to create the clustergram. This controls the colors used to display the heat map. *ColorMapValue* is either an M-by-3

matrix of RGB values or the name or function handle of a function that returns a color map. Default is redgreencmap.

clustergram(*Data*, ...'SymmetricRange', *SymmetricRangeValue*, ...), controls whether the color range of the heat map is symmetric around zero. *SymmetricRangeValue* can be true (default) or false.

clustergram(*Data*, ...'Dimension', *DimensionValue*, ...) specifies whether to create a one-dimensional or two-dimensional clustergram. Choices are 1 (default) or 2. The one-dimensional clustergram clusters the rows of the data. The two-dimensional clustergram creates the one-dimensional clustergram, and then clusters the columns of the row-clustered data.

clustergram(*Data*, ...'Ratio', *RatioValue*, ...) specifies the ratio of the space that the dendrogram(s) use in the *X* and *Y* directions, relative to the size of the heat map. If *RatioValue* is a scalar, it is used as the ratio for both directions. If *RatioValue* is a two-element vector, the first element is used for the *X* ratio, and the second element is used for the *Y* ratio. The *Y* ratio is ignored for one-dimensional clustergrams. Default ratio is 1/5.

Tip Click and hold the mouse button on the heat map to display the intensity value, column label, and row label for that area of the heat map. View row labels by using the zoom icon to zoom the right side of 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 during the metabolic shift from fermentation to respiration. Expression levels were measured at seven time points during the diauxic shift.

1 Load the filtered yeast data provided with Bioinformatics Toolbox, and then create a clustergram from the gene expression data in the yeastvalues matrix.

```
load filteredyeastdata
clustergram(yeastvalues)
```
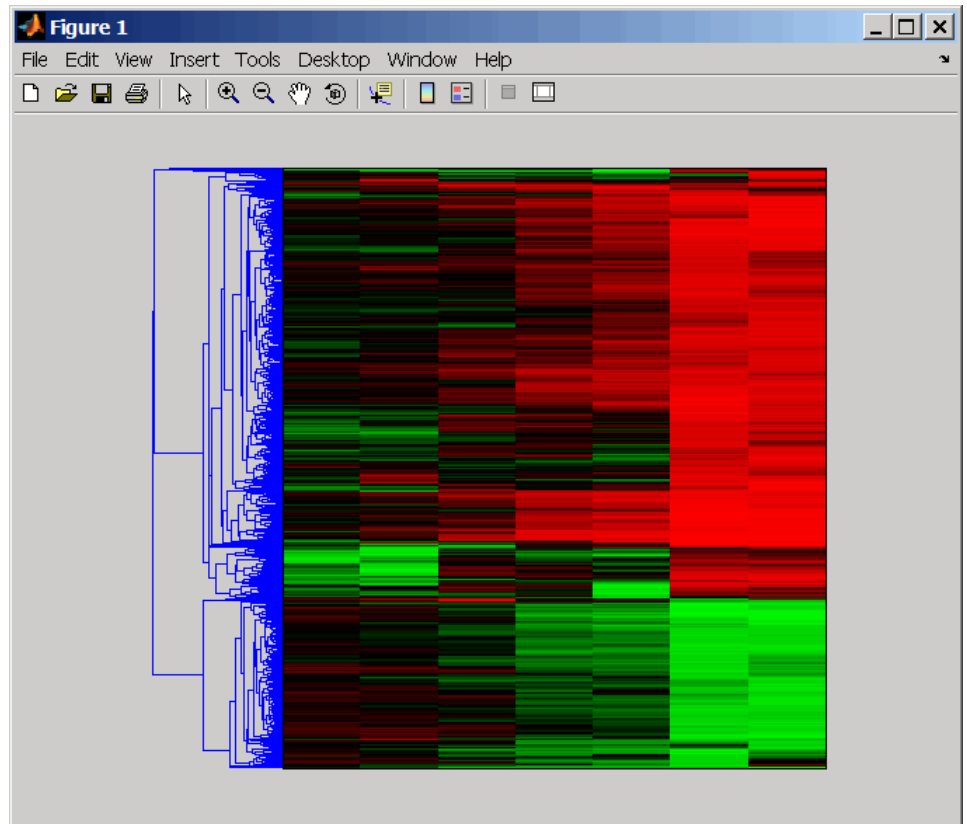

2 Add labels to the clustergram, then click and hold the mouse button on the heat map to display the intensity value, column label, and row label for that area of the heat map. View the row labels by using the **Zoom** icon to zoom the right side of the clustergram.

```
clustergram(yeastvalues,'RowLabels',genes,'ColumnLabels',times)
```
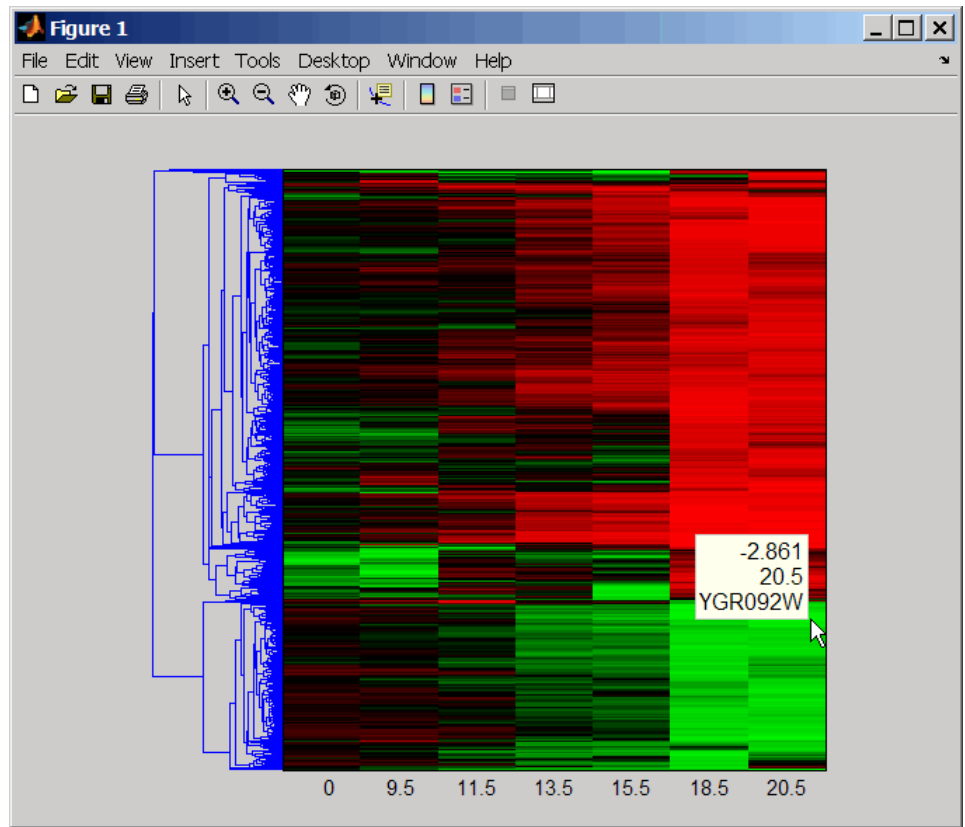
3 Change the clustering parameters.

```
clustergram(yeastvalues,'Linkage','complete')
```
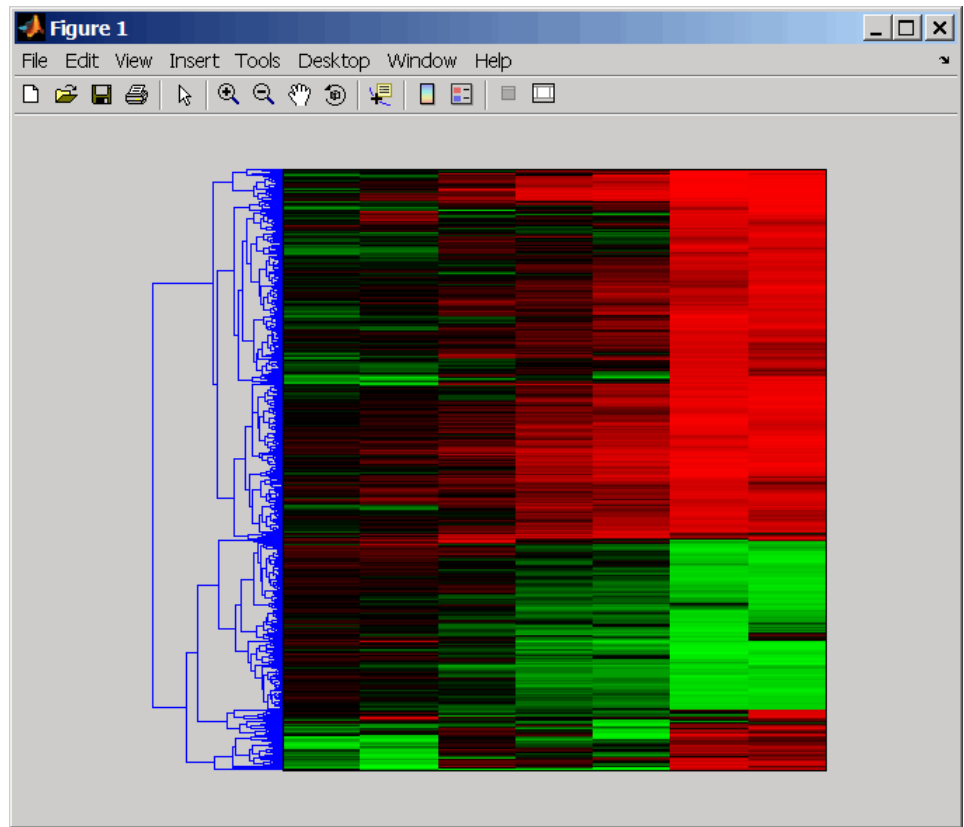
4 Change the color of the groups of nodes in the dendrogram whose linkage is less than a threshold of 5.

```
clustergram(yeastvalues,'RowLabels',genes,...
                        'Dendrogram',{'colorthreshold',5})
```

clustergram
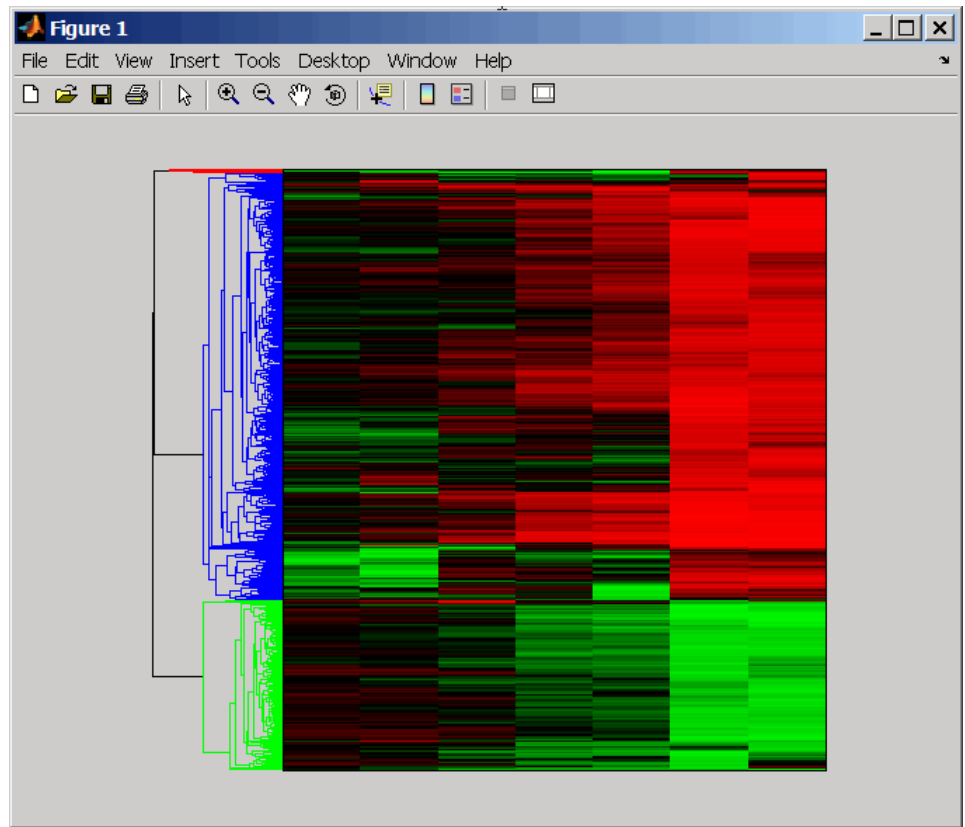
References

[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.

See Also   Bioinformatics Toolbox function: `redgreencmap`

Statistics Toolbox functions: `cluster`, `dendrogram`, `linkage`, `pdist`

codonbias

Purpose   Calculate codon frequency for each amino acid in DNA sequence

Syntax

```
codonbias(SeqDNA)
codonbias(..., 'PropertyName', PropertyValue,...)
codonbias(..., 'GeneticCode', GeneticCodeValue)
codonbias(..., 'Frame', FrameValue)
codonbias(..., 'Reverse', ReverseValue)
codonbias(..., 'Pie', PieValue)
```

Arguments

SeqDNA	Nucleotide sequence (DNA or RNA). Enter a character string with the letters A, T or U, C, and G or a vector of integers. You can also enter a structure with the field Sequence. codonbias does not count ambiguous bases or gaps.

Description   Many amino acids are coded by two or more nucleic acid codons. However, the probability that a codon (from the various possible codons for an amino acid) is used to code an amino acid is different between sequences. Knowing the frequency of each codon in a protein coding sequence for each amino acid is a useful statistic.

codonbias(*SeqDNA*) calculates the codon frequency in percent for each amino acid in a DNA sequence (*SeqDNA*).

codonbias(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

codonbias(..., 'GeneticCode', *GeneticCodeValue*) selects an alternative genetic code (*GenetidCodeValue*). The default value is 'Standard' or 1. For a list of genetic codes, see the table Genetic Code on page 2-6.

codonbias(..., 'Frame', *FrameValue*) selects a reading frame (*FrameValue*). *FrameValue* can be 1 (default), 2, or 3.

codonbias(..., 'Reverse', *ReverseValue*), when *ReverseValue* is true, returns the codon frequency for the reverse complement of the DNA sequence (*SeqDNA*).

codonbias(..., 'Pie', *PieValue*), when *PieValue* is true, creates a figure of 20 pie charts for each amino acid.

Examples

1 Import a nucleotide sequence from GenBank to MATLAB. For example, get the DNA sequence that codes for a human insulin receptor.

```
S = getgenbank('M10051');
```

2 Calculate the codon frequency for each amino acid and plot the results.

```
cb = codonbias(S.Sequence,'PIE',true)

cb.Ala
ans =
    Codon: {'GCA' "GCC" "GCG" 'GCT'}
     Freq: [0.1600 0.3867 0.2533 02000]
```

MATLAB draws a figure with 20 pie charts for the 20 amino acids.

codonbias
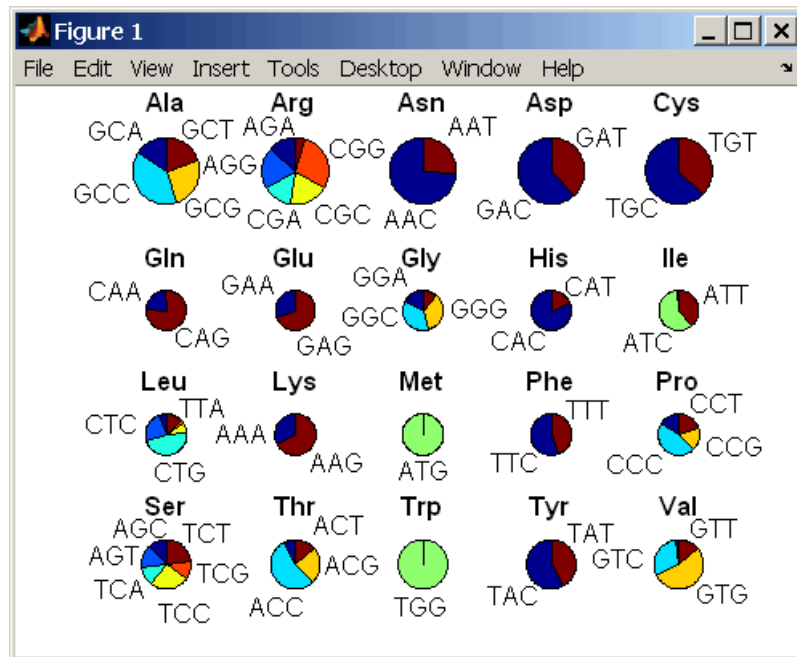
See Also   Bioinformatics Toolbox functions `aminolookup`, `codoncount`, `geneticcode`, `nt2aa`

Purpose   Count codons in nucleotide sequence

Syntax
Codons = codoncount(*SeqNT*)
[*Codons, CodonArray*] = codoncount(*SeqNT*)
codoncount(..., *'PropertyName'*, *PropertyValue*,...)
codoncount(..., 'Frame', *FrameValue*)
codoncount(..., 'Reverse', *ReverseValue*)
codoncount(..., 'Figure', *FigureValue*)

Arguments

SeqNT	Nucleotide sequence. Enter a character string or vector of integers. You can also enter a structure with the field Sequence.
FrameValue	Property to select a reading frame. Enter 1 (default), 2, or 3.
ReverseValue	Property to control returning the complement sequence. Enter true or false (default).
FigureValue	Property to control plotting a heat map. Enter either true or false (default).

Description   *Codons* = codoncount(*SeqNT*) counts the number of codons in a
sequence (*SeqNT*) and returns the codon counts in a structure with the
fields AAA, AAC, AAG, ..., TTG, TTT.

- For sequences that have codons with the character U, the U characters
  are added to codons with T characters.

- If the sequence contains ambiguous nucleotide characters (R Y K M
  S W B D H V N), or gaps indicated with a hyphen (-), this function
  creates a field Others and displays a warning message:

  ```
  Warning: Ambiguous symbols 'symbol' appear
  in the sequence.
  These will be in Others.
  ```

codoncount

- If the sequence contains undefined nucleotide characters (E F H I
  J L O P Q X Z), codoncount ignores the characters and displays a
  warning message:

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

[*Codons, CodonArray*] = codoncount(*SeqNT*) returns a 4-by-4-by-4
array (*CodonArray*) with the raw count data for each codon. The three
dimensions correspond to the three positions in the codon. For example,
the element (2,3,4) of the array gives the number of CGT codons, where
A <=> 1, C <=> 2, G <=> 3, and T <=> 4.

codoncount(..., '*PropertyName*', *PropertyValue*,...) defines
optional properties using property name/value pairs.

codoncount(..., 'Frame', *FrameValue*) counts the codons in a
specific reading frame.

codoncount(..., 'Reverse', *ReverseValue*), when *ReverseValue* is
true, counts the codons for the reverse complement of the sequence.

codoncount(..., 'Figure', *FigureValue*), when *FigureValue* is
true, displays a figure showing a heat map of the codon counts.

Examples   Count the number of standard 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);
```

Create a heat map for the codons in a nucleotide sequence.

```
a = randseq(1000);
codoncount(a,'Figure', true);
```

codoncount
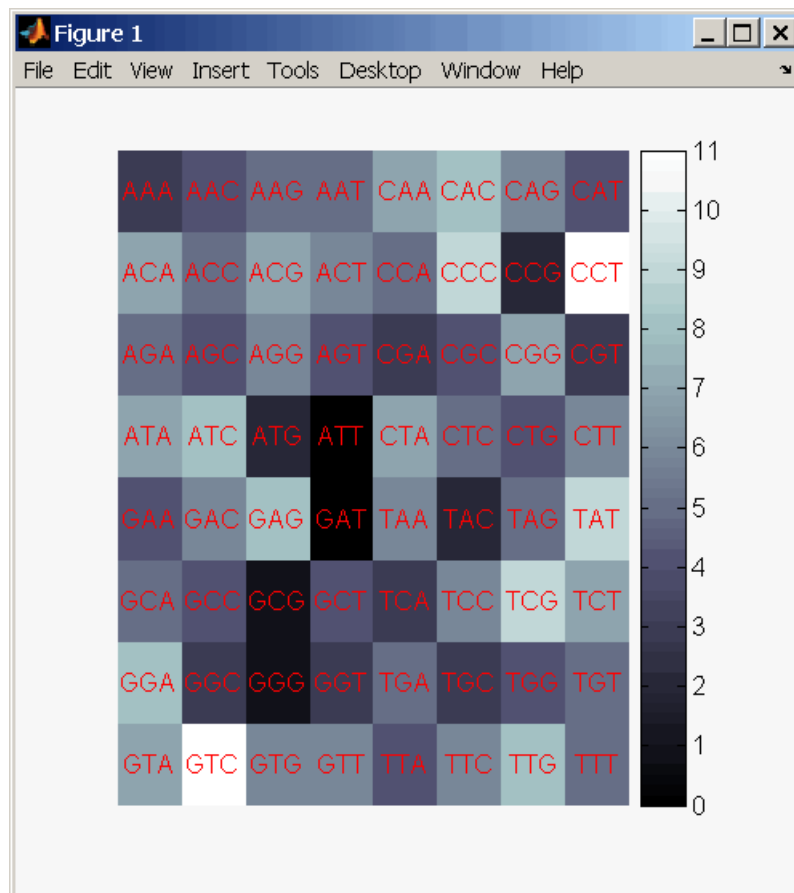
See Also   Bioinformatics Toolbox functions: aacount, basecount, baselookup, codonbias, dimercount, nmercount, ntdensity, seqrcomplement, seqwordcount

Purpose   Locate CpG islands in DNA sequence

Syntax
```
cpgisland(SeqDNA)
cpgisland(..., 'PropertyName', PropertyValue,...)
cpgisland(..., 'Window', WindowValue)
cpgisland(..., 'MinIsland', MinIslandValue)
cpgisland(..., 'CpGoe', CpGoeValue)
cpgisland(..., 'GCmin', GCminValue)
cpgisland(..., 'Plot', PlotValue)
```

Arguments

SeqDNA	DNA nucleotide sequence. Enter a character string with the letters A, T, C, and G. You can also enter a structure with the field Sequence. cpgisland does not count ambiguous bases or gaps.

Description   cpgisland(*SeqDNA*) finds CpG islands by marking bases within a moving window of 100 DNA bases with a GC content greater than 50% and a CpGobserved/CpGexpected ratio greater than 60%.

cpgisland(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

cpgisland(..., 'Window', *WindowValue*) specifies the window size for calculating GC percent and CpGobserved/CpGexpected ratios for a sequence. The default value is 100 bases. A smaller window size increases the noise in a plot.

cpgisland(..., 'MinIsland', *MinIslandValue*) specifies the minimum number of consecutive marked bases to report. The default value is 200 bases.

cpgisland(..., 'CpGoe', *CpGoeValue*) specifies the minimum CpGobserved/CpGexpected ratio in each window needed to mark a base. Enter a value between 0 and 1. The default value is 0.6. This ratio is defined as

```
CPGobs/CpGexp = (NumCpGs*Length)/(NumGs*NumCs)
```

cpgisland(..., 'GCmin', *GCminValue*) specifies the minimum GC percent in a window needed to mark a base. Enter a value between 0 and 1. The default value is 0.5.

cpgisland(..., 'Plot', *PlotValue*), when Plot is true, plots GC content, CpGoe content, CpG islands greater than the minimum island size, and all potential CpG islands for the specified criteria.

Examples

1 Import a nucleotide sequence from GenBank. For example, get 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)
```

MATLAB lists the CpG islands greater than 200 bases and draws a figure.

```
ans =
    Starts: [4470 28753 29347 36229]
    Stops: [5555 29064 29676 36450]
```

See Also   Bioinformatics Toolbox functions: `basecount`, `ntdensity`, `seqshoworfs`

crossvalind

Purpose　　　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. Use K-fold 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   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
```

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
```

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
sum(train) % Total groups allocated for training
```

See Also   Bioinformatics Toolbox functions: classperf, knnclassify,
svmclassify

Statistics Toolbox functions: classify, grp2idx

Purpose   Read cytogenetic banding information

Syntax   *CytoStruct* = cytobandread(*File*)

Arguments

File	String specifying a file containing cytogenetic G-banding data, such as an NCBI ideogram text file or a UCSC cytoband text file.

Return Values

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.
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.

Field	Description
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 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   Bioinformatics Toolbox function: chromosomeplot

dayhoff

Purpose   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   Bioinformatics Toolbox functions: `blosum`, `gonnet`, `pam`

Purpose   Count dimers in sequence

Syntax
Dimers = dimercount(*SeqNT*)
[*Dimers, Percent*] = dimercount(*SeqNT*)
dimercount(..., '*PropertyName*', *PropertyValue*,...)
dimercount(..., 'Chart', *ChartStyle*)

Arguments

SeqNT	Nucleotide sequence. Enter a character string or vector of integers.
	Examples: 'ACGT' and [1 2 3 4].You can also enter a structure with the field Sequence.
ChartStyleValue	Property to select the type of plot. Enter 'pie' or 'bar'.

Description   *Dimers* = dimercount(*SeqNT*) counts the number of nucleotide dimers in a 1-by-1 sequence and returns the dimer counts in a structure with the fields AA, AC, AG, AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, TT.

- For sequences that have dimers with the character U, the U characters are added to dimers with T characters.

- If the sequence contains ambiguous nucleotide characters (R Y K M S W B D H V N), or gaps indicated with a hyphen (-), this function creates a field Others and displays a warning message.

      Warning: Ambiguous symbols '*symbol list*' appear
      in the sequence.
      These will be in Others.

- If the sequence contains undefined nucleotide characters (E F H I J L O P Q X Z), codoncount ignores the characters and displays a warning message.

dimercount

```
Warning: Unknown symbols 'symbol list' appear
in the sequence.
These will be ignored.
```

[*Dimers, Percent*] = dimercount(*SeqNT*) returns 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.

dimercount(..., *'PropertyName'*, *PropertyValue*,...) defines optional properties using property name/value pairs.

dimercount(..., 'Chart', *ChartStyle*) creates a chart showing the relative proportions of the dimers.

Examples   Count the number of dimers in a nucleotide sequence.

```
dimercount('TAGCTGGCCAAGCGAGCTTG')

ans =
    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
```

See Also   Bioinformatics Toolbox functions `aacount`, `basecount`, `baselookup`, `codoncount`, `nmercount`, `ntdensity`

dna2rna

Purpose	Convert DNA sequence to RNA sequence
Syntax	*SeqRNA* = dna2rna(*SeqDNA*)

Arguments

SeqDNA	DNA sequence. Enter either a character string with the characters A, T, G, C, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N, or a vector of integers from the table . You can also enter a structure with the field Sequence.
SeqRNA	RNA sequence.

Description   *SeqRNA* = dna2rna(*SeqDNA*) converts a DNA sequence to an RNA sequence by converting any thymine nucleotides (T) in the DNA sequence to uracil (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   Bioinformatics Toolbox function: rna2dna

MATLAB functions: regexp, strrep

Purpose   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*, ...'Verbose',
                   VerboseValue, ...)

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 on page 2-6. 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.
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.

Return Values	*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. See Estimating Synonymous and Nonsynonymous Substitution Rates Between Two Nucleotide Sequences That Are Not Codon-Aligned on page 2-208.

dnds

- 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 on page 2-6. 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.

2-206

• 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*, ...'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   **Estimating Synonymous and Nonsynonymous Substitution Rates Between the gag Genes of Two HIV Viruses**

1 Retrieve two sequences from the GenBank database for the gag genes of two HIV viruses.

```
gag1 = getgenbank('L11768');
gag2 = getgenbank('L11770');
```

2 Estimate the synonymous and nonsynonymous substitution rates between the two sequences.

```
[dn ds vardn vards] = dnds(gag1, gag2)

dn =

    0.0241
```

```
ds =

    0.0739

vardn =

  2.2785e-005

vards =

  2.6447e-004
```

Estimating Synonymous and Nonsynonymous Substitution Rates Between Two Nucleotide Sequences That Are Not Codon-Aligned

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 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.

```
[dn,ds] = dnds(hk01_aligned,vt04_aligned,'verbose',true)
```

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   Bioinformatics Toolbox functions: dndsml, featuresparse, geneticcode, nt2aa, nwalign, seqinsertgaps, seqpdist

dndsml

Purpose   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, ...)

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 on page 2-6. 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.

Return Values

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 Yang-Nielsen method (2000). 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. See Estimating Synonymous and Nonsynonymous Substitution Rates Between Two Nucleotide Sequences That Are Not Codon-Aligned on page 2-213

- 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 on page 2-6. 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

Estimating Synonymous and Nonsynonymous Substitution Rates Between the gag Genes of Two HIV Viruses

1 Retrieve two sequences from the GenBank database for the gag genes of two HIV viruses

```
gag1 = getgenbank('L11768');
gag2 = getgenbank('L11770');
```

2 Estimate the synonymous and nonsynonymous substitution rates between the two sequences.

```
[dn ds like] = dndsml(gag1, gag2)

dn =
    0.0259
ds =
    0.0624
like =
    -2.1864e+003
```

Estimating Synonymous and Nonsynonymous Substitution Rates Between Two Nucleotide Sequences That Are Not Codon-Aligned

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 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.

```
[dn,ds] = dndsml(hk01_aligned,vt04_aligned,'verbose',true)
```

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.

See Also   Bioinformatics Toolbox functions: dnds, featuresparse, geneticcode, nt2aa, nwalign, seqinsertgaps, seqpdist

Purpose   Read data from EMBL file

Syntax

EMBLData = emblread('*File*')
EMBLSeq = emblread ('*File*', SequenceOnly',
SequenceOnlyValue)

Arguments

File	EMBL formatted file (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 name.
SequenceOnlyValue	Property to control reading EMBL file information. If *SequenceOnlyValue* is true, emblread returns only the sequence (*EMBLSeq*).
EMBLData	MATLAB structure with fields corresponding to EMBL data.
EMBLSeq	MATLAB character string without metadata for the sequence.

Description   *EMBLData* = emblread('*File*') reads data from an EMBL formatted file (*File*) and creates a MATLAB structure (*EMBLData*) 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.EntryName
Identification.Version
Identification.Topology
Identification.Molecule
Identification.DataClass

Field
Identification.Division
Identification.SequenceLength
Accession
SequenceVersion
DateCreated
DateUpdated
Description
Keyword
OrganismSpecies
OrganismClassification
Organelle
Reference{#}.Number
Reference{#}.Comment
Reference{#}.Position
Reference{#}.MedLine
Reference{#}.PubMed
Reference{#}.Authors
Reference{#}.Title
Reference{#}.Location
DatabaseCrossReference
Comments
Feature
Basecount.BP
Basecount.A
Basecount.C

Field
Basecount.G
Basecount.T
Basecount.Other
Sequence

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*), when *SequenceOnlyValue* is true, reads only the sequence information.

Examples   Get sequence information from the Web, save to a file, and then read back into MATLAB.

```
getembl('X00558','ToFile','rat_protein.txt');
EMBLData = emblread('rat_protein.txt')
```

See Also   Bioinformatics Toolbox functions: fastaread, genbankread, getembl, seqtool

evalrasmolscript

Purpose	Send RasMol script commands to Molecule Viewer window
Syntax	evalrasmolscript(*FigureHandle*, *Command*) evalrasmolscript(*FigureHandle*, 'File', *FileValue*)

Arguments

FigureHandle	Figure handle to a molecule viewer returned by the molviewer function.
Command	Either 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/

FileValue	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 Directory.

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.

evalrasmolscript(*FigureHandle*, 'File', *FileValue*) sends the RasMol script commands specified by *FileValue* 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

Bioinformatics Toolbox functions: `getpdb`, `molviewer`, `pdbread`, `pdbwrite`

exprprofrange

Purpose	Calculate range of gene expression profiles
Syntax	*Range* = exprprofrange(*Data*) [*Range, LogRange*] = exprprofrange(*Data*) ... = exprprofrange(*Data*, 'ShowHist', *ShowHistValue*)

Arguments

Data	Matrix where each row corresponds to a gene.
ShowHistValue	Controls the display of a histogram with range data. Choices are true or false (default).

Description

Range = exprprofrange(*Data*) calculates the range of each expression profile in *Data*, a data set.

[*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 (default).

Examples

Calculate the range of expression profiles for yeast data as gene expression changes during the metabolic shift from fermentation to respiration.

```
load yeastdata
range = exprprofrange(yeastvalues,'ShowHist',true);
```

See Also

Bioinformatics Toolbox functions: exprprofvar, generangefilter

Purpose   Calculate variance of gene expression profiles

Syntax   *Variance* = exprprofvar(*Data*)
               exprprofvar(..., '*PropertyName*', *PropertyValue*,...)
               exprprofvar(..., 'ShowHist', *ShowHistValue*)

Arguments

Data	Matrix where each row corresponds to a gene.
ShowHistValue	Property to control the display of a histogram with variance data. Enter either true or false (default).

Description *Variance* = exprprofvar(*Data*) calculates the variance of each
expression profile in a data set (*Data*). 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*), when ShowHist is
true, displays a histogram of the range data .

Examples   Calculate the variance of expression profiles for yeast data as gene
expression changes during the metabolic shift from fermentation to
respiration.

```
load yeastdata
datavar = exprprofvar(yeastvalues,'ShowHist',true);
```

See Also   Bioinformatics Toolbox functions exprprofrange, generangefilter,
genevarfilter

fastaread

Purpose	Read data from FASTA file

Syntax

```
FASTAData = fastaread(File)
[Header, Sequence] = fastaread(File)
... = fastaread(File, ...'IgnoreGaps',
IgnoreGapsValue, ...)
... = fastaread(File, ...'Blockread', BlockreadValue, ...)
```

Arguments

File	FASTA-formatted file (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 name.
FASTAData	MATLAB structure with the fields Header and Sequence.
IgnoreGapsValue	Property to control removing gap symbols. Enter either true or false (default).
BlockreadValue	Property to control reading a single entry or block of entries from a file containing multiple sequences. Enter a scalar N, to read the Nth entry in the file. Enter a 1-by-2 vector [*M1, M2*], to read the block of entries starting at entry *M1* and ending at entry *M2*. To read all remaining entries in the file starting at entry *M1*, enter a positive value for *M1* and enter Inf for *M2*.

Description

fastaread reads data from a FASTA-formatted file into a MATLAB structure with the following fields:

Field
Header
Sequence

A file with a FASTA format 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 are expected to 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 file with a FASTA format 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 more than one sequence, 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 entry or block of entries from a file containing multiple sequences. If *BlockreadValue* is a scalar N, then fastaread reads the Nth entry in the file. If *BlockreadValue* is a 1-by-2 vector [*M1, M2*], then fastaread reads the block of entries starting at entry *M1* and ending at entry *M2*. To read all remaining entries in the file starting at entry *M1*, enter a positive value for *M1* and enter Inf for *M2*.

fastaread

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 the human mitochondrion genome in FASTA format.

```
entrezSite = 'http://www.ncbi.nlm.nih.gov/entrez/viewer.fcgi?'
textOptions = '&txt=on&view=fasta'
genbankID = '&list_uids=NC_001807'
mitochondrion = fastaread([entrezSite textOptions genbankID])
```

See Also   Bioinformatics Toolbox functions: emblread, fastawrite, genbankread,
genpeptread, multialignread, seqprofile, seqtool

Purpose   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 supported by your operating system. If you specify only a file name, the file is saved to the MATLAB Current Directory.
Data	Any of the following: • String with a FASTA format • Sequence object • MATLAB structure containing the fields Header and Sequence • GenBank/GenPept structure
Header	String containing information about the sequence. This text will be included 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 on page 2-57 or Nucleotide Lookup on page 2-67.

Description   fastawrite(*File*, *Data*) writes the contents of *Data* to a
                   FASTA-formatted file (ASCII text file).

                   fastawrite(*File*, *Header*, *Sequence*) writes the specified header
                   and sequence information to a FASTA-formatted file (ASCII text file).

Examples   %get the sequence for the human p53 gene from GenBank.
                   seq = getgenbank('NM_000546')

fastawrite

```
%find the CDS line in the FEATURES information.
cdsline = strmatch('CDS',seq.Features)

%read the coordinates of the coding region.
[start,stop] = strread(seq.Features(cdsline,:),'%*s%d..%d')

%extract the coding region.
codingSeq = seq.Sequence(start:stop)

%write just the coding region to a FASTA file.
fastawrite('p53coding.txt','Coding region for p53',codingSeq);
```

Save multiple sequences.

```
data(1).Sequence = 'ACACAGGAAA'
data(1).Header = 'First sequence'
data(2).Sequence = 'ACGTCAGGTC'
data(2).Header = 'Second sequence'

fastawrite('my_sequences.txt', data)
type('my_sequences.txt')

>First sequence
ACACAGGAAA

>Second sequence
ACGTCAGGTC
```

See Also   Bioinformatics Toolbox functions: `fastaread`, `seqtool`

Purpose   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 on page 2-235.

ShowPositionsValue	Property to add the sequence position to the annotation label for each feature. Enter true to add the sequence position. Default is false.

featuresmap

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.

featuresmap

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 on page 2-235.

`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`.
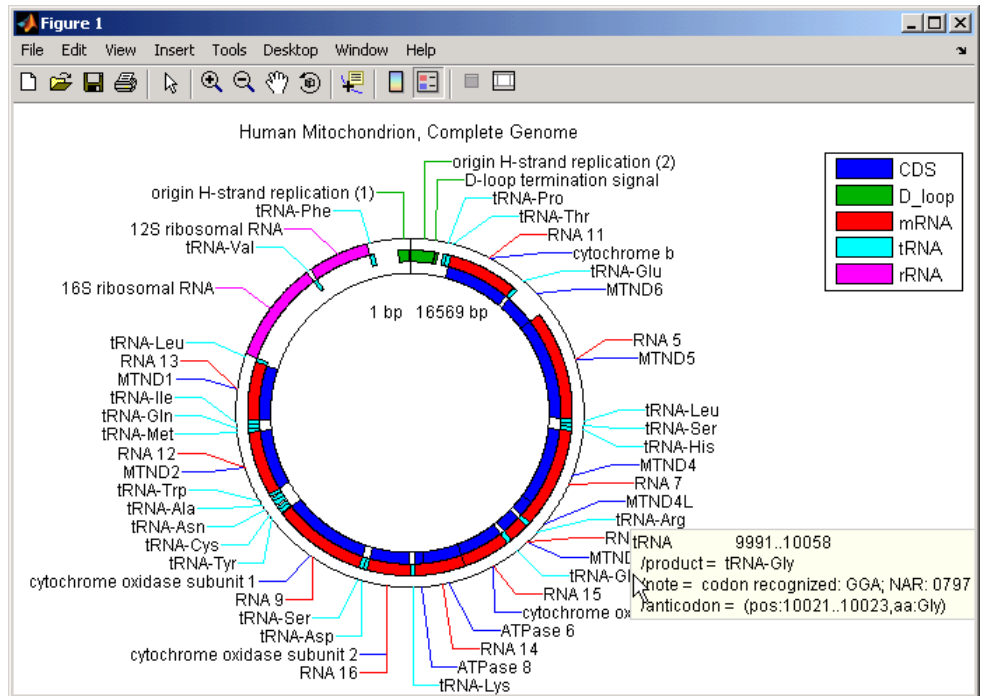
featuresmap
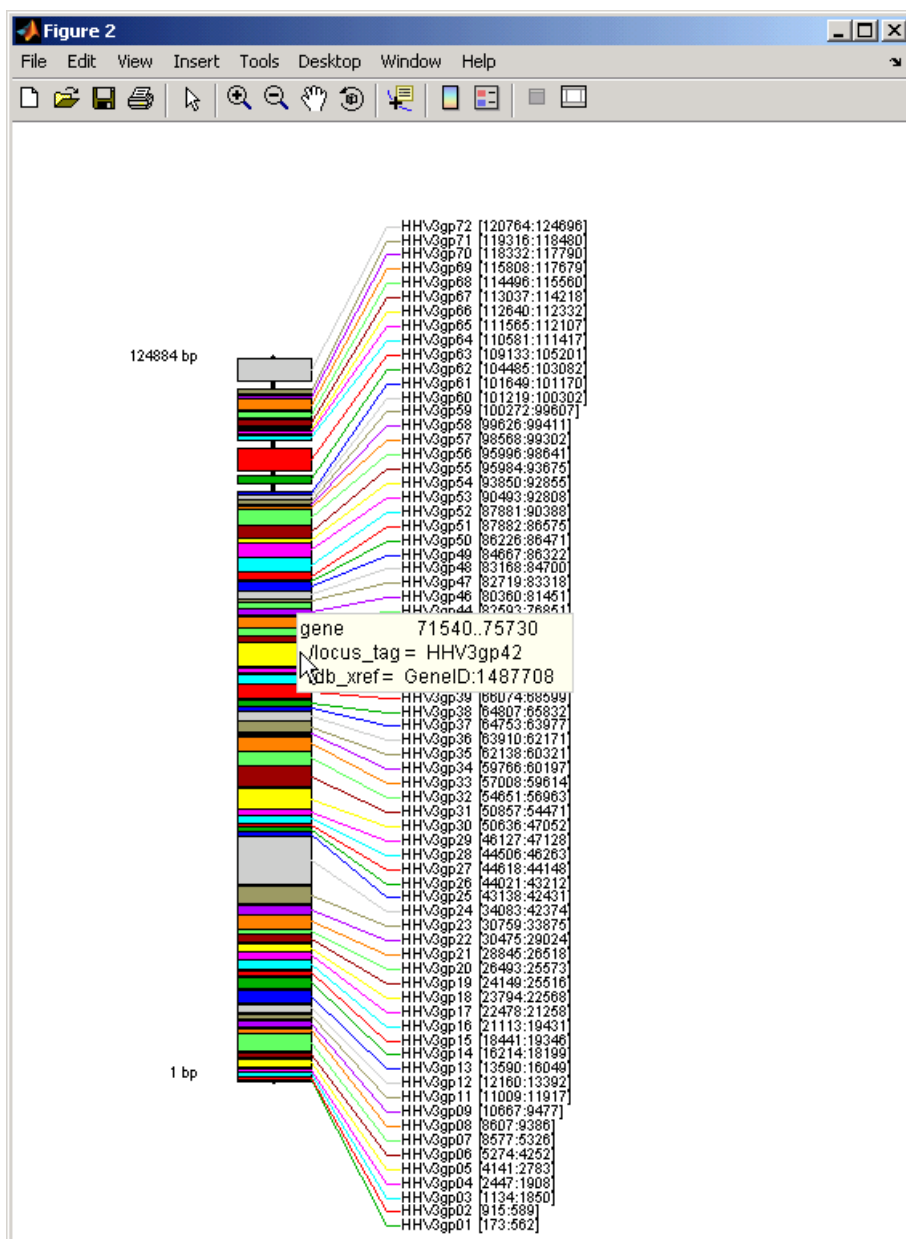
After creating a map:

- 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 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

featuresmap

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, seqtool

Purpose   Parse features from GenBank, GenPept, or EMBL data

Syntax   *FeatStruct* = featuresparse(*Features*)
                   FeatStruct = featuresparse(*Features*, ...'Feature',
                   FeatureValue, ...)
                   FeatStruct = featuresparse(*Features*, ...'Sequence',
                      SequenceValue, ...)

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).

featuresparse

Return Values

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
-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

featuresparse

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   Bioinformatics Toolbox functions: `emblread`, `genbankread`, `genpeptread`, `getgenbank`, `getgenpept`

Purpose   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://www.moleculardevices.com/pages/software/gn_genepix_file_formats.html#gal

                     For a list of supported file format versions, see

                     http://www.moleculardevices.com/pages/software/gn_genepix_file_formats.html

                     GenePix is a registered trademark of Molecular Devices Corporation.

See Also   Bioinformatics Toolbox functions: affyread, geosoftread, gprread,
                     imageneread, sptread

gcrma

Purpose

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*, ...)

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, assuming no affinity data is provided. 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 $\geq$ 0 and $\leq$ 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	Controls whether gene-specific binding (GSB) correction is performed on the nonspecific binding (NSB) data. Choices are true (default) or false.
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.

**Return
Values**

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*, assuming no affinity data is provided. 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*, ...) controls whether gene specific binding (GSB) correction is performed on the nonspecific binding (NSB) data. Choices are true (default) or false.

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 Bioinformatics Toolbox, 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

Bioinformatics Toolbox functions: `affyprobeseqread`, `affyread`, `celintensityread`, `gcrmabackadj`, `quantilenorm`, `rmabackadj`, `rmasummary`

Purpose　　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(...'Showplot', *ShowplotValue*, ...)
... = gcrmabackadj(...'Verbose', *VerboseValue*, ...)

gcrmabackadj

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).

gcrmabackadj

	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.

Return Values

	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.

[*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 $\geq$ 0 and $\leq$ 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

... = 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(...'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 Bioinformatics Toolbox, 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);
```
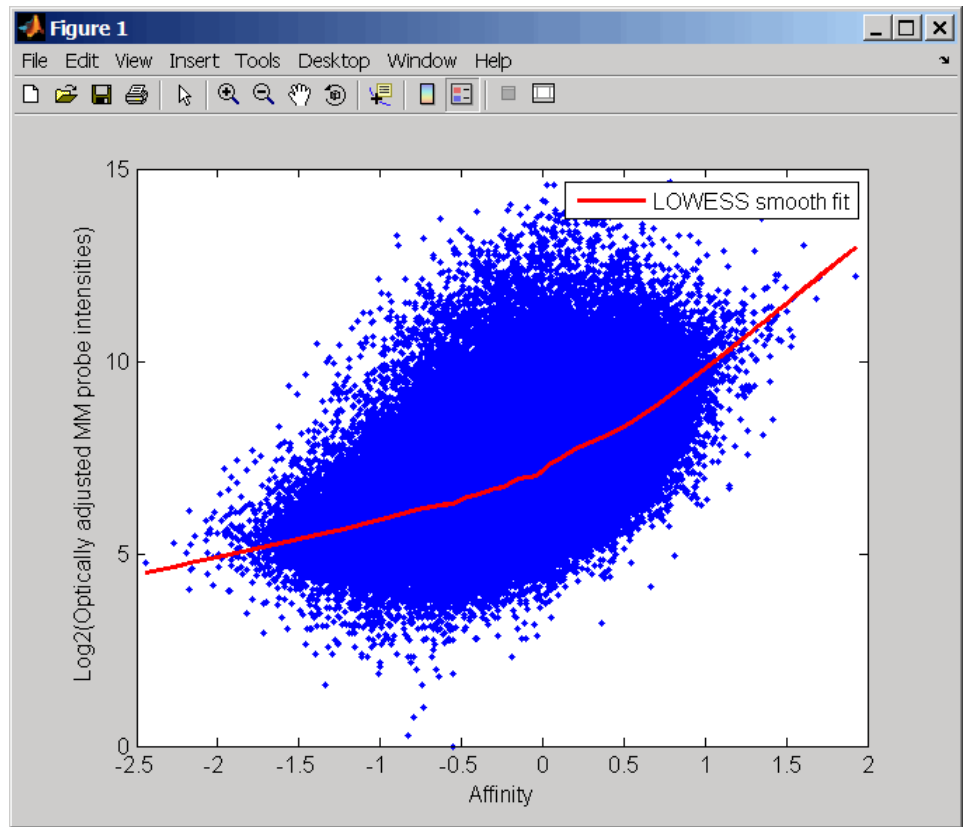
4 Perform GCRMA background adjustment again, using the slower, more formal, empirical Bayes method.

```
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   Bioinformatics Toolbox functions: `affyprobeseqread`, `affyread`, `celintensityread`, `probelibraryinfo`

genbankread

Purpose	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 Directory.
	• MATLAB character array that contains the text of a GenBank-formatted file.
GenBankData	MATLAB structure with fields corresponding to GenBank keywords.

Description

GenBankData = genbankread(*File*) reads in a GenBank-formatted file, *File*, and creates a structure, *GenBankData*, containing fields corresponding to the GenBank keywords. Each separate sequence listed in the output structure *GenBankData* is stored as a separate element of the structure.

Examples

1 Get sequence information for a gene (HEXA), store data in a file, and then read back into MATLAB.

```
getgenbank('nm_000520', 'ToFile', 'TaySachs_Gene.txt')
s = genbankread('TaySachs_Gene.txt')

s =

                 LocusName: 'NM_000520'
       LocusSequenceLength: '2255'
    LocusNumberofStrands: ''
```

```
                      LocusTopology: 'linear'
                 LocusMoleculeType: 'mRNA'
              LocusGenBankDivision: 'PRI'
            LocusModificationDate: '13-AUG-2006'
                        Definition: [1x63 char]
                         Accession: 'NM_000520'
                           Version: 'NM_000520.2'
                                GI: '13128865'
                           Project: []
                          Keywords: []
                           Segment: []
                            Source: 'Homo sapiens (human)'
                    SourceOrganism: [4x65 char]
                         Reference: {1x58 cell}
                           Comment: [15x67 char]
                          Features: [74x74 char]
                               CDS: [1x1 struct]
                          Sequence: [1x2255 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   Bioinformatics Toolbox functions: emblread, fastaread, genpeptread, getgenbank, scfread, seqtool

geneentropyfilter

Purpose   Remove genes with low entropy expression values

Syntax
```
Mask = geneentropyfilter(Data)
[Masks, FData] = geneentropyfilter(Data)
[Mask, FData, FNames] = geneentropyfilter(Data,Names)
geneentropyfilter(..., 'PropertyName', PropertyValue,...)
geneentropyfilter(..., 'Percentile', PercentileValue)
```

Arguments

Data	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 then the threshold are 0.

[*Masks, FData*] = geneentropyfilter(*Data*) returns a filtered data matrix (*FData*). *FData* can also be created using *FData* = Data(find(I),:).

[*Mask, FData, FNames*] = geneentropyfilter(*Data,Names*) returns a filtered names array (*FNames*). You can also create *FNames* using *FNames* = Names(I).

geneentropyfilter(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

geneentropyfilter(..., 'Percentile', *PercentileValue*)
removes from the experimental data (*Data*) gene expression profiles
with entropy values less than a given percentile (*PercentileValue*).

Examples
```
load yeastdata
[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   Bioinformatics Toolbox functions: exprprofrange, exprprofvar,
genelowvalfilter, generangefilter, genevarfilter

genelowvalfilter

Purpose	Remove gene profiles with low absolute values
Syntax	*Mask* = genelowvalfilter(*Data*) [*Mask*, *FData*] = genelowvalfilter(*Data*) [*Mask, FData, FNames*] = genelowvalfilter(*Data*, *Names*) genelowvalfilter(..., '*PropertyName*', *PropertyValue*,...) genelowvalfilter(..., 'Prctile', *PrctileValue*) genelowvalfilter(..., 'AbsValue', *AbsValueValue*) genelowvalfilter(..., 'AnyVal', *AnyValValue*)

Arguments

Data	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 same number of rows as *Data*. Each row contains the name or ID of the gene in the data set.
PrctileValue	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.
AnyValValue	Property to select the minimum or maximum absolute value for comparison with *AbsValueValue*. If *AnyValValue* is true, selects the minimum absolute value. If *AnyValValue* is false, selects the maximum absolute value. The default value is false.

Description

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.

Mask = genelowvalfilter(*Data*) identifies gene expression profiles in Data with all absolute 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 absolute expression levels greater than the threshold have a value of 1, and those with absolute expression levels less then the threshold are 0.

[*Mask*, *FData*] = genelowvalfilter(*Data*) returns a filtered data matrix (*FData*). You can create *FData* using *FData* = Data(find(*I*),:).

[*Mask, FData, FNames*] = genelowvalfilter(*Data*, *Names*) returns a filtered names array (*FNames*), 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(I).

genelowvalfilter(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

genelowvalfilter(..., 'Prctile', *PrctileValue*) removes from the experimental data (*Data*) gene expression profiles with all absolute values less than a specified percentile (*Percentile*).

genelowvalfilter(..., 'AbsValue', *AbsValueValue*) calculates the maximum absolute value for each gene expression profile and removes the profiles with maximum absolute values less than *AbsValValue*.

genelowvalfilter(..., 'AnyVal', *AnyValValue*), when *AnyValValue* is true, calculates the minimum absolute value for each gene expression profile and removes the profiles with minimum absolute values less than *AnyValValue*.

Examples

```
[data, labels, I, FI] = genelowvalfilter(data,labels,'AbsValue',5);
```

References

[1] Kohane I.S., Kho A.T., Butte A.J. (2003), Microarrays for an Integrative Genomics, Cambridge, MA:MIT Press.

See Also

Bioinformatics Toolbox functions: exprprofrange, exprprofvar, geneentropyfilter, generangefilter, genevarfilter

geneont

Purpose	Create geneont object

Syntax

GeneontObj = geneont
GeneontObj = geneont('File', *FileValue*)
GeneontObj = geneont('Live', *LiveValue*)
GeneontObj = geneont('Live', *LiveValue*, 'ToFile',
    ToFileValue)

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.

Return Values

GeneontObj	MATLAB object containing gene ontology information.

Description

GeneontObj = geneont creates *GeneontObj*, a geneont object, from the gene_ontology.obo file in the MATLAB current directory.

GeneontObj = geneont('File', *FileValue*) creates *GeneontObj*, a geneont object, from *FileValue*, a string specifying the file name of an 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.

Examples   **1** Download the current version of the Gene Ontology database from the Web into a geneont object in MATLAB.

    GO = geneont('LIVE', true)

MATLAB creates a geneont object and displays the number of terms in the database.

    Gene Ontology object with 24316 Terms.

2 Display information about the geneont object.

    get(GO)

    default_namespace: 'gene_ontology'
                 date: '12:06:2007 19:30'
       format_version: '1.0'
              version: '4.256'
            subsetdef: {5x1 cell}
                Terms: [24316x1 geneont.term]

3 Search for all GO terms in the geneont object that contain the string ribosome in the field name and create a MATLAB structure containing those terms.

```
comparison = regexpi(get(GO.Terms,'name'),'ribosome');
indices = find(~cellfun('isempty',comparison));
terms_with_ribosmome = GO.Term(indices)

22x1 struct array with fields:
    id
    name
    ontology
    definition
    comment
    synonym
    is_a
    part_of
    obsolete
```

See Also   Bioinformatics Toolbox functions: goannotread, num2goid

Bioinformatics Toolbox object: geneont object

Bioinformatics Toolbox methods of geneont object: getancestors, getdescendants, getmatrix, getrelatives

Purpose   Remove gene profiles with small profile ranges

Syntax
```
Mask = generangefilter(Data)
[Mask, FData] = generangefilter(Data)
[Mask, FData, FNames] = generangefilter(Data,Names)
generangefilter(..., 'PropertyName', PropertyValue,...)
generangefilter(..., 'Percentile', PercentileValue)
generangefilter(..., 'AbsValue', AbsValueValue)
generangefilter(..., 'LOGPercentile', LOGPercentileValue)
generangefilter(..., 'LOGValue', LOGValueValue)
```

Arguments

Data	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 LOG of a percentile.
LOGValueValue	Property to specify the LOG of an absolute value.

Description   *Mask* = generangefilter(*Data*) calculates the range for each gene expression profile in the experimental data (*Data*), and then identifies the expression profiles with ranges less than the 10th percentile.

generangefilter

Mask is a logical vector with one element for each row in *Data*. The elements of *Mask* corresponding to rows with a range greater then the threshold have a value of 1, and those with a range less then the threshold are 0.

[*Mask, FData*] = generangefilter(*Data*) returns a filtered data matrix (*FData*). *FData* can also be created using *FData* = Data(find(I),:).

[*Mask, FData, FNames*] = generangefilter(*Data,Names*) returns a filtered names array (*FNames*), where *Names* is a cell array with the names of the genes corresponding to each row in *Data*. You can also create *FNames* using *FNames* = Names(I).

generangefilter(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

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

```
load yeastdata
[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   Bioinformatics Toolbox functions: exprprofrange, exprprofvar, geneentropyfilter, genelowvalfilter, genevarfilter

Purpose   Nucleotide codon to amino acid mapping

Syntax   *Map* = geneticcode
                geneticcode(*GeneticCode*)

Arguments   *GeneticCode*   Enter a code number or code name from the table
                                  Genetic Code on page 2-273. If you use a code
                                  name, you can truncate the name to the first two
                                  characters 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

geneticcode

Genetic Code (Continued)

Code Number	Code Name
21	Trematode Mitochondrial
22	Scenedesmus Obliquus Mitochondrial
23	Thraustochytrium Mitochondrial

Description *Map* = geneticcode returns a structure with a mapping of nucleotide codons to amino acids for the standard genetic code.

geneticcode(*GeneticCode*) returns a structure of the mapping for alternate genetic codes, where *GeneticCode* is either of the following:

• The transl_table (code) number from the NCBI Genetics Web page

    http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c

• One of the supported names in the table Genetic Code on page 2-273

Examples List the mapping of nucleotide codons to amino acids for a specific genetic code.

    wormcode = geneticcode('Flatworm Mitochondrial');

See Also Bioinformatics Toolbox functions: aa2nt, aminolookup, baselookup, codonbias, dnds, dndsml, nt2aa, revgeneticcode, seqshoworfs, seqtool

Purpose   Filter genes with small profile variance

Syntax   *Mask* = genevarfilter(*Data*)
[*Mask, FData*] = genevarfilter(*Data*)
[*Mask, FData, FNames*] = genevarfilter(*Data,Names*)
genevarfilter(..., '*PropertyName*', *PropertyValue*,...)
genevarfilter(..., 'Prctile', *PrctileValue*)
genevarfilter(..., 'AbsValue', *AbsValValue*)

Arguments

Data	Matrix where each row corresponds to a gene. 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.
Prctile	Property to specify a percentile below which gene expression profiles are removed. Enter a value from 0 to 100.
AbsValue	Property to specify an absolute value below which gene expression profiles are removed.

Description   Gene profiling experiments have genes that exhibit little variation in the profile and are generally not of interest in the experiment. These genes are commonly removed from the data.

Mask = genevarfilter(*Data*) calculates the variance for each gene expression profile in Data and then identifies the 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.

genevarfilter

[*Mask, FData*] = genevarfilter(*Data*) returns the filtered data matrix (*FData*). You can also create *FData* using *FData = Data*(find(I),:).

[*Mask, FData, FNames*] = genevarfilter(*Data,Names*) returns a filtered names array (FNames). Names is a cell array of the names of the genes corresponding to each row of Data. FNames can also be created using FNames = Names(I).

genevarfilter(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

genevarfilter(..., 'Prctile', *PrctileValue*) removes from the experimental data (*Data*) gene expression profiles with a variance less than the percentile (*Prctile*).

genevarfilter(..., 'AbsValue', *AbsValValue*) removes from *Data* gene expression profiles with a variance less than AbsValue.

Examples
```
load yeastdata
[fyeastvalues, fgenes] = genevarfilter(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
Bioinformatics Toolbox functions: exprprofrange, exprprofvar, generangefilter, geneentropyfilter, genelowvalfilter

Purpose	Read data from GenPept file
Syntax	GenPeptData = genpeptread('*File*')
Arguments	*File* GenPept formatted file (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 of a GenPept file.

Description   genpeptread reads data from a GenPept formatted file into a MATLAB structure.

Note NCBI has changed the name of their protein search engine from GenPept to Entrez Protein. However, the function names in Bioinformatics Toolbox (getgenpept and genpeptread) are unchanged representing the still-used GenPept report format.

GenPeptData = genpeptread('*File*') reads in the GenPept formatted sequence from *File* and creates a structure GenPeptData, containing fields corresponding to the GenPept keywords. Each separate sequence listed in *File* is stored as a separate element of the structure. GenPeptDATA contains these fields:

Field
LocusName
LocusSequenceLength
LocusMoleculeType
LocusGenBankDivision
LocusModificationDate
Definition

Field
Accession
PID
Version
GI
DBSource
Keywords
Source
SourceDatabase
SourceOrganism
Reference.Number
Reference.Authors
Reference.Title
Reference.Journal
Reference.MedLine
Reference.PubMed
Reference.Remark
Comment
Features
Weight
Length
Sequence

Examples   Get sequence information for the protein coded by the gene HEXA, save to a file, and then read back into MATLAB.

```
getgenpept('p06865', 'ToFile', 'TaySachs_Protein.txt')
genpeptread('TaySachs_Protein.txt')
```

See Also   Bioinformatics Toolbox functions: `fastaread`, `genbankread`,
`getgenpept`, `pdbread`, `seqtool`

geosoftread

Purpose　　Read Gene Expression Omnibus (GEO) SOFT format data

Syntax　　*GEOSOFTData* = geosoftread(*File*)

Arguments

File	Gene Expression Omnibus (GEO) SOFT format Sample file (GSM) or Data Set file (GDS). Enter a file name, a path and file name, or a URL pointing to a file.

> **Note** *File* can also be a MATLAB character array that contains the text of a GEO file.

Description　　*GEOSOFTData* = geosoftread(*File*) reads a Gene Expression Omnibus (GEO) SOFT format Sample file (GSM) or Data Set file (GDS), and then creates a MATLAB structure, *GEOSOFTdata*, with the following fields.

Fields
Scope
Accession
Header
ColumnDescriptions
ColumnNames
Data
Identifier (GDS files only)
IDRef (GDS files only)

Fields correspond to the GenBank keywords. Each separate entry listed in *File* is stored as a separate element of the structure.

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   Bioinformatics Toolbox functions: `galread`, `getgeodata`, `gprread`, `sptread`

getblast

Purpose	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*, ...)

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 MATLAB 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*.

Return Values

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 ≤ 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 MATLAB 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://www.ncbi.nlm.nih.gov/Education/BLASTinfo/tut1.html

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.

Field	Description
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.

Field	Description
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 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://www.ncbi.nlm.nih.gov/Education/BLASTinfo/Blast_output.html
```

See Also   Bioinformatics Toolbox functions: blastformat, blastlocal, blastncbi, blastread, blastreadlocal

getembl

Purpose	Retrieve sequence information from EMBL database
Syntax	*Data* = getembl(*AccessionNumber*) getembl(..., *'PropertyName'*, *PropertyValue*,...) getembl(..., 'ToFile', *ToFileValue*) getembl(..., 'SequenceOnly', *SequenceOnlyValue*)

Arguments

AccessionNumber	Unique identifier for a sequence record. Enter a unique combination of letters and numbers.
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).
SequenceOnlyValue	Property to control getting a sequence without the metadata. Enter either true or false (default).

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-Bank database, see

   http://www.ebi.ac.uk/embl/Documentation/index.html

Data = getembl(*AccessionNumber*) searches for the accession number in the EMBL database (http://www.ebi.ac.uk/embl) and returns a MATLAB structure containing the following fields:

Field
Comments
Identification
Accession

Field
SequenceVersion
DateCreated
DateUpdated
Description
Keyword
OrganismSpecies
OrganismClassification
Organelle
Reference
DatabaseCrossReference
Feature
BaseCount
Sequence

getembl(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

getembl(..., 'ToFile', *ToFileValue*) returns a structure containing information about the sequence and saves the information in a file using an EMBL data format. If you do not give a location or path to the file, the file is stored in the MATLAB current directory. Read an EMBL formatted file back into MATLAB using the function emblread.

getembl(..., 'SequenceOnly', *SequenceOnlyValue*) , if *SequenceOnlyValue* is true, returns the sequence information without the metadata.

Examples   Retrieve data for the rat liver apolipoprotein A-I.

```
emblout = getembl('X00558')
```

Retrieve data for the rat liver apolipoprotein and save in the file rat_protein. If a file name is given without a path, the file is stored in the current directory.

```
Seq = getembl('XOO558','ToFile','c:\project\rat_protein.txt')
```

Retrieve only the sequence for the rat liver apolipoprotein.

```
Seq = getembl('XOO558','SequenceOnly',true)
```

See Also   Bioinformatics Toolbox functions: emblread, getgenbank, getgenpept, getpdb, seqtool

Purpose   Retrieve sequence information from GenBank database

Syntax   *Data* = getgenbank('*AccessionNumber*')
                  getgenbank('*AccessionNumber*')
                  getgenbank(..., '*PropertyName*', *PropertyValue*,...)
                  getgenbank(..., 'ToFile', *ToFileValue*)
                  getgenbank(..., 'FileFormat', *FileFormatValue*)
                  getgenbank(..., 'SequenceOnly', *SequenceOnlyValue*)

Arguments

AccessionNumber	Unique identifier for a sequence record. Enter a unique combination of letters and numbers.
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).
FileFormatValue	Property to select the format for the file specified with the property ToFileValue. Enter either 'GenBank' or 'FASTA'.
SequenceOnlyValue	Property to control getting the sequence only. Enter either true or false.

Description   getgenbank retrieves nucleotide and amino acid sequence 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 a MATLAB structure containing information for the sequence. If an error occurs while retrieving the GenBank formatted information, then an attempt is make to retrieve the FASTA formatted data.

getgenbank('*AccessionNumber*') displays information in the MATLAB Command Window without returning data to a variable. The displayed information includes hyperlinks to the URLs for searching and retrieving data.

getgenbank(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

getgenbank(..., 'ToFile', *ToFileValue*) saves the data returned from GenBank in a file. If you do not give a location or path to the file, the file is stored in the MATLAB current directory. Read a GenBank formatted file back into MATLAB using the function genbankread.

getgenbank(..., 'FileFormat', *FileFormatValue*) returns the sequence in the specified format (*FileFormatValue*).

getgenbank(..., 'SequenceOnly', *SequenceOnlyValue*) when SequenceOnly is true, returns only the sequence as a character array. When the properties SequenceOnly and ToFile are used together, the output file is in the FASTA format.

Examples   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: []
             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: [1x105 char]
          RetrieveURL: [1x95 char]
```

See Also Bioinformatics Toolbox functions: genbankread, getembl, getgenpept, getpdb,seqtool

getgenpept

Purpose	Retrieve sequence information from GenPept database
Syntax	*Data* = getgenpept('*AccessionNumber*') getgenpept(...) getgenpept(..., '*PropertyName*', *PropertyValue*,...) getgenpept(..., 'ToFile', *ToFileValue*) getgenpept(..., 'FileFormat', *FileFormatValue*) getgenpept(..., 'SequenceOnly', *SequenceOnlyValue*)

Arguments

AccessionNumber	Unique identifier for a sequence record. Enter a combination of letters and numbers.
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).
FileFormatValue	Property to select the format for the file specified with the property *ToFileValue*. Enter either 'GenBank' or 'FASTA'.
SequenceOnlyValue	Property to control getting the sequence without metadata. Enter either true or false.

Description getgenpept retrieves a protein (amino acid) sequence and sequence information from the GenPept database. This database is a translation of the nucleotide sequences in GenBank 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 Bioinformatics Toolbox (getgenpept and genpeptread) are unchanged representing the still-used GenPept report format.

For more details about the GenBank database, see

```
http://www.ncbi.nlm.nih.gov/Genbank/
```

Data = getgenpept('*AccessionNumber*') searches for the accession number in the GenPept database and returns a MATLAB structure containing for the sequence. If an error occurs while retrieving the GenBank formatted information, then an attempt is make to retrieve the FASTA formatted data.

getgenpept(...) displays the information to the screen without returning data to a variable. The displayed information includes hyperlinks to the URLs used to search for and retrieve the data.

getgenpept(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

getgenpept(..., 'ToFile', *ToFileValue*) saves the information in a file. If you do not give a location or path to the file, the file is stored in the MATLAB current directory. Read a GenPept formatted file back into MATLAB using the function genpeptread

getgenpept(..., 'FileFormat', *FileFormatValue*) returns the sequence in the specified format FileFormatValue.

getgenpept(..., 'SequenceOnly', *SequenceOnlyValue*) returns only the sequence information without the metadata if *SequenceOnlyValue* is true. When the properties SequenceOnly and ToFile are used together, the output file is in the FASTA format.

Examples   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]
```

See Also   Bioinformatics Toolbox functions: `genpeptread`, `getembl`, `getgenbank`, `getpdb`

getgeodata

Purpose	Retrieve Gene Expression Omnibus (GEO) Sample (GSM) data

Syntax

```
Data = getgeodata('AccessionNumber')
getgeodata(..., 'PropertyName', PropertyValue,...)
getgeodata(..., 'ToFile', ToFileValue)
```

Arguments

AccessionNumber	Unique identifier for a sequence record. Enter a combination of letters and numbers.
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).

Description

Data = getgeodata('*AccessionNumber*') searches for the accession number in the Gene Expression Omnibus database and returns a MATLAB structure containing the following fields:

Field
Scope
Accession
Header
ColumnDescriptions
ColumnNames
Data

getgeodata(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

getgeodata(..., 'ToFile', *ToFileValue*) saves the data returned from the database to a file. Read a GenPept formatted file back into MATLAB using the function gensoftread.

2-299

getgeodata

For more information, see

```
http://www.ncbi.nlm.nih.gov/About/disclaimer.html
```

Examples   geoStruct = getgeodata('GSM1768')

See Also   Bioinformatics Toolbox functions: geosoftread, getgenbank, getgenpept

Purpose   Retrieve multiple sequence alignment associated with hidden Markov
                model (HMM) profile from PFAM database

Syntax   *AlignStruct* = gethmmalignment(*PFAMNumber*)
                AlignStruct = gethmmalignment(*PFAMAccessNumber*)
                AlignStruct = gethmmalignment(..., 'ToFile',
                ToFileValue, ...)
                AlignStruct = gethmmalignment(..., 'Type', *TypeValue*, ...)
                AlignStruct = gethmmalignment(..., 'Mirror', *MirrorValue*,
                   ...)
                AlignStruct = gethmmalignment(..., 'IgnoreGaps',
                IgnoreGaps,
                   ...)

Arguments

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.
PFAMAccessNumber	String specifying a protein family accession number of an HMM profile record in the PFAM database. For example, 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 Directory.
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.

gethmmalignment

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).

Return Values

AlignStruct	MATLAB structure containing the multiple sequence alignment associated with an HMM profile.

Description

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 containing the following fields:

Field
Header
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.

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 MATLAB 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://www.sanger.ac.uk/Software/Pfam/
http://pfam.janelia.org/
```

gethmmalignment

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   Bioinformatics Toolbox functions: fastaread, gethmmprof, gethmmtree, multialignread, pfamhmmread

Purpose	Retrieve hidden Markov model (HMM) profile from PFAM database
Syntax	*HMMStruct* = gethmmprof(*PFAMName*) *HMMStruct* = gethmmprof(*PFAMNumber*) *HMMStruct* = gethmmprof(*PFAMAccessNumber*) *HMMStruct* = gethmmprof(..., 'ToFile', *ToFileValue*, ...) *HMMStruct* = gethmmprof(..., 'Mode', *ModeValue*, ...) *HMMStruct* = gethmmprof(..., 'Mirror', *MirrorValue*, ...)

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 PF0002.
PFAMAccessNumber	String specifying a protein family accession number of an HMM profile record in the PFAM database. The string must include a version number appended at the end of the accession number. For example, PF00002.14.

> **Note** While this is the most efficient way to query the PFAM database, version numbers can change, making your input invalid.

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 Directory.

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

Return Values

HMMStruct	MATLAB structure containing information retrieved from the PFAM database.

Description *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, with the following fields:

Field
Name
PfamAccessionNumber
ModelDescription
ModelLength
Alphabet
MatchEmission
InsertEmission
NullEmission
BeginX

Field
MatchX
InsertX
DeleteX
FlankingInsertX
LoopX
NullX

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(*PFAMAccessNumber*) searches the PFAM database for the record represented by *PFAMAccessNumber*, a protein family accession number, retrieves the HMM profile information, and stores it in *HMMStruct*, a MATLAB structure.

Note While this is the most efficient way to query the PFAM database, version numbers can change, making your input invalid.

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*.

gethmmprof

> **Note** You can read an HMM-formatted file back into MATLAB 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://www.sanger.ac.uk/Software/Pfam/
http://pfam.janelia.org/
```

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   Bioinformatics Toolbox functions: gethmmalignment, hmmprofalign,
hmmprofstruct, pfamhmmread, showhmmprof

gethmmtree

Purpose   Retrieve phylogenetic tree data from PFAM database

Syntax
Tree = gethmmtree(*AccessionNumber*)
Tree = gethmmtree(*AccessionNumber*, ...'ToFile',
ToFileValue, ...)
Tree = gethmmtree(*AccessionNumber*, ...'Type',
TypeValue, ...)

Arguments

AccessionNumber	Accession number in the PFAM database.
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.

Description   *Tree* = gethmmtree(*AccessionNumber*) searches for the PFAM family accession number in the PFAM database and returns an object, Tree, containing a phylogenetic tree representative of the protein family.

Tree = gethmmtree(*AccessionNumber*, ...'*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(*AccessionNumber*, ...'ToFile', *ToFileValue*, ...) saves the data returned from the PFAM database in the file *ToFileValue*.

Tree = gethmmtree(*AccessionNumber*, ...'Type', *TypeValue*, ...) specifies which alignments to include int he 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.

Examples   Retrieve a 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(2, 'type', 'seed')
tree  = gethmmtree('PF00002', 'type', 'seed')
```

See Also   Bioinformatics Toolbox functions: gethmmalignment, phytreeread

getpdb

Purpose   Retrieve protein structure data from Protein Data Bank (PDB) database

Syntax
PDBStruct = getpdb(*PDBid*)
PDBStruct = getpdb(*PDBid*, ...'ToFile', *ToFileValue*, ...)
PDBStruct = getpdb(*PDBid*, ...'SequenceOnly',
    SequenceOnlyValue, ...)

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 Directory.

> **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 MATLAB 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.

Return Values	*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

PDB Database Record	Field in the MATLAB Structure
JRNL	Journal
REMARK 1	Remark1
REMARK *N*	Remark*n*
Note *N* equals 2 through 999.	**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

PDB Database Record	Field in the MATLAB Structure
CRYST1	Cryst1
ORIGXn	OriginX
SCALEn	Scale
MTRIXn	Matrix
TVECT	TranslationVector
MODEL	Model
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 MATLAB
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`.

getpdb

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 Directory.

```
pdbstruct = getpdb('5CYT', 'ToFile', 'electron_transport.pdb')
```

See Also   Bioinformatics Toolbox functions: getembl, getgenbank, getgenpept, molviewer, pdbdistplot, pdbread, pdbwrite

Purpose   Annotations from Gene Ontology annotated file

Syntax   *Annotation* = goannotread(*File*)

Arguments
<table>
<tr><td>*File*</td><td>String specifying a file name of a Gene Ontology annotated file.</td></tr>
</table>

Description *Annotation* = goannotread(*File*) converts the contents of *File*, a Gene Ontology annotated file into *Annotation*, an array of structs. 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

Examples

1 Open a Web browser to

    http://www.geneontology.org/GO.current.annotations.shtml

2 Download the file containing GO annotations for the gene products of *Saccharomyces cerevisiae* (gene_association.sgd.gz) to your MATLAB Current Directory.

3 Uncompress the file using the gunzip function.

```
gunzip('gene_association.sgd.gz')
```

4 Read the file into MATLAB.

```
SGDGenes = goannotread('gene_association.sgd');
```

5 Create a structure with GO annotations and get a list of genes.

```
S = struct2cell(SGDGenes);
genes = S(3,:)'
```

goannotread

See Also   Bioinformatics Toolbox

- functions — geneont (object constructor), num2goid

- geneont object methods — getancestors, getdescendants, getmatrix, getrelatives

Purpose   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   Bioinformatics Toolbox functions: blosum, dayhoff, pam

gprread

Purpose	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	Property to control creating column names that MATLAB can use as variable names.

Description   *GPRData* = gprread('*File*') reads GenePix results data from *File* and creates a MATLAB structure (*GPRData*) with the following fields:

Field
Header
Data
Blocks
Columns
Rows
Names
IDs
ColumnNames
Indices
Shape

gprread(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

gprread(..., 'CleanColNames', *CleanColNamesValue*). A GPR file may contain column names with spaces and some characters

that MATLAB 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 MATLAB indices that can be used for plotting heat maps of the data.

For more details on the GPR format, see

```
http://www.moleculardevices.com/pages/software/gn_genepix_file_formats.html#gpr
```

```
http://www.moleculardevices.com/pages/software/gn_gpr_format_history.html
```

For a list of supported file format versions, see

```
http://www.moleculardevices.com/pages/software/gn_genepix_file_formats.html
```

GenePix is a registered trademark of Molecular Devices Corporation.

Examples

```
% Read in a sample GPR file and plot the median foreground
% intensity for the 635 nm channel.
gprStruct = gprread('mouse_a1pd.gpr')
maimage(gprStruct,'F635 Median');

% Alternatively you can create a similar plot using
% more basic graphics commands.
F635Median = magetfield(gprStruct,'F635 Median');
imagesc(F635Median(gprStruct.Indices));
colormap bone
colorbar;
```

See Also

Bioinformatics Toolbox functions: affyread, agferead, celintensityread, galread, geosoftread, imageneread, magetfield, sptread

graphallshortestpaths

Purpose   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" in the *Bioinformatics Toolbox User's Guide*.

[*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 node S to node T. A 0 in this matrix indicates the source node; an Inf is an unreachable node. The *pred* output is the predecessor map of the winning paths.

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'))
```
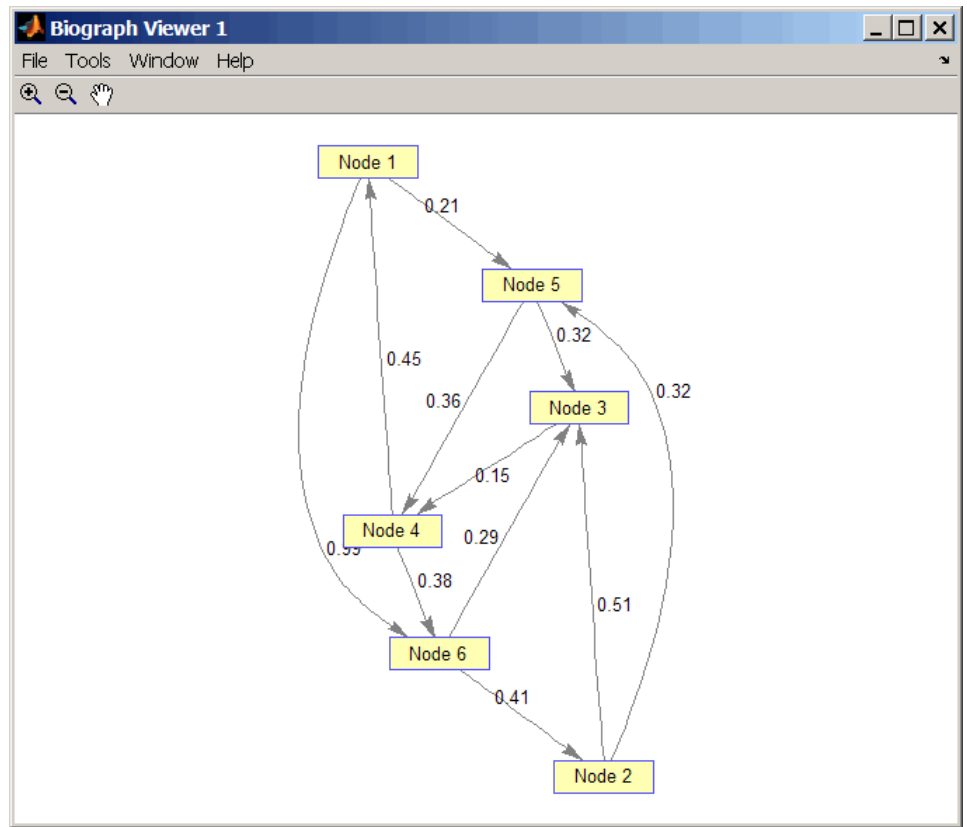
2 Find all the shortest paths between every pair of nodes in the directed graph.

```
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'))
```
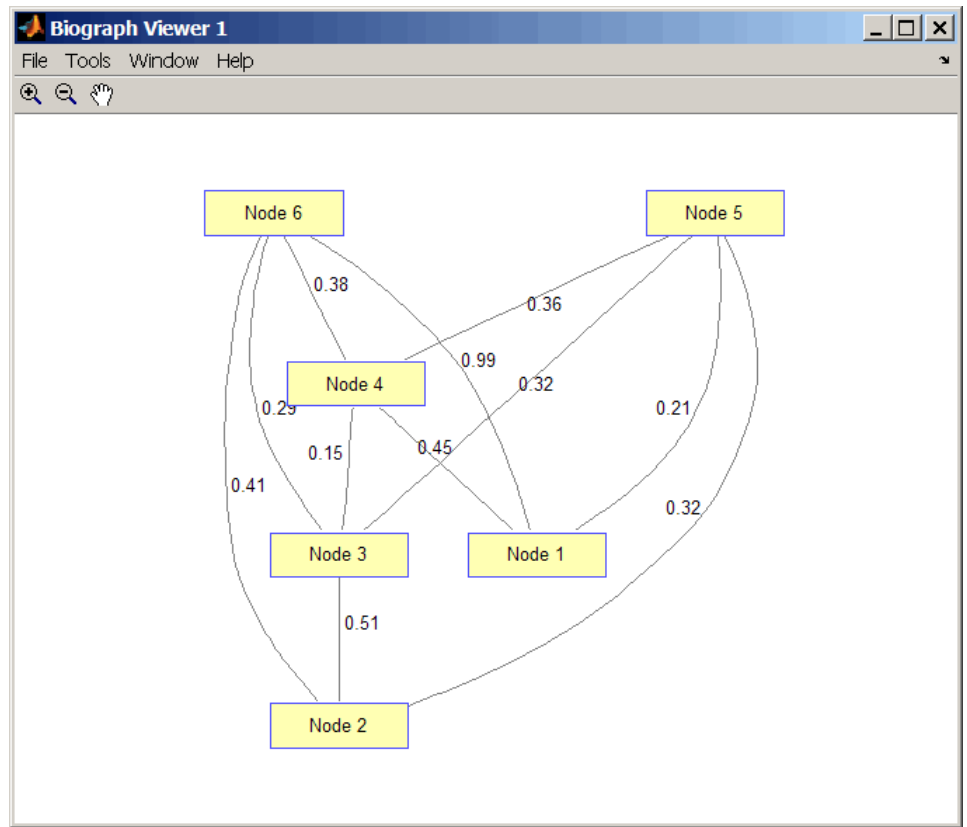
2 Find all the shortest paths between every pair of nodes in the undirected graph.

```
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   Bioinformatics Toolbox functions: graphconncomp, graphisdag, graphisomorphism, graphisspantree, graphmaxflow, graphminspantree, graphpred2path, graphshortestpath, graphtopoorder, graphtraverse

Bioinformatics Toolbox method of biograph object: allshortestpaths

Purpose	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.

graphcluster

Description

> **Tip** For introductory information on graph theory functions, see "Graph Theory Functions" in the Bioinformatics Toolbox documentation.

[*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
```

graphcluster

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   Bioinformatics Toolbox functions: graphallshortestpaths, graphisdag, graphisomorphism, graphisspantree, graphmaxflow, graphminspantree, graphpred2path, graphshortestpath, graphtopoorder, graphtraverse

Bioinformatics Toolbox method of biograph object: conncomp

graphconncomp

Purpose
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" in the *Bioinformatics Toolbox User's Guide*.

[*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).

graphconncomp

See Also Bioinformatics Toolbox functions: `graphallshortestpaths`, `graphisdag`, `graphisomorphism`, `graphisspantree`, `graphmaxflow`, `graphminspantree`, `graphpred2path`, `graphshortestpath`, `graphtopoorder`, `graphtraverse`

Bioinformatics Toolbox method of `biograph object`: `conncomp`

Purpose   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" in the *Bioinformatics Toolbox User's Guide*.

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))
```
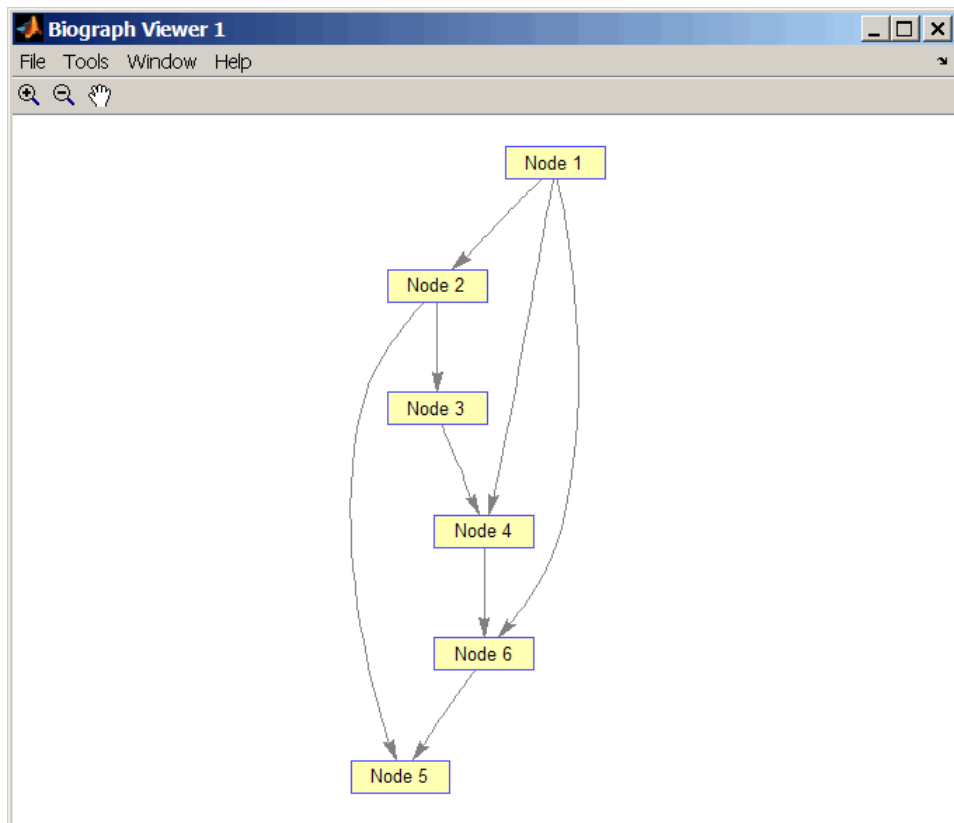

2 Test for cycles in the DAG.

```
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))
```
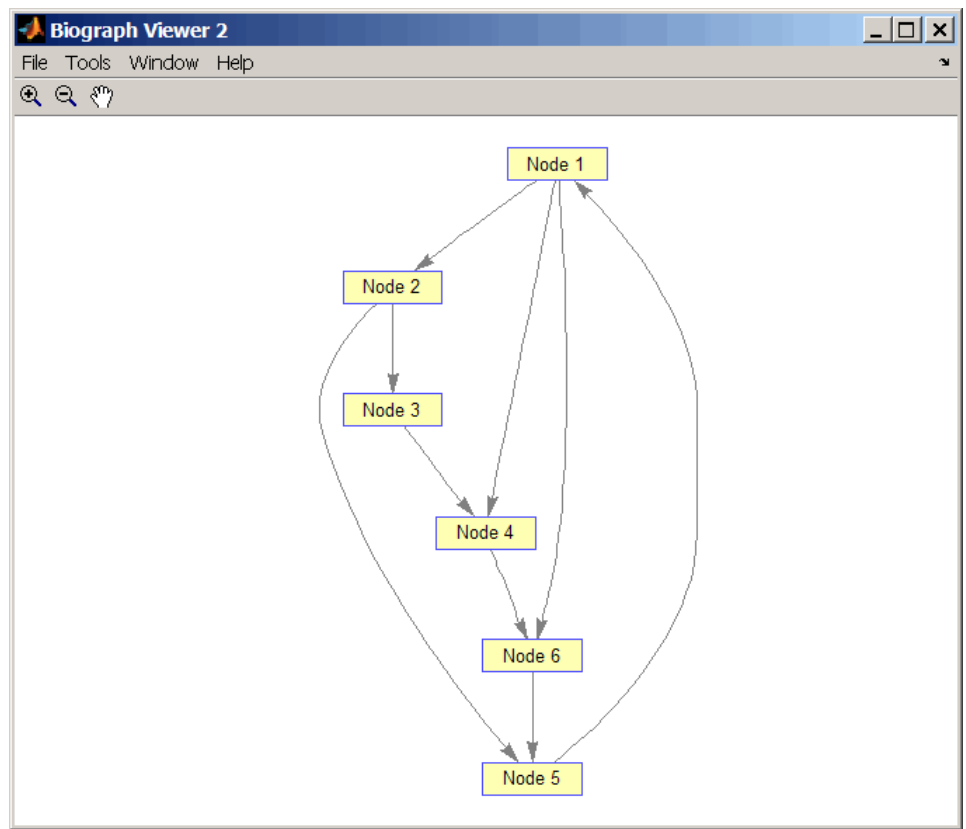
4 Test for cycles in the new graph.

```
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   Bioinformatics Toolbox functions: graphallshortestpaths, graphconncomp, graphisomorphism, graphisspantree, graphmaxflow,

graphisdag

graphminspantree, graphpred2path, graphshortestpath, graphtopoorder, graphtraverse

Bioinformatics Toolbox method of `biograph` object: isdag

Purpose   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" in the *Bioinformatics Toolbox User's Guide*.

[*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*. When *Isomorphic* is false, the worst-case time complexity is O(N!), where N is the number of nodes.

graphisomorphism

[*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'))
```
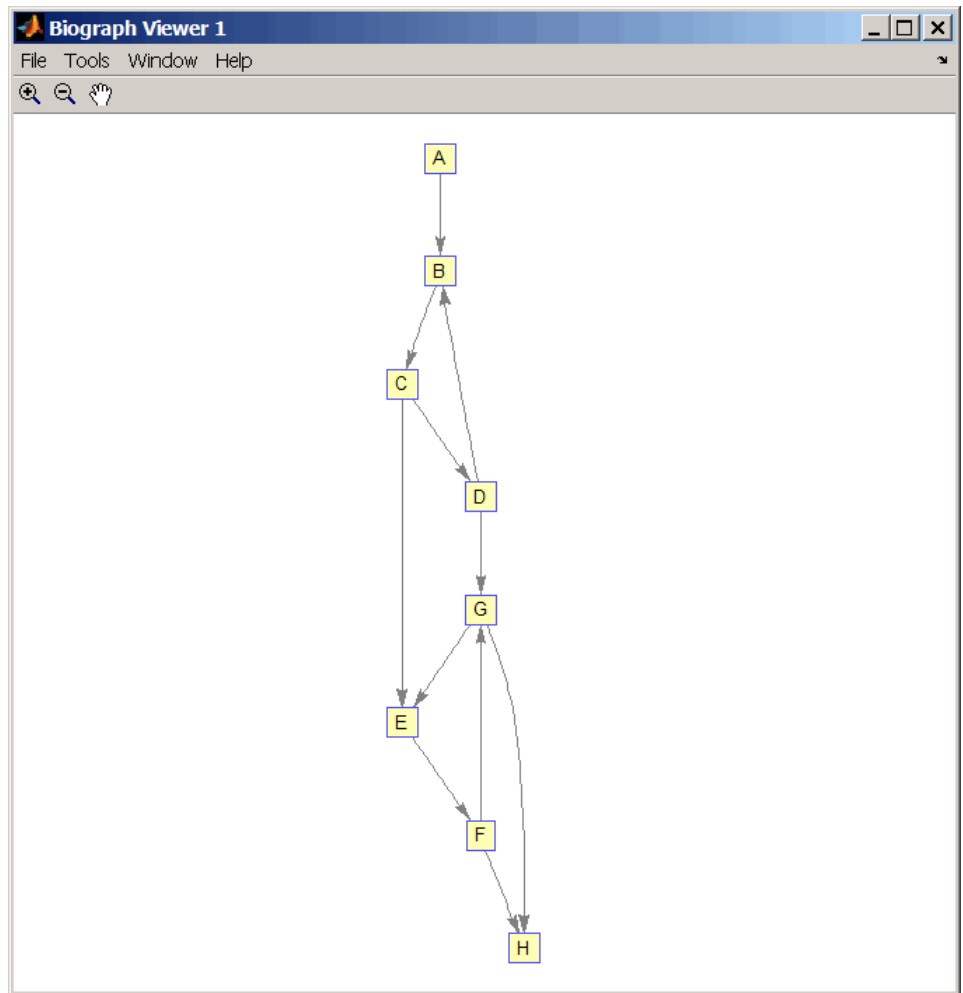
2 Set a random permutation vector and then create and view a new permuted graph.

```
p = randperm(8)
```

```
p =

       7     8     2     3     6     4     1     5

g2 = g1(p,p);
view(biograph(g2,'12345678'))
```
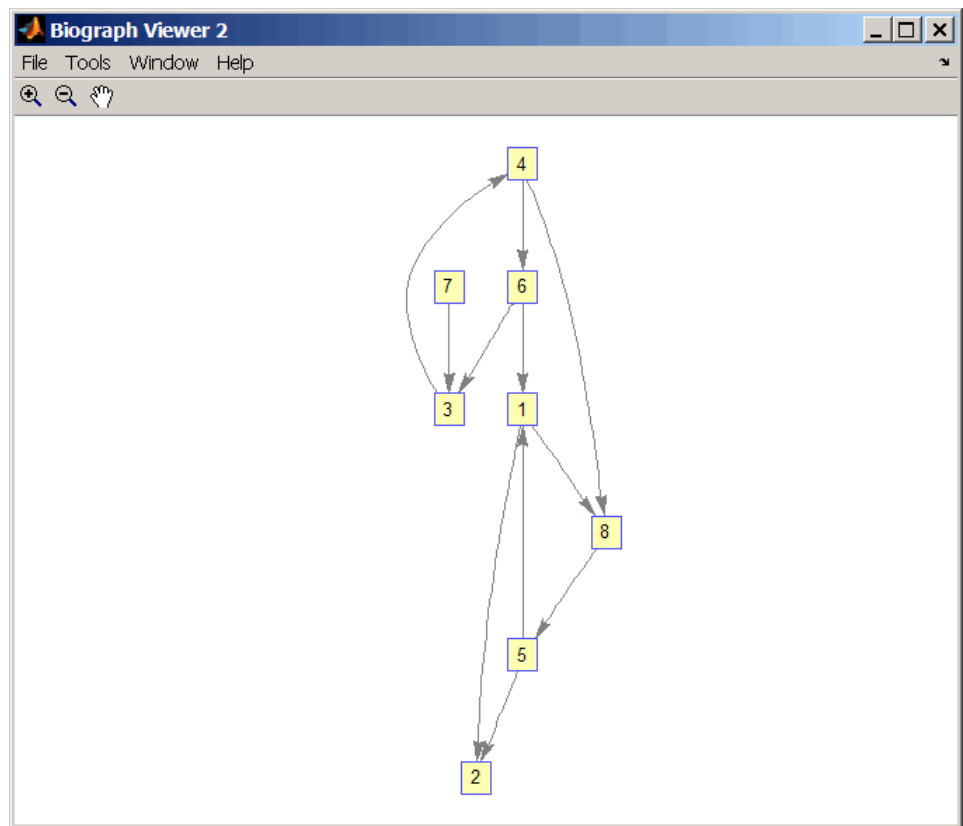

3 Check if the two graphs are isomorphic.

```
[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'))
```

graphisomorphism
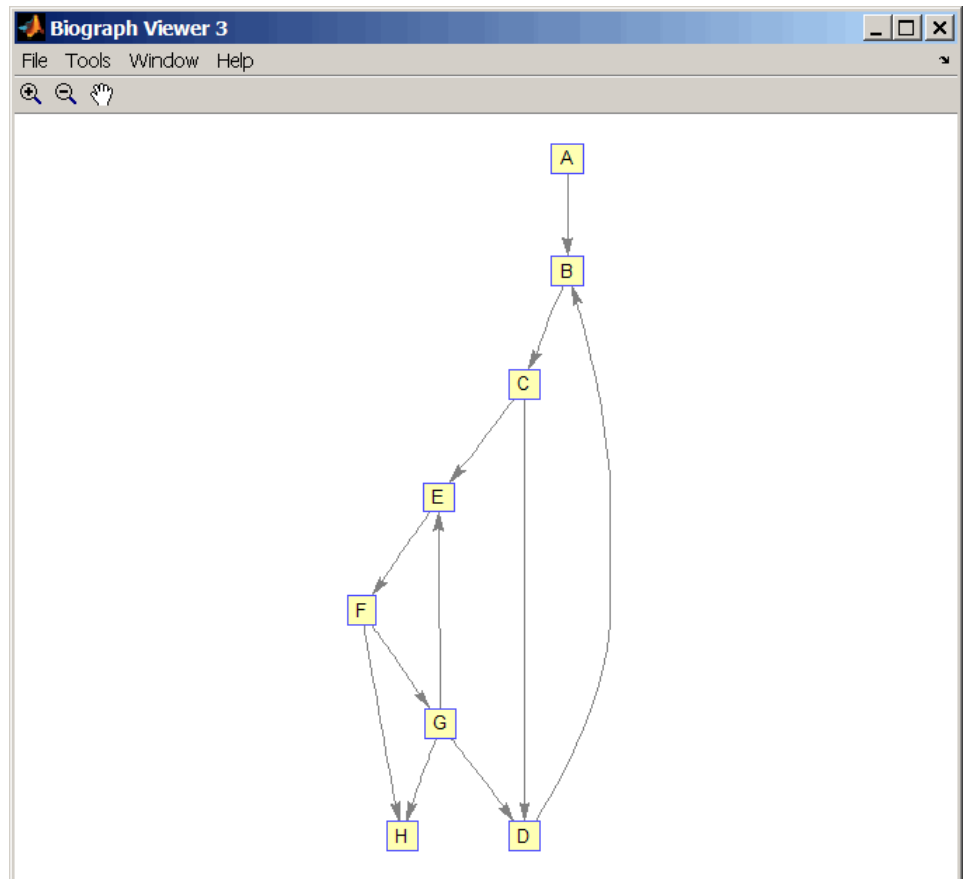

```
[F,M] = graphisomorphism(g2,g1)

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 Bioinformatics Toolbox functions: graphallshortestpaths,
graphconncomp, graphisdag, graphisspantree, graphmaxflow,
graphminspantree, graphpred2path, graphshortestpath,
graphtopoorder, graphtraverse

Bioinformatics Toolbox methods of biograph object: isomorphism

graphisspantree

Purpose   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" in the *Bioinformatics Toolbox User's Guide*.

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   Bioinformatics Toolbox functions: `graphallshortestpaths`, `graphconncomp`, `graphisdag`, `graphisomorphism`, `graphmaxflow`, `graphminspantree`, `graphpred2path`, `graphshortestpath`, `graphtopoorder`, `graphtraverse`

Bioinformatics Toolbox methods of `biograph object`: `isspantree`

graphmaxflow

Purpose   Calculate maximum flow and minimum cut 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

Tip For introductory information on graph theory functions, see "Graph Theory Functions" in the *Bioinformatics Toolbox User's Guide*.

[*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.

[...] = 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*

graphmaxflow

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'))
```
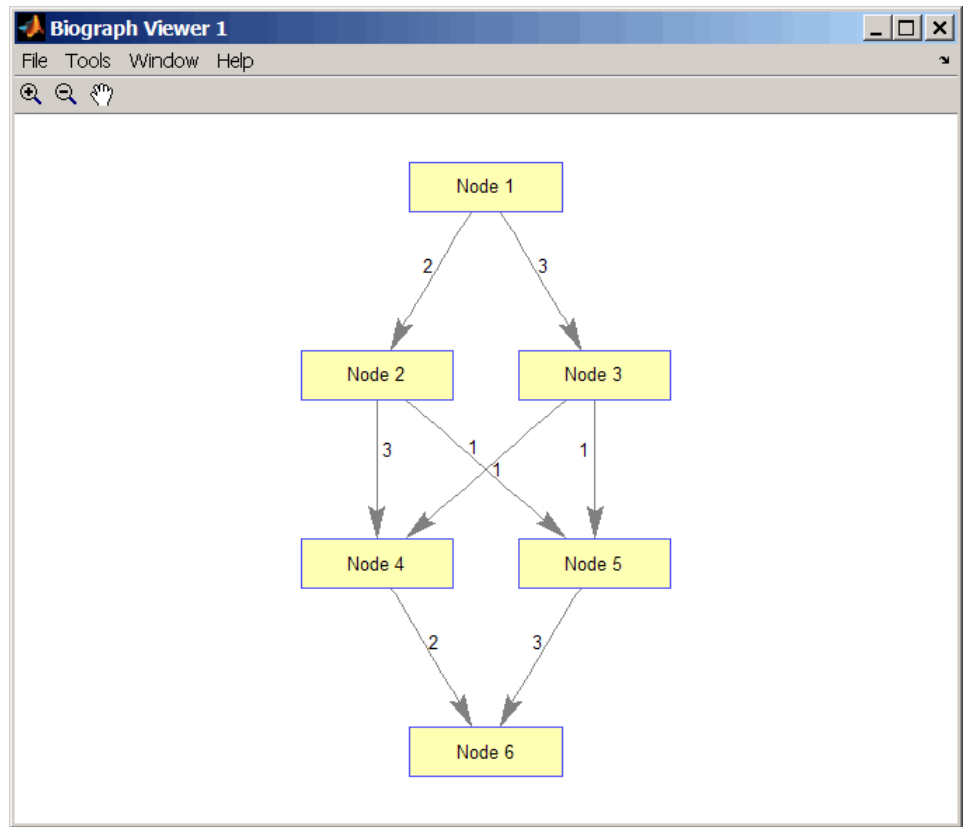
4 View the graph with the calculated maximum flows.

```
view(biograph(F,[],'ShowWeights','on'))
```
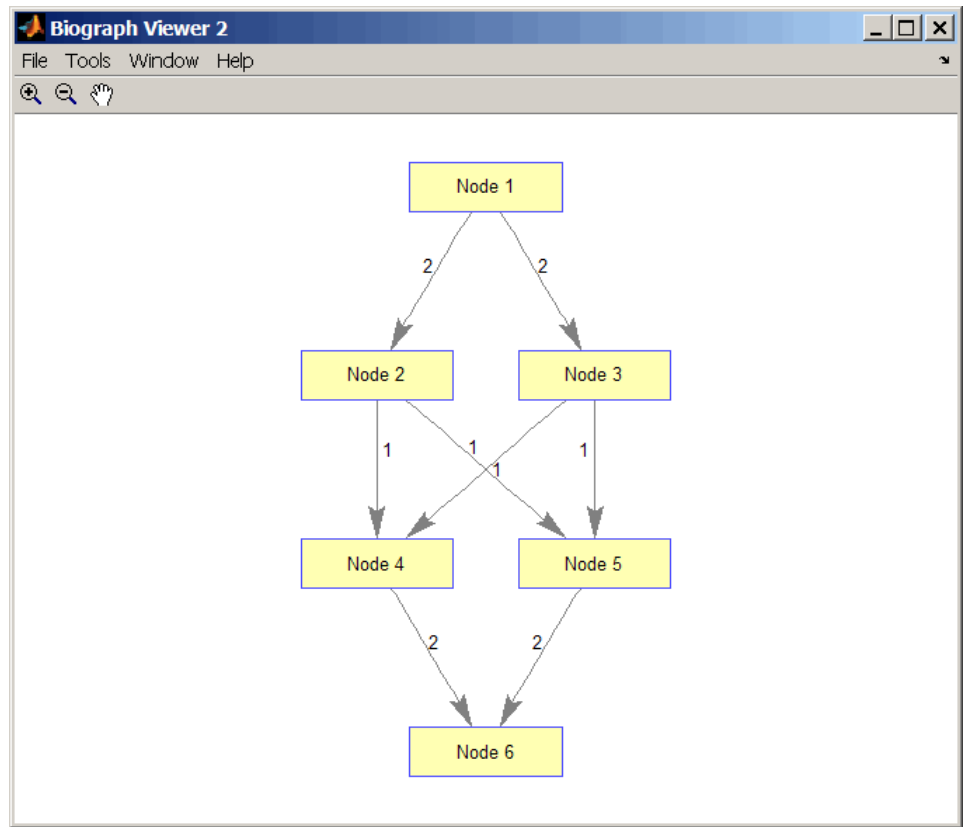
5 Show one solution to the minimum cut problem in the original graph.

```
set(h.Nodes(K(1,:)),'Color',[1 0 0])
```
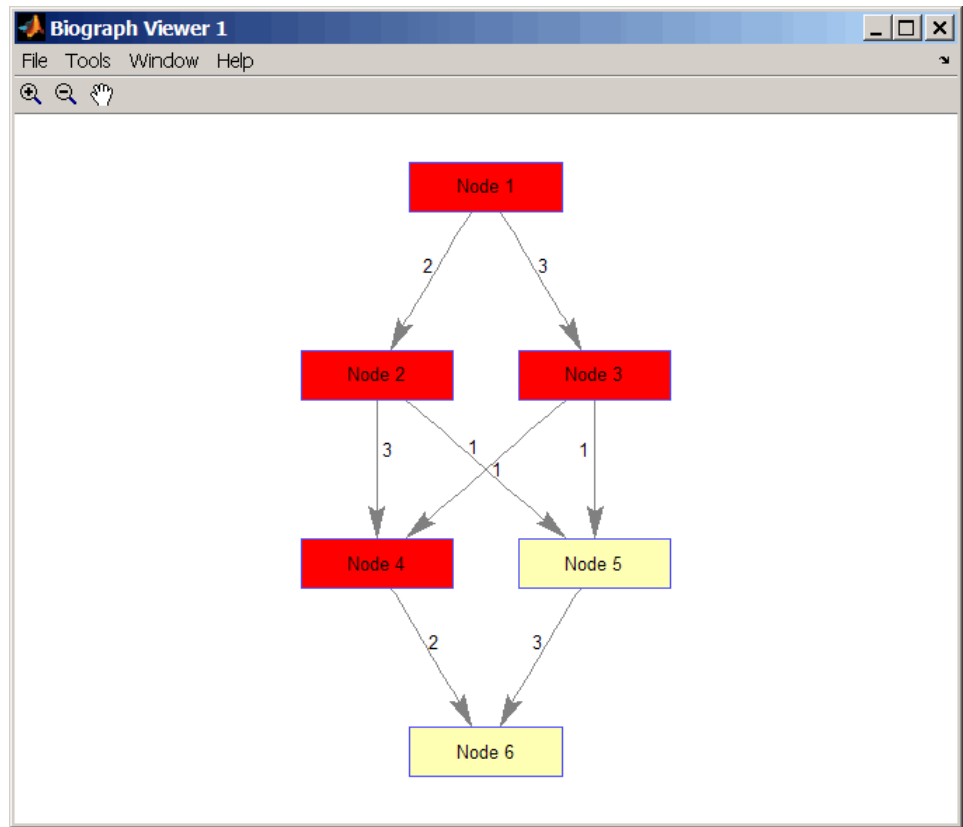
Notice that in the three edges that connect the source nodes (red) to the destination nodes (yellow), the original capacities and the calculated maximum flows are the same.

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   Bioinformatics Toolbox functions: `graphallshortestpaths`, `graphconncomp`, `graphisdag`, `graphisomorphism`, `graphisspantree`, `graphminspantree`, `graphpred2path`, `graphshortestpath`, `graphtopoorder`, `graphtraverse`

Bioinformatics Toolbox method of `biograph` object: `maxflow`

graphminspantree

Purpose   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" in the *Bioinformatics Toolbox User's Guide*.

[*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'))
```
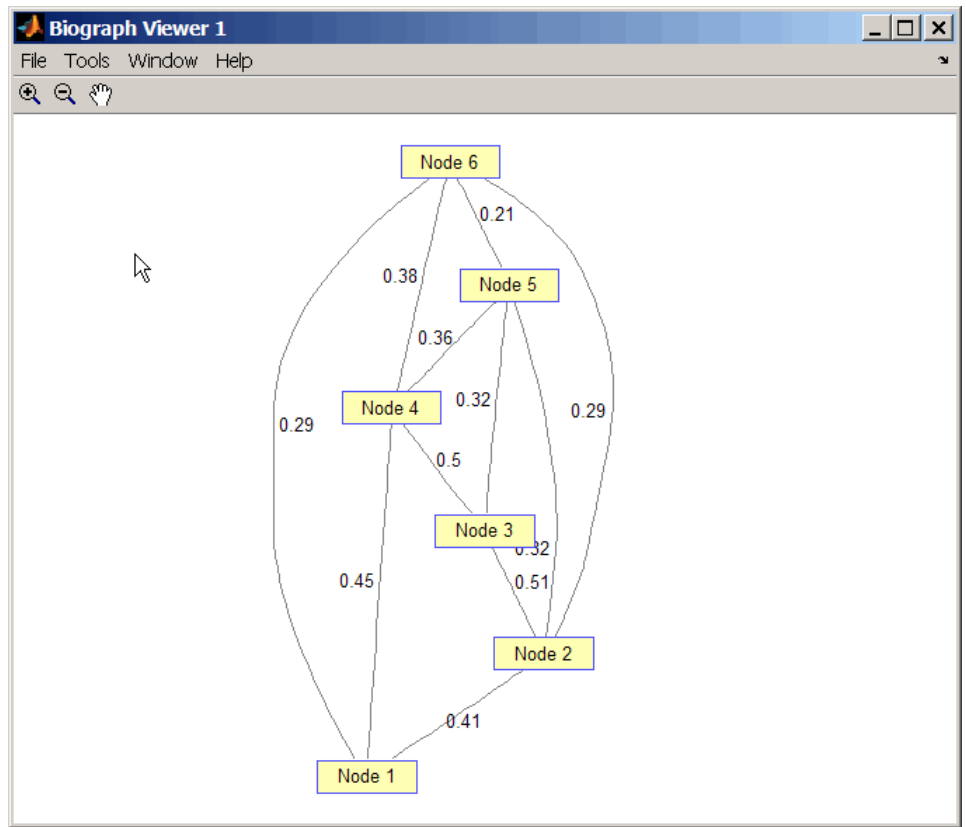
2 Find and view the minimal spanning tree of the undirected graph.

```
[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'))
```
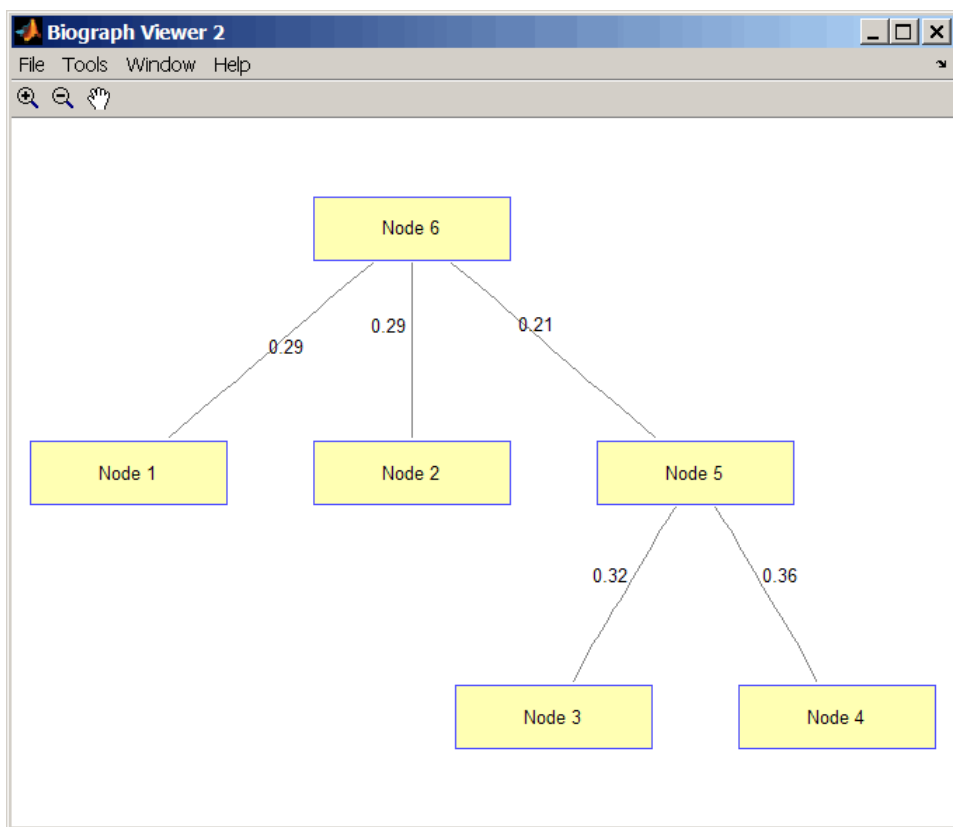
References   [1] Kruskal, J.B. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society *7*, 48-50.

[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   Bioinformatics Toolbox functions: `graphallshortestpaths`, `graphconncomp`, `graphisdag`, `graphisomorphism`, `graphisspantree`, `graphmaxflow`, `graphpred2path`, `graphshortestpath`, `graphtopoorder`, `graphtraverse`

Bioinformatics Toolbox method of `biograph` object: `minspantree`

graphpred2path

Purpose   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" in the *Bioinformatics Toolbox User's Guide*.

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)
```
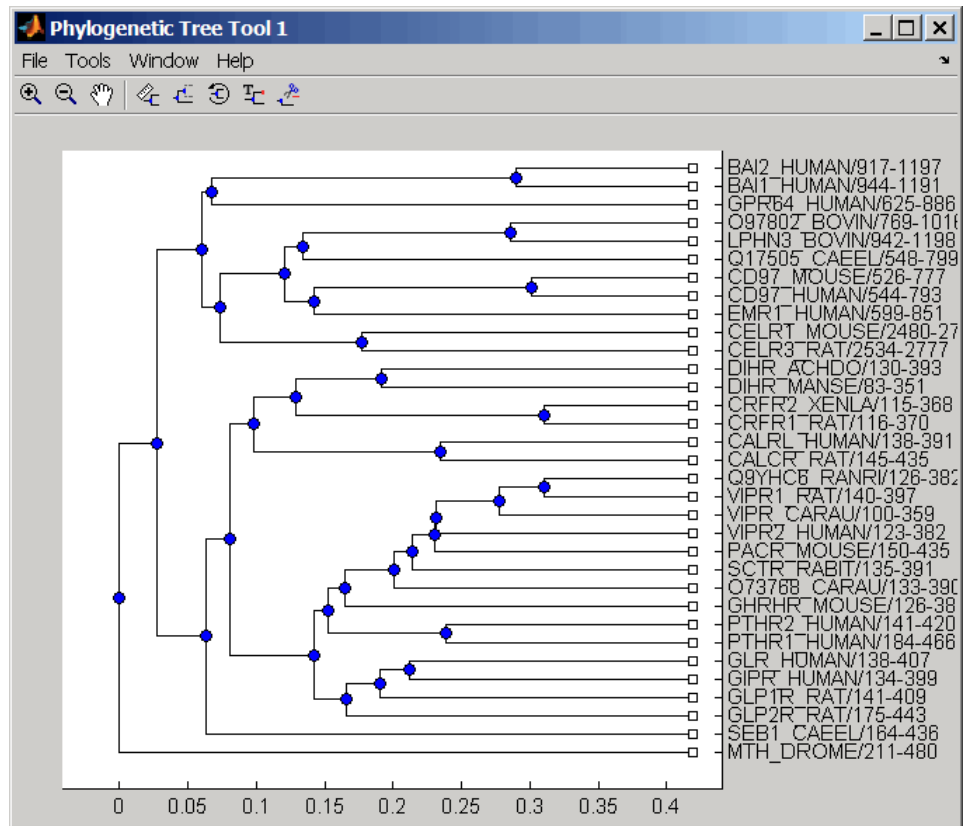
3 From the phytree object, create a connection matrix to represent the phylogenetic tree.

```
[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 = strmatch('GLR',labels)

glr_loc =

              28

[T,PRED]=graphminspantree(CM,root_loc);
PATH = graphpred2path(PRED,glr_loc)

PATH =

     65    64    53    52    46    45    44    43    28
```

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   Bioinformatics Toolbox functions: graphallshortestpaths,
                graphconncomp, graphisdag, graphisomorphism, graphisspantree,
                graphmaxflow, graphminspantree, graphshortestpath,
                graphtopoorder, graphtraverse

graphshortestpath

Purpose	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*.

graphshortestpath

Description

> **Tip** For introductory information on graph theory functions, see "Graph Theory Functions" in the *Bioinformatics Toolbox User's Guide*.

[*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 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*] = 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.
```
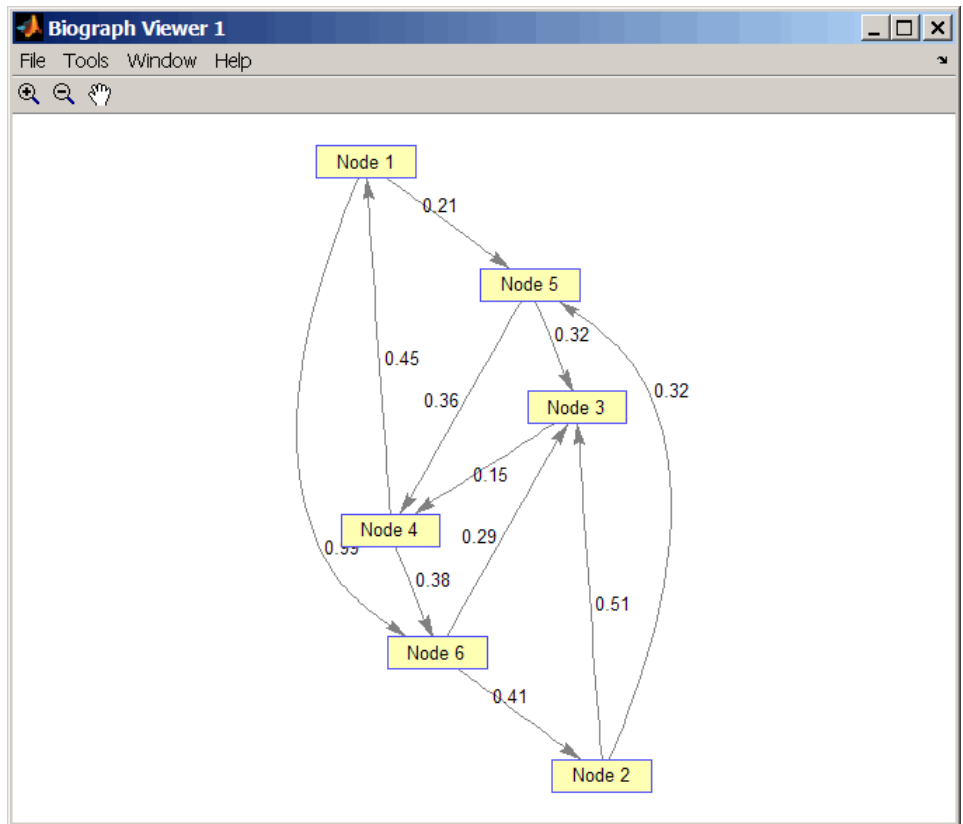

2 Find the shortest path in the graph from node 1 to node 6.

```
[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)
```
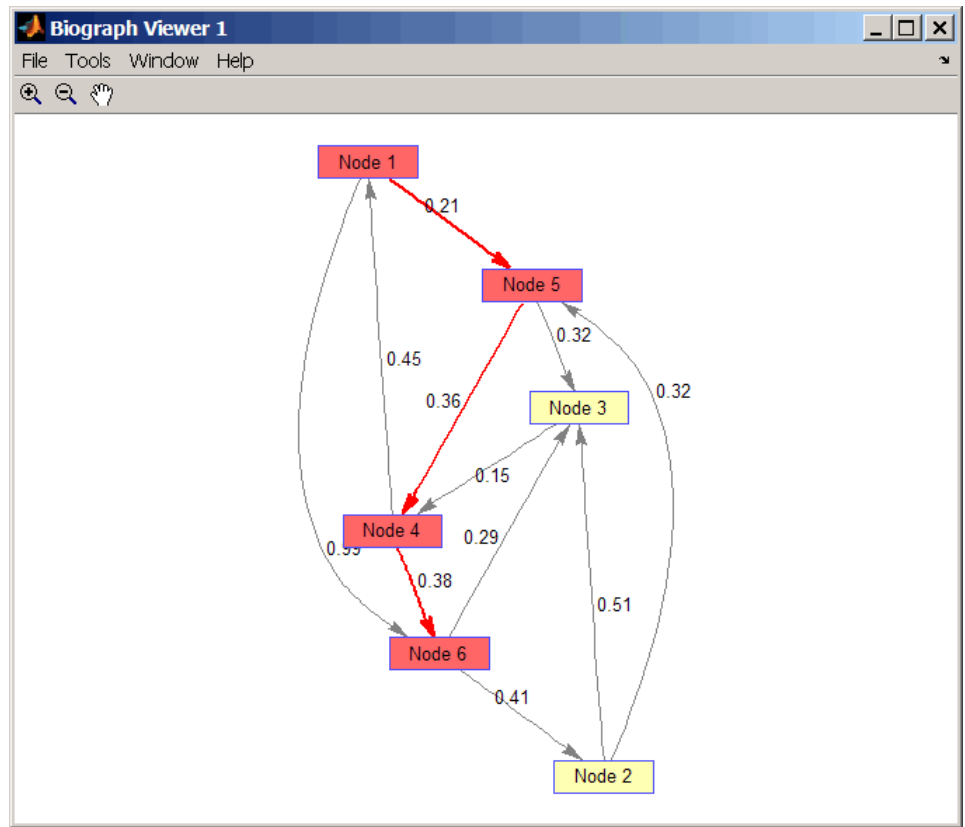
Finding the Shortest Path in an Undirected Graph

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.
```

graphshortestpath
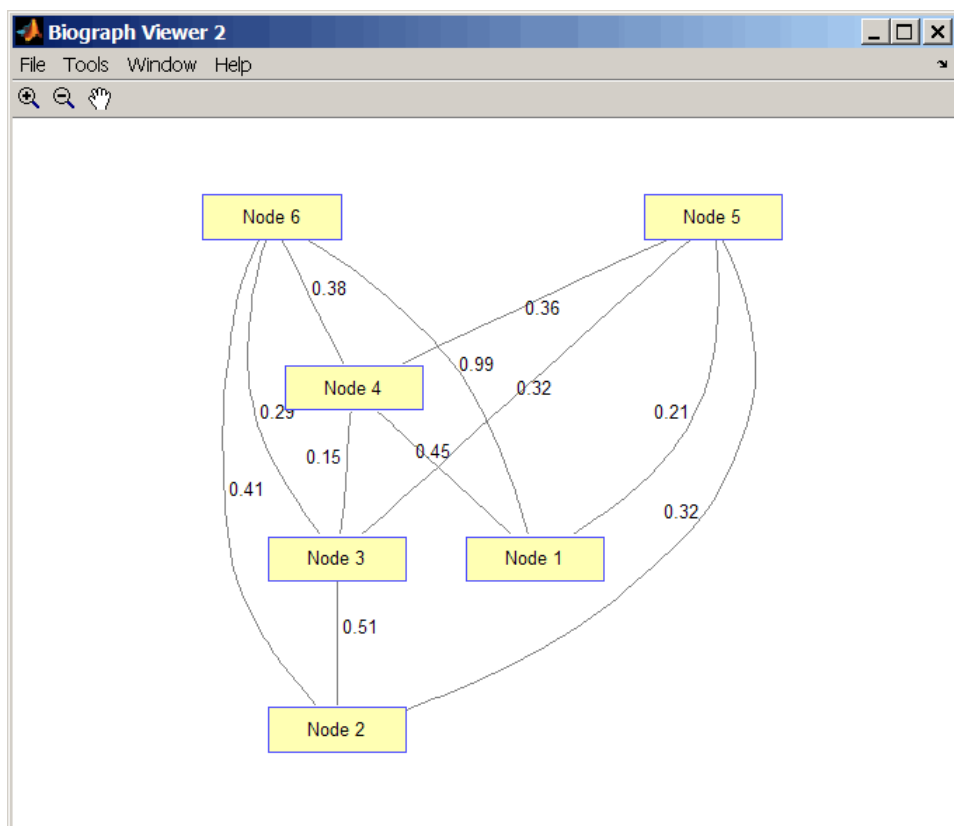
2 Find the shortest path in the graph from node 1 to node 6.

```
[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)
```
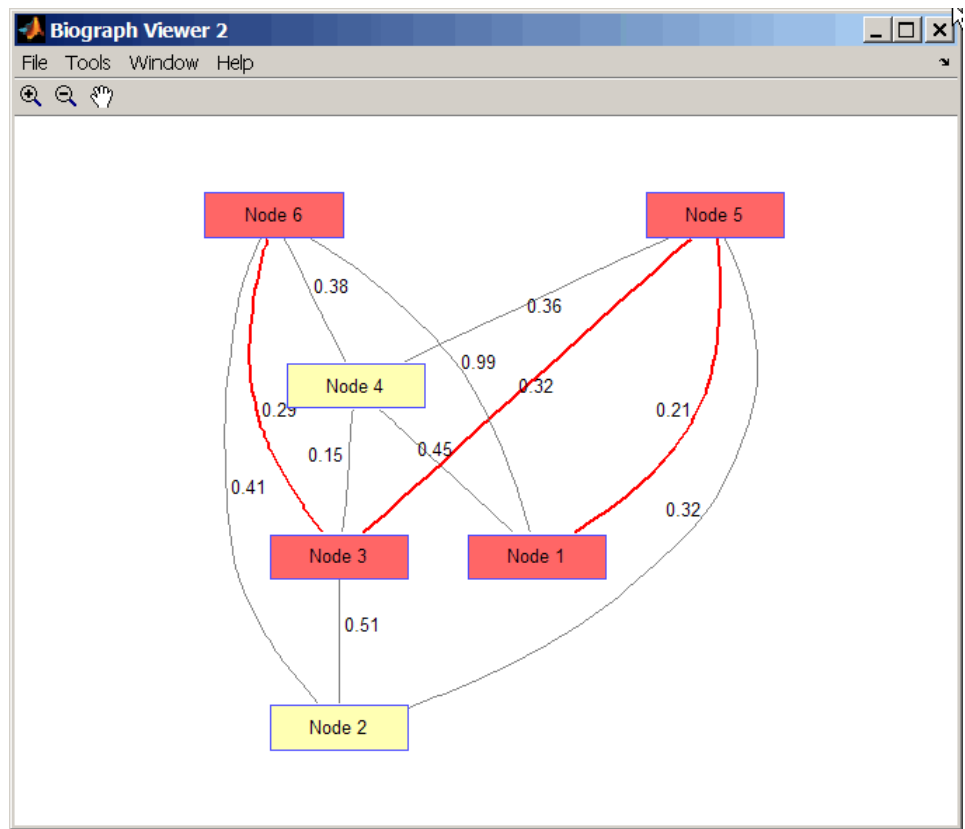
References   [1] Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik *1*, 269-271.

[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   Bioinformatics Toolbox functions: `graphallshortestpaths`, `graphconncomp`, `graphisdag`, `graphisomorphism`, `graphisspantree`, `graphmaxflow`, `graphminspantree`, `graphpred2path`, `graphtopoorder`, `graphtraverse`

Bioinformatics Toolbox method of `biograph` object: `shortestpath`

graphtopoorder

Purpose	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" in the *Bioinformatics Toolbox User's Guide*.

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))
```
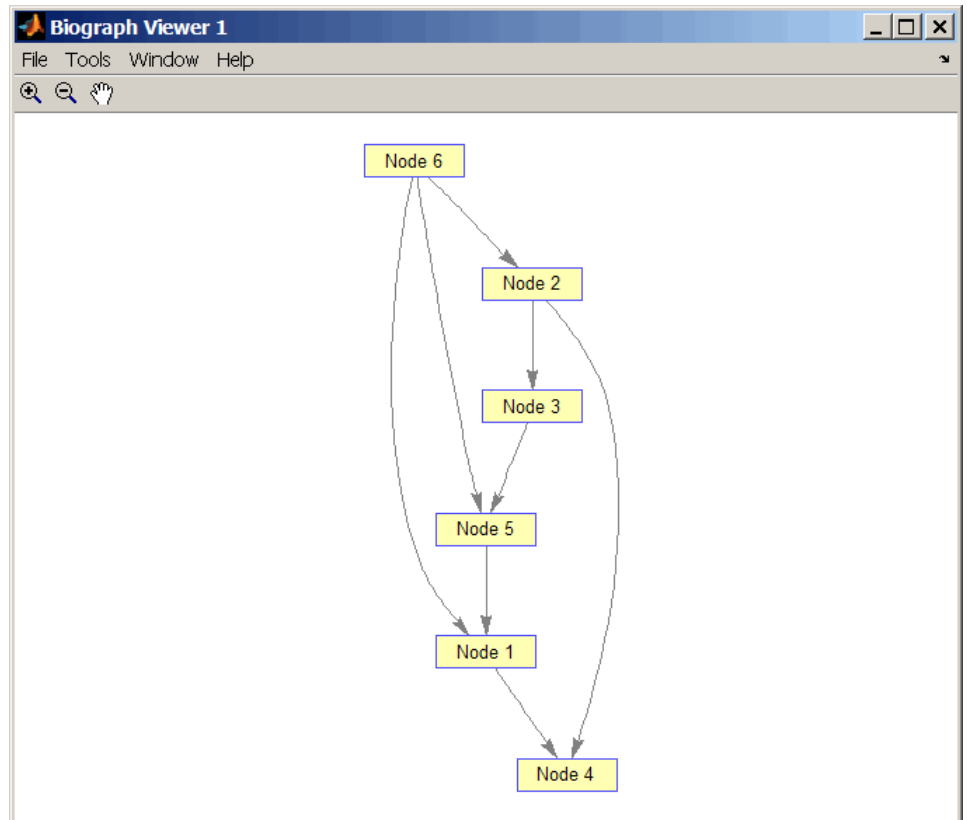

2 Find the topological order of the DAG.

```
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))
```
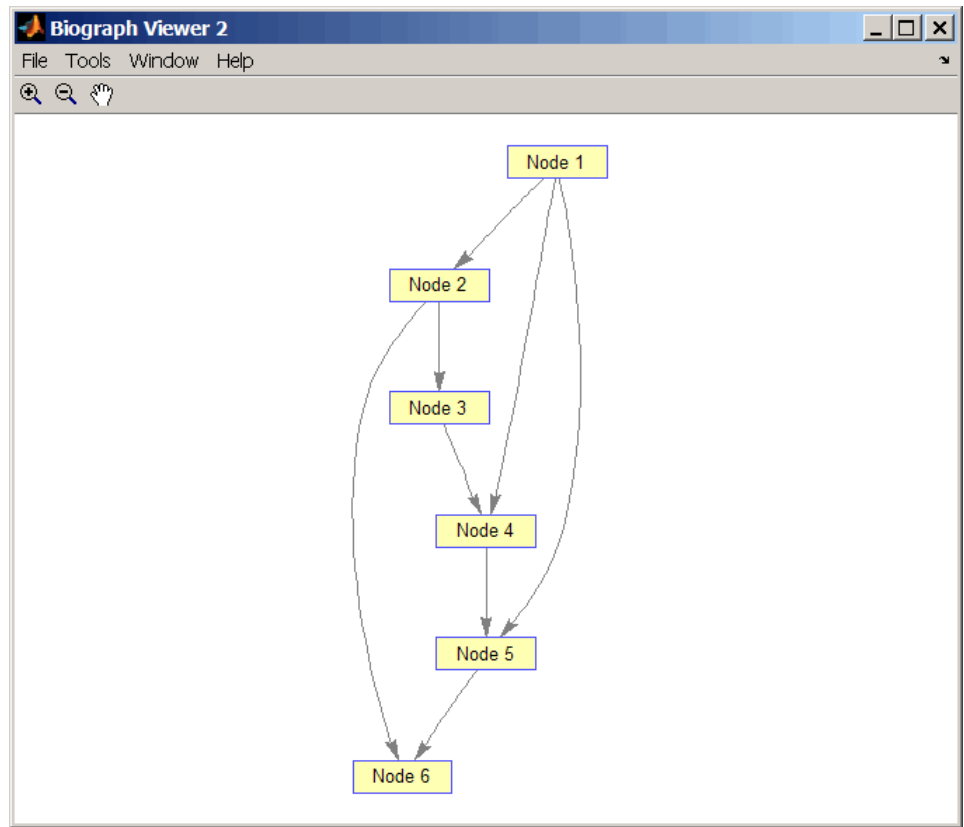
References   [1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also   Bioinformatics Toolbox functions: graphallshortestpaths, graphconncomp, graphisdag, graphisomorphism, graphisspantree, graphmaxflow, graphminspantree, graphpred2path, graphshortestpath, graphtraverse

Bioinformatics Toolbox method of biograph object: topoorder

graphtraverse

Purpose	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" in the *Bioinformatics Toolbox User's Guide*.

[*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.
```
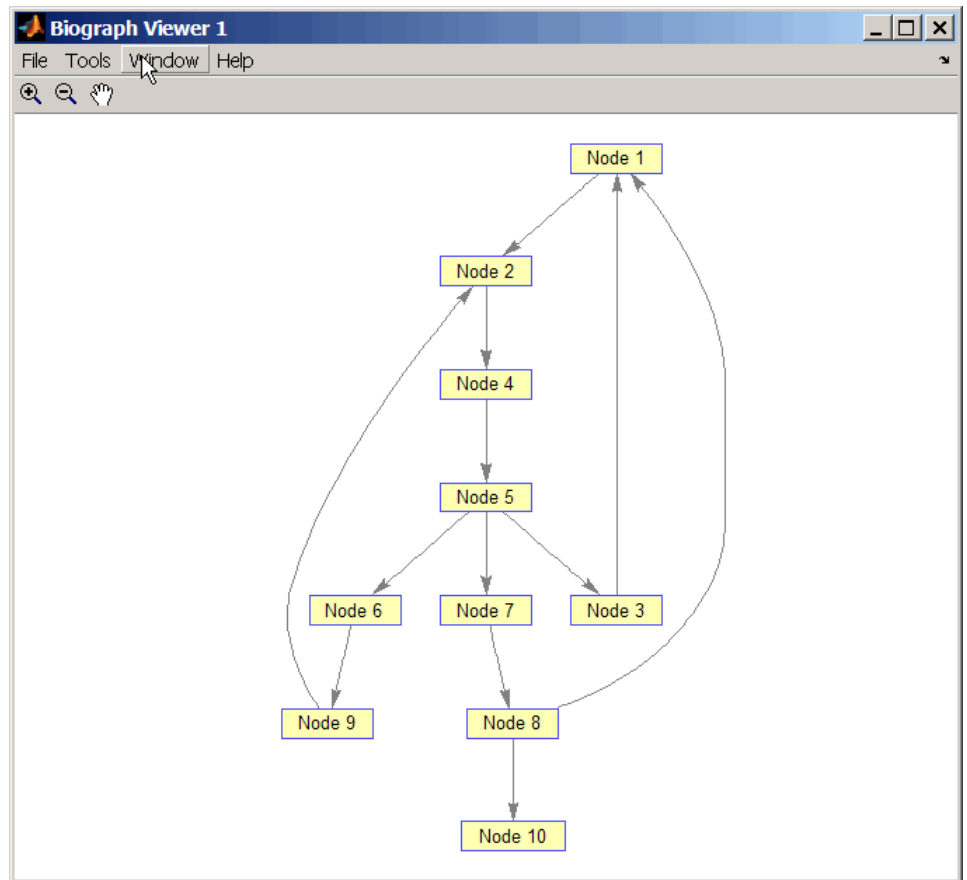
2 Traverse the graph to find the depth-first search (DFS) discovery order starting at node 4.

```
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)
```
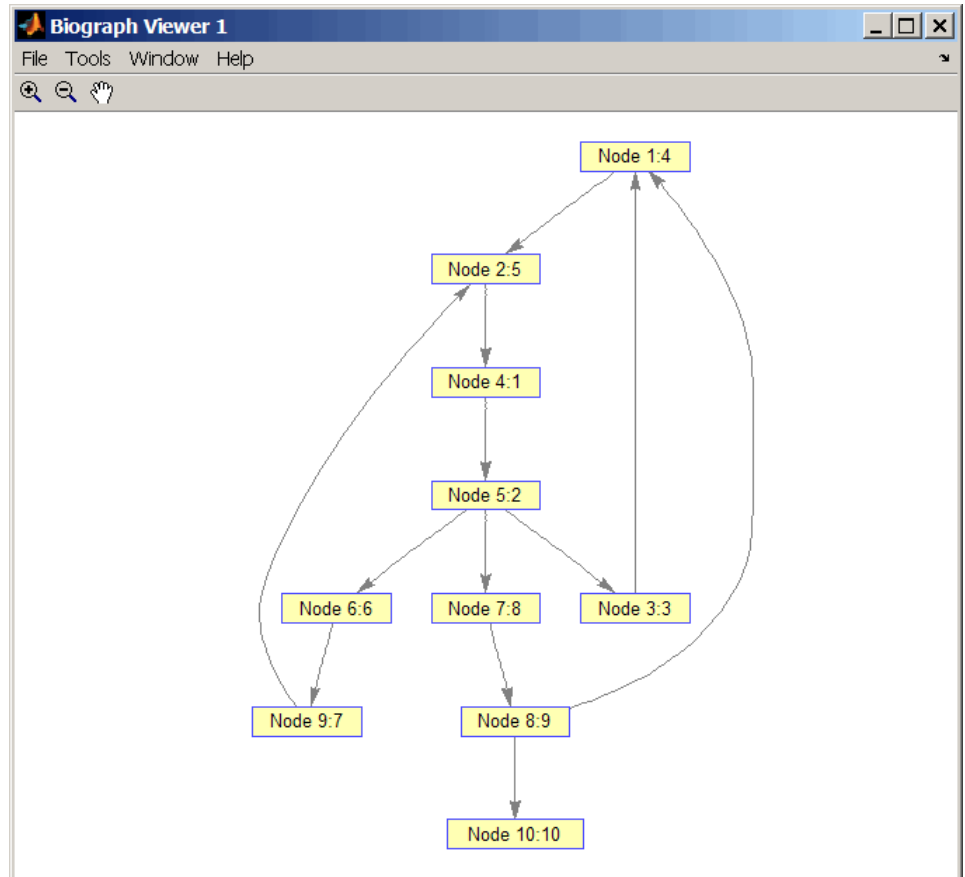
4 Traverse the graph to find the breadth-first search (BFS) discovery order starting at node 4.

```
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)
```
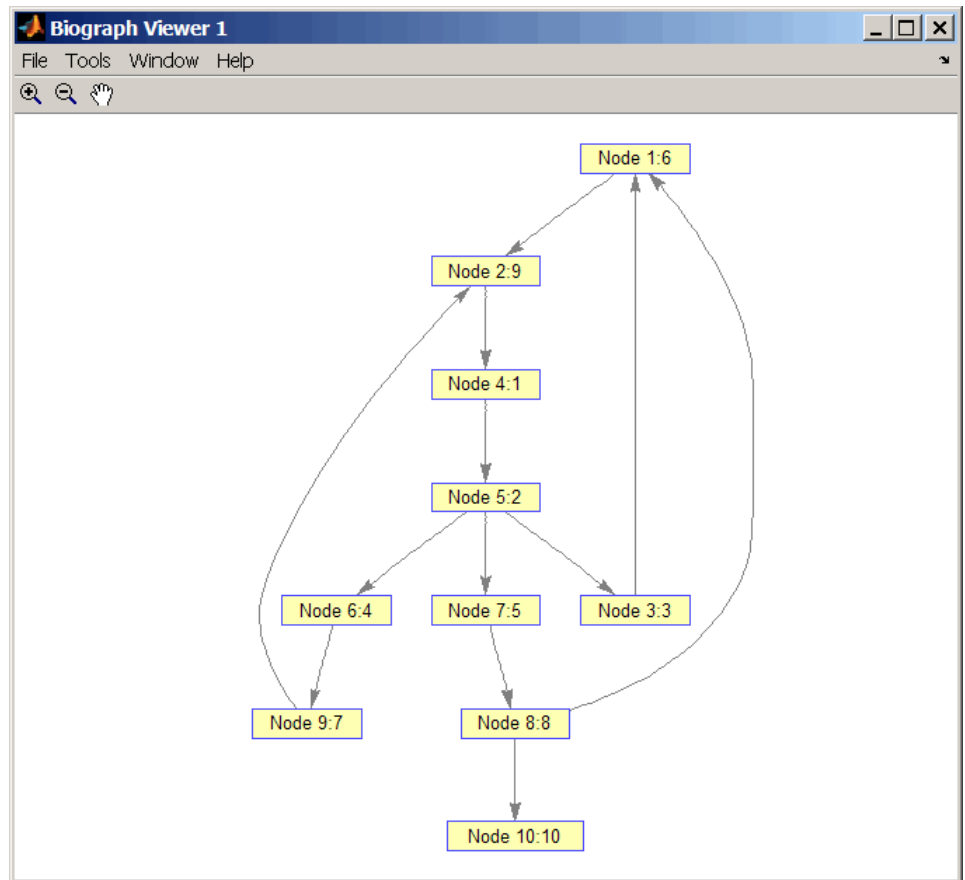
6 Find and color nodes that are close to (within two edges of) node 4.

```
node_idxs = graphtraverse(DG,4,'depth',2)

node_idxs =

     4     5     3     6     7

set(h.nodes(node_idxs),'Color',[1 0 0])
```
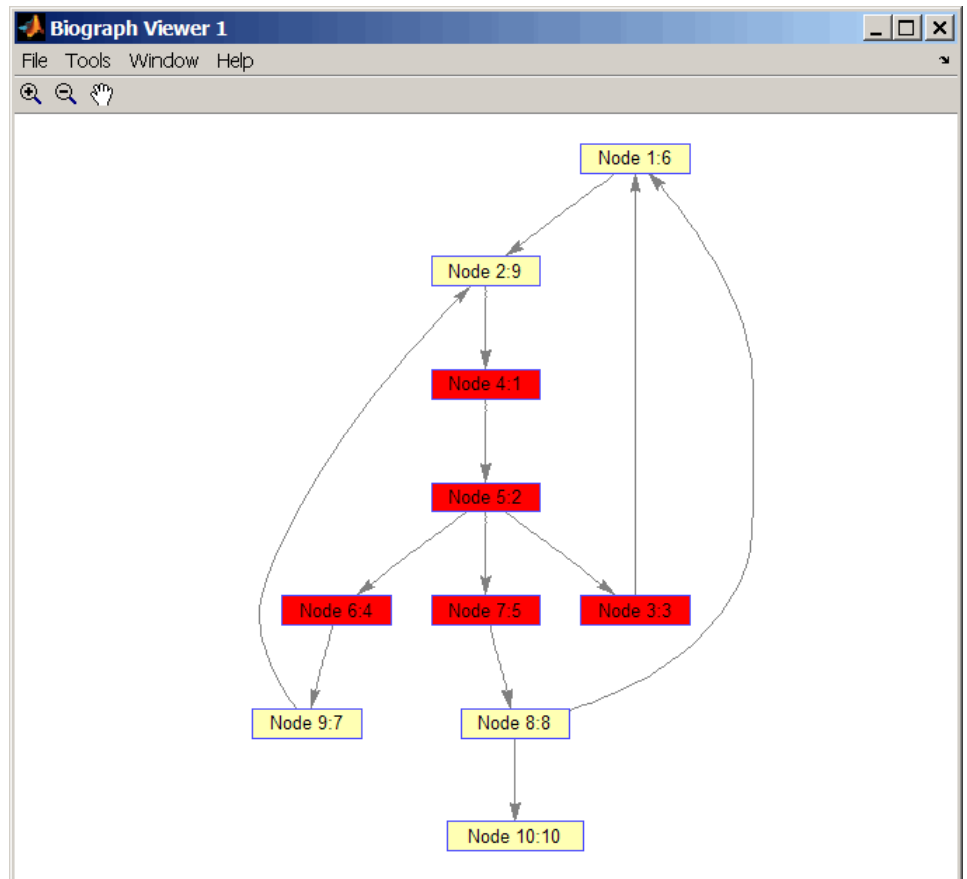
References   [1] Sedgewick, R., (2002). Algorithms in C++, Part 5 Graph Algorithms (Addison-Wesley).

[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).

graphtraverse

See Also
Bioinformatics Toolbox functions: `graphallshortestpaths`, `graphconncomp`, `graphisdag`, `graphisomorphism`, `graphisspantree`, `graphmaxflow`, `graphminspantree`, `graphpred2path`, `graphshortestpath`, `graphtopoorder`

Bioinformatics Toolbox method of `biograph` object: `traverse`

Purpose   Align query sequence to profile using hidden Markov model alignment

Syntax   *Alignment* = hmmprofalign(*Model,Seq*)
[*Alignment, Score*] = hmmprofalign(*Model,Seq*)
[*Score, Alignment, Prointer*] = hmmprofalign(*Model,Seq*)
hmmprofalign(..., '*PropertyName*', *PropertyValue*,...)
hmmprofalign(..., 'ShowScore', *ShowScoreValue*)
hmmprofalign(..., 'Flanks', *FlanksValue*)
hmmprofalign(..., 'ScoreFlanks', *ScoreFlanksValue*)
hmmprofalign(..., 'ScoreNullTransitions',
    ScoreNullTransitionValue)

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	Property to control displaying the scoring space and the winning path. Enter either true or false (default).
FlanksValue	Property to control including the symbols generated by the FLANKING INSERT states in the output sequence. Enter either true or false (default).
ScoreFlanksValue	Property to control including the transition probabilities for the flanking states in the raw score. Enter either true or false (default).
ScoreNullTransValue	Property to control adjusting the raw score using the null model for transitions (Model.NullX). Enter either true or false (default).

hmmprofalign

Description   *Alignment* = 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.

[*Alignment, Score*] = hmmprofalign(*Model,Seq*) 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, Prointer*] = hmmprofalign(*Model,Seq*) 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*,...) defines optional properties using property name/value pairs.

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', *ScoreNullTransitionValue*), 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

```
load('hmm_model_examples','model_7tm_2') % load a model example
load('hmm_model_examples','sequences')  % load a sequence example
SCCR_RABIT=sequences(2).Sequence;
[a,s]=hmmprofalign(model_7tm_2,SCCR_RABIT,'showscore',true)
```

See Also

Bioinformatics Toolbox functions gethmmprof, hmmprofestimate, hmmprofgenerate, hmmprofgenerate, hmmprofstruct, pfamhmmread, showhmmprof, multialign, profalign

hmmprofestimate

Purpose
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

hmmprofestimate(..., 'BDx', *BDxValue*) sets the background transition probabilities from any DELETE state ([D->M D->D]). Default values are taken from hmmprofstruct.

See Also   Bioinformatics Toolbox functions: hmmprofalign, hmmprofstruct, showhmmprof

Purpose	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	Property to control using uppercase letters for matches and lowercase letters for inserted letters. Enter either true or false. Default is false.
FlanksValue	Property to control including the symbols generated by the FLANKING INSERT states in the output sequence. Enter either true or false. Default is false.
SignatureValue	Property to control returning the most likely path and symbols. Enter either true or false. Default is false.

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 move 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

hmmprofgenerate

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   Bioinformatics Toolbox functions: `hmmprofalign`, `hmmprofstruct`, `showhmmprof`

Purpose   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*) 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.

hmmprofmerge

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   Bioinformatics Toolbox functions: hmmprofalign, hmmprofstruct

hmmprofstruct

Purpose   Create profile Hidden Markov Model (HMM) structure

Syntax
```
Model = hmmprofstruct(Length)
Model = hmmprofstruct(Length, 'Field1', FieldValues1,...)
hmmprofstruct(Model, 'Field1', Field1Values1,...)
```

Arguments

Length	Number of match states in the model.
Model	Hidden Markov model created with the function hmmprofstruct.
Field1	Field name in the structure Model. Enter a name from the table below.

Description   Model = hmmprofstruct(Length) returns a structure with the fields containing the required parameters of a profile HMM. Length specifies the number of match states in the model. All other mandatory model parameters are initialized to the default values.

Model = hmmprofstruct(Length, 'Field1', *FieldValues1*, ...) creates a profile HMM using the specified fields and parameters. All other mandatory model parameters are initialized to default values.

hmmprofstruct(Model, 'Field1', *Field1Values1*, ...) returns the updated profile HMM with the specified fields and parameters. All other mandatory model parameters are taken from the reference MODEL.

HMM Profile Structure Format

Model parameters fields (mandatory). All probability values are in the [0 1] range.

Field Name	Description
ModelLength	Length of the profile (number of MATCH states)
Alphabet	'AA' or 'NT'. Default is 'AA'.

Field Name	Description
MatchEmission	Symbol emission probabilities in the MATCH states.
	Size is [ModelLength x AlphaLength]. Defaults to uniform distributions. May accept a structure with residue counts (see aacount or basecount).
InsertEmission	Symbol emission probabilities in the INSERT state.
	Size is [ModelLength x AlphaLength]. Defaults to uniform distributions. May accept a structure with residue counts (see aacount or basecount).
NullEmission	Symbol emission probabilities in the MATCH and INSERT states for the NULL model. NULL model, size is [1 x AlphaLength]. Defaults to a uniform distribution. May accept a structure with residue counts (see aacount or basecount). The NULL model is used to compute the log-odds ratio at every state and avoid overflow when propagating the probabilities through the model.
BeginX	BEGIN state transition probabilities.
	Format is
	`[B->D1 B->M1 B->M2 B->M3 .... B->Mend]`
	Notes:
	`sum(S.BeginX) = 1`
	For fragment profiles
	`sum(S.BeginX(3:end)) = 0`
	Default is [0.01 0.99 0 0 ... 0].

Field Name	Description
MatchX	MATCH state transition probabilities
	Format is
	``` [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] ```
	Notes:
	``` sum(S.MatchX) = [1 1 ... 1] ```
	For fragment profiles
	``` sum(S.MatchX(4,:)) = 0 ```
	Default is repmat([0.998 0.001 0.001 0],profLength-1,1).
InsertX	INSERT state transition probabilities
	Format is
	``` [I1->M2 I2->M3 ... I[end-1]->Mend; [I1->I1 I2->I2 ... I[end-1]->I[end-1]] ```
	Note:
	``` sum(S.InsertX) = [1 1 ... 1] ```
	Default is repmat([0.5 0.5],profLength-1,1).

hmmprofstruct

Field Name	Description

DeleteX	DELETE state transition probabilities. The format is
	
	`[D1->M2 D2->M3 ... D[end-1]->Mend ;`
	`[D1->D2 D2->D3 ... D[end-1]->Dend   ]`
	
	--- 	
	Note sum(S.DeleteX) = [1 1 ... 1]
	
	--- 	
	
	Default is repmat([0.5 0.5],profLength-1,1).

FlankingInsertX	Flanking insert states (N and C) used for LOCAL profile alignment. The format is
	
	`[N->B  C->T ;`
	`[N->N  C->C ]`
	
	--- 	
	Note sum(S.FlankingInsertsX) = [1 1]
	
	--- 	
	
	To force global alignment use
	
	`S.FlankingInsertsX = [1 1; 0 0]`
	
	Default is [0.01 0.01; 0.99 0.99].

Field Name	Description
LoopX	Loop states transition probabilities used for multiple hits alignment. The format is ``` [E->C J->B ; E->J J->J] ``` **Note** sum(S.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. The format is ``` [G->F ; G->G] ``` **Note** sum(S.NullX) = 1 Default is [0.01; 0.99]

Annotation Fields (Optional)

Name	Model Name
IDNumber	Identification Number
Description	Short description of the model

A profile Markov 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 4. Insert, match, and delete states are in the regular part (middle 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 symbol).

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, N, E, and T are silent while N and C are used to insert symbols at the flanks.
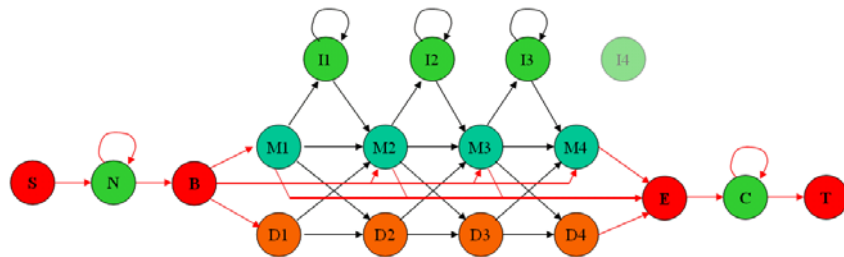
Examples   `hmmprofstruct(100,'Alphabet','AA')`

See Also   Bioinformatics Toolbox functions: `aacount`, `basecount`, `gethmmprof`, `hmmprofalign`, `hmmprofestimate`, `hmmprofgenerate`, `hmmprofmerge`, `pfamhmmread`, `showhmmprof`

Purpose	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.
CleanColNameValue	Property to control creating column names that MATLAB can use as variable names.

Description   *imagenedata* = imageneread('*File*') reads ImaGene results data from *File* and creates a MATLAB structure imagedata containing the following fields:

Field
HeaderAA
Data
Blocks
Rows
Columns
Fields
IDs
ColumnNames

imageneread

Field
Indices
Shape

imagenedata = imageneread(..., '*PropertyName*', *PropertyValue*, ...) defines optional properties using property name/value pairs, described as follows:

imagenedata = imageneread(..., 'CleanColNames', *CleanColNamesValue*, ...). An ImaGene file may contain column names with spaces and some characters that MATLAB cannot use in MATLAB variable names. If *CleanColNamesValue* is true, imagene returns, in the field ColumnNames, names 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 not valid for MATLAB variable names.

The field Indices of the structure contains MATLAB 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 User Manual.

ImaGene is a registered trademark of BioDiscovery, Inc.

Examples

1 Read in a sample ImaGene Results file. Note, the file cy3.txt is not provided with Bioinformatics Toolbox.

```
cy3Data = imageneread('cy3.txt');
```

2 Plot the signal mean.

```
maimage(cy3Data,'Signal Mean');
```

3 Read in a sample ImaGene Results file. Note, the file cy5.txt is not provided with Bioinformatics Toolbox.

```
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   Bioinformatics Toolbox functions: gprread, maboxplot, maimage, sptread

int2aa

Purpose	Convert amino acid sequence from integer to letter representation

Syntax

```
SeqChar = int2aa(SeqInt)
SeqChar = int2aa(SeqInt, 'Case', CaseValue)
```

Arguments

SeqInt	Row vector of integers specifying an amino acid sequence. See the table Mapping Amino Acid Integers to Letters on page 2-422 for valid integers. Integers are arbitrarily assigned to IUB/IUPAC letters.
CaseValue	String that specifies the case of the returned character string. Choices are 'upper' (default) or 'lower'.

Return Values

SeqChar	Character string of single-letter codes specifying an amino acid sequence.

Mapping Amino Acid Integers to Letters

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

Mapping Amino Acid Integers to Letters (Continued)

Amino Acid	Integer	Code
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
Any amino acid	23	X
Translation stop	24	*
Gap of indeterminate length	25	-
Unknown or any integer not in table	0	?

Description   *SeqChar* = int2aa(*SeqInt*) converts a 1-by-N array of integers specifying an amino acid sequence to a character string of single-letter codes specifying the same amino acid sequence. See the table Mapping Amino Acid Integers to Letters on page 2-422 for valid integers.

SeqChar = int2aa(*SeqInt*, 'Case', *CaseValue*) specifies the case of the returned character string representing an amino acid sequence. Choices are 'upper' (default) or 'lower'.

Examples   Convert an amino acid sequence from integer to letter representation.

```
s = int2aa([13 1 17 11 1 21])

s =

MATLAB
```

See Also   Bioinformatics Toolbox functions: aa2int, aminolookup, int2nt, nt2int

Purpose   Convert nucleotide sequence from integer to letter representation

Syntax
```
int2nt(SeqNT)
int2nt(..., 'PropertyName', PropertyValue,...)
int2nt(..., 'Alphabet', AlphabetValue)
int2nt(..., 'Unknown', UnknownValue)
int2nt(..., 'Case', CaseValue)
```

Arguments

SeqNT	Nucleotide sequence represented by integers. Enter a vector of integers from the table Mapping Nucleotide Integers to Letters below. The array does not have to be of type integer, but it does have to contain only integer numbers. Integers are arbitrarily assigned to IUB/IUPAC letters.
AlphabetValue	Property to select the nucleotide alphabet. Enter 'DNA' or 'RNA'.
UnknownValue	Property to select the integer value for the unknown character. Enter a character to map integers 16 or greater to an unknown character. The character must not be one of the nucleotide characters A, C, G, T, or the ambiguous nucleotide characters N, R, Y, K, M, S, W, B, D, H, or V. The default character is *.
CaseValue	Property to select the letter case for the nucleotide sequence. Enter 'upper' (default) or 'lower' .

Mapping Nucleotide Integers to Letters

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)	15	N
Gap of indeterminate length	16	-
Unknown (default)	0 or ≥ 17	*

Description   int2nt(*SeqNT*) converts a 1-by-N array of integers to a character string using the table Mapping Nucleotide Integers to Letters on page 2-426.

int2nt(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

int2nt(..., 'Alphabet', *AlphabetValue*) selects the nucleotide alphabet to use. Default is 'DNA', which uses the symbols A, C, G, and T. If *AlphabetValue* is set to 'RNA', int2nt uses the symbols A, C, G, and U instead.

int2nt(..., 'Unknown', *UnknownValue*) specifies the character to represent an unknown nucleotide base.

int2nt(..., 'Case', *CaseValue*) selects the output case of the nucleotide string.

Examples

Enter a sequence of integers as a MATLAB vector (space or comma-separated list with square brackets).

```
s = int2nt([1 2 4 3 2 4 1 3 2])

s =
    ACTGCTAGC
```

Define a symbol for unknown numbers 16 and greater.

```
si = [1 2 4 20 2 4 40 3 2];
s = int2nt(si, 'unknown', '#')

s =
ACT#CT#GC
```

See Also

Bioinformatics Toolbox functions: aa2int, int2aa, nt2int

isoelectric

Purpose 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 Letters to Integers on page 2-2. Examples: 'ARN' or [1 2 3].
PKValsValue	Property to provide alternative pK values.
ChargeValue	Property to select a specific pH for estimating charge. Enter a number between 0 and 14. The default value is 7.2.
ChartValue	Property to control 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. 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 the alternative pK table stored in the text file *PKValValues*. For an example of a pK text file, see the file Emboss.pK.

```
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
```

isoelectric(..., 'Charge', *ChargeValue*) returns the estimated charge of a sequence for a given pH (*ChargeValue*).

isoelectric

isoelectric(..., 'Chart', *ChartValue*) when *ChartValue* is true, returns a graph plotting the charge of the protein versus the pH of the solvent.

Example

```
% 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   Bioinformatics functions aacount, molweight

Purpose   Read JCAMP-DX formatted files

Syntax   *JCAMPData* = jcampread(*File*)

Arguments   *File*   JCAMP-DX formatted file (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 of 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 and 5 of the JCAMP-DX format. For more details, see:

                   http://www.jcamp-dx.org/

                 JCAMPData = jcampread(*File*) reads data from *File*, a JCAMP-DX
                 formatted file, and creates *JCAMPData*, a MATLAB structure containing
                 the following fields:

Field
Title
DataType
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 Download test data in the file isa_ms1.dx from:

```
http://www.jcamp-dx.org/testdata.html
```

2 Read a JCAMP-DX file (isas_ms1.dx) into MATLAB and plot the mass spectrum.

```
jcampStruct = jcampread('isas_ms1.dx')
data = jcampStruct.Blocks(1);
stem(data.XData,data.YData, '.', 'MarkerEdgeColor','w');
title(jcampStruct.Title);
xlabel(data.XUnits);
ylabel(data.YUnits);
```

A Figure window opens with the mass spectrum.
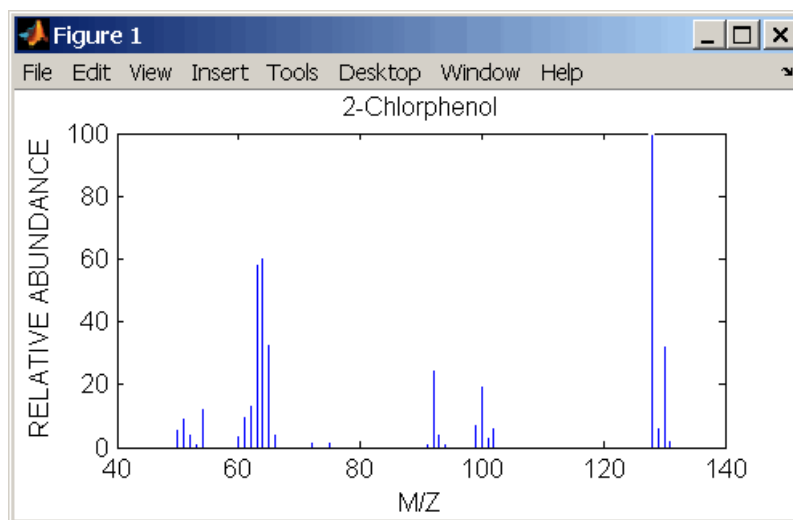
See Also   Bioinformatics Toolbox functions: `mslowess`, `mssgolay`, `msviewer`, `mzxmlread`

joinseq

Purpose	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

```
seq1 = 'ACGTAAA';
seq2 = 'AAATGCA';
joined = joinseq(seq1,seq2)

joined =
    ACGTAAATGCA
```

See Also   MATLAB functions cat, strcat, strfind

Purpose   Classify data using nearest neighbor method

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.

knnclassify

distance	String to specify 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 (only suitable 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 (only suitable 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 or '' for string named 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.

knnclassify

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;
```
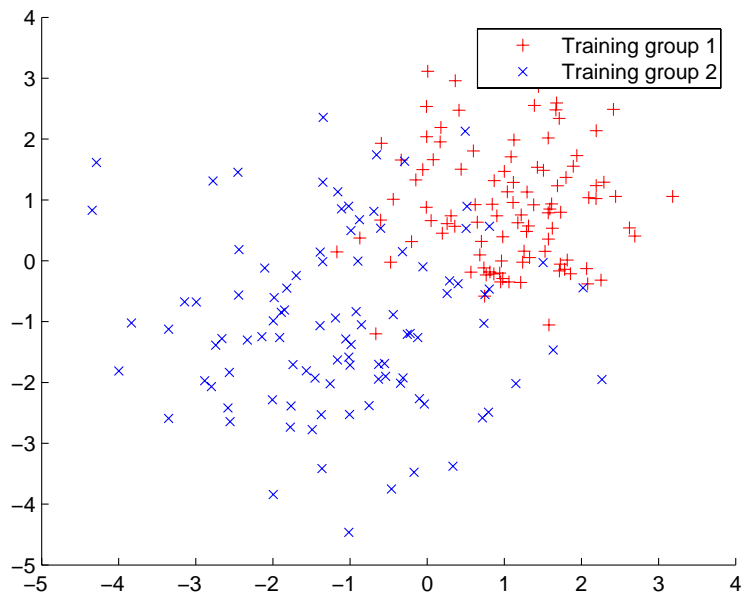

The following commands create the matrix `sample`, classify its rows into two groups, and plot the result.

```
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;
```
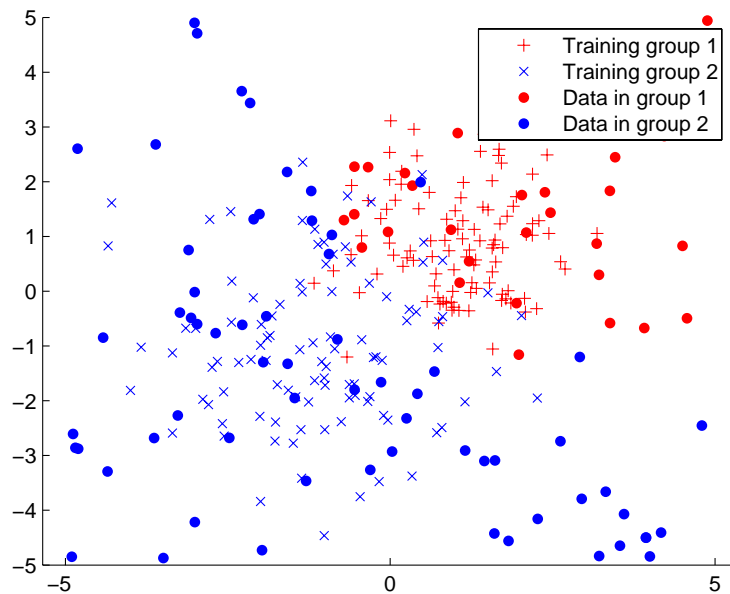

Classifying Rows Using the Three Nearest Neighbors

The following example uses the same data as in Example 2, 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');
```
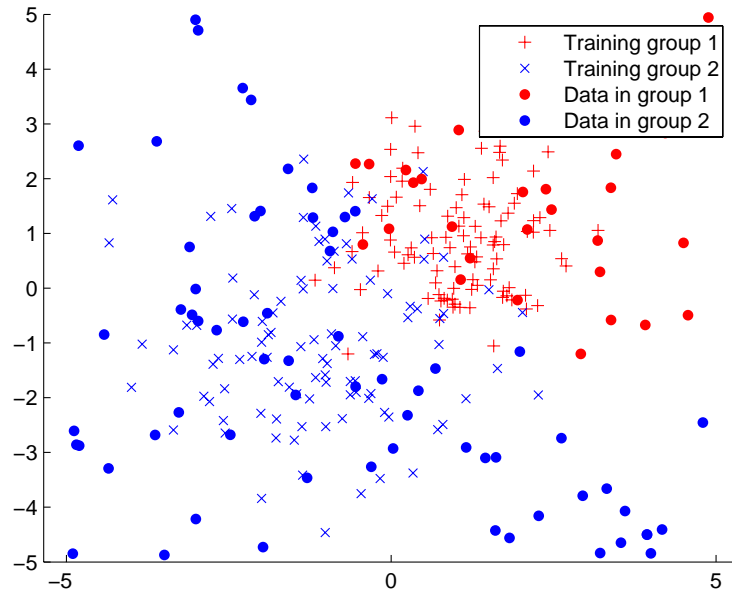

If you compare this plot with the one in Example 2, you see that some of the data points are classified differently using three nearest neighbors.

References   [1] Mitchell T (1997), Machine Learning, McGraw-Hill.

See Also   Bioinformatics Toolbox functions: `knnimpute`, `classperf`, `crossvalind`, `svmclassify`, `svmtrain`

Statistics Toolbox functions: `classify`

knnimpute

Purpose	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

k

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.

knnimpute

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 Bioinformatics Toolbox function: `knnclassify`

MATLAB function: `isnan`

Statistics Toolbox functions: `nanmean`, `nanmedian`, `pdist`

maboxplot

Purpose	Box plot for microarray data

Syntax

```
maboxplot(MAData)
maboxplot(MAData, ColumnName)
maboxplot(MAStruct, FieldName)
H = maboxplot(...)
[H, HLines] = maboxplot(...)
maboxplot(..., 'PropertyName', PropertyValue, ...)
maboxplot(..., 'Title', TitleValue, ...)
maboxplot(..., 'Notch', NotchValue, ...)
maboxplot(..., 'Symbol', SymbolValue, ...)
maboxplot(..., 'Orientation', OrientationValue, ...)
maboxplot(..., 'WhiskerLength', WhiskerLengthValue, ...)
```

Arguments

MAData	A numeric array or a structure containing a field called Data. The values in the columns of *MAData* will be used to create box plots.
ColumnName	An array of column names corresponding to the data in *MAData*.
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	A string to use as the title for the plot. The default title is FieldName.
NotchValue	Property to control the type of boxes drawn. Enter either true for notched boxes, or false, for square boxes. Default is false.

OrientationValue	Property to specify the orientation of the box plot. Enter `'Vertical'` or `'Horizontal'`. Default is `'Horizontal'`.
WhiskerLengthValue	Property to specify 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.

Description maboxplot(*MAData*) displays a box plot of the values in the columns of data (*MAData*). *MAData* can be a numeric array or a structure containing a field called 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.

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*, ...) defines optional properties using property name/value pairs in any order. These property name/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

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.

Examples

```
load yeastdata
maboxplot(yeastvalues,times);
xlabel('Sample Times');

% Using a structure
geoStruct = getgeodata('GSM1768');
maboxplot(geoStruct, 'title', 'GSM1768');

% For block-based data
madata = gprread('mouse_a1wt.gpr');
maboxplot(madata,'F635 Median');
figure
maboxplot(madata,'F635 Median - B635','TITLE',...
          'Cy5 Channel FG - BG');
```

See Also

Bioinformatics Toolbox functions magetfield, maimage, mairplot, maloglog, malowess, manorm, mavolcanoplot

Statistics Toolbox function boxplot

Purpose   Estimate false discovery rate (FDR) of differentially expressed genes from two experimental conditions or phenotypes

Syntax
```
FDR = mafdr(PValues)
[FDR, Q] = mafdr(PValues)
[FDR, Q, Pi0] = mafdr(PValues)
[FDR, Q, Pi0, R2] = mafdr(PValues)
... = mafdr(PValues, ...'BHFDR', BHFDRValue, ...)
... = mafdr(PValues, ...'Lambda', LambdaValue, ...)
... = mafdr(PValues, ...'Method', MethodValue, ...)
... = mafdr(PValues, ...'Showplot', ShowplotValue, ...)
```

Arguments

PValues	Column vector of p-values for each gene in two microarray data sets, such as returned by mattest.
BHFDRValue	Property to control the use of the linear step-up (LSU) procedure originally introduced by Benjamini and Hochberg, 1995. Choices are true or false (default).

> **Note** If *BHFDRValue* is set to true, the Lambda and Method properties are ignored.

LambdaValue   Input that specifies lambda, λ, the tuning parameter used to estimate the true null hypotheses, $\hat{\pi}_0(\lambda)$. *LambdaValue* can be either:

- A single value that is > 0 and < 1.
- A series of values. Each value must be > 0 and < 1. There must be at least four values in the series.

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 *LambdaValue* is set to a single value, the Method property is ignored.

MethodValue	String that specifies a method to calculate the true null hypothesis, $\hat{\pi}_0(\lambda)$, from the tuning parameter, *LambdaValue*, when *LambdaValue* is a series of values. Choices are:

- bootstrap (default)
- polynomial

ShowplotValue	Property to display two plots:

- Plot of the estimated true null hypotheses, $\hat{\pi}_0(\lambda)$, 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).

Return Values

FDR	Column vector of positive FDR (pFDR) values.
Q	Column vector of q-values.
Pi0	Estimated true null hypothesis, $\hat{\pi}_0$.
R2	Square of the correlation coefficient.

Description *FDR* = mafdr(*PValues*) computes a positive FDR (pFDR) value for each value in *PValues*, a column vector of p-values for each gene in two microarray data sets, using a procedure introduced by Storey, 2002. *FDR* is a column vector of positive FDR (pFDR) values.

[*FDR, Q*] = mafdr(*PValues*) also returns a q-value for each p-value in *PValues*. *Q* is a column vector.

[*FDR, Q, Pi0*] = mafdr(*PValues*) also returns *Pi0*, the estimated true null hypothesis, $\hat{\pi}_0$, if using the procedure introduced by Storey, 2002.

[*FDR, Q, Pi0, R2*] = mafdr(*PValues*) also returns *R2*, the square of the correlation coefficient, if using the procedure introduced by Storey, 2002, and the `polynomial` method to calculate the true null hypothesis, $\hat{\pi}_0$, from 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:

... = mafdr(*PValues*, ...'BHFDR', *BHFDRValue*, ...) controls the use of the linear step-up (LSU) procedure originally introduced by Benjamini and Hochberg, 1995, to computes an FDR-adjusted p-value for each value in *PValues*. Choices are `true` or `false` (default).

Note If *BHFDRValue* is set to `true`, the `Lambda` and `Method` properties are ignored.

... = mafdr(*PValues*, ...'Lambda', *LambdaValue*, ...) specifies lambda, λ, the tuning parameter used to estimate the true null hypotheses, $\hat{\pi}_0(\lambda)$. *LambdaValue* can be either:

- A single value that is > `0` and < `1`.
- A series of values. Each value must be > `0` and < `1`. There must be at least four values in the series.

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 *LambdaValue* is set to a single value, the Method property is ignored.

... = mafdr(*PValues*, ...'Method', *MethodValue*, ...) specifies a method to calculate the true null hypothesis, $\hat{\pi}_0$, from the tuning parameter, *LambdaValue*, when *LambdaValue* is a series of values. Choices are bootstrap (default) or polynomial.

... = mafdr(*PValues*, ...'Showplot', *ShowplotValue*, ...) controls the display of two plots:

- Plot of the estimated true null hypotheses, $\hat{\pi}_0(\lambda)$, versus the tuning parameter, lambda, with a cubic polynomial fitting curve
- Plot of q-values versus p-values

Choices are true or false (default).

Examples

1 Load the MAT file, included with Bioinformatics Toolbox, 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

Bioinformatics Toolbox functions: `gcrma`, `mairplot`, `maloglog`, `mapcaplot`, `mattest`, `mavolcanoplot`, `rmasummary`

magetfield

Purpose	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');
cy3data = magetfield(maStruct,'F635 Median');
cy5data = magetfield(maStruct,'F532 Median');
mairplot(cy3data,cy5data,'title','R vs G IR plot');
```

See Also   Bioinformatics Toolbox functions: agferead, gprread, imageneread, maboxplot, mairplot, maloglog, malowess, sptread

Purpose   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

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
```
madata = gprread('mouse_a1wt.gpr');
maimage(madata,'F635 Median');
figure;
maimage(madata,'F635 Median - B635',...
        'Title','Cy5 Channel FG - BG');
colormap hot
```

See Also
Bioinformatics Toolbox functions: maboxplot, magetfield, mairplot, maloglog, malowess

MATLAB function: imagesc

Purpose   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(..., 'Prctile', *PrctileValue*, ...)
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	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 [0.03, 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.
PrctileValue	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 (*StopPrctileValue*) 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*.

    $prd = \text{abs}(RankX - RankY)$

- 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*, ...) defines optional properties that use property name/value pairs in any order. These property name/value pairs are as follows:

NormDataY = mainvarsetnorm(..., '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 [0.03, 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(..., 'Prctile', *PrctileValue*, ...) 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 (*PrctileValue*) is reached. When *IterateValue* is false, performs only one iteration of the process. Default is true.

mainvarsetnorm

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`.

The following example illustrates how mainvarsetnorm can correct for dye bias or scanning differences between two channels of data from a two-color microarray experiment. 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.
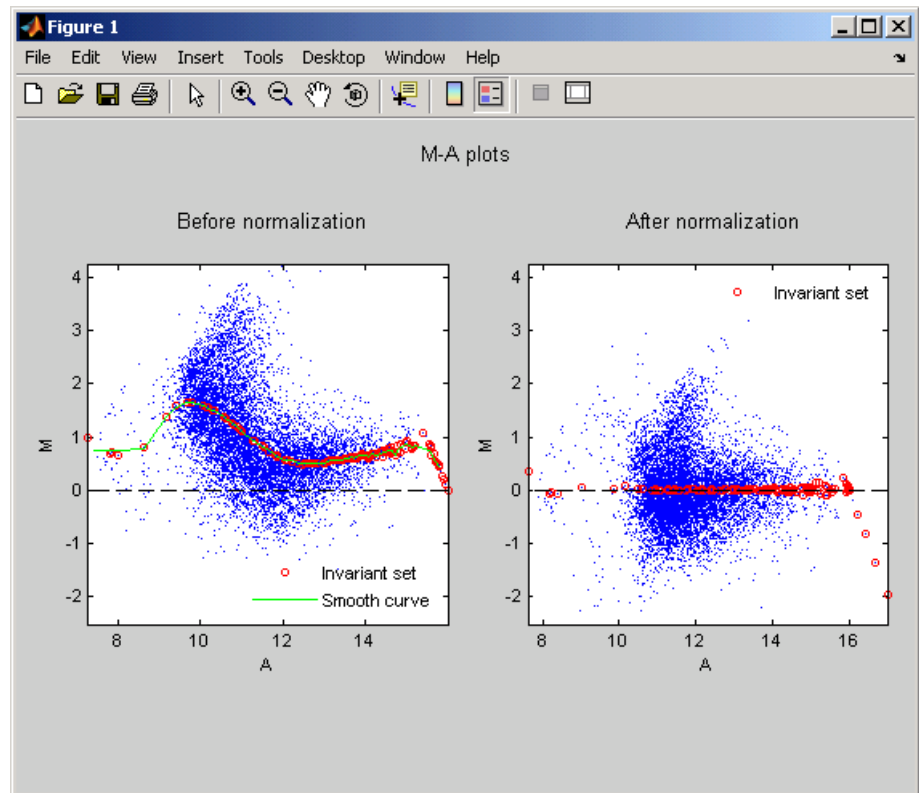
Examples   The following example extracts data from a GPR file and creates two
column vectors of gene expression values from different experimental
conditions. It then normalizes one of the data sets.

```
maStruct = gprread('mouse_a1wt.gpr');
cy3data = magetfield(maStruct, 'F635 Median');
cy5data = magetfield(maStruct, 'F532 Median');
Normcy5data = mainvarsetnorm(cy3data, cy5data);
```

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`

Purpose   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, ...)
```

Arguments

DataX, *DataY*	Vectors 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.

mairplot

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.

Return Values	*Intensity*	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')
	Ratio	Vector containing ratios of the microarray gene expression data, calculated as log2(*DataX./DataY*).
	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`.

Following is an IR plot of normalized data.
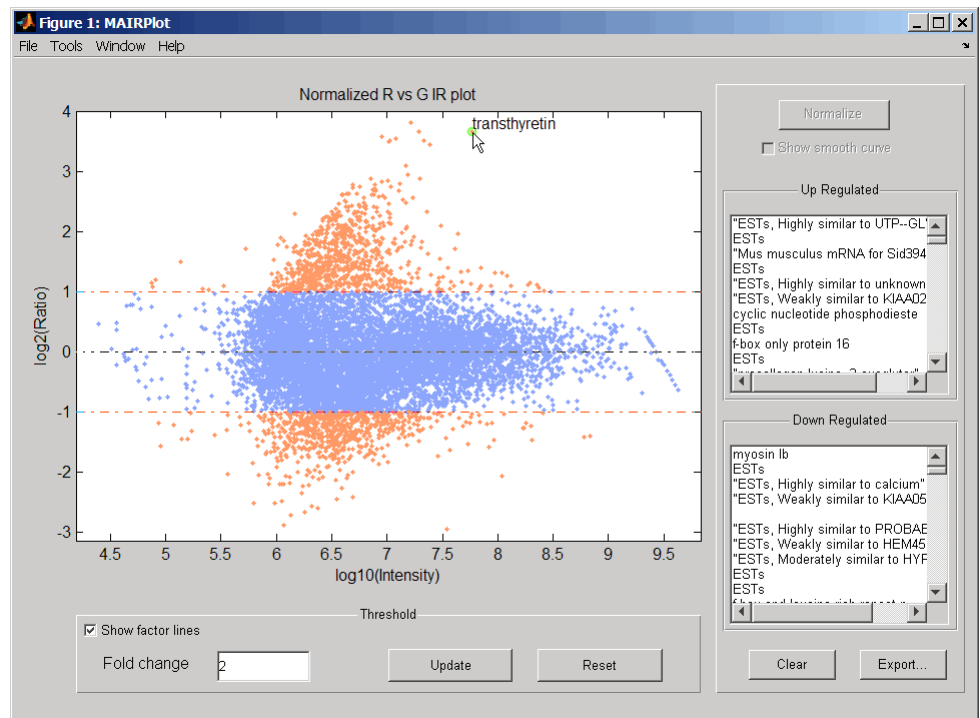
Following is an M-A plot of unnormalized data.
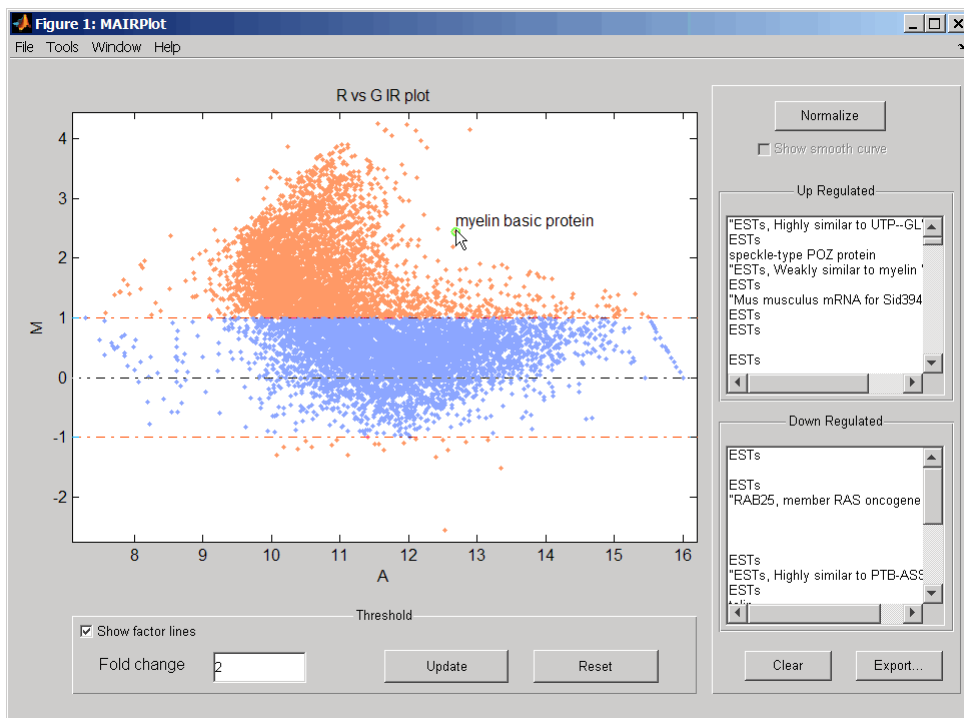
The intensity versus ratio scatter plot displays the following:

- $\log_{10}$ (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. For Order, enter **1** for linear or **2** for quadratic.

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.

```
cy3data = magetfield(maStruct,'F635 Median');
cy5data = 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(cy3data, cy5data, '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(cy3data, cy5data, 'Showplot', false);
```

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 Bioinformatics Toolbox functions: `maboxplot`, `magetfield`, `maimage`, `mainvarsetnorm`, `maloglog`, `malowess`, `manorm`, `mattest`, `mavolcanoplot`

Purpose   Create loglog plot of microarray data

Syntax
```
maloglog(X, Y, 'PropertyName', PropertyValue...)
maloglog(..., 'FactorLines', N)
maloglog(..., 'Title', TitleValue)
maloglog(..., 'Labels', LabelsValues)
maloglog(..., 'HandleGraphicsName', HGValue)
H = maloglog(...)
```

Arguments

X	A numeric array of microarray expression values from a single experimental condition.
Y	A 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, 'PropertyName', PropertyValue...) creates a loglog scatter plot of X versus Y. X and Y are numeric arrays of microarray expression values from two different experimental conditions.

maloglog(..., 'FactorLines', N) adds two lines to the plot showing a factor of N change.

maloglog(..., 'Title', TitleValue) allows you to specify a title for the plot.

maloglog(..., '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(..., 'HandleGraphicsName', HGValue) allows you to pass optional Handle Graphics property name/property value pairs to the function.

maloglog

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
Bioinformatics Toolbox functions maboxplot, magetfield, mainvarsetnorm, maimage, mairplot, malowess, manorm, mattest, mavolcanoplot

MATLAB function loglog

malowess

Purpose   Smooth microarray data using Lowess method

Syntax   *YSmooth* = malowess(*X*, *Y*)
                     malowess(..., '*PropertyName*', *PropertyValue*,...)
                     malowess(..., 'Order', *OrderValue* ...)
                     malowess(..., 'Robust', *RobustValue* ...)
                     malowess(..., 'Span', *SpanValue* ...)

Arguments

X, *Y*	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 (X, Y) using the
                     Lowess smoothing method. The default window size is 5% of the length
                     of X.

                     malowess(..., '*PropertyName*', *PropertyValue*,...) defines
                     optional properties using property name/value pairs.

                     malowess(..., 'Order', *OrderValue* ...) chooses the order of the
                     algorithm. Note that Curve Fitting Toolbox refers to Lowess smoothing
                     of order 2 as Loess smoothing.

                     malowess(..., 'Robust', *RobustValue* ...) uses a robust fit when
                     RobustValue is set to true. This option can take a long time to calculate.

                     malowess(..., '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*.

malowess

Examples
```
maStruct = gprread('mouse_a1wt.gpr');
cy3data = magetfield(maStruct, 'F635 Median');
cy5data = magetfield(maStruct, 'F532 Median');
[x,y] = mairplot(cy3data, cy5data);
drawnow
ysmooth = malowess(x,y);
hold on;
plot(x, ysmooth, 'rx')
ynorm = y - ysmooth;
```

See Also   Bioinformatics Toolbox functions affyinvarsetnorm, maboxplot, magetfield, maimage, mainvarsetnorm, mairplot, maloglog, manorm, quantilenorm

Statistics Toolbox function robustfit

Purpose   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)

Description   XNorm = manorm(X) scales the values in each column of microarray
data (X) by dividing by the mean column intensity.

- X — Microarray data. Enter a vector or matrix.

- XNorm — Normalized microarray data.

XNorm = manorm(MAStruct, FieldName) scales the data for a field
(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.

- MAStruct — Microarray structure.

[XNorm, ColVal] = manorm(...) returns the values used to normalize
the data.

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 Bioinformatics Toolbox functions affyinvarsetnorm, maboxplot, magetfield, mainvarsetnorm, mairplot, maloglog, malowess, quantilenorm, rmasummary

mapcaplot

Purpose   Create Principal Component Analysis (PCA) plot of microarray data

Syntax   mapcaplot(*Data*)
                   mapcaplot(*Data*, *Label*)

Arguments

Data	Microarray expression profile data.
Label	Cell array of strings representing labels for the data points.

Description   mapcaplot(*Data*) creates 2-D scatter plots of principal components of
                   the array *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.

mapcaplot
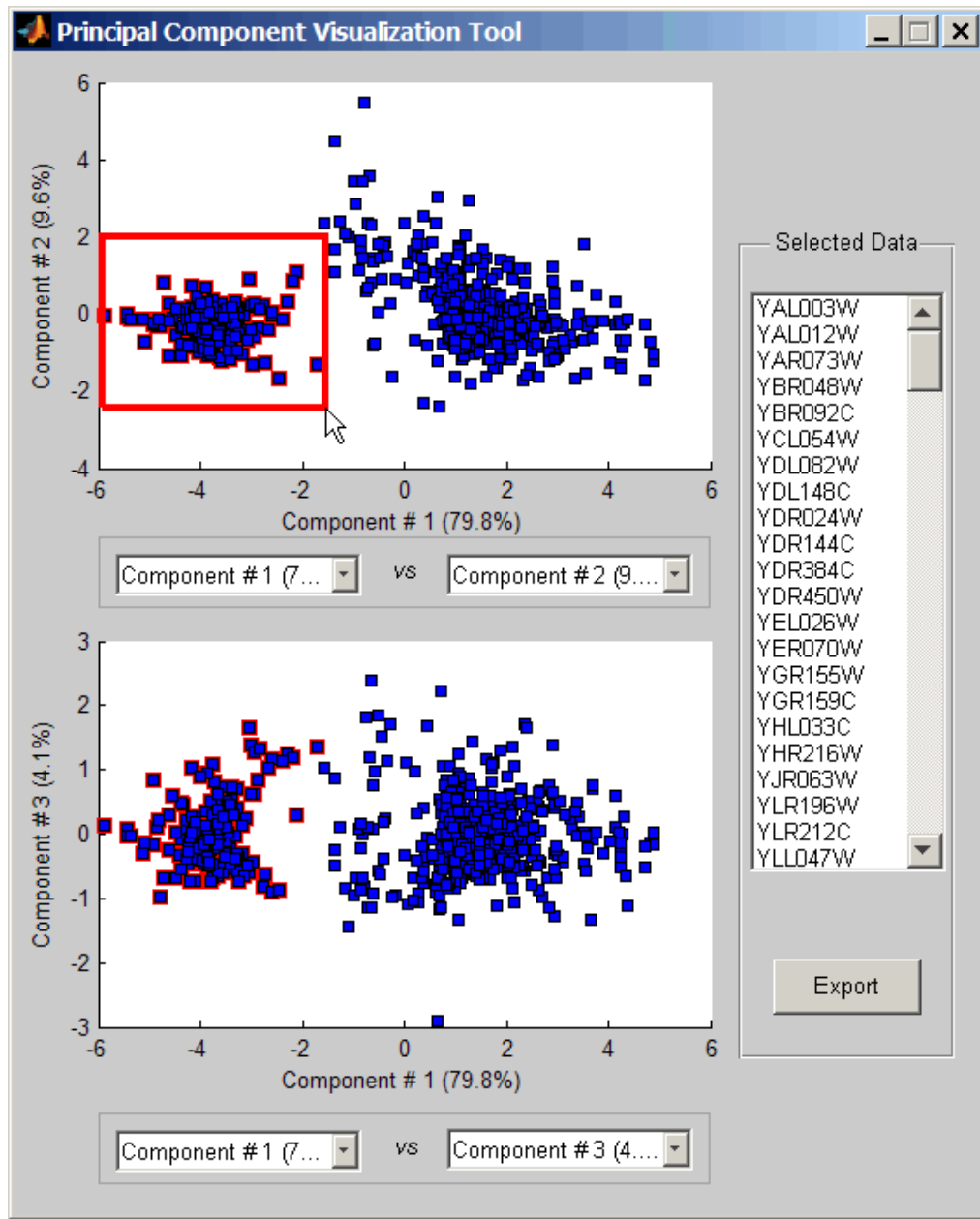
Once you plot the principal components, you can:

- 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
```
load filteredyeastdata
mapcaplot(yeastvalues, genes)
```

See Also
Bioinformatics Toolbox functions: `clustergram`, `mattest`, `mavolcanoplot`

Statistics Toolbox function: `princomp`

Purpose   Perform two-tailed 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(..., 'Permute', *PermuteValue*, ...)
... = mattest(..., 'Showhist', *ShowhistValue*, ...)
... = mattest(..., 'Showplot', *ShowplotValue*, ...)
... = mattest(..., 'Labels', *LabelsValue*, ...)

Arguments

DataX, *DataY*	Matrices 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, in a two-color microarray experiment, *DataX* could be cy3 intensity values and *DataY* could be cy5 intensity values.
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.
ShowhistValue	Controls the display of histograms of t-score distributions and p-value distributions. Choices are true or false (default).

mattest

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.	

Return Values

PValues	Column vector of p-values for each gene in *DataX* and *DataY*.
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*) compares the gene expression profiles in *DataX* and *DataY* and returns a p-value for each gene. *DataX* and *DataY* are matrices 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 with equal variances. *PValues* is a column vector of p-values for each gene.

[*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(..., '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(..., '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.
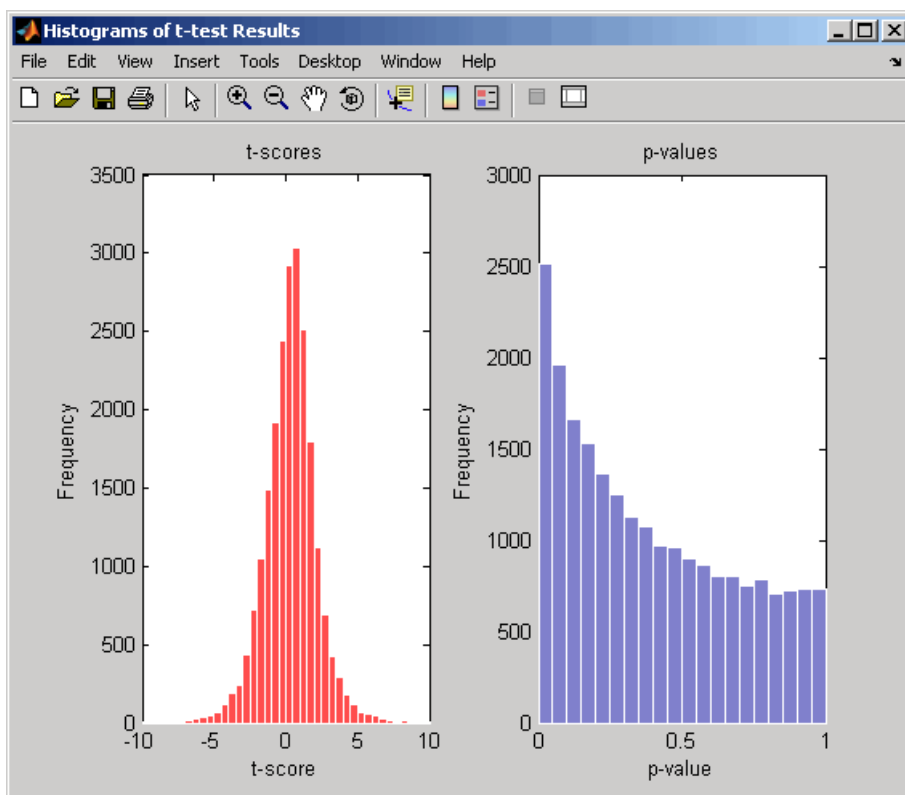
... = mattest(..., 'Showplot', *ShowplotValue*, ...) controls the display of a normal t-score quantile plot. When *ShowplotValue* is true, mattest displays a quantile-quantile plot. Default is false. In the t-score quantile plot, the black diagonal line represents the sample quantile being equal to the 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.
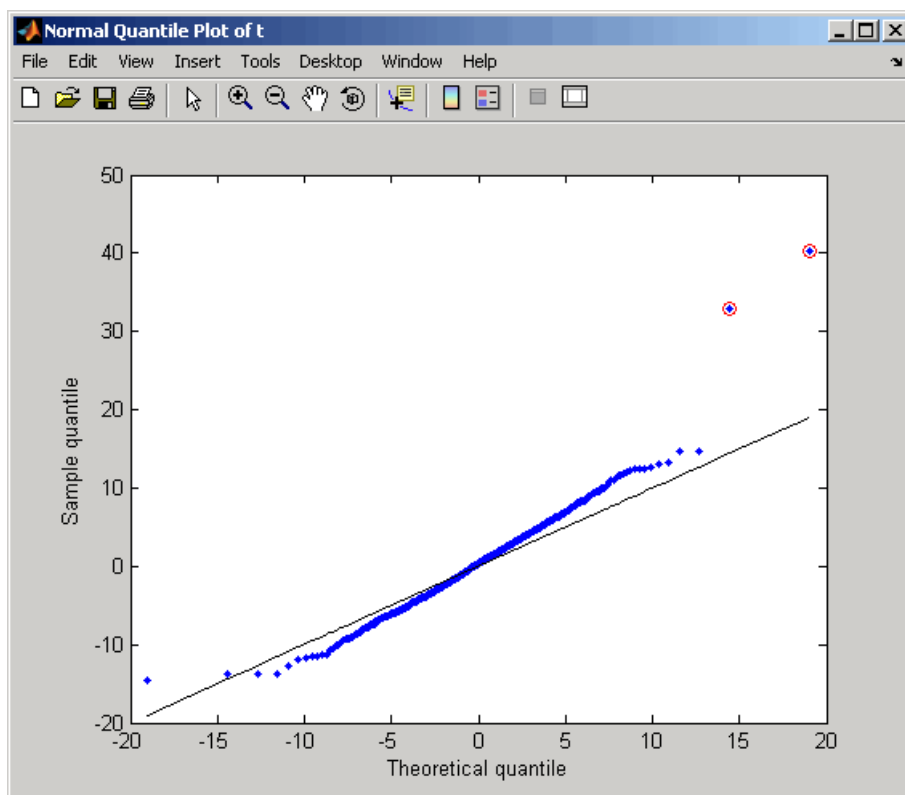
`...  = mattest(..., 'Labels', `*`LabelsValue`*`, ...)` controls the
display of labels when you click a data point in the t-score quantile plot.
LabelsValue is a cell array of labels (typically gene names or probe set
IDs) for each row in *DataX* and *DataY*.

Examples **1** Load the MAT file, included with Bioinformatics Toolbox, 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);
```

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

References   [1] 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 Suppl1*, 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   Bioinformatics Toolbox functions: `maboxplot`, `mafdr`, `mainvarsetnorm`, `mairplot`, `maloglog`, `malowess`, `manorm`, `mavolcanoplot`, `rmasummary`

Purpose   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*, ...)

Arguments

DataX	Matrix or vector of gene expression values from a single experimental condition. If *DataX* is 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* are natural scale, use the LogTrans property to convert them to log 2 scale.

DataY	Matrix or vector of gene expression values from a single experimental condition. If 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 *DataY* are natural scale, use the LogTrans property to convert them to log 2 scale.

PValues	Vector of p-values for each gene in data sets from two different experimental conditions.
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.

Description   mavolcanoplot(*DataX, DataY, PValues*) creates a scatter plot of gene expression data, plotting significance versus fold change of gene expression ratios. It uses the average gene expression values from two data sets, *DataX* and *DataY*, for each gene in the data sets. It plots significance as the $-\log_{10}$ (p-value) from the vector, *PValues*. *DataX* and *DataY* can be vectors or matrices.

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

... 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.
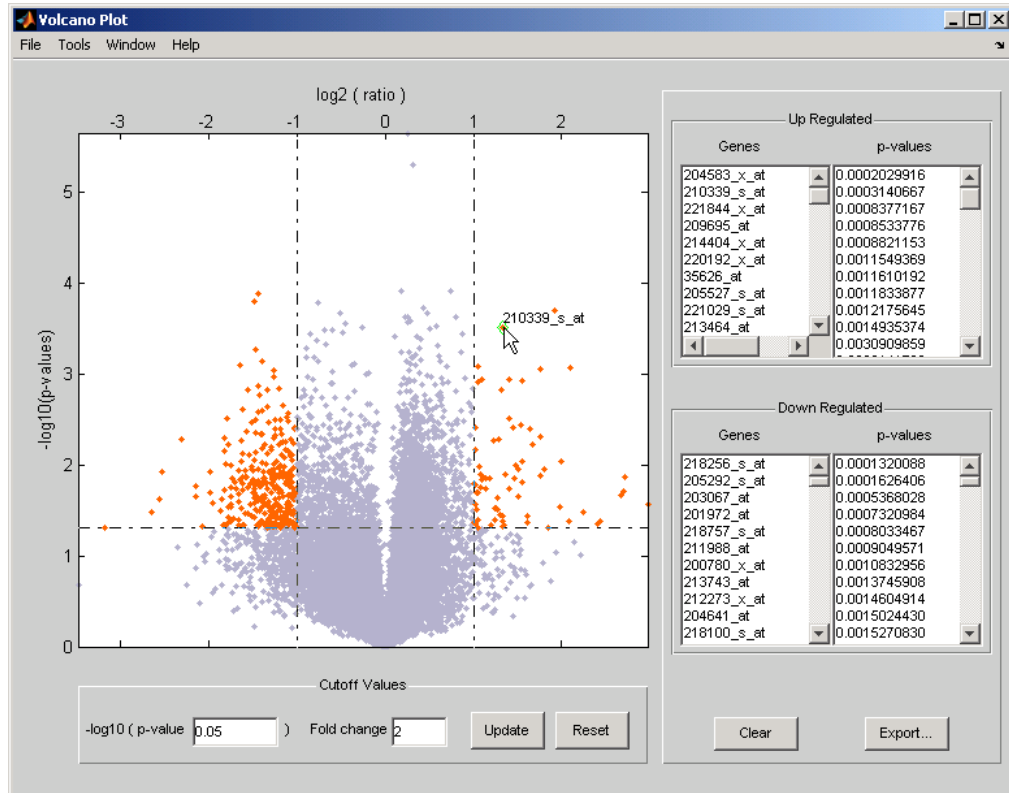
The volcano plot displays the following:

- $-\log_{10}$ (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 Bioinformatics Toolbox, 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)
```

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   Bioinformatics Toolbox functions: `maboxplot`, `maimage`, `mainvarsetnorm`, `mairplot`, `maloglog`, `malowess`, `manorm`, `mapcaplot`, `mattest`

molweight

Purpose	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 on page 2-57. 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** Get 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   Bioinformatics Toolbox functions: aacount, atomiccomp, isoelectric, proteinplot

Purpose   Display and manipulate 3-D molecule structure

Syntax
```
molviewer
molviewer(File)
molviewer(pdbID)
molviewer(pdbStruct)
FigureHandle = molviewer(...)
```

molviewer

Arguments

File	String specifying one of the following:	

- File name of a file on the MATLAB search path or in the MATLAB Current Directory

- 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.

Return Values   *FigureHandle*   Figure handle to a Molecule Viewer window.

Description   `molviewer` opens a blank Molecule Viewer window. 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 a Molecule Viewer window displaying the 3-D molecular structure for viewing and manipulation.

`molviewer(`*pdbID*`)` retrieves the data for a protein structure record, *pdbID*, from the PDB database and opens a Molecule Viewer window 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 a Molecule Viewer window 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

---

molviewer
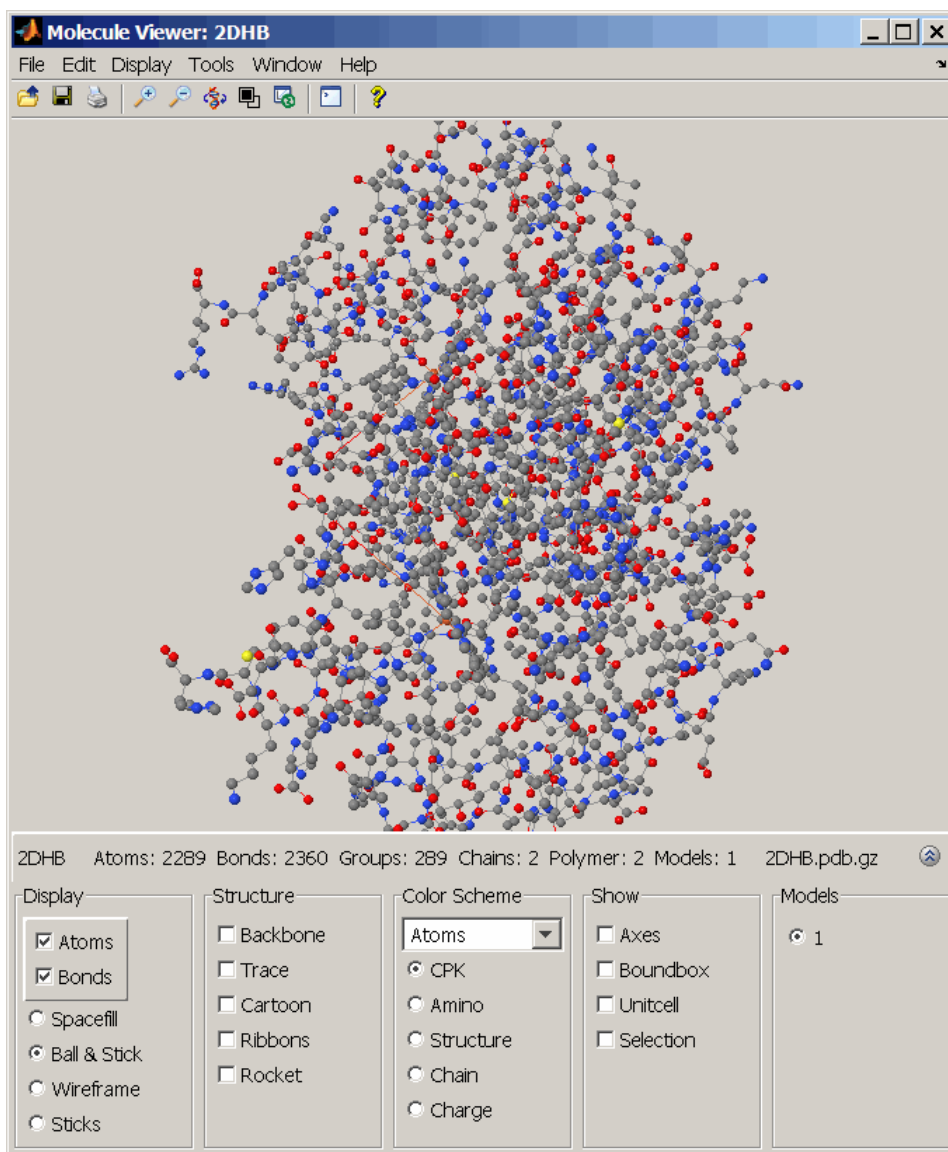
After displaying the 3-D molecule structure, you can:

- Click-drag the molecule to spin, rotate, and view it from different angles.

- Hover the mouse over a subcomponent of the molecule to display an identification label for it.

- Zoom the plot by turning the mouse scroll wheel or clicking the following buttons:

   or

- Spin the molecule by clicking .

- 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 by right-clicking to select commands:
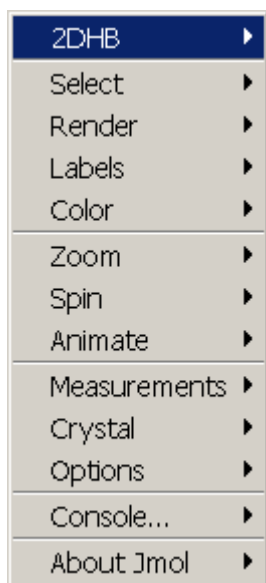
molviewer

- Display the RasMol Scripts console by clicking .
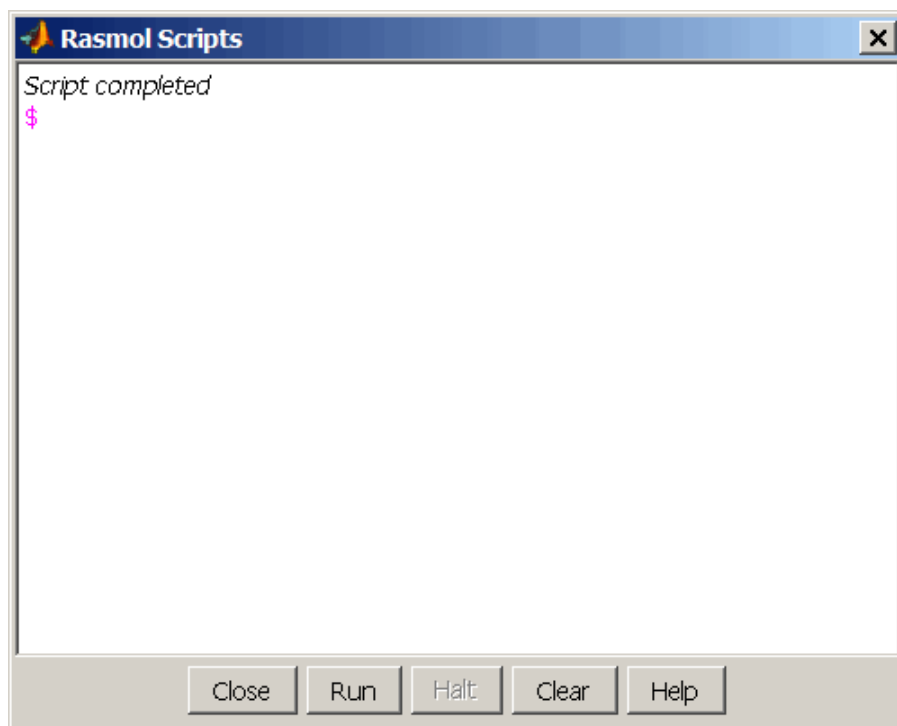
Examples   View the acetylsalicylic acid (aspirin) molecule, whose structural information is contained in the Elsevier MDL molecule file aspirin.mol.

```
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   Bioinformatics Toolbox functions: evalrasmolscript, getpdb, pdbread, pdbwrite

Purpose	Align peaks in mass spectrum to reference peaks

Syntax

```
IntensitiesOut = msalign(MZ, Intensities, RefMZ)
... = msalign(..., 'Weights', WeightsValue, ...)
... = msalign(..., 'Range', RangeValue, ...)
... = msalign(..., 'WidthOfPulses',
WidthOfPulsesValue, ...)
... = msalign(..., 'WindowSizeRatio', WindowSizeRatioValue,
  ...)
... = msalign(..., 'Iterations', IterationsValue, ...)
... = msalign(..., 'GridSteps', GridStepsValue, ...)
... = msalign(..., 'SearchSpace', SearchSpaceValue, ...)
... = msalign(..., 'ShowPlot', ShowPlotValue, ...)
[IntensitiesOut, RefMZOut] = msalign(...,
'Group', GroupValue,
  ...)
```

Arguments

MZ	Vector of mass/charge (m/z) values for a spectrum or set of spectra. The number of elements in the vector equals *n* or the number of rows in the matrix *Intensities*.
Intensities	Either of the following: • Column vector of intensity values for a spectrum, where each row corresponds to an m/z value. • Matrix of intensity values for a set of mass spectra that share the same m/z range, where each row corresponds to an m/z value, and each column corresponds to a spectrum. The number of rows equals *n* or the number of elements in vector *MZ*.

msalign

RefMZ	Vector of m/z values of known reference masses in a sample spectrum.

Tip For reference peaks, select compounds that do not undergo structural transformation, such as phosphorylation. Doing so will increase the accuracy of your alignment and allow you to detect compounds that do exhibit structural transformations among the sample spectra.

WeightsValue	Vector of positive values, with the same number of elements as *RefMZ*. The default vector is ones(size(*RefMZ*)).
RangeValue	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 m/z units, relative to each peak. No peak will shift beyond these limits. Default is [-100 100].
WidthOfPulsesValue	Positive value that specifies the width, in m/z units, for all the Gaussian pulses used to build the correlating synthetic spectrum. 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 spectrum is compared to the sample spectrum only within these regions, which saves computation time. The size of the window is given in m/z 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 spectrum over the reference masses specified by *RefMZ*. 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 return values are specified.

- true — When return values are not specified.

GroupValue	Controls the creation of *RefMZOut*, a new vector of m/z values to be used as reference masses for aligning the peaks. This vector is created by adjusting the values in *RefMZ*, based on the sample data from multiple spectra 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 spectra, and you are not confident of the m/z values used for your reference peaks in *RefMZ*. Leave *GroupValue* set to false if you are confident of the m/z values used for your reference peaks in *RefMZ*.

Return Values	*IntensitiesOut*	Either of the following: • Column vector intensity values for a spectrum, where each row corresponds to an m/z value.
		• Matrix of intensity values for a set of mass spectra that share the same mass/charge (m/z) range, where each row corresponds to an m/z value, and each column corresponds to a spectrum.
		The intensity values represent a shifting and scaling of the data.
	RefMZOut	Vector of m/z values of reference masses, calculated from *RefMZ* and the sample data from multiple spectra in *Intensities*, when *GroupValue* is set to true.

Description   *IntensitiesOut* = msalign(*MZ*, *Intensities*, *RefMZ*) aligns the peaks in a raw mass spectrum or spectra, represented by *Intensities* and *MZ*, to reference peaks, provided by *RefMZ*. First, it creates a synthetic spectrum from the reference peaks using Gaussian pulses centered at the m/z values specified by *RefMZ*. Then, it shifts and scales the m/z scale to find the maximum alignment between the input spectrum or spectra and the synthetic spectrum. (It uses an iterative multiresolution grid search until it finds the best scale and shift factors for each spectrum.) Once the new m/z scale is determined, the corrected spectrum or spectra are created by resampling their intensities at the original m/z values, creating *IntensitiesOut*, a vector or matrix of corrected intensity values. The resampling method preserves the shape of the peaks.

msalign

Note The msalign function works best with three to five reference peaks (marker masses) that you know will appear in the spectrum. 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 *MZ* vector. If using a single reference peak, you might need to only shift the *MZ* vector. To do this, use *IntensitiesOut* = interp1(*MZ*, *Intensities*, *MZ*-(*ReferenceMass*-*ExperimentalMass*)). For more information, see Aligning Mass Spectrum with One Reference Peak on page 2-517.

... = 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(..., 'Weights', *WeightsValue*, ...) specifies the relative weight for each mass in *RefMZ*, the vector of reference m/z values. *WeightsValue* is a vector of positive values, with the same number of elements as *RefMZ*. The default vector is ones(size(*RefMZ*)), 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(..., 'Range', *RangeValue*, ...) specifies the lower and upper limits of the range, in m/z units, relative to each peak. No peak will shift beyond these limits. *RangeValue* 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 m/z units, for all the Gaussian pulses used to build the correlating synthetic spectrum. The point of the peak where the Gaussian pulse reaches `60.65%` of its maximum is set to the width specified by *WidthOfPulsesValue*. Choices are any positive value. Default is `10`. *WidthOfPulsesValue* may also be a function handle. The function is evaluated at the respective m/z values and returns a variable width for the pulses. Its evaluation should give reasonable values between `0` and `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 spectrum. 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 spectrum is compared to the sample spectrum only within these regions, which saves computation time. The size of the window is given in m/z 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 spectrum over the reference masses specified by *RefMZ*. 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 return values are specified.
- `true` — When return values are not specified.

`[`*IntensitiesOut, RefMZOut*`] = msalign(..., 'Group', ` *GroupValue*`, ...)` controls the creation of *RefMZOut*, a new vector of m/z values to be used as reference masses for aligning the peaks. This vector is created by adjusting the values in *RefMZ*, based on the sample data from multiple spectra 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 spectra, and you are not confident of the m/z values used for your reference peaks in *RefMZ*. Leave *GroupValue* set to false if you are confident of the m/z values used for your reference peaks in *RefMZ*.

Examples   **Aligning Mass Spectrum with Three or More Reference Peaks**

1 Load 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')
```
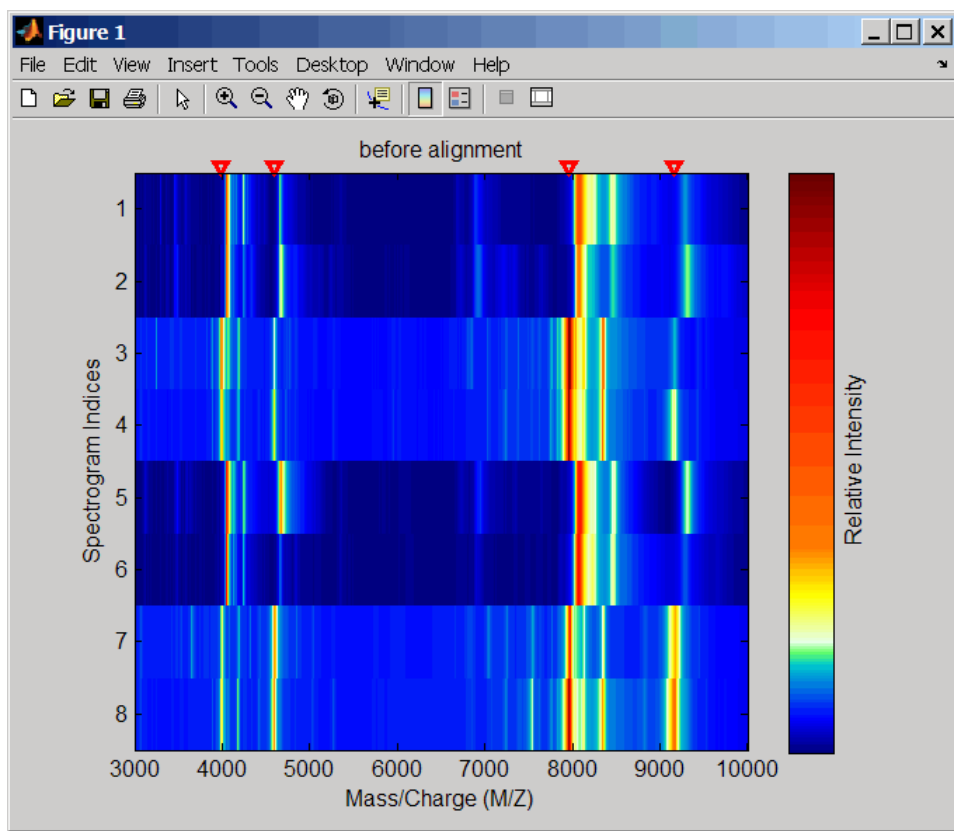
3 Align spectra with reference masses and display a color image of mass spectra after alignment.

```
YA = msalign(MZ_lo_res,Y_lo_res,R,'weights',W);
msheatmap(MZ_lo_res,YA,'markers',R,'range',[3000 10000])
title('after alignment')
```
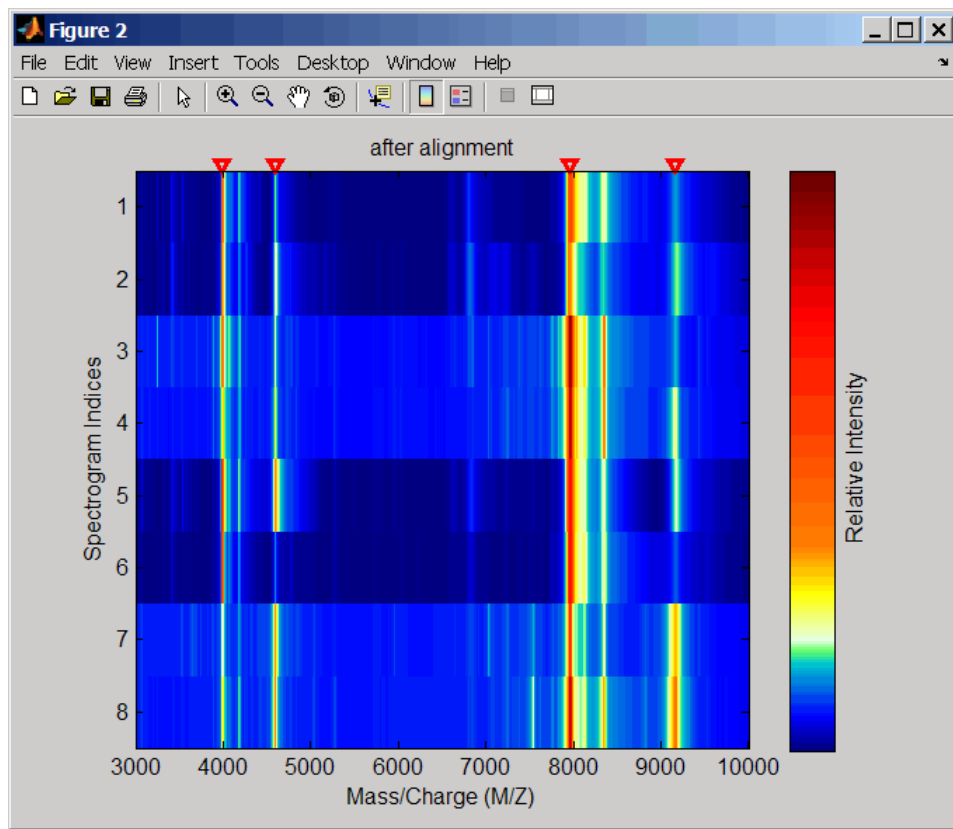
Aligning Mass Spectrum with One Reference Peak

It is not recommended to use the msalign function if you have only
one reference peak. Instead, use the following procedure, which shifts
the *MZ* 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)
```
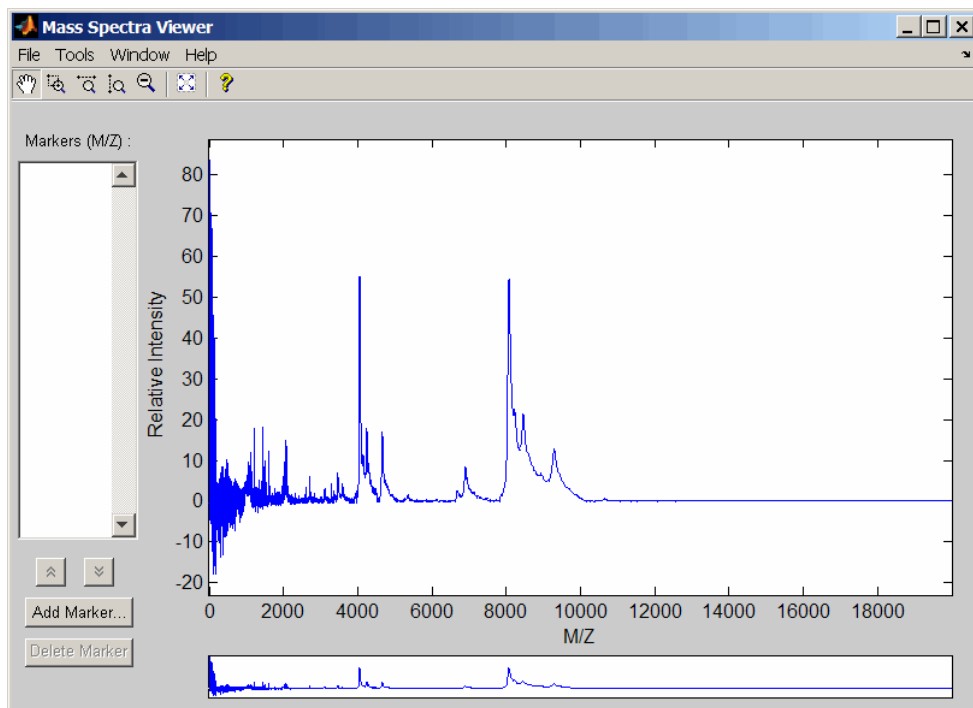

2 Use the tall peak around 4000 m/z as the reference peak. To determine the reference peak's m/z value, click [Q], and then click-drag to zoom in on the peak. Right-click in the center of the peak, and then click **Add Marker** to label the peak with its m/z value.
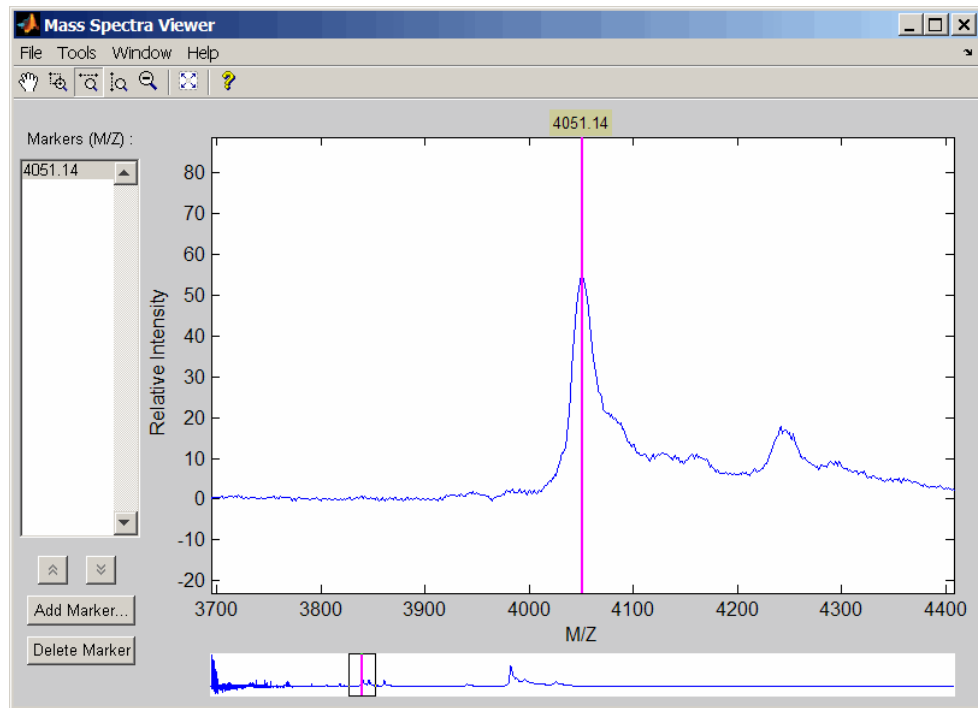
3 Shift a spectrum by the difference between RP, the known reference mass of 4000 m/z, and SP, the experimental mass of 4051.14 m/z.

```
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')
```

msalign

```
legend('Y','YOut')
axis([3600 4800 -2 60])
```
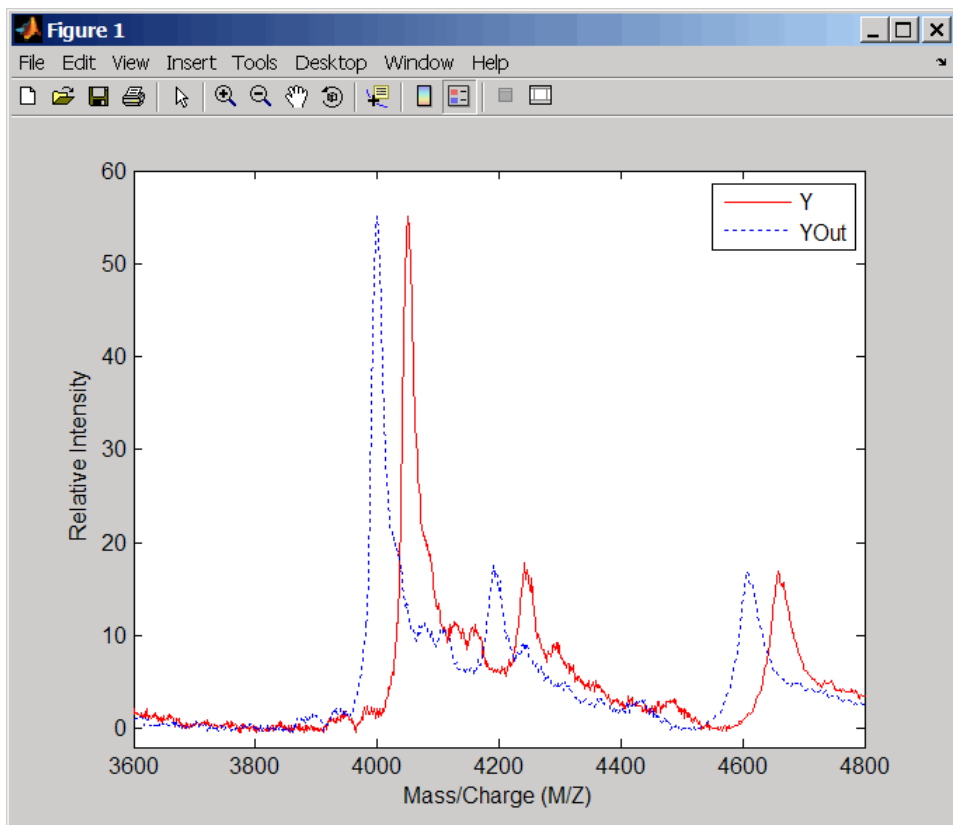

References   [1] Monchamp, P., Andrade-Cetto, L., Zhang, J.Y., and Henson, R. (2007) Signal Processing Methods for Mass Spectrometry. In Systems Bioinformatics: An Engineering Case-Based Approach, G. Alterovitz and M.F. Ramoni, eds. (Artech House Publishers).

See Also   Bioinformatics Toolbox functions: `msbackadj`, `msheatmap`, `mspalign`, `mspeaks`, `msresample`, `msviewer`

Purpose   Correct baseline of mass spectrum

Syntax
```
Yout = msbackadj(MZ, Y)
Yout = msbackadj(MZ, Y, ...'WindowSize',
WindowSizeValue, ...)
Yout = msbackadj(MZ, Y, ...'StepSize', StepSizeValue, ...)
Yout = msbackadj(MZ, Y, ...'RegressionMethod',
   RegressionMethodValue, ...)
Yout = msbackadj(MZ, Y, ...'EstimationMethod',
   EstimationMethodValue, ...)
Yout = msbackadj(MZ, Y, ...'SmoothMethod',
SmoothMethodValue,
   ...)
Yout = msbackadj(MZ, Y, ...'QuantileValue',
   QuantileValueValue, ...)
Yout = msbackadj(MZ, Y, ...'PreserveHeights',
   PreserveHeightsValue, ...)
Yout = msbackadj(MZ, Y, ...'ShowPlot', ShowPlotValue, ...)
```

Arguments

MZ	Range of mass/charge ions. Enter a vector with the range of ions in the spectra.
Y	Ion intensity vector with the same length as the mass/charge vector (*MZ*). *Y* can also be a matrix with several spectra that share the same mass/charge (*MZ*) range.

Description   Yout = msbackadj(*MZ*, *Y*) adjusts the variable baseline of a raw mass spectrum by following three steps:

1 Estimates the baseline within multiple shifted windows of width 200 m/z

2 Regresses the varying baseline to the window points using a spline approximation

3 Adjusts the baseline of the spectrum (Y)

Yout = msbackadj(*MZ*, *Y*, ...'*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(*MZ*, *Y*, ...'WindowSize', *WindowSizeValue*, ...) specifies the width for the shifting window. *WindowSizeValue* can also be a function handler. The function is evaluated at the respective MZ 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 of the spectrogram. The default value is 200 (baseline point estimated for windows with a width of 200 m/z).

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 spectrum and the presence of possible drifts. If you have wider peaks toward the end of the spectrum, you may want to use variable parameters.

Yout = msbackadj(*MZ*, *Y*, ...'StepSize', *StepSizeValue*, ...) specifies the steps for the shifting window. The default value is 200 m/z (baseline point is estimated for windows placed every 200 m/z). *StepSizeValue* may also be a function handle. The function is evaluated at the respective m/z values and returns the distance between adjacent windows.

Yout = msbackadj(*MZ*, *Y*, ...'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(*MZ*, *Y*, ...'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(*MZ*, *Y*, ...'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(*MZ*, *Y*, ...'QuantileValue', *QuantileValueValue*, ...) specifies the quantile value. The default value is 0.10.

Yout = msbackadj(*MZ*, *Y*, ...'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(*MZ*, *Y*, ...'ShowPlot', *ShowPlotValue*, ...) plots the baseline estimated points, the regressed baseline, and the original spectrum. When msbackadj is called without output arguments, the spectra are plotted unless *ShowPlotValue* is false. When *ShowPlotValue* is true, only the first spectrum in Y is plotted. *ShowPlotValue* can also contain an index to one of the spectra in Y.

Examples

1 Load 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);
```
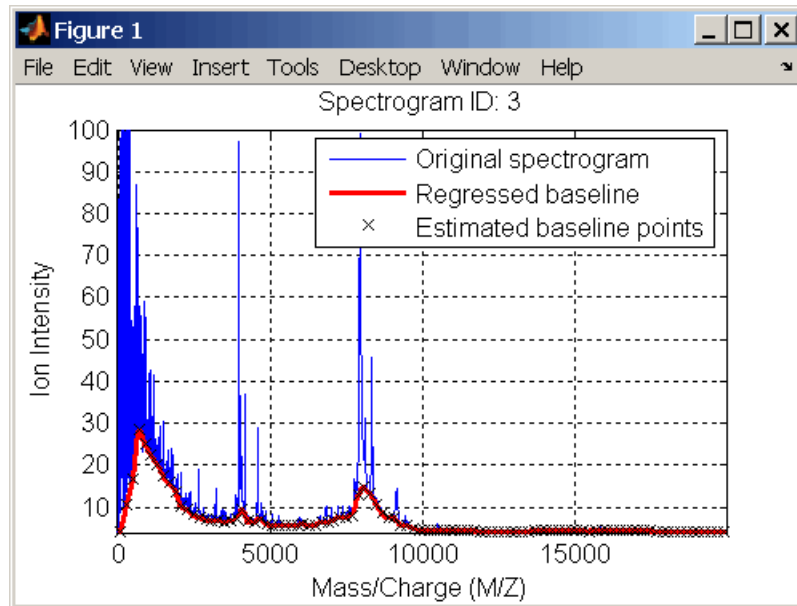

3 Plot the estimated baseline for the fourth spectrum in Y_lo_res using an anonymous function to describe an m/z dependent parameter.

```
wf = @(mz) 200 + .001 .* mz;
msbackadj(MZ_lo_res,Y_lo_res(:,4),'STEPSIZE',wf);
```
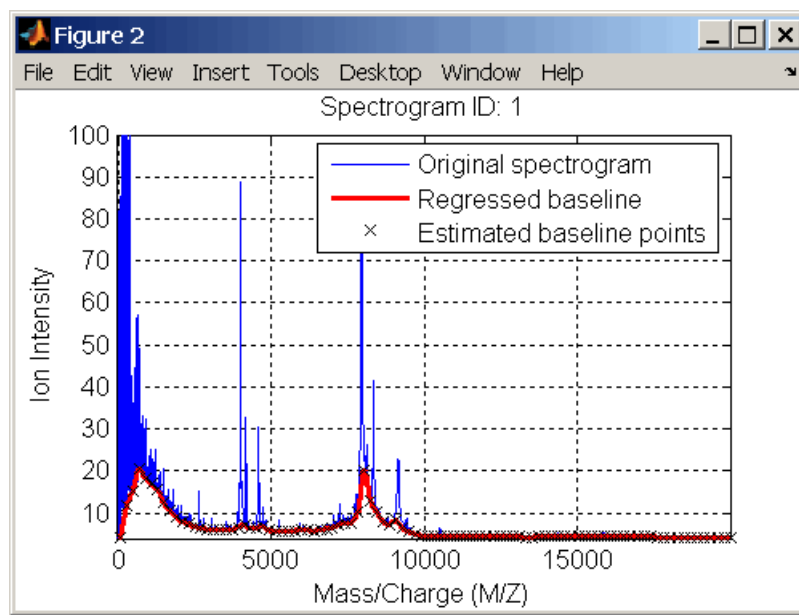
See Also   Bioinformatics Toolbox functions: `msalign`, `mslowess`, `msheatmap`, `msnorm`, `mspeaks`, `msresample`, `mssgolay`, `msviewer`

msdotplot

Purpose	Plot set of peak lists from LC/MS or GC/MS data set

Syntax

```
msdotplot(Peaks, Times)
msdotplot(FigHandle, Peaks, Times)
msdotplot(..., 'Quantile', QuantileValue)
PlotHandle = msdotplot(...)
```

Arguments

Peaks	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 *Peaks* 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 *Peaks*.

> **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 $\geq$ 0 and $\leq$ 1. Default is 0. For example, setting *QuantileValue* = 0 plots all peaks, and setting *QuantileValue* = 0.8 plots only the 20% most intense peaks.

Return Values

PlotHandle	Handle to the line series object (figure plot).

Description   msdotplot(*Peaks*, *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 *Peaks*, 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. *Peaks* 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*, *Peaks*, *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 Bioinformatics Toolbox, 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(peaks,ret_time,'Quantile',0.95)
```
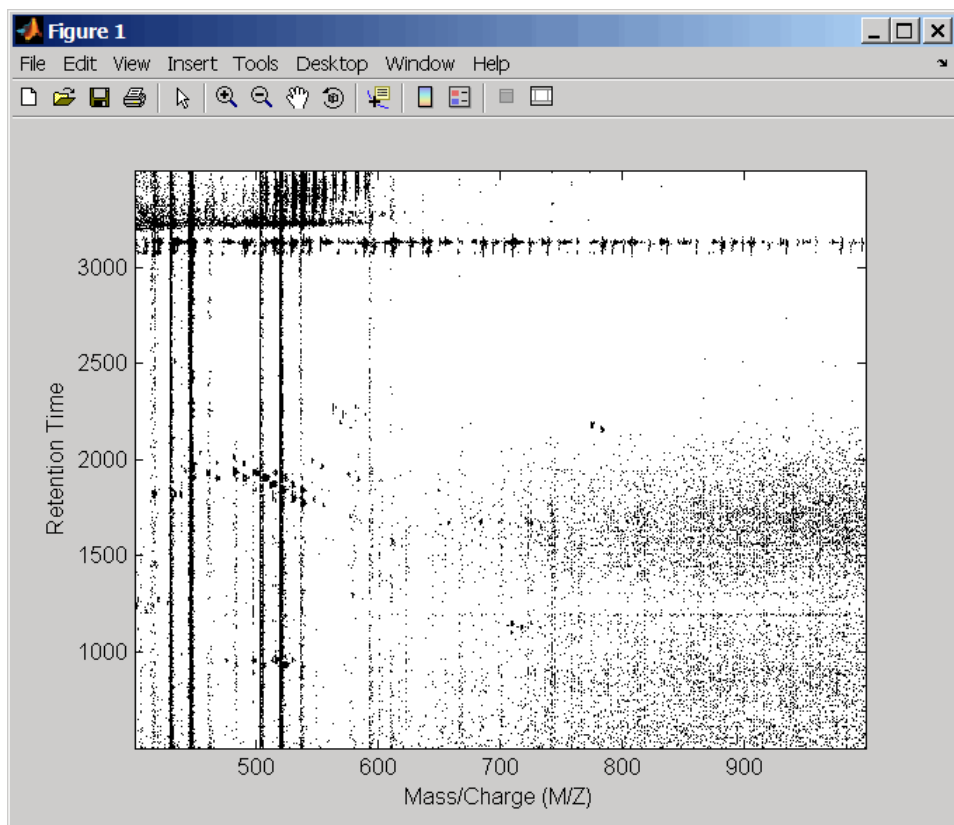

3 Resample the data, then create a heat map of the LC/MS data.

```
[MZ,Y] = msppresample(peaks,5000);
msheatmap(MZ,ret_time,log(Y))
```
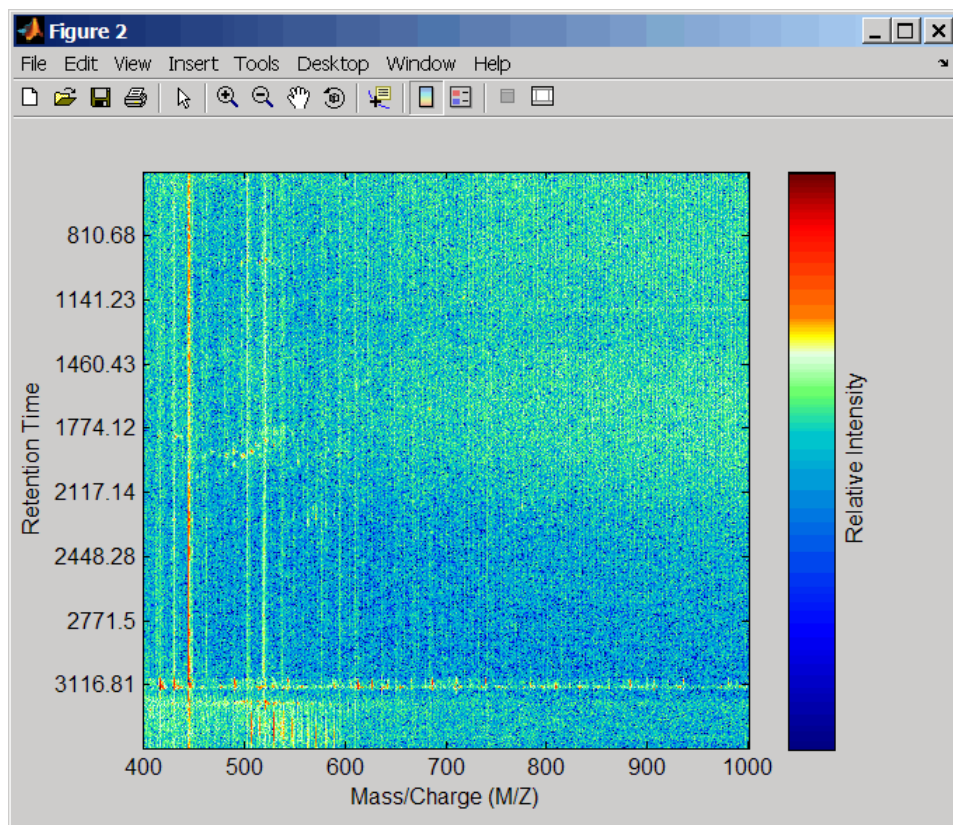

4 Overlay the dot plot on the heat map, and then zoom in to see the detail.

```
msdotplot(peaks,ret_time)
axis([480 532 375 485])
```

msdotplot
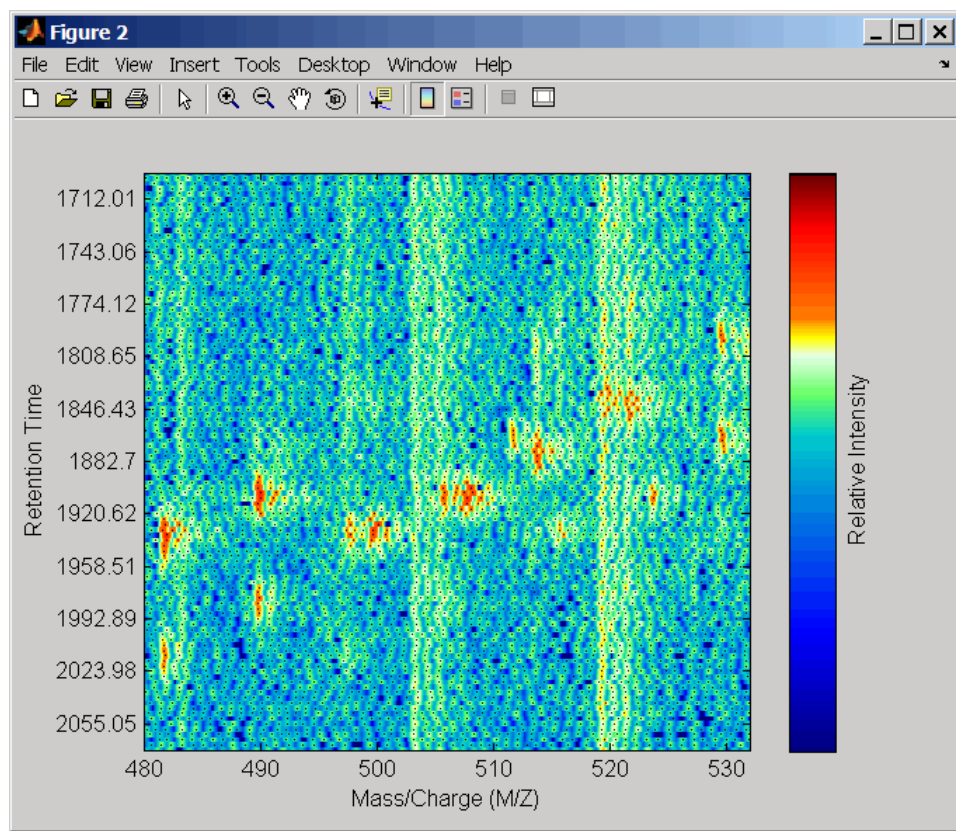
See Also   Bioinformatics Toolbox functions: msheatmap, mspalign, mspeaks, msppresample, mzxml2peaks, mzxmlread

msheatmap

Purpose   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 color map, meaning they do not represent peaks. msheatmap uses a custom color map 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 color map, meaning they do not represent peaks. msheatmap uses a custom color map 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.

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])
```
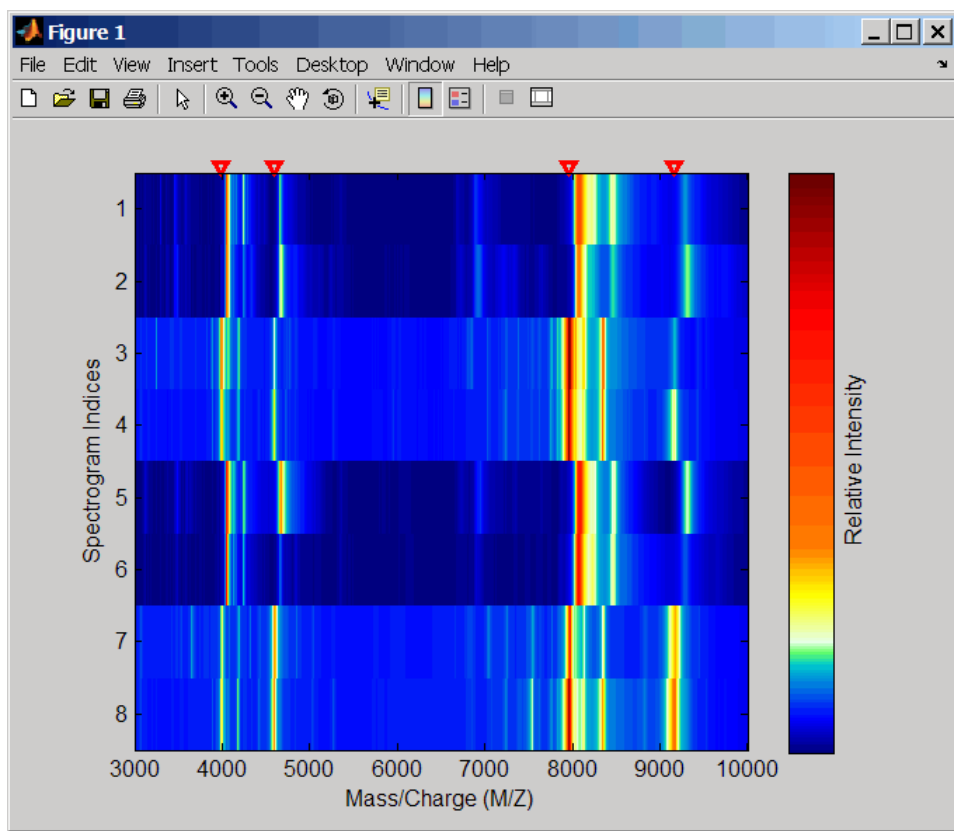
4 Display the heat map again grouping each spectrum into one of two
groups.

```
TwoGroups = [1 1 2 2 1 1 2 2];
msheatmap(MZ_lo_res,Y_lo_res,'markers',M,'group',TwoGroups)
```
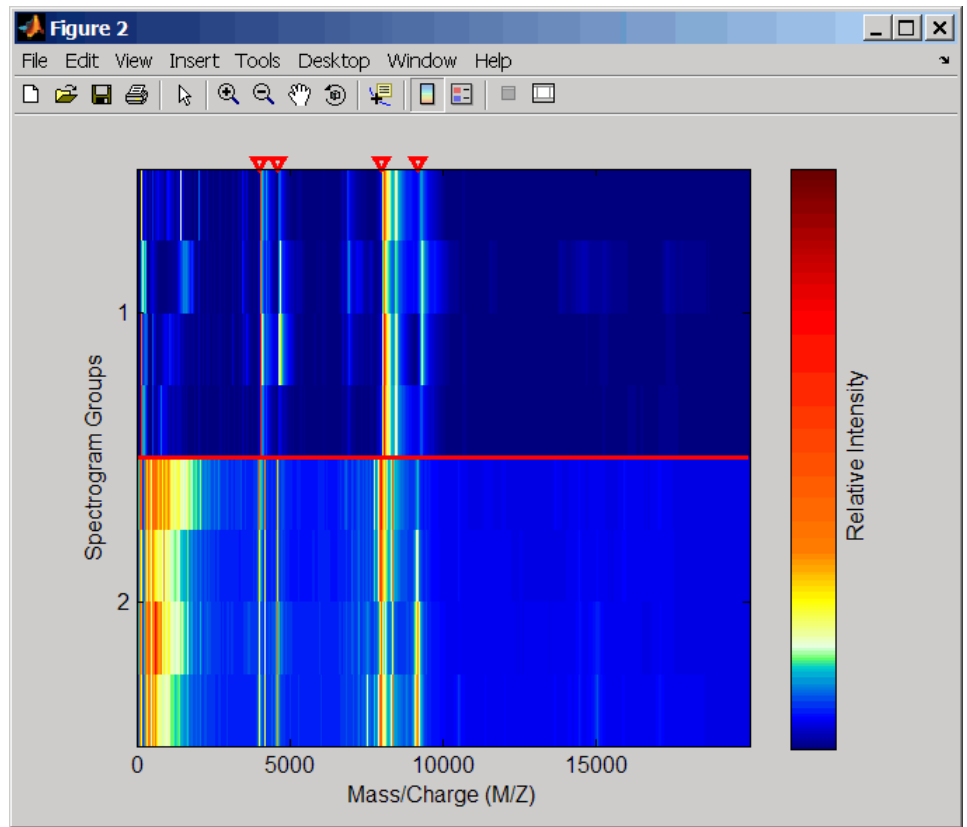
Liquid Chromatography/Mass Spectrometry (LC/MS) Data

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(peaks, 5000);
```

3 Display the heat map showing mass spectra at different retention times.

```
msheatmap(MZ, ret_time, log(Intensities))
```
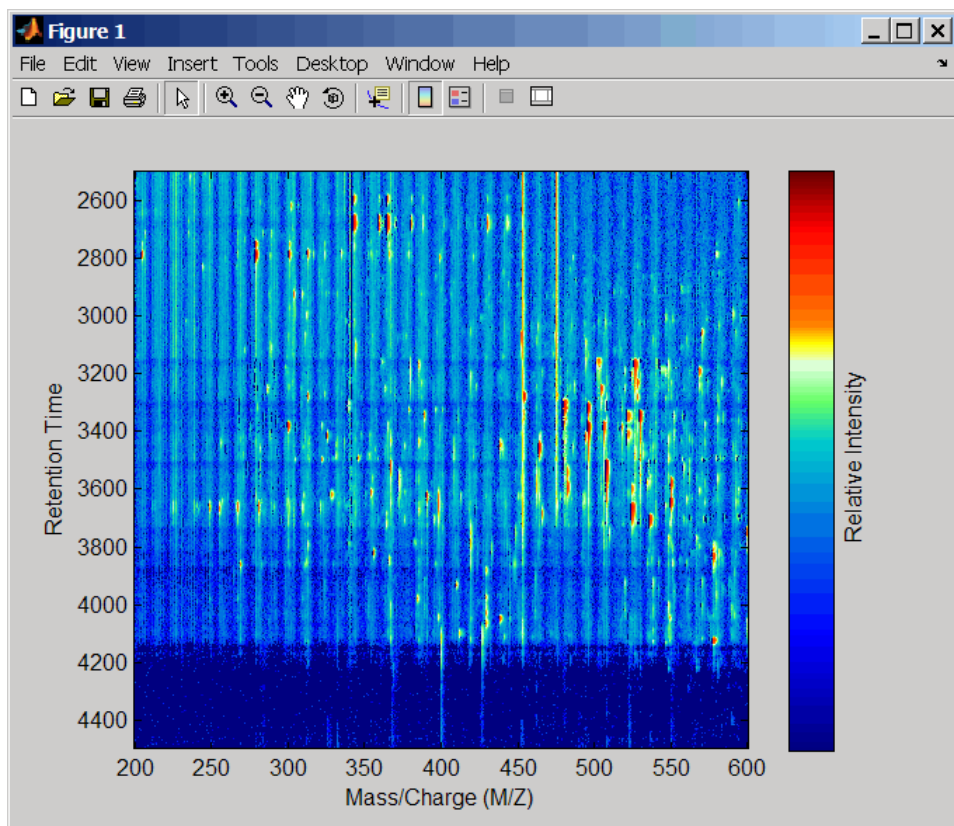

See Also   Bioinformatics Toolbox functions: msalign, msbackadj, msdotplot, mslowess, msnorm, mspalign, msresample, mssgolay, msviewer

Purpose   Smooth mass spectrum using nonparametric method

Syntax   Yout = mslowess(MZ, Y, '*PropertyName*', *PropertyValue*...)
                mslowess(..., 'Order', *OrderValue*)
                mslowess(..., 'Span', *SpanValue*)
                mslowess(..., 'Kernel', *KernelValue*)
                mslowess(..., 'RobustIterations', *RobustIterationsValue*)
                mslowess(..., 'ShowPlot', *ShowPlotValue*)

Arguments

MZ	Mass/charge vector with the range of ions in the spectra.
Y	Ion intensity vector with the same length as the mass/charge vector (MZ). Y can also be a matrix with several spectra that share the same mass/charge (MZ) range.

Description Yout = mslowess(MZ, Y, '*PropertyName*', *PropertyValue*...)
smoothes a mass spectrum (Y) using a locally weighted linear regression
(lowess) method with a default span of 10 samples.

Note 1) mslowess assumes that a mass/charge vector (MZ) might not
be uniformly spaced. Therefore, the sliding window for smoothing is
centered using the closest samples in terms of the MZ value and not
in terms of the MZ indices.

2) When the vector MZ does not have repeated values or NaNs, the
algorithm is approximately twice as fast.

---

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 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 mass/charge vector
(MZ). 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 MZ.

mslowess(..., 'Kernel', *KernelValue*) selects the function
(*KernelValue*) for weighting the observed ion intensities. Samples close
to the MZ location being smoothed have the most weight in determining
the estimate. Enter

```
'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 a uniformly spaced MZ vector, a nonrobust smoothing with Order equal to 0 is equivalent to filtering the signal with the kernel vector.

mslowess(..., 'ShowPlot', *ShowPlotValue*)plots the smoothed spectrum over the original spectrum. When mslowess is called without output arguments, the spectra are plotted unless *ShowPlotValue* is false. When *ShowPlotValue* is true, only the first spectrum in Y is plotted. *ShowPlotValue* can also contain an index to one of the spectra in Y.

Example

1 Load sample data.

```
load sample_lo_res
```

2 Smooth spectrum and draw figure with unsmoothed and smoothed spectra.

```
YS = mslowess(MZ_lo_res,Y_lo_res(:,1),'Showplot',true);
```

mslowess
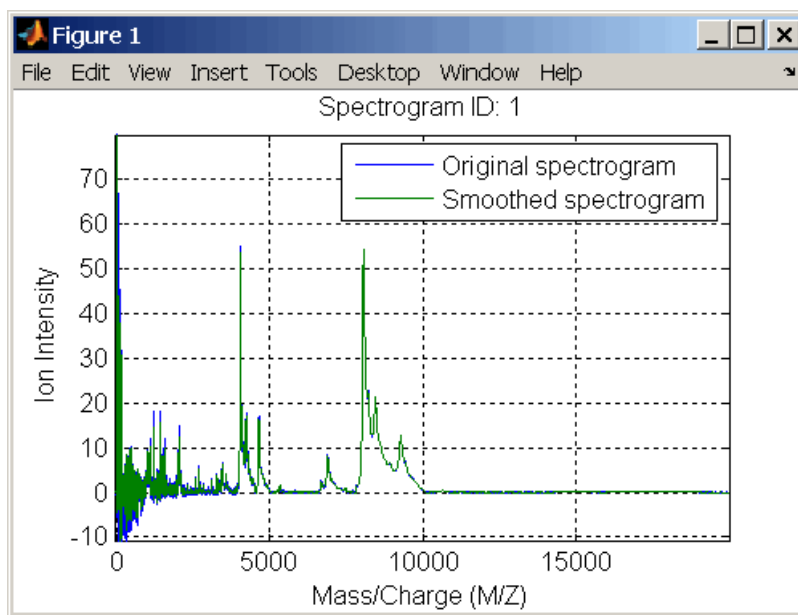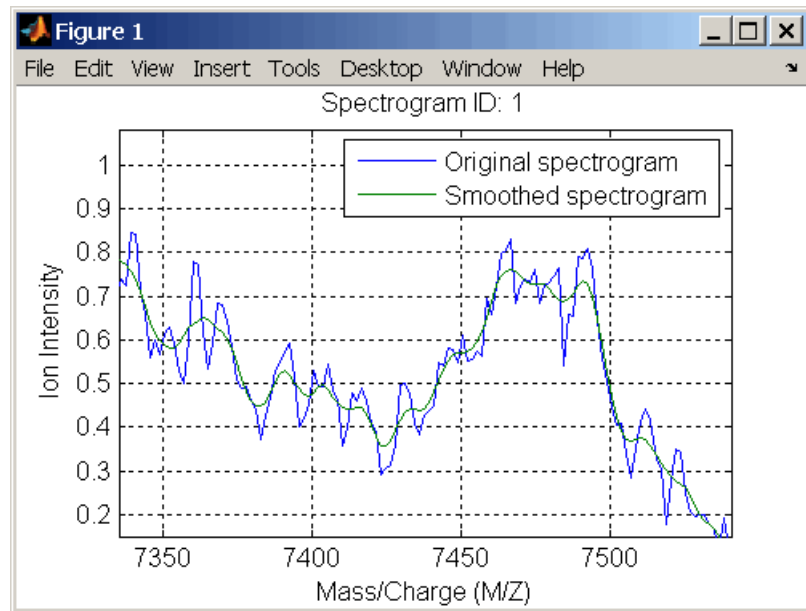
See Also   Bioinformatics Toolbox functions `msalign`, `msbackadj`, `msheatmap`, `msheatmap`,`msnorm`, `mspeaks`, `msresample`, `mssgolay`, `msviewer`

msnorm

Purpose	Normalize set of mass spectra

Syntax

Yout = msnorm(*MZ*, *Y*)
[*Yout*, *NormParameters*] = msnorm(...)
msnorm(*MZ*, *NewY*, *NormParameters*)
msnorm(..., '*PropertyName*', *PropertyValue*,...)
msnorm(..., 'Quantile', *QuantileValue*)
msnorm(..., 'Limits', *LimitsValue*)
msnorm(..., 'Consensus', *ConsensusValue*)
msnorm(..., 'Method', *MethodValue*)
msnorm(..., 'Max', *MaxValue*)

Arguments

MZ	Mass/charge vector with the range of ions in the spectra.
Y	Ion intensity vector with the same length as the mass/charge vector (*MZ*). Y can also be a matrix with several spectra that share the same mass/charge (*MZ*) range.

Description

Yout = msnorm(*MZ*, *Y*) normalizes a group of mass spectra by standardizing the area under the curve (AUC) to the group median.

[*Yout*, *NormParameters*] = msnorm(...) returns a structure with the parameters to normalize another group of spectra.

msnorm(*MZ*, *NewY*, *NormParameters*) uses the parameter information from a previous normalization (NormParameters) to normalize a new set of spectra (NewY) with the MZ 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 MZ positions are selected, and normalization is performed using the same MZ positions.

msnorm(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

msnorm(..., 'Quantile', *QuantileValue*)specifies a 1-by-2 vector with the quantile limits for reducing the set of MZ values. For example, when *QuantileValue* is [0.9 1], only the largest 10% of ion intensities in every spectrum 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 an MZ range for picking normalization points. This parameter is useful to eliminate low-mass noise from the AUC calculation. The default value is [1, max(MZ)].

msnorm(..., 'Consensus', *ConsensusValue*) selects MZ positions with a consensus rule to include an MZ position into the AUC. Its ion intensity must be within the quantile limits of at least part (*ConsensusValue*) of the spectra in Y. The same MZ positions are used to normalize all the spectrums. Enter a scalar between 0 and 1.

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 spectrum. Enter either 'Median' (default) or 'Mean'.

msnorm(..., 'Max', *MaxValue*), after individually normalizing every spectrum, scales each spectrum to an overall maximum intensity (Max). Max 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 Max.

Example 1 **1** Load sample data and plot one of the spectra.

```
load sample_lo_res;
Y = Y_lo_res(:,[1 2 5 6]);
MZ = MZ_lo_res;
plot(MZ, Y(:, 4));
```
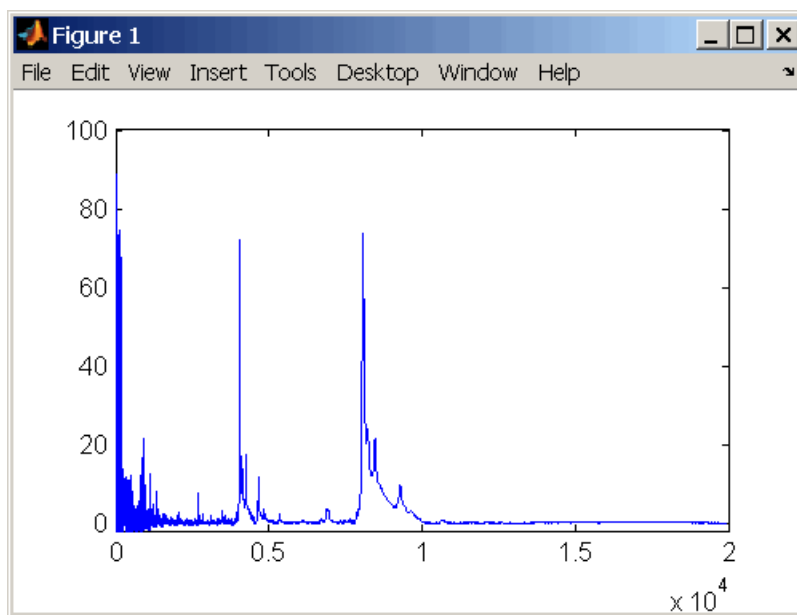
2 Normalize the AUC of every spectrum to its median, eliminating low-mass noise, and post-rescaling such that the maximum intensity is 100.

```
Y1 = msnorm(MZ,Y,'Limits',[1000 inf],'Max',100);
plot(MZ, Y1(:, 4));
```
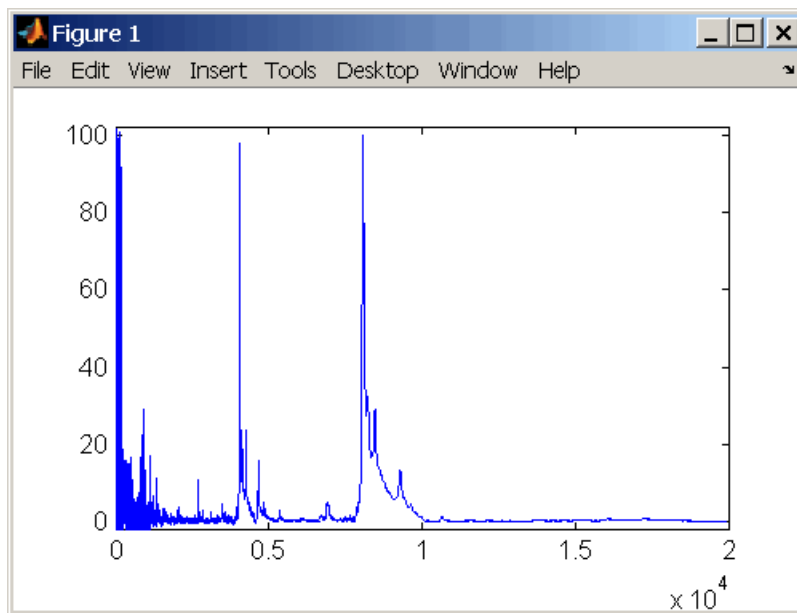
3 Normalize the ion intensity of every spectrum to the maximum intensity of the single highest peak from any of the spectra in the range above 100 m/z.

```
Y2 = msnorm(MZ,Y,'QUANTILE', [1 1],'LIMITS',[1000 inf]);
```

Example 2 **1** Select MZ regions where the intensities are within the third quartile in at least 90% of the spectrograms.

```
[Y3,S] = msnorm(MZ,Y,'Quantile',[0.5 0.75],'Consensus',0.9);
```

2 Use the same MZ regions to normalize another set of spectrograms.

```
Y4 = msnorm(MZ,Y,S);
```

See Also Bioinformatics Toolbox functions msalign, msbackadj, msheatmap, mslowess, msresample, mssgolay, msviewer

mspalign

Purpose	Align mass spectra from multiple peak lists from LC/MS or GC/MS data set
Syntax	[*CMZ*, *AlignedPeaks*] = mspalign(*Peaks*) [*CMZ*, *AlignedPeaks*] = mspalign(*Peaks*, ...'Quantile', *QuantileValue*, ...) [*CMZ*, *AlignedPeaks*] = mspalign(*Peaks*, ...'EstimationMethod', *EstimationMethodValue*, ...) [*CMZ*, *AlignedPeaks*] = mspalign(*Peaks*, ...'CorrectionMethod', *CorrectionMethodValue*, ...) [*CMZ*, *AlignedPeaks*] = mspalign(*Peaks*, ...'ShowEstimation', *ShowEstimationValue*, ...)

Arguments

Peaks	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 *Peaks* 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.

mspalign

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.

Return Values

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 *Peaks*, but with corrected m/z values in the first column of each matrix.

Description [*CMZ*, *AlignedPeaks*] = mspalign(*Peaks*) 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 *Peaks*, 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(*Peaks*, ...*'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(*Peaks*, ...'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(*Peaks*, ...'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(*Peaks*, ...'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.

mspalign

• 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(*Peaks*, ...'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 Bioinformatics Toolbox, 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(peaks,5000);
msheatmap(MZ,ret_time,log(Y))
```
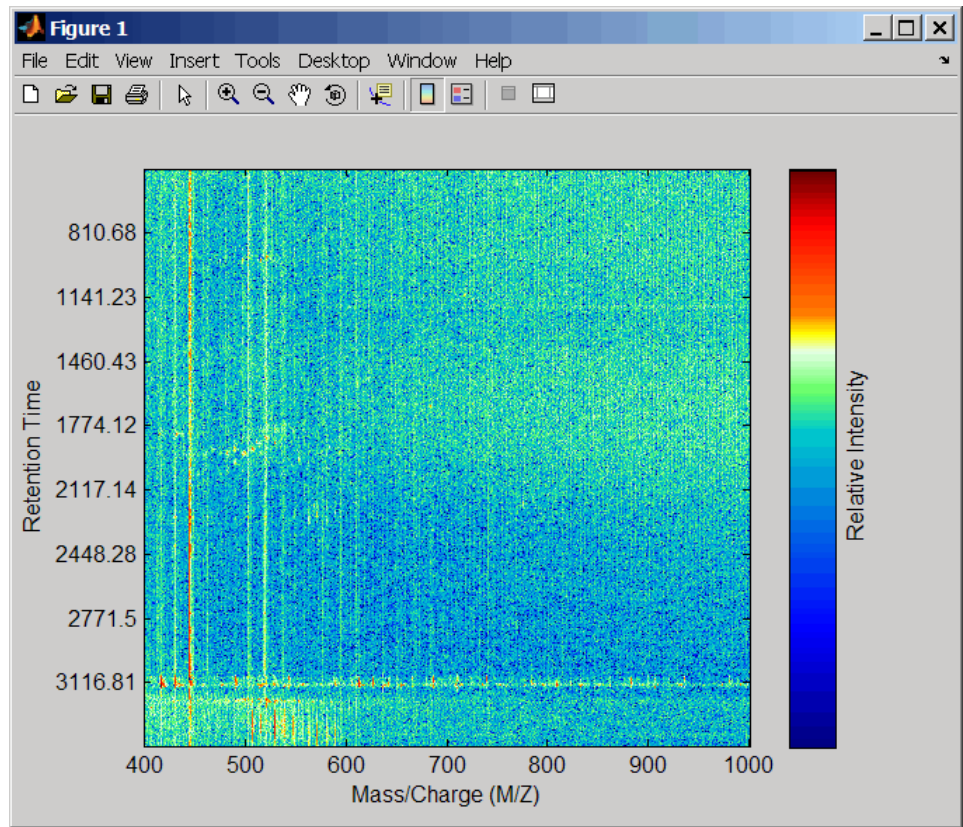
```
msdotplot(peaks,ret_time)
```

3 Align the peak lists from the mass spectra using the default estimation and correction methods.

```
[CMZ, aligned_peaks] = mspalign(peaks);
```

4 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))
```
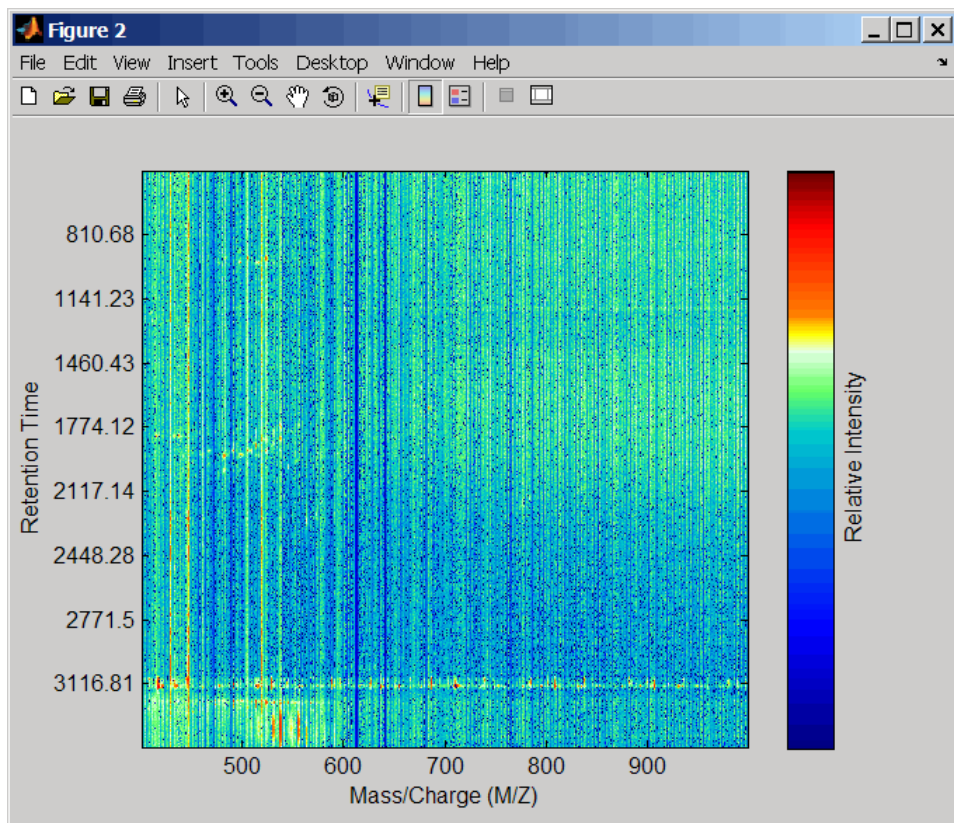


```
msdotplot(aligned_peaks,ret_time)
```

5 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])
```

mspalign
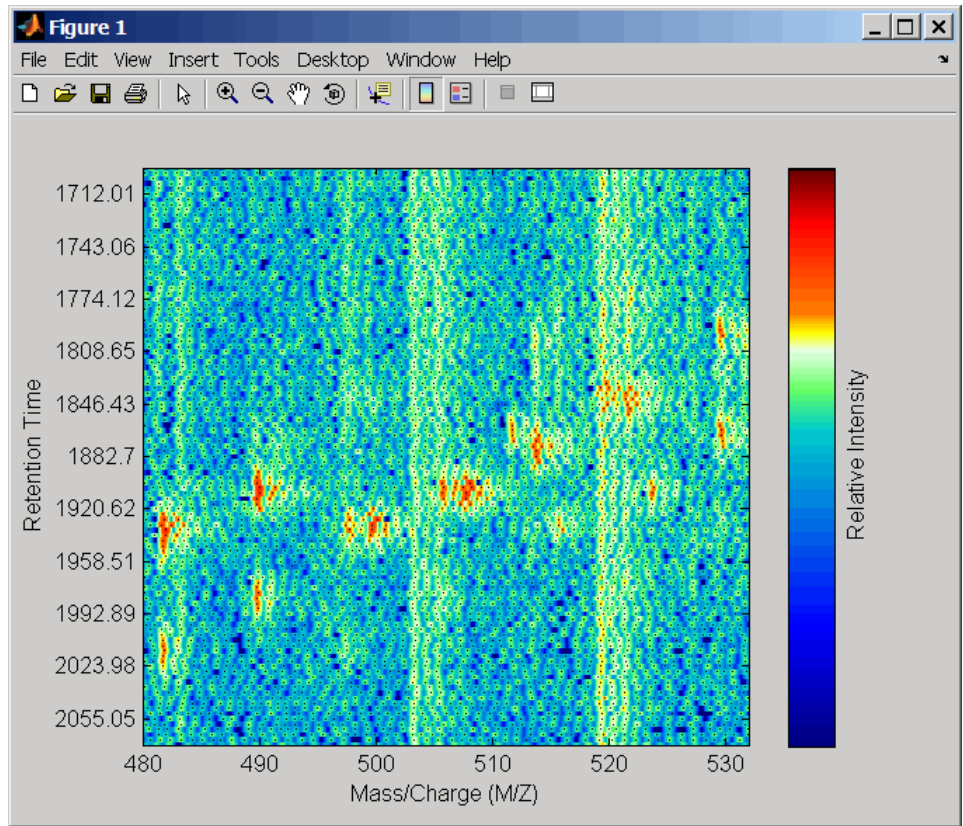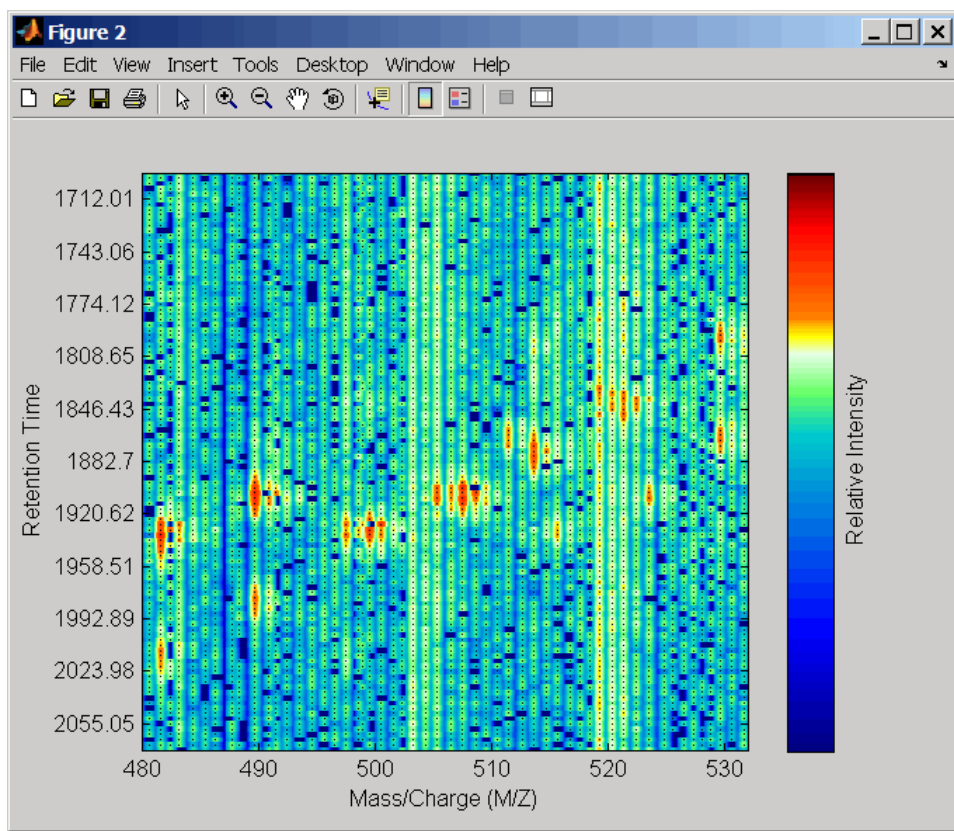
References   [1] Jeffries, N. (2005) Algorithms for alignment of mass spectrometry proteomic data. Bioinfomatics *21:14*, 3066–3073.

[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   Bioinformatics Toolbox functions: `msalign`, `msdotplot`, `msheatmap`, `mspeaks` , `msppresample`, `mzxml2peaks`

Purpose   Convert raw mass spectrometry data to peak list (centroided data)

Syntax   *Peaks* = mspeaks(*MZ*, *Intensities*)
Peaks = mspeaks(*MZ*, *Intensities*, ...'Base', *BaseValue*, ...)
Peaks = mspeaks(*MZ*, *Intensities*, ...'Levels', *LevelsValue*, ...)
Peaks = mspeaks(*MZ*, *Intensities*, ...'NoiseEstimator', *NoiseEstimatorValue*, ...)
Peaks = mspeaks(*MZ*, *Intensities*, ...'Multiplier', *MultiplierValue*, ...)
Peaks = mspeaks(*MZ*, *Intensities*, ...'Denoising', *DenoisingValue*, ...)
Peaks = mspeaks(*MZ*, *Intensities*, ...'PeakLocation', *PeakLocationValue*, ...)
Peaks = mspeaks(*MZ*, *Intensities*, ...'FWHHFilter', *FWHHFilterValue*, ...)
Peaks = mspeaks(*MZ*, *Intensities*, ...'OverSegmentationFilter', *OverSegmentationFilterValue*, ...)
Peaks = mspeaks(*MZ*, *Intensities*, ...'HeightFilter', *HeightFilterValue*, ...)
Peaks = mspeaks(*MZ*, *Intensities*, ...'ShowPlot', *ShowPlotValue*, ...)

mspeaks

Arguments

MZ	Vector of mass/charge (m/z) values for a set of spectra. The number of elements in the vector equals *n* or the number of rows in matrix *Intensities*.
Intensities	Matrix of intensity values for a set of mass spectra that share the same mass/charge (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 *n* or the number of elements in vector *MZ*.
BaseValue	An integer between 2 and 20 that specifies the wavelet base. Default is 4.
LevelsValue	An integer between 1 and 12 that specifies the number of levels for the wavelet decomposition. Default is 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 = sqrt(2*log(*n*))*mad(*y_h*) / 0.6745, where *n* = the number of rows in the *Intensities* matrix.

- std — Standard deviation, which calculates T = std(*y_h*).

- A positive real value.

MultiplierValue	A positive real value that specifies the threshold multiplier constant. Default is 1.0.
DenoisingValue	Controls the use of wavelet denoising to smooth the signal. Choices are true (default) or false.

Note If your data has previously been smoothed, for example, with the mslowess or mssgolay function, it is not necessary to use wavelet denoising. Set this property to false.

---

mspeaks

PeakLocationValue	Value that specifies the proportion of the peak height that selects the points used to compute the centroid mass of the respective peak. The value must be ≥ 0 and ≤ 1. Default is 1.0.
FWHHFilterValue	Positive real value that specifies the minimum full width at half height (FWHH), in m/z units, for reported peaks. Peaks with FWHH below this value are not included in the output list *Peaks*. Default is 0.
OverSegmentationFilterValue	Positive real value that specifies the minimum distance, in m/z units, between neighboring peaks. When the signal is not smoothed appropriately, multiple maxima can appear to represent the same peak. By increasing this filter value, oversegmented peaks are joined into a single peak. Default is 0.

HeightFilterValue		Positive real value that specifies the minimum height for reported peaks. Default is 0.
ShowPlotValue		Controls the display of a plot of the original and the smoothed signal, with the peaks included in the output matrix *Peaks* 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 return values are specified.

- true — When return values are not specified.

Return Values

Peaks	Two-column matrix where each row corresponds to a peak. The first column contains mass/charge (m/z) values, and the second column contains ion intensity values.

Description   *Peaks* = mspeaks(*MZ*, *Intensities*) finds relevant peaks in raw mass spectrometry data, and creates *Peaks*, a two-column matrix, containing the m/z value and ion intensity for each peak.

mspeaks finds peaks by first smoothing the signal using undecimated wavelet transform with Daubechies coefficients, then assigning peak locations, and lastly, eliminating peaks that do not satisfy specified criteria.

Peaks = mspeaks(*MZ*, *Intensities*, ...'*PropertyName*', *PropertyValue*, ...) calls mspeaks with optional properties that

mspeaks

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:

Peaks = mspeaks(*MZ*, *Intensities*, ...'Base', *BaseValue*, ...) specifies the wavelet base. *BaseValue* must be an integer between 2 and 20. Default is 4.

Peaks = mspeaks(*MZ*, *Intensities*, ...'Levels', *LevelsValue*, ...) specifies the number of levels for the wavelet decomposition. *LevelsValue* must be an integer between 1 and 12. Default is 10.

Peaks = mspeaks(*MZ*, *Intensities*, ...'NoiseEstimator', *NoiseEstimatorValue*, ...) 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 = sqrt(2*log(*n*))*mad(*y_h*) / 0.6745, where *n* = the number of rows in the *Intensities* matrix.

- std — Standard deviation, which calculates T = std(*y_h*).

- A positive real value.

Peaks = mspeaks(*MZ*, *Intensities*, ...'Multiplier', *MultiplierValue*, ...) specifies the threshold multiplier constant. *MultiplierValue* must be a positive real value. Default is 1.0.

Peaks = mspeaks(*MZ*, *Intensities*, ...'Denoising', *DenoisingValue*, ...) controls the use of wavelet denoising to smooth the signal. Choices are true (default) or false.

Note If your data has previously been smoothed, for example, with the `mslowess` or `mssgolay` function, it is not necessary to use wavelet denoising. Set this property to `false`.

Peaks = mspeaks(*MZ*, *Intensities*, ...'PeakLocation', *PeakLocationValue*, ...) specifies the proportion of the peak height that selects the points used to compute the centroid mass of the respective peak. *PeakLocationValue* must be a value ≥ 0 and ≤ 1. Default is 1.0.

Note When *PeakLocationValue* = 1.0, the peak location is exactly at the maximum of the peak, while when *PeakLocationValue* = 0, the peak location is computed with all the points from the closest minimum to the left of the peak to the closest minimum to the right of the peak.

Peaks = mspeaks(*MZ*, *Intensities*, ...'FWHHFilter', *FWHHFilterValue*, ...) specifies the minimum full width at half height (FWHH), in m/z units, for reported peaks. Peaks with FWHH below this value are not included in the output list *Peaks*. *FWHHFilterValue* must be a positive real value. Default is 0.

Peaks = mspeaks(*MZ*, *Intensities*, ...'OverSegmentationFilter', *OverSegmentationFilterValue*, ...) specifies the minimum distance, in m/z units, between neighboring peaks. When the signal is not smoothed appropriately, multiple maxima can appear to represent the same peak. By increasing this filter value, oversegmented peaks are joined into a single peak. *OverSegmentationFilterValue* must be a positive real value. Default is 0.

Peaks = mspeaks(*MZ*, *Intensities*, ...'HeightFilter', *HeightFilterValue*, ...) specifies the minimum height for reported

peaks. Peaks with heights below this value are not included in the output list *Peaks*. *HeightFilterValue* must be a positive real value. Default is 0.

Peaks = mspeaks(*MZ, Intensities*, ...'ShowPlot', *ShowPlotValue*, ...) controls the display of a plot of the original and the smoothed signal, with the peaks included in the output matrix *Peaks* 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 either:

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

Examples

1 Load a MAT file, included with Bioinformatics Toolbox, which contains mass spectrometry data variables, 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 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);
```
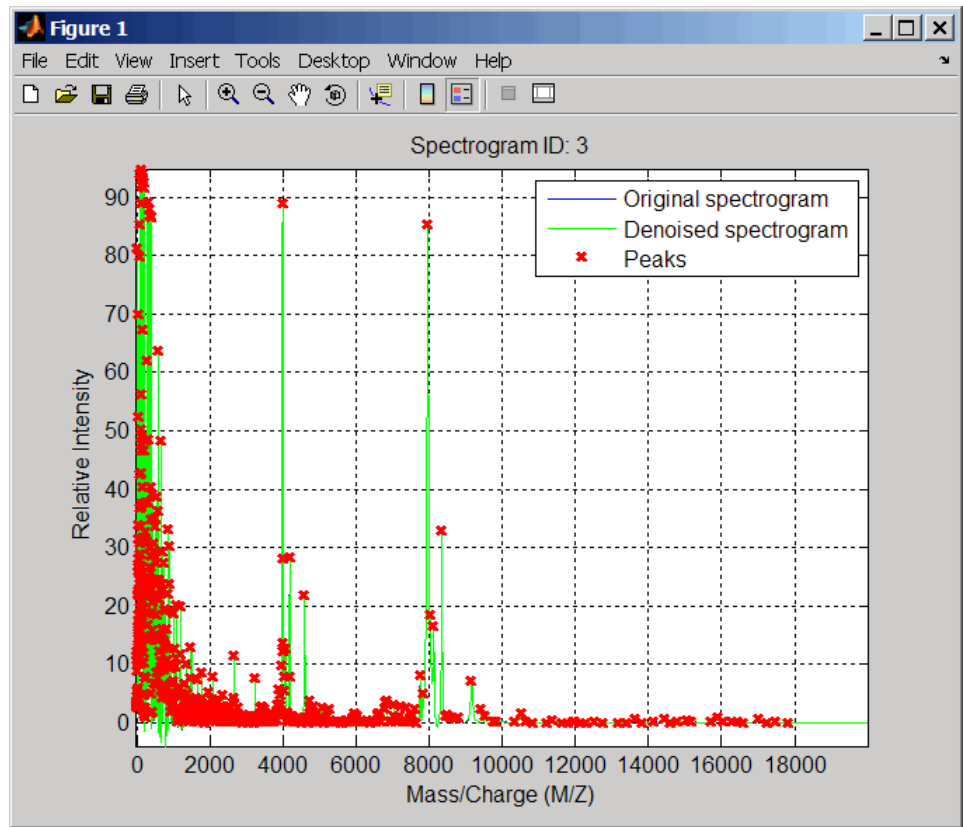
5 Smooth the signal using the `mslowess` function. Then convert the smoothed data to a peak list by finding relevant peaks and plot the third spectrum.

```
YS = mslowess(MZ_lo_res,YB,'SHOWPLOT',3);
```
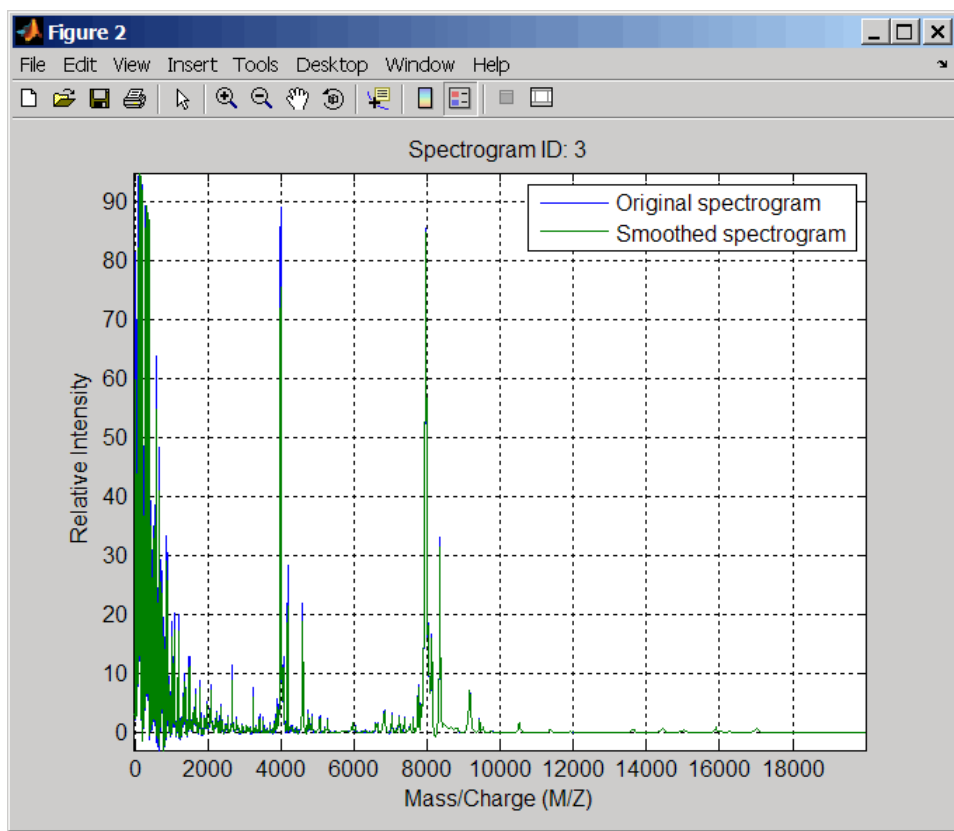
```
P = mspeaks(MZ_lo_res,YS,'DENOISING',false,'SHOWPLOT',3);
```
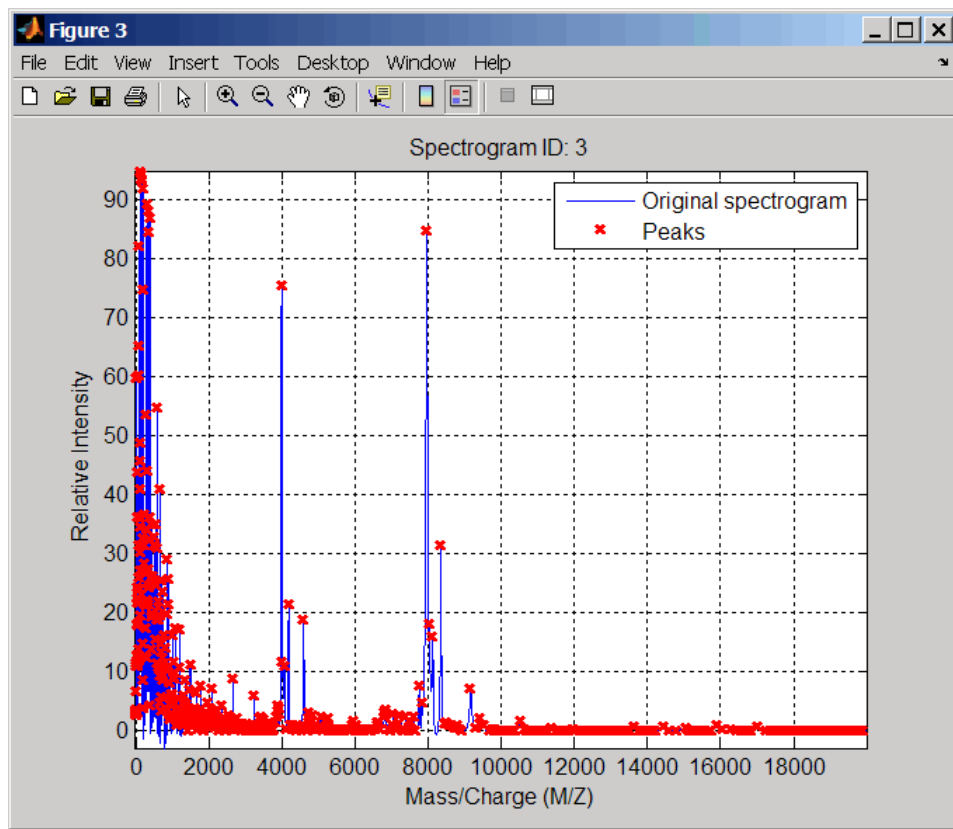
6 Use the cellfun function to remove all peaks with m/z values less than 2000 from the eight peaks lists in output P. Then plot the peaks of the third spectrum (in red) over its smoothed signal (in blue).

```
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])
```

mspeaks
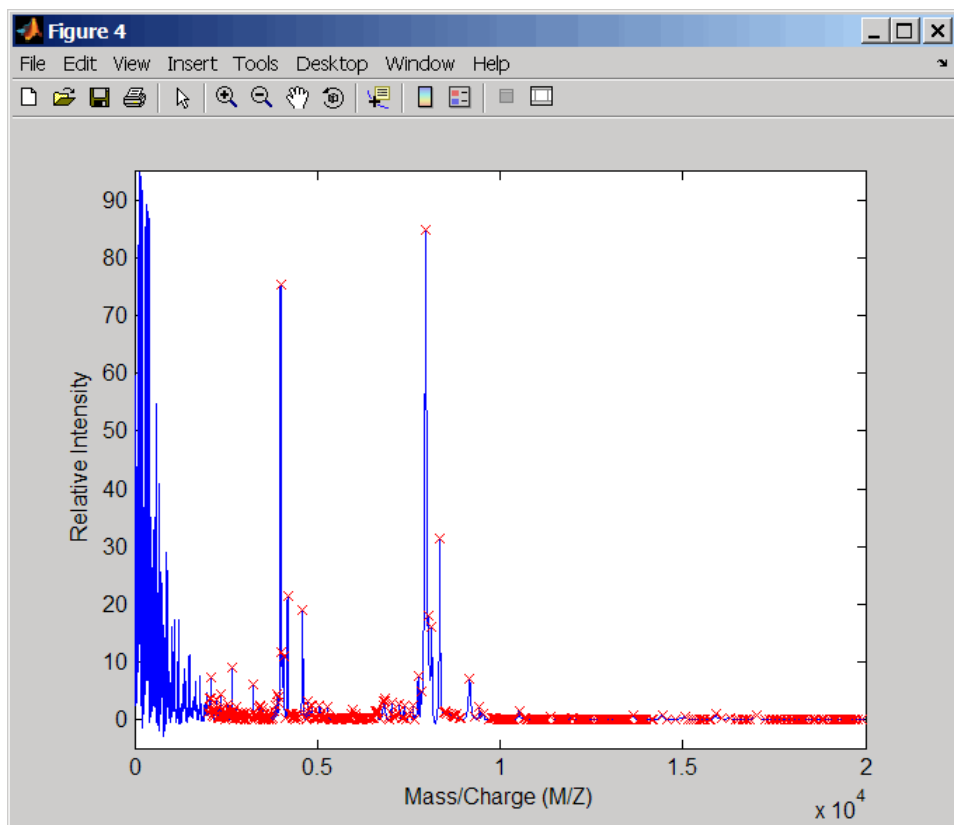
References   [1] Morris, J.S., Coombes, K.R., Koomen, J., Baggerly, K.A., and Kobayash, R. (2005) Feature extraction and quantification for mass spectrometry in biomedical applications using the mean spectrum. Bioinfomatics *21:9*, 1764–1775.

[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   Bioinformatics Toolbox functions: `msbackadj`, `msdotplot`, `mslowess`, `mspalign`, `msppresample`, `mssgolay`

msppresample

Purpose	Resample mass spectrometry signal while preserving peaks

Syntax	[*MZ*, *Intensities*] = msppresample(*Peaks*, *N*)
	[*MZ*, *Intensities*] = msppresample(*Peaks*, *N*,
	...'Range', *RangeValue*, ...)
	[*MZ*, *Intensities*] = msppresample(*Peaks*, *N*, ...'FWHH',
	    FWHHValue, ...)
	[*MZ*, *Intensities*] = msppresample(*Peaks*, *N*, ...'ShowPlot',
	    ShowPlotValue, ...)

Arguments

Peaks	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.

> **Note** You can use the mzxml2peaks function or the mspeaks function to create the *Peaks* matrix or cell array.

N	Integer specifying the number of equally spaced points (m/z values) in the resampled signal.
RangeValue	1-by-2 vector specifying the minimum and maximum m/z values for the output matrix *Intensities*. *RangeValue* must be within [min(*inputMZ*) max(*inputMZ*)], where *inputMZ* is the concatenated m/z values from the input *Peaks*. Default is the full range [min(*inputMZ*) max(*inputMZ*)].

FWHHValue   Value that specifies the full width at half height (FWHH) in m/z units. The FWHH is used to convert each peak to a Gaussian shaped curve. Default is `median(diff(`*inputMZ*`))/2`, where *inputMZ* is the concatenated m/z values from the input *Peaks*. The default is a rough approximation of resolution observed in the input data, *Peaks*.

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 spectrum. 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 return values are specified.

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

Return Values

MZ   Vector of equally spaced, common mass/charge (m/z) values for a set of spectra. 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 mass spectra that share the same mass/charge (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 *N* or the number of elements in vector *MZ*.

msppresample

Description [*MZ*, *Intensities*] = msppresample(*Peaks*, *N*) resamples *Peaks*, a mass spectrometry peak list, by converting centroided peaks to a semicontinuous, raw signal that preserves peak information. The resampled signal has *N* equally spaced points. Output *MZ* is a vector of *N* elements specifying the equally spaced, common m/z values for the spectra. Output *Intensities* is a matrix of reconstructed 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 *N*.

msppresample uses a Gaussian kernel to reconstruct the signal. The ion intensity at any given m/z value is taken from the maximum intensity of any contributing (overlapping) peaks.

Tip msppresample is useful to prepare a set of spectra for imaging functions such as msheatmap and preprocessing functions such as msbackadj and msnorm.

[*MZ*, *Intensities*] = msppresample(*Peaks*, *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:

[*MZ*, *Intensities*] = msppresample(*Peaks*, *N*, ...'Range', *RangeValue*, ...) specifies an m/z range for the output matrix *Intensities* using the minimum and maximum m/z values specified in the 1-by-2 vector *RangeValue*. *RangeValue* must be within [min(*inputMZ*) max(*inputMZ*)], where *inputMZ* is the concatenated m/z values from the input *Peaks*. Default is the full range [min(*inputMZ*) max(*inputMZ*)]

[*MZ*, *Intensities*] = msppresample(*Peaks*, *N*, ...'FWHH', *FWHHValue*, ...) sets the full width at half height (FWHH) in m/z units. The FWHH is used to convert each peak

to a Gaussian shaped curve. Default is median(diff(*inputMZ*))/2, where *inputMZ* is the concatenated m/z values from the input *Peaks*. The default is a rough approximation of resolution observed in the input data, *Peaks*.

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.

[*MZ*, *Intensities*] = msppresample(*Peaks*, *N*, ...'ShowPlot', *ShowPlotValue*, ...) controls the display of a plot of an original and resampled spectrum. 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 return values are specified.

- true — When return values are not specified.

Examples

1 Load a MAT file, included with Bioinformatics Toolbox, which contains liquid chromatography/mass spectrometry (LC/MS) data variables, including 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(peaks,5000);
msheatmap(MZ,ret_time,log(Y))
```
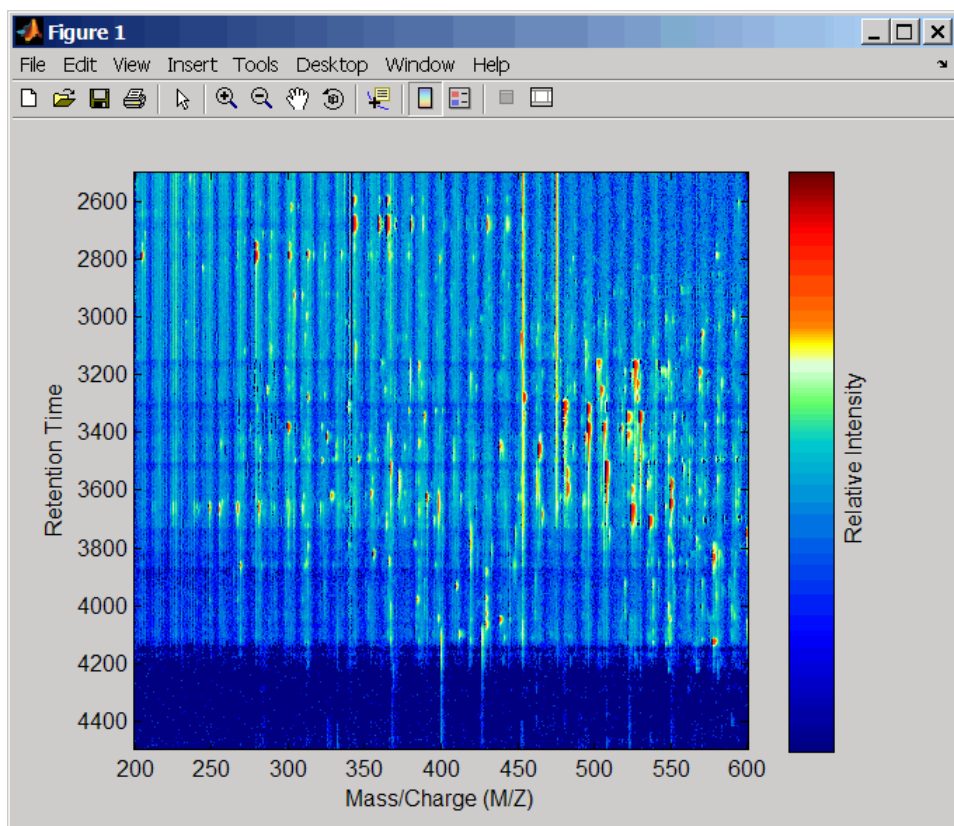
3 Plot the reconstructed profile spectra between two retention times.

```
figure
t1 = 3370;
t2 = 3390;
h = find(ret_time>t1 & ret_time<t2);
[MZ,Y] = msppresample(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')
```
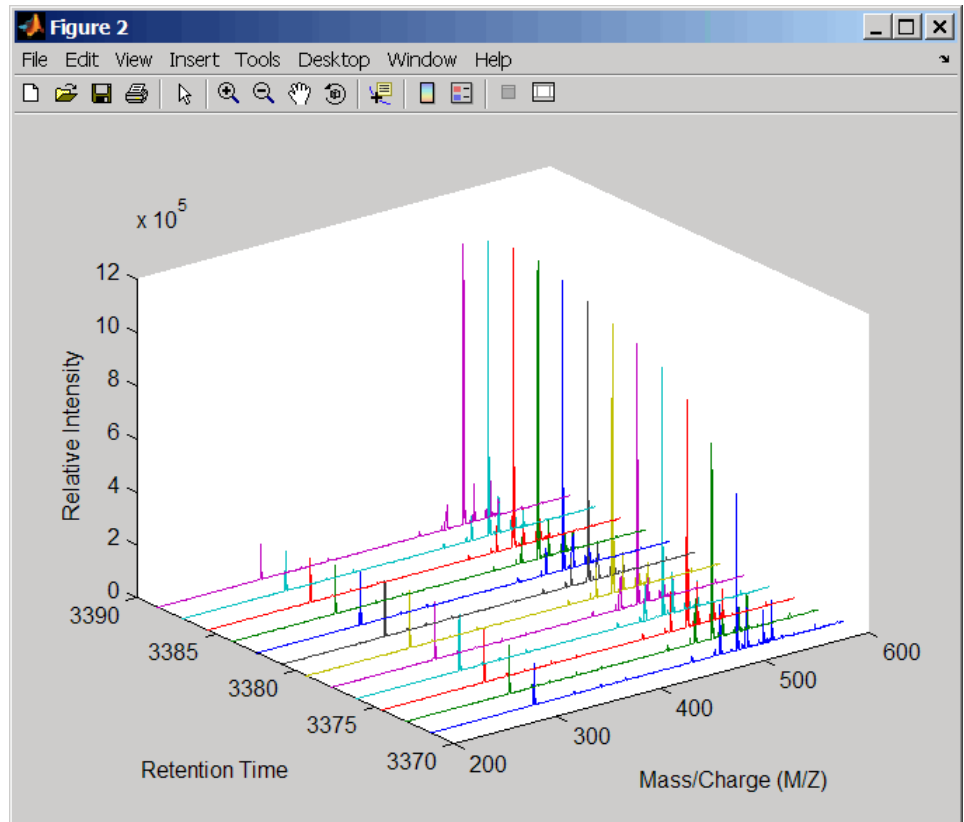

4 Resample the data to plot the Total Ion Chromatogram (TIC).

```
figure
[MZ,Y] = msppresample(peaks,5000);
plot(ret_time,sum(Y))
title('Total Ion Chromatogram (TIC)')
xlabel('Retention Time')
ylabel('Relative Intensity')
```
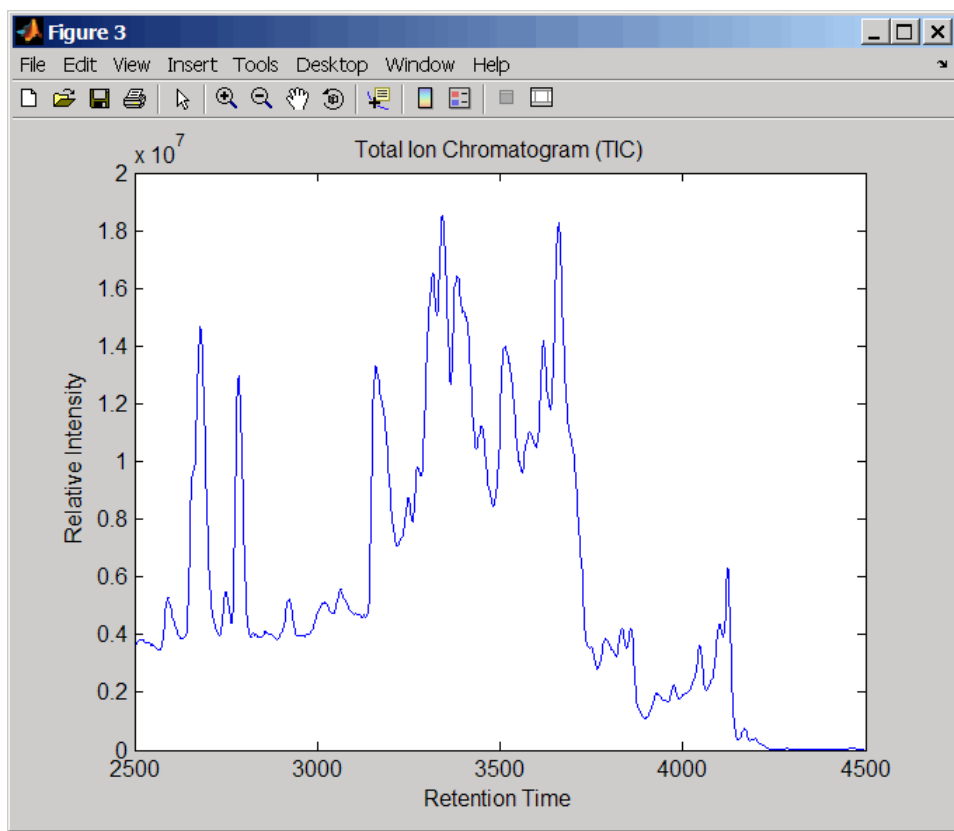
5 Resample the data to plot the Extracted Ion Chromatogram (XIC) in the 450 to 500 m/z range.

```
figure
[MZ,Y] = msppresample(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')
```
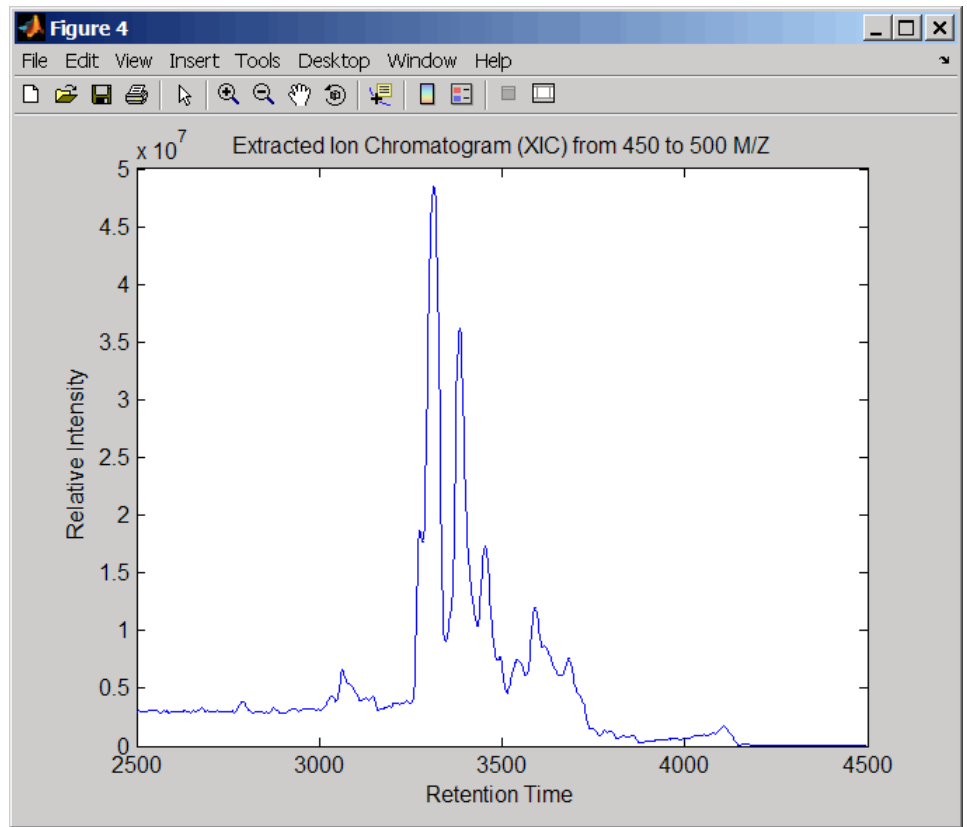
See Also   Bioinformatics Toolbox functions: `msdotplot`, `mspeaks`, `mspalign`, `msresample`, `mzxml2peaks`, `mzxmlread`

msresample

Purpose	Resample mass spectrometry signal

Syntax

```
[MZout, Yout] = msresample(MZ, Y, N)
msresample(..., 'PropertyName', PropertyValue,...)
msresample(..., 'Uniform', UniformValue)
msresample(..., 'Range', RangeValue)
msresample(..., 'Missing', MissingValue)
msresample(..., 'Window', WindowValue)
msresample(..., 'Cutoff', CutoffValue)
msresample(..., 'ShowPlot', ShowPlotValue)
```

Arguments

MZ	Mass/charge vector with the range of ions in the spectra.
Y	Ion intensity vector with the same length as the mass/charge vector (*MZ*). *Y* can also be a matrix with several spectra that share the same mass/charge (*MZ*) range.
N	Total number of samples.

Description   [*MZout*, *Yout*] = msresample(*MZ*, *Y*, *N*) resamples a raw mass spectrum (*Y*). The output spectrum will have *N* samples with a spacing that increases linearly within the range [min(*MZ*) max(*MZ*)]. *MZ* can be a linear or a quadratic function of its index. When input arguments are set 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 cu-off frequency is set by the largest down-sampling ratio when comparing the same regions in the *MZ* and *MZout* vectors.

Note msresample is particularly useful when you have spectra with different mass/charge vectors and you want to match the scales.

---

msresample(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

msresample(..., 'Uniform', *UniformValue*), when *UniformValue* is true, forces the vector *MZ* to be uniformly spaced. The default value is false.

msresample(..., 'Range', *RangeValue*) specifies a 1-by-2 vector with the mass/charge range for the output spectrum (Yout). *RangeValue* must be within [min(*MZ*) max(*MZ*)]. The default value is the full range [min(*MZ*) max(*MZ*)].

msresample(..., 'Missing', *MissingValue*), when *MissingValue* is true, analyzes the mass/charge vector (*MZ*) 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 MZ values follow a linear or a quadratic function of the *MZ* 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 between 0 and 1 (Nyquist frequency or half the sampling frequency). By default, msresample estimates the cutoff value by inspecting the mass/charge vectors (*MZ*, MZout). However, the cutoff frequency might be underestimated if *MZ* has anomalies.

msresample(..., 'ShowPlot', *ShowPlotValue*) plots the original and the resampled spectrum. When msresample is called without output arguments, the spectra are plotted unless *ShowPlotValue* is false. When *ShowPlotValue* is true, only the first spectrum in Y is plotted. *ShowPlotValue* can also contain an index to one of the spectra in Y.

Examples   1 Load mass spectrometry data and extract m/z and intensity value vectors

```
load sample_hi_res;
mz = MZ_hi_res;
y = Y_hi_res;
```

2 Plot original data to a lower resolution.

```
plot(mz, y, '.')
```

MATLAB draws a figure.
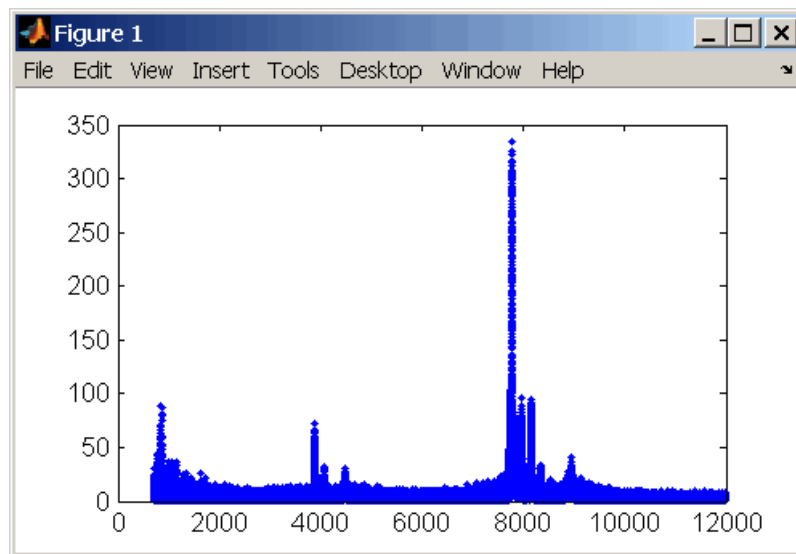
3 Resample data

```
[mz1,y1] = msresample(mz, y, 10000, 'range',[2000 max(mz)]);
```

4 Plot resampled data

```
plot(mz1,y1,'.')
```

MATLAB draws a figure with the down sampled data.
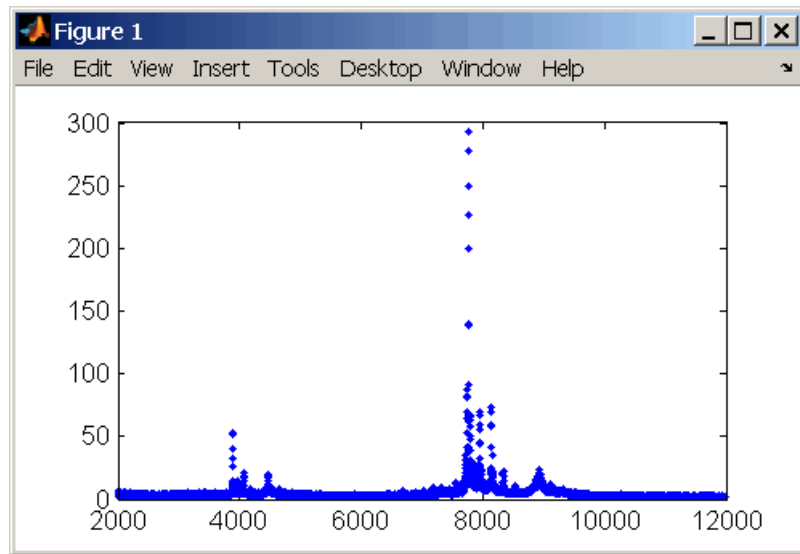
See Also Bioinformatics Toolbox functions: msalign, msbackadj, msheatmap, mslowess, msnorm, msppresample, mssgolay, msviewer

mssgolay

Purpose	Smooth mass spectrum with least-squares polynomial
Syntax	*Yout* = mssgolay(*MZ, Y*) mssgolay(..., '*PropertyName*', *PropertyValue*,...) mssgolay(..., 'Span', *SpanValue*) mssgolay(..., 'Degree', *DegreeValue*) mssgolay(..., 'ShowPlot', *ShowPlotValue*)

Arguments

MZ	Mass/charge vector with the range of ions in the spectra.
Y	Ion intensity vector with the same length as the mass/charge vector (*MZ*). *Y* can also be a matrix with several spectra that share the same mass/charge (*MZ*) range.

Description   *Yout* = mssgolay(*MZ, Y*) smoothes a raw mass spectrum (*Y*) using a least squares digital polynomial filter (Savitzky and Golay filters). The default span or frame is 15 samples.

mssgolay(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

mssgolay(..., '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 *MZ* vector. Higher values will 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 data (*MZ*). For example, if *SpanValue* is 0.05, the window size is equal to 5% of the number of points in *MZ*.

> **Note** 1) The original algorithm by Savitzky and Golay assumes a uniformly spaced mass/charge vector (*MZ*), 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 *MZ* value and not in terms of the *MZ* index.
>
> 2) When the vector *MZ* does not have repeated values or NaNs, the algorithm is approximately twice as fast.
>
> 3) When the vector *MZ* is evenly spaced, the least-squares fitting is performed once so that the spectrum is filtered with the same coefficients, and the speed of the algorithm increases considerably.
>
> 4) If the vector *MZ* is evenly spaced and *SpanValue* is even, Span is incriminated by 1 to include both edge samples in the frame.

mssgolay(..., '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(..., 'ShowPlot', *ShowPlotValue*) plots smoothed spectra over the original. When mssgolay is called without output arguments, the spectra are plotted unless *ShowPlotValue* is false. When *ShowPlotValue* is true, only the first spectrum in *Y* is plotted. *ShowPlotValue* can also contain an index to one of the spectra in *Y*.

Examples
```
load sample_lo_res
YS = mssgolay(MZ_low_res, Y_low_res(:,1));
plot(MZ,[Y(:,1) YS])
```

See Also   Bioinformatics Toolbox functions msalign, msbackadj, msheatmap, mslowess, msnorm, mspeaks, msresample, msviewer

msviewer

Purpose	Explore mass spectrum or set of mass spectra

Syntax	`msviewer(MZ, Y)`
	`msviewer(..., 'Markers', MarkersValue)`
	`msviewer(..., 'Group', GroupValue)`

Arguments

MZ	Mass/charge vector with the range of ions in the spectra.
Y	Ion intensity vector with the same length as the mass/charge vector (MZ). Y can also be a matrix with several spectra that share the same mass/charge (MZ) range.

Description   msviewer(MZ, Y) creates a GUI to display and explore a mass spectrum (Y).

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 class label for every spectrum with a different color for every class. Enter a column vector of size [numSpectra x 1] with integers. The default value is [numSpectra].

MSViewer GUI features include the following:

- Plot mass spectra. The spectra are plotted with different colors according to their class 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: Zoom XY, Zoom X, Zoom Y, Reset view, Zoom out, 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 **1** Load and plot sample data

```
load sample_lo_res
msviewer(MZ_lo_res, Y_lo_res)
```

2 Add a marker by pointing to a mass peak, right-clicking, and then clicking **Add Marker**.

3 From the **File** menu, select

- **Import Markers from Workspace** — Opens the Import Markers From MATLAB Workspace dialog. The dialog should display 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. You can enter a variable name for the markers. All markers are saved. If there is no marker available, this menu item should be disabled.

- **Print to Figure** — Prints the spectra plot in the main display to a MATLAB Figure window

4 From the **Tools** menu, click

- **Add Marker** — Opens the Add Marker dialog. 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** (m/z) list.

- **Zoom XY**, **Zoom X**, **Zoom Y**, or **Zoom Out** — Changes the cursor from an arrow to crosshairs. Left-click and drag a rectangle box over an area and then release the mouse button. The display zooms the area covered by the box.

5 Move the cursor to the range window at the bottom. Click and drag the view box to a new location.

See Also   Bioinformatics Toolbox functions `msalign`, `msbackadj`, `mslowess`, `msnorm`, `msheatmap`, `msresample`, `mssgolay`

multialign

Purpose   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(..., 'JobManager', *JobManagerValue*)
multialign(..., 'WaitInQueue', *WaitInQueueValue*)
multialign(..., 'Verbose', *VerboseValue*)
multialign(..., 'ExistingGapAdjust',
ExistingGapAdjustValue)
multialign(..., 'TerminalGapAdjust',
TerminalGapAdjustValue)

Arguments

Seqs	Vector of structures with the fields 'Sequence' for the residues and 'Header' or 'Name' for the labels.
	Seqs may also be a cell array of strings or a char array.
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.

Tree	Phylogenetic tree calculated with either of the functions `seqlinkage` or `seqneighjoin`.
WeightsValue	Property to select the sequence weighting method. Enter either `'THG'` (default) or `'equal'`.
ScoringMatrixValue	Property to select or specify the scoring matrix. Enter an `[MxM]` matrix or `[MxMxN]` array of matrixes with N user-defined scoring matrices. *ScoringMatrixValue* may also be a cell array of strings with matrix names. The default is the `BLOSUM80` to `BLOSUM30` series for amino acids or a fixed matrix `NUC44` for nucleotides. When passing your own series of scoring matrices make sure all of them share the same scale.
SMInterpValue	Property to specify whether linear interpolation of the scoring matrices is on or off. When `false`, 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. The default is unity where sequences with the closest sequence farther than the median distance are delayed.
JobManagerValue	JobManager object representing an available distributed MATLAB resource. Enter a jobmanager object returned by the Distributed Computing Toolbox function findResource.
WaitInQueueValue	Property to control waiting for a distributed MATLAB resource to be available. Enter either true or false. The default value is false.
VerboseValue	Property to control displaying the sequences with sequence information. Default value is false.
ExistingGagAdjustValue	Property to control automatic adjustment based on existing gaps. Default value is true.
TerminalGapAdjustValue	Property to adjusts the penalty for opening a gap at the ends of the sequence. Default value is false.

multialign

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/value pairs.

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:

- 'THG'(default) — Thompson-Higgins-Gibson method using the
  phylogenetic tree branch distances weighted by their thickness.

- 'equal' — Assigns 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(..., 'JobManager', *JobManagerValue*) distributes pairwise alignments into a cluster of computers using Distributed Computing Toolbox.

multialign(..., 'WaitInQueue', *WaitInQueueValue*) when *WaitInQueueValue* is true, waits in the job manager queue for an available worker. When *WaitInQueueValue* is false (default) and there are no workers immediately available, multialign errors out. Use this property with Distributed Computing Toolbox and the multialign property WaitInQueue.

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*), 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, 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.

Example 1

1 Align seven cellular tumor antigen p53 sequences.

```
p53 = fastaread('p53samples.txt')
ma = multialign(p53,'verbose',true)
showalignment(ma)
```
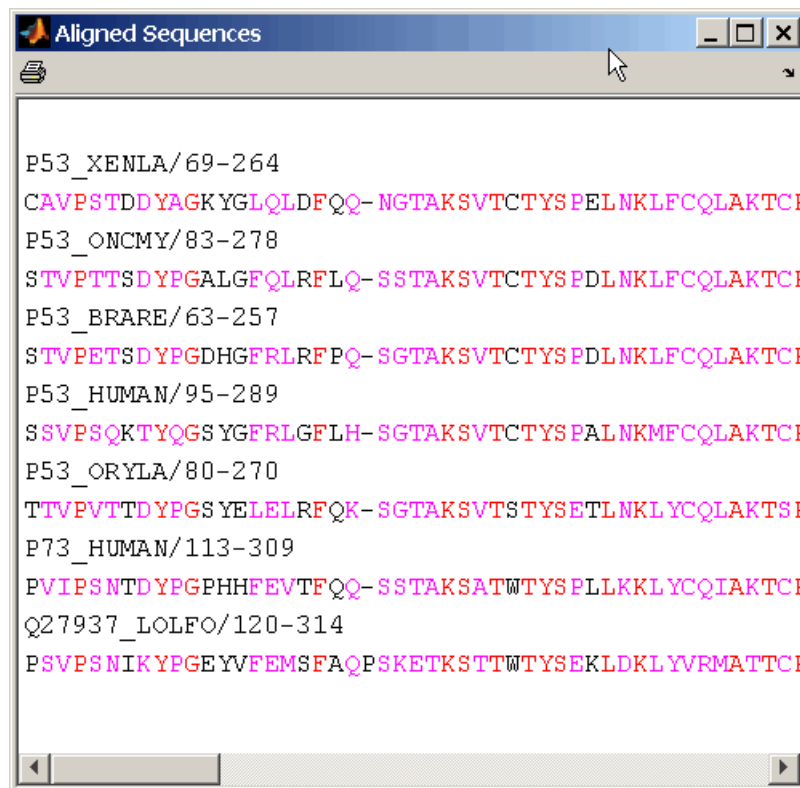
2 Use an UPGMA phylogenetic tree instead as a guiding tree.

```
dist = seqpdist(p53,'ScoringMatrix',gonnet);
tree = seqlinkage(dist,'UPGMA',p53)

Phylogenetic tree object with 7 leaves (6 branches)
```

3 Score the progressive alignment with the PAM family.

```
ma = multialign(p53,tree,'ScoringMatrix',...
               {'pam150','pam200','pam250'})
showalignment(ma)
```
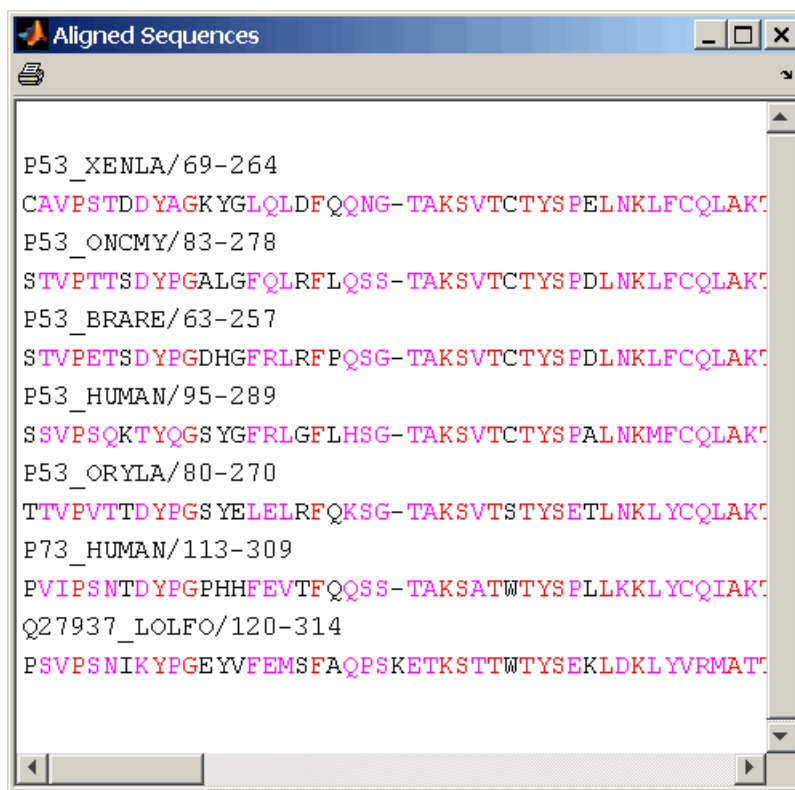
Example 2

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 alignment without termination gap adjustment.

```
multialign(seqs)

ans =
CA--CGTAACATCT--C
ACGACGTAACATCTTCT
AA-ACGTAACATCTCGC
```

See Also   Bioinformatics Toolbox functions: `hmmprofalign`, `multialignread`, `nwalign`, `profalign`, `seqprofile`, `seqconsensus`, `seqneighjoin`, `showalignment`

multialignread

Purpose   Read multiple-sequence alignment file

Syntax   *S* = multialignread(*File*)
                   [Headers, Sequences] = multialignread(*File*)
                   multialignread(..., '*PropertyName*', *PropertyValue*,...)
                   multialignread(..., 'IgnoreGaps', *IgnoreGapsValue*)

Arguments

File	Multiple sequence alignment file (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 of a multiple sequence alignment file. You can read common multiple alignment file types, such as ClustalW (.aln) and GCG (.msf).
IgnoreGapsValue	Property to control removing gap symbols.

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. (For an example, type open aagag.aln.) 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.

multialignread(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

multialignread(..., 'IgnoreGaps', *IgnoreGapsValue*), when *IgnoreGapsValue* is true, removes any gap symbol ('-' or '.') from the sequences. Default is false.

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   Bioinformatics Toolbox functions: `fastaread`, `gethmmalignment`, `multialign`, `seqconsensus`, `seqdisp`, `seqprofile`

multialignviewer

Purpose	Open viewer for multiple sequence alignments

Syntax

multialignviewer(*Alignment*)
multialignviewer*Alignment*, 'Alphabet', *AlphabetValue*)

Arguments

Alignment	A structure with a field Sequence, a character array, or a file name.
AlphabetValue	String specifying the alphabet type for the sequences. Choices are 'AA' for amino acids or 'NT' for nucleotides. Default is 'AA'.

Description

The multialignviewer is an interactive graphical user interface (GUI) for viewing multiple sequence alignments.

multialignviewer(*Alignment*) loads a group of previously multiple aligned sequences into the viewer. *Alignment* is a structure with a field Sequence, a character array, or a file name.

multialignviewer*Alignment*, 'Alphabet', *AlphabetValue*) specifies the alphabet type for the sequences. *AlphabetValue* can be 'AA' for amino acids or 'NT' for nucleotides. Default is 'AA'. If *AlphabetValue* is not specified, multialignviewer guesses the alphabet type.

Examples

multialignviewer('aagag.aln')

See Also

Bioinformatics Toolbox functions: fastaread, gethmmalignment, multialign, multialignread, seqtool

Purpose   Convert mzXML structure to peak list

Syntax   [*Peaks*, *Times*] = mzxml2peaks(*mzXMLStruct*)
                [*Peaks*, *Times*] = mzxml2peaks(*mzXMLStruct*,
                'Levels', *LevelsValue*)

Arguments

mzXMLStruct	mzXML structure, such as one created by the mzxmlread function. *mzXMLStruct* includes the following fields:

                      • scan

                      • offset

                      • mzXML

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).

mzxml2peaks

Return Values	*Peaks*	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 *Peaks*.

Description [*Peaks*, *Times*] = mzxml2peaks(*mzXMLStruct*) extracts peak information from *mzXMLStruct*, an mzXML structure, and creates *Peaks*, 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.

[*Peaks*, *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 **1** Use the mzxmlread function to read an mzXML file into MATLAB 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);
```

> **Note** The file results.mzxml is not provided. Sample mzXML files can be found at
>
>   http://sashimi.sourceforge.net/repository.html

2 Create a dotplot of the LC/MS data.

    msdotplot(peaks,time)

See Also   Bioinformatics Toolbox functions: msdotplot, mspalign, msppresample, mzxmlread

mzxmlread

Purpose   Read mzXML file into MATLAB as structure

Syntax   *mzXMLStruct* = mzxmlread(*File*)

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 directory.

**Return   *mzXMLStruct*   MATLAB structure containing information from an
Values**   mzXML file. It includes the following fields:

                                • 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

Description   *mzXMLStruct* = mzxmlread(*File*) reads an mzXML file, *File*, and then creates a MATLAB structure, *mzXMLStruct*.

File can be 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

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

mzxmlread

Examples
```
out = mzxmlread('results.mzxml');
% view a scan
m = out.scan(1).peaks.mz(1:2:end);
z = out.scan(1).peaks.mz(2:2:end);
bar(m,z)
```

Note The file results.mzxml is not provided. Sample mzXML files can be found at:

- Open Proteomics Database
- Peptide Atlas Repository at the Institute for Systems Biology (ISB)
- The Sashimi Project

See Also Bioinformatics Toolbox functions: jcampread, mzxml2peaks

MATLAB function: xmlread

Purpose   Count number of n-mers in nucleotide or amino acid sequence

Syntax   nmercount(*Seq*, *Length*)
                   nmercount(*Seq*, *Length*, *C*)

Arguments

Seq	Nucleotide or amino acid sequence. Enter a character string or a structure with the field Sequence.
Length	Length of n-mer to count. Enter an integer.

Description   nmercount(*Seq*, *Length*) counts the number of n-mers or patterns of a specific length in a sequence.

                   nmercount(*Seq*, *Length*, *C*) returns only the n-nmers with cardinality at least *C*.

Examples   Count the number of n-mers in an amino acid sequence and display the first six rows in the cell array.

```
S = getgenpept('AAA59174','SequenceOnly',true)
nmers = nmercount(S,4);
nmers(1:6,:)

ans =
    'apes'    [2]
    'dfrd'    [2]
    'eslk'    [2]
    'frdl'    [2]
    'gnys'    [2]
    'lkel'    [2]
```

See Also   Bioinformatics Toolbox functions: basecount, codoncount, dimercount

Purpose   Convert nucleotide sequence to amino acid sequence

Syntax
```
SeqAA = nt2aa(SeqNT)
SeqAA = nt2aa(..., 'Frame', FrameValue, ...)
SeqAA = nt2aa(..., 'GeneticCode', GeneticCodeValue, ...)
SeqAA = nt2aa(..., 'AlternativeStartCodons',
    AlternativeStartCodonsValue, ...)
```

—

Arguments

SeqNT

Either of the following:
- String specifying a nucleotide sequence
- MATLAB structure containing the field `Sequence`

Valid characters include:

- `A`
- `C`
- `G`
- `T`
- `U`
- hyphen (`-`)

> **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

Property to specify a reading frame. Choices are 1, 2, 3, or `'all'`. Default is 1.

If *FrameValue* is `'all'`, then *SeqAA* is a 3-by-1 cell array.

| *GeneticCodeValue* | Property to specify a genetic code. Enter a Code Number or a string with a Code Name from the table Genetic Code on page 2-610. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard. |
| *AlternativeStartCodonsValue* | Property to control the translation of alternative codons. Choices are true or false. Default is true. |

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

Genetic Code (Continued)

Code Number	Code Name
16	Chlorophycean Mitochondrial
21	Trematode Mitochondrial
22	Scenedesmus Obliquus Mitochondrial
23	Thraustochytrium Mitochondrial

Return Values

SeqAA   String specifying an amino acid
                                          sequence.

Description   *SeqAA* = nt2aa(*SeqNT*) converts a nucleotide sequence to an amino
acid sequence 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*, ...)
converts a nucleotide sequence to an amino acid sequence using a
specific genetic code.

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 details of alternative start codons, see

www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=t#SG1

Examples   The following example converts the gene ND1 on the human mitochondria genome to an amino acid sequence.

```
mitochondria = getgenbank('NC_001807', 'SequenceOnly', true)
ND1gene = mitochondria (3308:4264)
protein1 = nt2aa(ND1gene,'GeneticCode', 2)
protein2 = getgenpept('NP_536843', 'SequenceOnly', true)
```

The following example converts the gene ND2 on the human mitochondria genome to an amino acid sequence. In this case, the first codon is ATT, which is translated to M, while the following ATT codons are converted to I. If you set 'AlternativeStartCodons' to `false`, then the first codon ATT is translated to I, the corresponding amino acid in the Vertebrate Mitochondrial genetic code.

```
mitochondria = getgenbank('NC_001807', 'SequenceOnly', true)
ND2gene = mitochondria (4471:5514)
protein1 = nt2aa(ND2gene, 'GeneticCode', 2)
protein2 = getgenpept('NP_536844', 'SequenceOnly', true)
```

See Also   Bioinformatics Toolbox functions: aa2int, aminolookup, baselookup, codonbias, dnds, dndsml, geneticcode, revgeneticcode, seqtool

Purpose   Convert nucleotide sequence from letter to integer representation

Syntax   *SeqInt* = nt2int(SeqChar, '*PropertyName*', *PropertyValue*)
                   nt2int(..., 'Unknown', *UnknownValue*)
                   nt2int(..., 'ACGTOnly', *ACGTONlyValue*)

Arguments
SeqChar	Nucleotide sequence represented with letters. Enter a character string from the table Mapping Nucleotide Letters to Integers below. Integers are arbitrarily assigned to IUB/IUPAC letters. If the property ACGTOnly is true, you can only enter the characters A, C, T, G, and U.
UnknownValue	Property to select the integer for unknown characters. Enter an integer. Maximum value is 255. Default is 0.
ACGTOnlyValue	Property to control the use of ambiguous nucleotides. Enter true or false (default).

Mapping Nucleotide Letters 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

Mapping Nucleotide Letters to Integers (Continued)

Nucleotide	Code	Integer
Amino (A or C)	M	8
Strong interaction (3 H bonds) (G or C)	S	9
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)	N	15
Gap of indeterminate length	-	16
Unknown (default)	*	0 or ≥ 17

Description *SeqInt* = nt2int(SeqChar, '*PropertyName*', *PropertyValue*) converts a character string of nucleotides to a 1-by-N array of integers using the table Mapping Nucleotide Letters to Integers on page 2-613. Unknown characters (characters not in the table) are mapped to 0. Gaps represented with hyphens are mapped to 16.

nt2int(..., 'Unknown', *UnknownValue*) defines the number used to represent unknown nucleotides. Default is 0.

nt2int(..., 'ACGTOnly', *ACGTOnlyValue*) if ACGTOnly is true, the ambiguous nucleotide characters (N, R, Y, K, M, S, W, B, D, H, and V) are represented by the unknown nucleotide number.

Examples Convert a nucleotide sequence with letters to integers.

```
s = nt2int('ACTGCTAGC')

s =
     1     2     4     3     2     4     1     3     2
```

See Also   Bioinformatics Toolbox functions: `aa2int`, `baselookup`, `int2aa`, `int2nt`

ntdensity

Purpose	Plot density of nucleotides along sequence
Syntax	*Density* = ntdensity(SeqNT, '*PropertyName*', *PropertyValue*) ntdensity(..., 'Window', *WindowValue*) [Density, HighCG] = ntdensity(..., 'CGThreshold', *CGThresholdValue*)
Description	ntdensity(SeqNT) plots the density of nucleotides A, T, C, G in sequence SeqNT. *Density* = ntdensity(SeqNT, '*PropertyName*', *PropertyValue*) returns a MATLAB structure with the density of nucleotides A, C, G, and T. ntdensity(..., 'Window', *WindowValue*) uses a window of length Window for the density calculation. The default value is length(SeqNT)/20. [Density, HighCG] = ntdensity(..., 'CGThreshold', *CGThresholdValue*) returns indices for regions where the CG content of SeqNT is greater than CGThreshold. The default value for CGThreshold is 5.
Examples	s = randseq(1000, 'alphabet', 'dna'); ndensity(s)
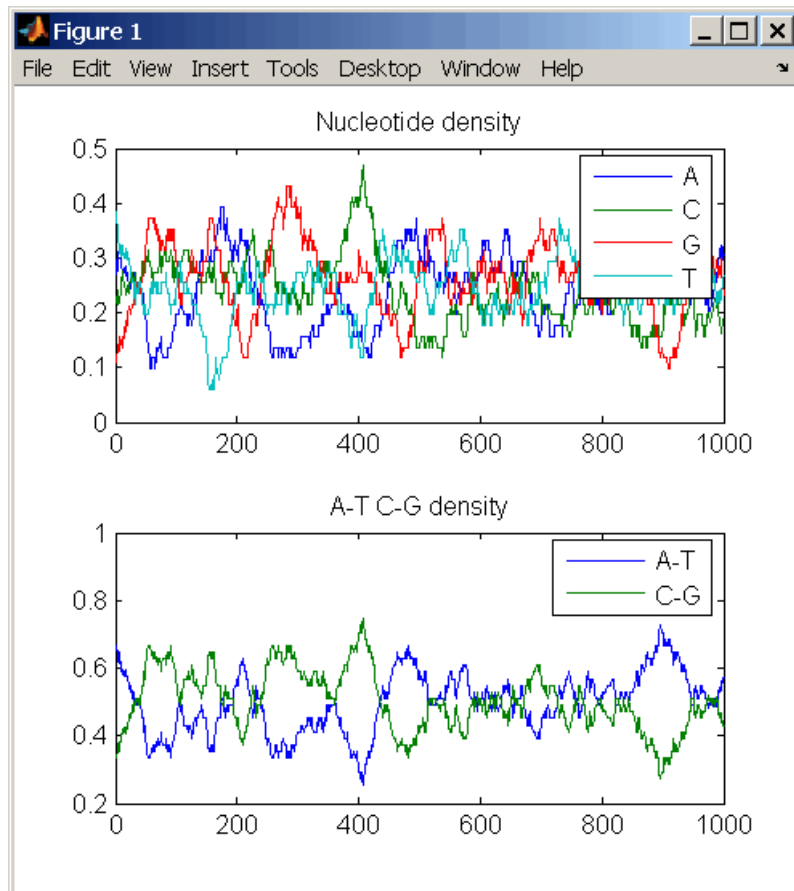
See Also   Bioinformatics Toolbox functions basecount, codoncount, cpgisland, dimercount

MATLAB function filter

Purpose	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.

Purpose   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 a 7-digit number preceded by the prefix 'GO:'.

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)
```

See Also   Bioinformatics Toolbox functions: geneont (object constructor),
                  goannotread

                  Bioinformatics Toolbox methods of geneont object: getancestors,
                  getdescendants, getmatrix, getrelatives

nwalign

Purpose   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*, ...'Showscore',
ShowscoreValue, ...)

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 on page 2-57 or Nucleotide Lookup on page 2-67.

AlphabetValue	String specifying the type of sequence. Choices are 'AA' (default) or 'NT'.

ScoringMatrixValue	String specifying the scoring matrix to use for the global alignment. Choices for amino acid sequences are:

- `'PAM40'`
- `'PAM250'`
- `'DAYHOFF'`
- `'GONNET'`
- `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'`
- `'BLOSUM62'`
- `'BLOSUM100'`

Default is:

- `'BLOSUM50'` (when *AlphabetValue* equals `'AA'`)
- `'NUC44'` (when *AlphabetValue* equals `'NT'`)

Note All of the above scoring matrices have a built-in scale factor that returns *Score* in bits.

ScaleValue	Positive value that specifies the scale factor used to return *Score* in arbitrary units other than bits. For example, if you enter `log(2)` for *ScaleValue*, then nwalign returns *Score* in nats.
GapOpenValue	Positive integer specifying the penalty for opening a gap in the alignment. Default is 8.

nwalign

	ExtendGapValue	Positive integer specifying the penalty for extending a gap. Default is equal to *GapOpenValue*.
	ShowscoreValue	Controls the display of the scoring space and the winning path of the alignment. Choices are `true` or `false` (default).

Return Values

	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 the scale factor used to return *Score* in arbitrary units other than bits. Choices are any positive value.

... = nwalign(*Seq1*,*Seq2*, ...'GapOpen', *GapOpenValue*, ...) specifies the penalty for opening a gap in the alignment. Choices are any positive integer. Default is 8.

... = nwalign(*Seq1*,*Seq2*, ...'ExtendGap', *ExtendGapValue*, ...) specifies the penalty for extending a gap in the alignment. Choices are any positive integer. Default is equal to *GapOpenValue*.

... = nwalign(*Seq1*,*Seq2*, ...'Showscore', *ShowscoreValue*, ...) controls the display of the scoring space and winning path of the alignment. Choices are true or false (default)
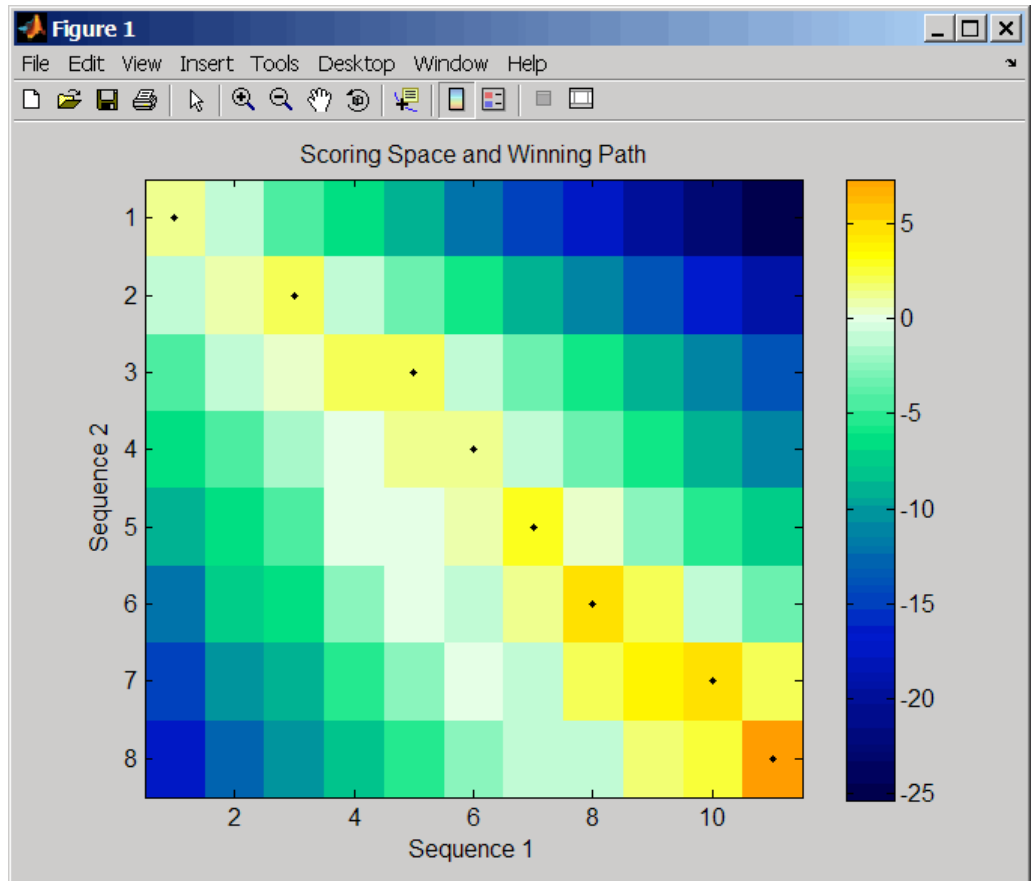
The scoring space is a heat map displaying the best scores for all the partial alignments of two sequences. The color of each (n1,n2) coordinate in the scoring space represents the best score for the pairing of subsequences Seq1(1:n1) and Seq2(1:n2), where n1 is a position in Seq1 and n2 is a position 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 represents 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.

Tip 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 Bioinformatics Toolbox functions: `blosum`, `multialign`, `nt2aa`, `pam`, `profalign`, `seqdotplot`, `showalignment`, `swalign`

Purpose   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, ...)

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.	

Return Values

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.

Field	Description
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 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*.

Field	Description
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
           GCAlpha: 0
          Hairpins: [0x25 char]
            Dimers: 'tAGCTtcatcgttgacttctactaa'
         MolWeight: 7.5820e+003
    MolWeightAlpha: 0
                Tm: [52.7640 60.8629 62.2493 55.2870 54.0293 61.0614]
           TmAlpha: [0 0 0 0 0 0]
             Thermo: [4x3 double]
        ThermoAlpha: [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
           GCAlpha: 3.5714
          Hairpins: 'ACGTagaggACGTn'
```

```
            Dimers: [3x14 char]
         MolWeight: 4.3329e+003
    MolWeightAlpha: 20.0150
                Tm: [38.8357 42.2958 57.7880 52.4180 49.9633 55.1330]
           TmAlpha: [1.4643 1.4643 10.3885 3.4633 0.2829 3.8074]
             Thermo: [4x3 double]
        ThermoAlpha: [4x3 double]
```

2 List the potential dimers for the sequence.

```
S2.Dimers

ans =

ACGTagaggacgtn
ACGTagaggACGTn
acgtagagGACGTN
```

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   Bioinformatics Toolbox functions: `isoelectric`, `molweight`, `ntdensity`, `palindromes`, `randseq`

optimalleaforder

Purpose   Determine optimal leaf ordering for hierarchical binary cluster tree

Syntax

Order = optimalleaforder(*Tree*, *Dist*)
Order = optimalleaforder(*Tree*, *Dist*, ...'Criteria',
CriteriaValue, ...)
Order = optimalleaforder(*Tree*, *Dist*, ...'Transformation',
   TransformationValue, ...)

Arguments

Tree	Hierarchical binary cluster tree represented by an (M - 1)-by-3 matrix, created by the linkage function, where M is the number of leaves.
Dist	Distance matrix, such as that created by the pdist function.

	CriteriaValue	String that specifies the optimization criteria. Choices are:

        • adjacent (default) — Minimizes the sum of distances between adjacent leaves.

        • group — Minimizes the sum of distances between every leaf and all other leaves in the adjacent cluster.

TransformationValue Either of the following:

        • String that specifies the algorithm to transform the distances in *Dist* into similarity values. Choices are:

            ▪ linear (default) — Similarity = max(all distances) - distance

            ▪ quadratic — Similarity = (max(all distances) - distance)^2

            ▪ inverse — Similarity = 1/distance

        • A function handle created using @ to a function that transforms the distances in *Dist* into similarity values. The function is typically a monotonic decreasing function within the range of the distance values. The function must accept a vector input and return a vector of the same size.

Return Values

    Order   Optimal leaf ordering for the hierarchical binary cluster tree represented by *Tree*.

Description   *Order* = optimalleaforder(*Tree*, *Dist*) returns the optimal leaf ordering for the hierarchical binary cluster tree represented by *Tree*, an (*M* - 1)-by-3 matrix, created by the linkage function, where *M* is the number of leaves. Optimal leaf ordering of a binary tree maximizes the

optimalleaforder

similarity between adjacent elements (clusters or leaves) by flipping tree branches, but without dividing the clusters. The input *Dist* is a distance matrix, such as that created by the pdist function.

Order = optimalleaforder(*Tree*, *Dist*, ...'*PropertyName*', *PropertyValue*, ...) calls optimalleaforder 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:

Order = optimalleaforder(*Tree*, *Dist*, ...'Criteria', *CriteriaValue*, ...) specifies the optimization criteria.

Order = optimalleaforder(*Tree*, *Dist*, ...'Transformation', *TransformationValue*, ...) specifies the algorithm to transform the distances in *Dist* into similarity values. The transformation is necessary because optimalleaforder maximizes the similarity between adjacent elements, which is comparable to minimizing the sum of distances between adjacent elements.

Examples

1 Use the rand function to create a 10-by-2 matrix of random values.

```
X = rand(10,2);
```

2 Use the pdist function to create a distance matrix containing the city block distances between the pairs of objects in matrix X.

```
Dist = pdist(X,'cityblock');
```

3 Use the linkage function to create a matrix, Tree, that represents a hierarchical binary cluster tree, from the distance matrix, Dist.

```
Tree = linkage(Dist,'average');
```

4 Use the optimalleaforder function to determine the optimal leaf ordering for the hierarchical binary cluster tree represented by Tree, using the distance matrix Dist.

```
order = optimalleaforder(Tree,Dist)
```

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.

See Also   Bioinformatics Toolbox function: `clustergram`

Statistics Toolbox functions: `linkage`, `pdist`

palindromes

Purpose	Find palindromes in sequence

Syntax

```
[Position, Length] = palindromes(SeqNT, 'PropertyName',
    PropertyValue)
[Position, Length, Pal] = palindromes(SeqNT)
palindromes(..., 'Length',LengthValue)
palindromes(..., 'Complement', ComplementValue)
```

Description

[Position, Length] = palindromes(SeqNT, '*PropertyName*', *PropertyValue*) 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(..., 'Length',*LengthValue*) finds all palindromes longer than or equal to Length. The default value is 6.

palindromes(..., 'Complement', *ComplementValue*) finds complementary palindromes if Complement is true, that is, where the elements match their complementary pairs A-T(or U) and C-G instead of an exact nucleotide match.

Examples

```
[p,l,s] = palindromes('GCTAGTAACGTATATATAAT')


p =
    11
    12
l =
     7
     7
s =
    'TATATAT'
    'ATATATA'

[pc,lc,sc] = palindromes('GCTAGTAACGTATATATAAT',...
                         'Complement',true);
```

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   Bioinformatics Toolbox functions seqrcomplement, seqshowwords

MATLAB functions regexp, strfind

pam

Purpose   Return PAM scoring matrix

Syntax   *ScoringMatrix* = pam(*N*)
                [*ScoringMatrix*, *MatrixInfo*] = pam(*N*)
                ... = pam(*N*, ...'Extended', *ExtendedValue*, ...)
                ... = pam(*N*, ...'Order', *OrderValue*, ...)

Arguments

N	Enter values 10:10:500. 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 *.
	Entering a larger value for *N* to allow sequence alignments with larger evolutionary distances.
ExtendedValue	Property to add ambiguous characters to the scoring matrix. Enter either true or false. Default is false.
OrderValue	Property to control the order of amino acids in the scoring matrix. Enter a string with at least the 20 standard amino acids.

Description *ScoringMatrix* = pam(*N*) returns a PAM 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*, ...) if Extended is true, returns a scoring matrix with the 20 amino acid characters, the

ambiguous characters, and stop character (B, Z, X, *), . If Extended is false, only the standard 20 amino acids are included in the matrix.

... = pam(*N*, ...'Order', *OrderValue*, ...) returns a PAM matrix ordered by the amino acid sequence in Order. If Order 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 a PAM matrix with N = 50.

```
PAM50 = pam(50)

PAM250 = pam(250,'Order','CSTPAGNDEQHRKMILVFYW')
```

See Also   Bioinformatics Toolbox functions: blosum, dayhoff, gonnet, nwalign, swalign

pdbdistplot

Purpose	Visualize intermolecular distances in Protein Data Bank (PDB) file

Syntax

```
pdbdistplot('PDBid')
pdbdistplot('PDBid', Distance)
```

Arguments

PDBid	Unique identifier for a protein structure record. Each structure in the PDB is represented by a 4-character alphanumeric identifier.
	For example, 4hhb is the identification code for hemoglobin.
Distance	Threshold distance in Angstroms shown on a spy plot. Default value is 7.

Description   pdbdistplot displays the distances between atoms and amino acids in a PDB structure.

pdbdistplot('PDBid') retrieves the entry PDBid from the Protein Data Bank (PDB) database and creates a heat map showing interatom distances and a spy plot showing the residues where the minimum distances apart are less than 7 Angstroms. PDBid can also be the name of a variable or a file containing a PDB MATLAB structure.

pdbdistplot('PDBid', Distance) specifies the threshold distance shown on a spy plot.

Examples   Show spy plot at 7 Angstroms of the protein cytochrome C from albacore tuna.

```
pdbdistplot('5CYT');
```

Now take a look at 10 Angstroms.

```
pdbdistplot('5CYT',10);
```

See Also   Bioinformatics Toolbox functions: `getpdb`, `molviewer`, `pdbread`, `proteinplot`, `ramachandran`

pdbread

Purpose	Read data from Protein Data Bank (PDB) file

Syntax

PDBStruct = pdbread(*File*)
PDBStruct = pdbread(*File*, 'ModelNum', *ModelNumValue*)

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 Directory.
	• MATLAB character array that contains the text of a PDB-formatted file.
ModelNumValue	Positive integer specifying a model in a PDB-formatted file.

Return Values

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*	Remark*n*
Note *N* equals 2 through 999.	**Note** *n* equals 2 through 999.
DBREF	DBReferences
SEQADV	SequenceConflicts
SEQRES	Sequence
FTNOTE	Footnote
MODRES	ModifiedResidues

pdbread

PDB Database Record	Field in the MATLAB Structure
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

PDB Database Record	Field in the MATLAB Structure
HETATM	HeterogenAtom
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 Directory.

```
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
```

pdbread

```
            Atom: [1x1205 struct]
        Terminal: [1x2 struct]
```

5 Read the data from an URL into a MATLAB structure,
gfl_pdbstruct.

```
gfl_pdbstruct = pdbread('http://www.rcsb.org/pdb/files/1gfl.pdb');
```

See Also   Bioinformatics Toolbox functions: genpeptread, getpdb, molviewer,
pdbdistplot, pdbwrite

Purpose   Write to file using Protein Data Bank (PDB) format

Syntax   pdbwrite(*File*, *PDBStruct*)
                  PDBArray = pdbwrite(*File*, *PDBStruct*)

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 Directory.

                                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*.

**Return
Values**   *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

pdbwrite

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 directory 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   Bioinformatics Toolbox functions: getpdb, molviewer, pdbread

pfamhmmread

Purpose	Read data from PFAM-HMM file
Syntax	Data = pfamhmmread('*File*')

Arguments

File	PFAM-HMM formatted 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 of a PFAM-HMM file.

Description pfamhmmread reads data from a PFAM-HMM formatted file (file saved with the function gethmmprof) and creates a MATLAB structure.

Data = pfamhmmread('*File*') reads from *File* a Hidden Markov Model described by the PFAM format, and converts it to the MATLAB structure Data, containing fields corresponding to annotations and parameters of the model. For more information about the model structure format, see hmmprofstruct. *File* can also be a URL or a MATLAB cell array that contains the text of a PFAM formatted file.

pfamhmmread is based on the HMMER 2.0 file formats.

Examples pfamhmmread('pf00002.ls')

```
site='http://www.sanger.ac.uk/';
pfamhmmread([site 'cgi-bin/Pfam/download_hmm.pl?mode=ls&id=7tm_2'])
```

See Also Bioinformatics Toolbox functions: gethmmalignment, gethmmprof, hmmprofalign, hmmprofstruct, showhmmprof

Purpose   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.
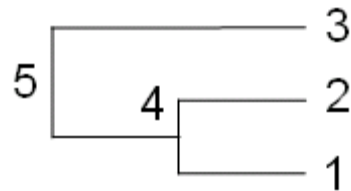
In the MATLAB Command Window, type

```
B = [1 2 ; 3 4]
tree = phytree(B)
view(tree)
```
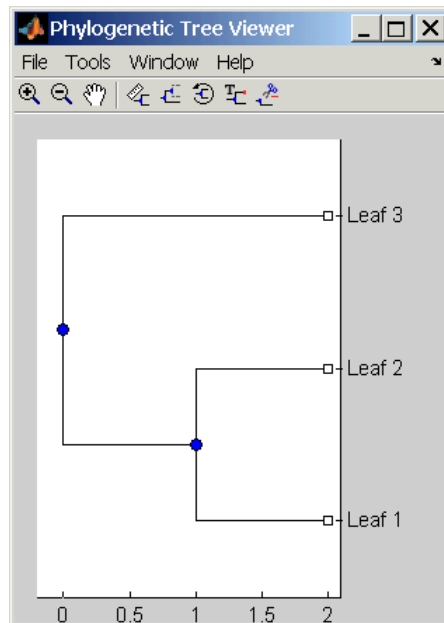
Tree = phytree(*B*, *D*) creates an additive (ultrametric or nonultrametric) phylogenetic tree object with branch distances defined by *D*. *D* is a numeric array of size [NUMNODES X 1] with the distances of every child node (leaf or branch) to its parent branch equal to NUMNODES = NUMLEAVES + NUMBRANCHES. The last distance in *D* is the distance of the root node and is meaningless.

```
b = [1 2 ; 3 4 ]: d = [1 2 1.5 1 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]; 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 for a set of multiply aligned sequences.

```
Sequences = multialignread('aagag.aln')
distances = seqpdist(Sequences)
tree = seqlinkage(distances)
phytreetool(tree)
```

phytree

See Also   Bioinformatics Toolbox functions: `phytreeread`, `phytreetool`,
`phytreewrite`, `seqlinkage`, `seqneighjoin`, `seqpdist`

Bioinformatics Toolbox object: `phytree object`

Bioinformatics Toolbox methods of phytree object: `get`, `getbyname`,
`getcanonical`, `getmatrix`, `getnewickstr`, `pdist`, `plot`, `prune`, `reroot`,
`select`, `subtree`, `view`, `weights`

Purpose	Read phylogenetic tree file
Syntax	*Tree* = phytreeread(*File*)

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.
Tree	phytree object created with the function phytree.

Description   *Tree* = phytreeread(*File*) reads a Newick formatted tree file and returns a phytree object in the MATLAB workspace with data from the file.

The NEWICK tree format can be found at

```
http://evolution.genetics.washington.edu/phylip/newicktree.html
```

> **Note** This implementation only allows binary trees. Non-binary trees are translated into a binary tree with extra branches of length 0.

Examples   tr = phytreeread('pf00002.tree')

See Also   Bioinformatics Toolbox functions: phytree (object constructor), gethmmtree, phytreetool, phytreewrite

phytreetool

Purpose	View, edit, and explore phylogenetic tree data
Syntax	phytreetool(*Tree*) phytreetool(*File*)

Arguments

Tree	Phytree object created with the functions phytree or phytreeread.
File	Newick or ClustalW tree formatted file (ASCII text file) with phylogenetic tree data. 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 Newick file.

Description phytreetool is an interactive GUI that allows you to view, edit, and explore phylogenetic tree data. This GUI allows branch pruning, reordering, renaming, and distance exploring. It can also open or save Newick formatted files.

phytreetool(*Tree*) loads data from a phytree object in the MATLAB workspace into the GUI.

phytreetool(*File*) loads data from a Newick formatted file into the GUI.

Examples
```
tr= phytreeread('pf00002.tree')
phytreetool(tr)
```
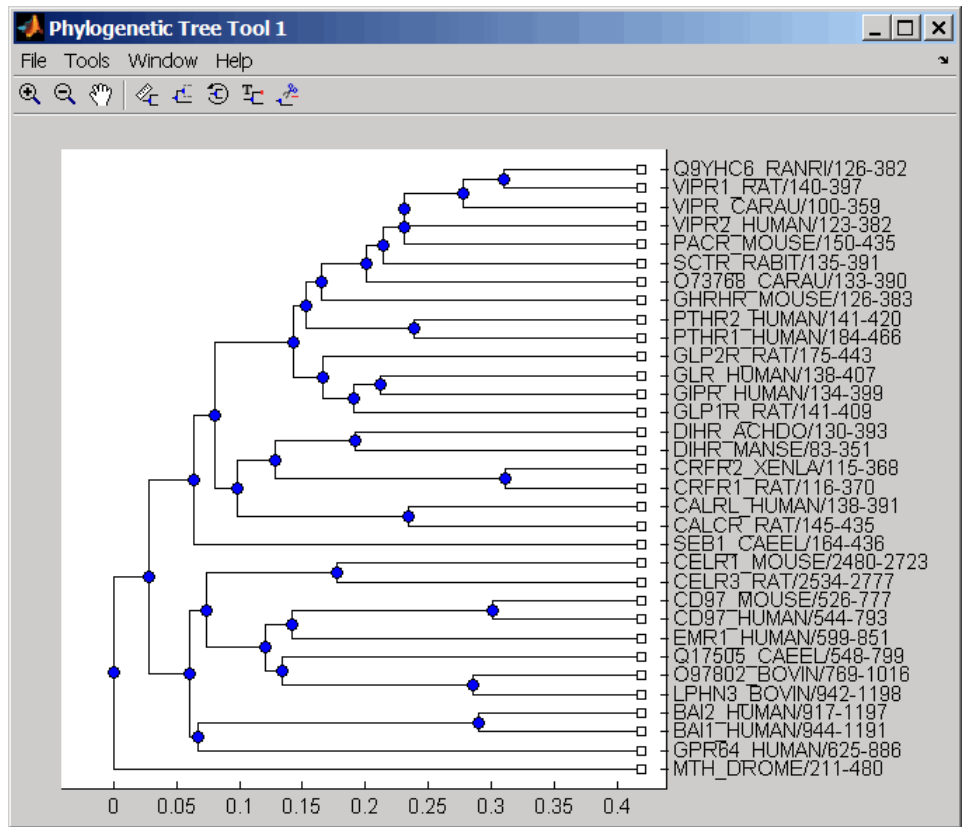
See Also   Bioinformatics Toolbox functions: phytree (object constructor), phytreeread, phytreewrite

Bioinformatics Toolbox methods of phytree object: plot, view

phytreewrite

Purpose	Write phylogenetic tree object to Newick-formatted file
Syntax	phytreewrite('*File*', *Tree*) phytreewrite(*Tree*)

Arguments

File	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.

Examples   Read tree data from a Newick-formatted file.

```
tr = phytreeread('pf00002.tree')
```

Remove all the mouse proteins

```
ind = getbyname(tr,'mouse');
tr = prune(tr,ind);
```

```
view(tr)
```

Write pruned tree data to a file.

```
phytreewrite('newtree.tree', tr)
```

See Also   Bioinformatics Toolbox functions: phytree (object constructor),
phytreeread, phytreetool, seqlinkage

Bioinformatics Toolbox object: phytree object

Bioinformatics Toolbox methods of phytree object: getnewickstr

probelibraryinfo

Purpose	Create table of probe set library information

Syntax   *ProbeInfo* = probelibraryinfo(*CELStruct*, *CDFStruct*)

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.

Return Values

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 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 demo 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/products/software/specific/dtt.affx

The following example assumes that the Ecoli-antisense-121502.CEL file is stored on the MATLAB search path or in the current directory. 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   Bioinformatics Toolbox functions: affyread, celintensityread, probesetlink, probesetlookup, probesetplot, probesetvalues

probesetlink

Purpose	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, ...)
```

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 directory 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).

Return Values

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 directory 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.

---

probesetlink

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 demo 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/products/software/specific/dtt.affx

The following example assumes that the Ecoli-antisense-121502.CHP file is stored on the MATLAB search path or in the current directory. 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   Bioinformatics Toolbox functions: affyread, celintensityread, probelibraryinfo, probesetlookup, probesetplot, probesetvalues

Purpose   Look up information for probe set

Syntax   *PSStruct* = probesetlookup(*AffyStruct*, *ID*)

Arguments   *AffyStruct* Structure created by the affyread function from an
                               Affymetrix CHP file or an Affymetrix CDF library file
                               for expression assays.

                   ID   Probe set ID/name or gene ID.

**Return
Values**   *PSStruct* Structure 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
                   containing information for a probe set specified by ID, a probe set
                   ID/name or gene ID, and by *AffyStruct*, a structure created from an
                   Affymetrix CHP file or Affymetrix CDF library file for expression assays.

probesetlookup

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 directory 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 Bioinformatics Toolbox functions: affyread, celintensityread, probelibraryinfo, probesetlink, probesetplot, probesetvalues, rmabackadj

Purpose   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 directory as the CDF library file from which *CDFStruct* was created.

probesetplot

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 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 directory 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 use 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 demo 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/products/software/specific/dtt.affx

probesetplot

The following example assumes that the `Ecoli-antisense-121502.CEL` file is stored on the MATLAB search path or in the current directory. 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)
```
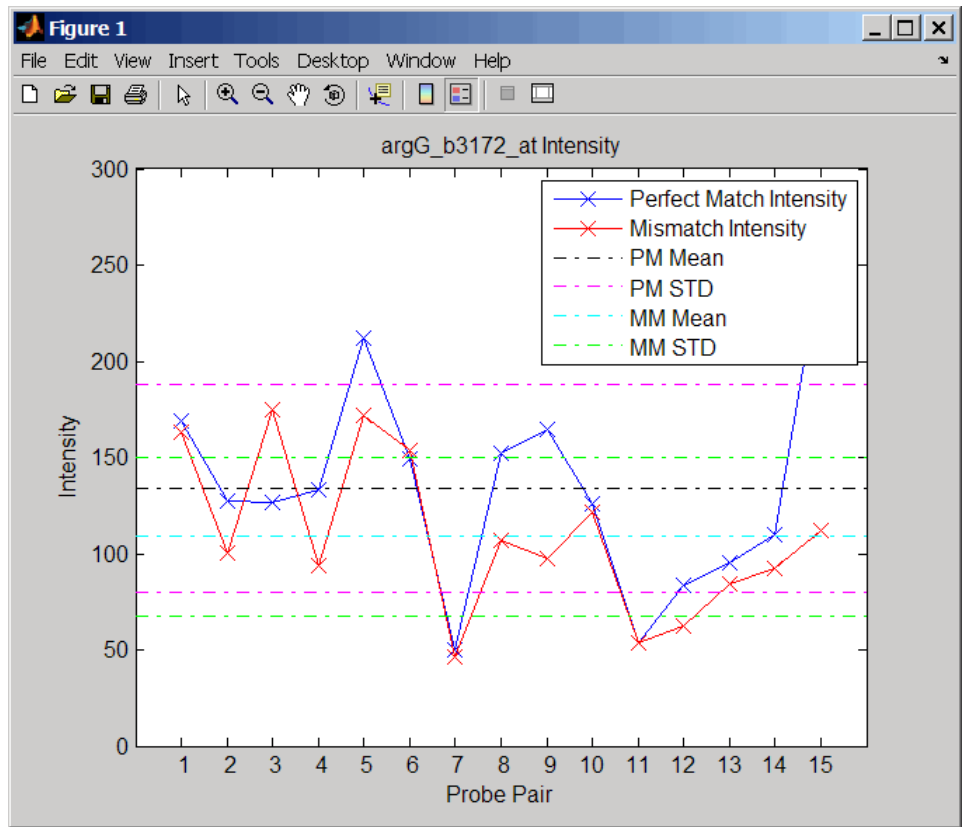
See Also   Bioinformatics Toolbox functions: `affyread`, `celintensityread`, `probesetlink`, `probesetlookup`, `probesetvalues`

probesetvalues

Purpose Create table of Affymetrix probe set intensity values

Syntax *PSValues* = probesetvalues(*CELStruct*, *CDFStruct*, *PS*)
ColumnNames = probesetvalues

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.

Return Values

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'`	Background-adjusted 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.
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.

Column	Field	Description
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 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.

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 demo 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/products/software/specific/dtt.affx
```

The following example assumes that the `Ecoli-antisense-121502.CEL`
file is stored on the MATLAB search path or in the current directory.
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 Create a table of intensity values for the `argG_b3172_at` probe set.

```
psvals = probesetvalues(celStruct, cdfStruct, 'argG_b3172_at');
```

See Also   Bioinformatics Toolbox functions: `affyread`, `celintensityread`,
`probelibraryinfo`, `probesetlink`, `probesetlookup`, `probesetplot`,
`rmabackadj`

profalign

Purpose	Align two profiles using Needleman-Wunsch global alignment
Syntax	*Prof* = profalign(*Prof1*, *Prof2*) [*Prof, H1, H2*] = profalign(*Prof1*, *Prof2*) profalign(..., '*PropertyName*', *PropertyValue*,...) 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*,...) defines optional properties using property name/value pairs.

profalign(..., 'ScoringMatrix', *ScoringMatrixValue*) defines the scoring matrix (*ScoringMatrixValue*) to be used for the alignment. The default is 'BLOSUM50' for amino acids or 'NUC44' for nucleotide sequences.

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, 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)
```

See Also   Bioinformatics Toolbox functions hmmprofalign, multialign, nwalign, seqprofile, seqconsensus

Purpose   Characteristics for amino acid sequences

Syntax   proteinplot (SeqAA)

Arguments   SeqAA   Amino acid sequence or a structure with a field Sequence
                           containing an amino acid sequence.

Description proteinplot (SeqAA) loads an amino acid sequence into the protein plot
                GUI. proteinplot is a tool for analyzing a single amino acid sequence.
                You can use the results from proteinplot to compare the properties
                of several amino acid sequences. It displays smoothed line plots of
                various properties such as the hydrophobicity of the amino acids in
                the sequence.

Importing Sequences into proteinplot

1 In the MATLAB Command Window, type

    proteinplot(Seq_AA)

The proteinplot interface opens and the sequence Seq_AA is shown
in the **Sequence** text box.

2 Alternatively, type or paste an amino acid sequence into the
Sequence text box.

You can import a sequence with the Import dialog box:

1 Click the **Import Sequence** button. The Import dialog box opens.

2 From the **Import From** list, select a variable in the MATLAB
workspace, ASCII text file, FASTA formatted file, GenPept formatted
file, or accession number in the GenPept database.

proteinplot

Information About the Properties

You can also access information about the properties from the **Help** menu.

1 From the **Help** menu, click **References**. The Help Browser opens with a list of properties and references.

2 Scroll down to locate the property you are interested in studying.

Working with Properties

When you click on a property a smoothed plot of the property values along the sequence will be displayed. Multiple properties can be selected from the list by holding down Shift or Ctrl while selecting properties. When two properties are selected, 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 three or more properties are displayed, the values are normalized to the range 0-1.

You can add your own property values by clicking on the Add button next to the property list. This will open up a dialog that allows you to specify the values for each of the amino acids. The Display Text box allows you to specify the text that will be displayed in the selection box on the main `proteinplot` window. You can also save the property values to an m-file for future use by typing a file name into the Filename box.

The Terminal Selection boxes allow you to choose to plot only part of the sequence. By default all of the sequence is plotted. The default smoothing method is an unweighted linear moving average with a window length of five residues. You can change this using the "Configuration Values" dialog from the Edit menu. The dialog allows you to select the window length from 5 to 29 residues. You can modify the shape of the smoothing window by changing the edge weighting 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 allows you to Import a sequence, save the plot that you have created to a `FIG` file, you can export the data values in the figure to a workspace variable or to a MAT file, you can export the figure to a normal Figure window for customizing, and you can print the figure.

The Edit menu allows you to 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 allows you to turn the toolbar on and off, and to add a legend to the plot.

The Tools menu allows you to 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 allows you to view this document and to see the references for the sequence properties built into `proteinplot`

See Also   Bioinformatics Toolbox functions: `aacount`, `atomiccomp`, `molviewer`, `molweight`, `pdbdistplot`, `seqtool`

MATLAB function: `plotyy`

proteinpropplot

Purpose   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 m-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

proteinpropplot

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 m-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)
```

proteinpropplot
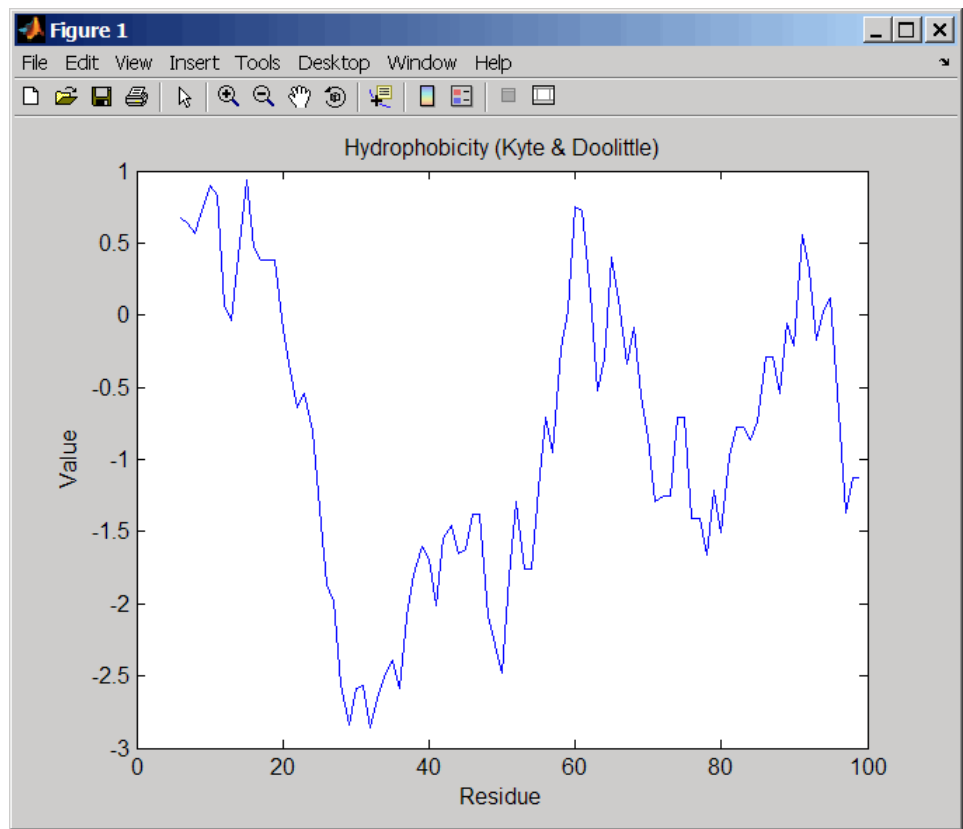
Plotting Parallel Beta Strand

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')
```
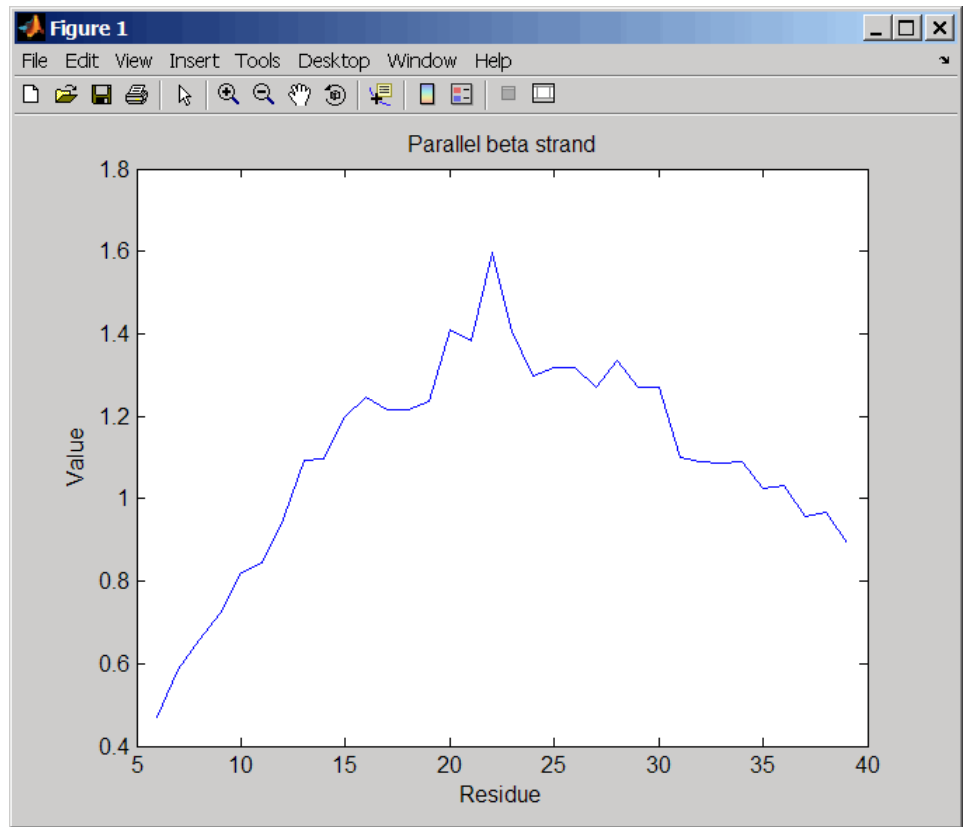
References   [1] Kyte, J., and Doolittle, R.F. (1982). A simple method for displaying the hydropathic character of a protein. J Mol Biol *157(1)*, 105–132.

See Also   Bioinformatics Toolbox functions: `aacount`, `atomiccomp`, `molviewer`, `molweight`, `pdbdistplot`, `proteinplot`, `ramachandran`, `seqtool`

MATLAB function: `plotyy`

quantilenorm

Purpose	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** | malowess, manorm, rmabackadj, rmasummary |

Purpose   Draw Ramachandran plot for Protein Data Bank (PDB) data

Syntax
```
ramachandran(PDBid)
ramachandran(File)
ramachandran(PDBStruct)
Angles = ramachandran(...)
[Angles, Handle] = ramachandran(...)
[Angles, Handle, ResSeqNum] = ramachandran(...)
```

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	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 Directory.
- MATLAB character array that contains the text of a PDB-formatted file.

PDBStruct	MATLAB structure containing PDB-formatted data, such as returned by getpdb or pdbread.

Return Values

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 *ResSeqNum* 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.

Handle   Handle to the plot.

ResSeqNum Column vector containing the residue sequence numbers from both the ATOM and HETATM records in the PDB file.

Note The *ResSeqNum* vector starts with one of the following:

- 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 *ResSeqNum* 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 *ResSeqNum* vector. Therefore, you can use the *ResSeqNum* vector to determine which residue corresponds to each row in the *Angles* matrix.

Description   ramachandran generates 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) of the 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.

Angles = ramachandran(...) returns a 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 *ResSeqNum* 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.

[*Angles*, *Handle*] = ramachandran(...) returns a handle to the plot.

[*Angles*, *Handle*, *ResSeqNum*] = ramachandran(...) returns *ResSeqNum*, a column vector containing the residue sequence numbers from both the ATOM and HETATM records in the PDB file.

Note The *ResSeqNum* vector starts with one of the following:

- 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 *ResSeqNum* 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 *ResSeqNum* vector. Therefore, you can use the *ResSeqNum* vector to determine which residue corresponds to each row in the *Angles* matrix.

Examples   Generate the Ramachandran plot for the human serum albumin complexed with octadecanoic acid, which has a PDB database identifier of 1E7I.

```
ramachandran('1E7I')
```

See Also   Bioinformatics Toolbox functions: getpdb, molviewer, pdbdistplot, pdbread

randfeatures

Purpose   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. Defaults are

{5,`'correlation'`,`'consensus'`} for KNN and {`'linear'`} for DA. See `knnclassify` and `classify` for more information.

`randfeatures(..., 'PerformanceThreshold', PT)` sets the correct classification threshold used to pick the subsets included in the final pool. Default is `0.8` (80%).

`randfeatures(..., 'ConfidenceThreshold', CT)` uses the posterior probability of the discriminant analysis to invalidate classified subvectors with low confidence. This option is only valid when `Classifier` is `'da'`. Using it has the same effect as using `'consensus'` in KNN; i.e., it makes the selection of approved subsets very stringent. Default is `0.95.^(number of classes)`.

`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
```

See Also   Bioinformatics Toolbox functions: classperf, crossvalind,
knnclassify, rankfeatures, svmclassify

Statistics Toolbox function: classify

Purpose   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	Number of amino acids or nucleotides in random sequence .
AlphabetValue	Property to select the alphabet for the sequence. Enter '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	Property to select the case of letters in a sequence whenAlphabet is 'char'. Values are'upper' (default) or 'lower'.
DataTypeValue	Property to select the data type for a sequence. Values 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 quotes 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   Statistics Toolbox functions: `hmmgenerate`, `randsample`

MATLAB functions: `rand`, `randperm`

rankfeatures

Purpose	Rank key features by class separability criteria
Syntax	[*IDX*, *Z*] = rankfeatures(*X*, *Group*)

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-Lieber distance or divergence.

- 'brattacharyya' — 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 u-statistic of a two-sample unpaired Wilcoxon test, also known as Mann-Whitney.

Note 'ttest', 'entropy', and 'brattacharyya' 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　**1** 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
```

2 Get a logical index vector to the breast cancer cells.

```
BC = GROUP == 8;
```

3 Select features.

```
I = rankfeatures(X,BC,'NumberOfIndices',12);
```

4 Test features with a linear discriminant classifier.

```
C = classify(X(I,:)',X(I,:)',double(BC));
cp = classperf(BC,C);
cp.CorrectRate

ans =

    1
```

5 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
```

6 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')
```

rankfeatures

See Also   Bioinformatics Toolbox functions: `classperf`, `crossvalind`, `randfeatures`, `svmclassify`

Statistics Toolbox function: `classify`

Purpose   Find restriction enzymes that cut protein sequence

Syntax   [*Enzymes*, *Sites*] = rebasecuts(*SeqNT*)
rebasecuts(*SeqNT*, *Group)*
rebasecuts(*SeqNT*, [*Q*, *R*])
rebasecuts(*SeqNT*, *S*)

Arguments

SeqNT	Nucleotide sequence.
Enzymes	Cell array with the names of restriction enzymes from REBASE Version 412.
Sites	Vector of cut sites with the base number before every cut relative to the sequence.
Group	Cell array with the names of valid restriction enzymes.
Q, *R*, *S*	Base positions.

Description   [*Enzymes*, *Sites*] = rebasecuts(*SeqNT*) finds all the restriction enzymes that cut a nucleotide sequence (*SeqNT*).

rebasecuts(*SeqNT*, *Group)* limits the search to a specified list of enzymes (*Group*).

rebasecuts(*SeqNT*, [*Q*, *R*]) limits the search to those enzymes that cut after a specified base position (*Q*) and before a specified base position (*R*) relative to the sequence.

rebasecuts(*SeqNT*, *S*) limits the search to those enzymes that cut just after a specified base position (*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

rebasecuts

Example

1 Enter a nucleotide sequence.

```
seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA'
```

2 Look for all possible cleavage sites in the sequence seq.

```
[enzymes sites] = rebasecuts(seq)
```

3 Find where restriction enzymes CfoI and Tru9I cut the sequence.

```
[enzymes sites] = rebasecuts(seq, {'CfoI','Tru9I'})
```

4 Search for any possible enzymes that cut after base 7.

```
enzymes  = rebasecuts(seq, 7)
```

5 Get the subset of enzymes that cut between base 11 and 37.

```
enzymes  = rebasecuts(seq, [11 37])
```

See Also   Bioinformatics Toolbox functions: cleave, restrict, seq2regexp, seqshowwords

MATLAB function: regexp

redgreencmap

Purpose
Create red and green color map

Syntax
redgreencmap(*Length*)
redgreencmap(..., 'Interpolation', *InterpolationValue*, ...)

Arguments

Length	Length of the color map. Enter either 256 or 64. Default is the length of the color map 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 an *Length*-by-3 matrix containing a red and green color map. 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 color map of the current figure.

redgreencmap(..., '*PropertyName*', *PropertyValue*, ...) defines optional properties that use property name/value pairs in any order. These property name/value pairs are as follows:

redgreencmap(..., 'Interpolation', *InterpolationValue*, ...) lets you set the algorithm for color interpolation. Choices are:

- 'linear'

redgreencmap

- 'quadratic'

- 'cubic'

- 'sigmoid' (default)

Note The sigmoid interpolation is tanh.

Examples Reset the color map of the current figure.

```
pd =gprread('mouse_a1pd.gpr')
maimage(pd,'F635 Median')
colormap(redgreencmap)
```

See Also Bioinformatics Toolbox function: clustergram

MATLAB functions: colormap, colormapeditor

Purpose   Split nucleotide sequence at restriction site

Syntax
Fragments = restrict(*SeqNT*, *Enzyme*)
Fragments = restrict(*SeqNT*, *Pattern*, *Position*)
[*Fragments, CuttingSites*] = restrict(...)
[*Fragments, CuttingSites, Lengths*] = restrict(...)
... = restrict(..., 'PartialDigest', *PartialDigestValue*)

Arguments

SeqNT	Nucleotide sequence. Enter either a character string with the characters A, T, 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.
Enzyme	Enter the name of a restriction enzyme from REBASE Version 412.
Pattern	Enter a short nucleotide pattern. Pattern can be a regular expression.
Position	Defines the position on Pattern where the sequence is cut. Position=0 corresponds to the 5' end of the Pattern.
PartialDigestValue	Property to specify a probability for partial digestion. Enter a value from 0 to 1.

Description   *Fragments* = restrict(*SeqNT*, *Enzyme*) cuts a sequence (*SeqNT*) into fragments at the restriction sites of a restriction enzyme (*Enzyme*). The returned values are stored in a cell array of sequences (*Fragments*).

Fragments = restrict(*SeqNT*, *Pattern*, *Position*) cuts a sequence (*SeqNT*) into fragments at restriction sites specified by a nucleotide pattern (*Pattern*).

[*Fragments, CuttingSites*] = restrict(...) returns a numeric vector with the indices representing the cutting sites. A 0 (zero) is added to the list so numel(*Fragments*)==numel(*CuttingSites*). 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 probability (*PartialDigestValue*) 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, go to the REBASE Web site at

```
http://rebase.neb.com/rebase/rebase.html
```

Examples   **1** Enter a nucleotide sequence.

```
Seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
```

2 Use the recognition pattern (sequence) `GCGC` with the point of cleavage at position 3 to cleave a nucleotide sequence.

```
fragmentsPattern = restrict(Seq,'GCGC',3)

fragmentsPattern =
    'AGAGGGGTACGCG'
    'CTCTGAAAAGCGGGAACCTCGTGGCG'
    'CTTTATTAA'
```

3 Use the restriction enzyme `HspAI` (recognition sequence `GCGC` with the point of cleavage at position 1) to cleave a nucleotide sequence.

```
fragmentsEnzyme = restrict(Seq,'HspAI')

fragmentsEnzyme =
    'AGAGGGGTACG'
    'CGCTCTGAAAAGCGGGAACCTCGTGG'
    'CGCTTTATTAA'
```

4 Use a regular expression for the enzyme pattern.

```
fragmentsRegExp = restrict(Seq,'GCG[^C]',3)

fragmentsRegExp =

    'AGAGGGGTACGCGCTCTGAAAAGCG'
    'GGAACCTCGTGGCGCTTTATTAA'
```

5 Capture the cutting sites and fragment lengths with the fragments.

```
[fragments, cut_sites, lengths] = restrict(Seq,'HspAI')

fragments =
    'AGAGGGGTACG'
    'CGCTCTGAAAAGCGGGAACCTCGTGG'
    'CGCTTTATTAA'

cut_sites =
     0
    11
    37

lengths =
    11
    26
    11
```

See Also   Bioinformatics Toolbox functions: cleave, rebasecuts, seq2regexp, seqshowwords

MATLAB function: regexp

revgeneticcode

Purpose   Reverse mapping for genetic code

Syntax

map = revgeneticcode
revgeneticcode(*GeneticCode*)
revgeneticcode(..., 'Alphabet', *AlphabetValue*, ...)
revgeneticcode(..., 'ThreeLetterCodes',
ThreeLetterCodesValue,
   ...)

Arguments

GeneticCode	Genetic code for translating nucleotide codons to amino acids. Enter a code number or code name from the table Genetic Code on page 2-718. If you use a code name, you can truncate the name to the first two characters of the name.
AlphabetValue	Property to select the nucleotide alphabet. Enter either 'dna' or 'rna'. The default value is 'dna'.
ThreeLetterCodesValue	Property to select one- or three-letter amino acid codes. Enter true for three-letter codes or false for one-letter codes.

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

Genetic Code (Continued)

Code Number	Code Name
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`

Description *map* = revgeneticcode returns a structure containing the reverse mapping for the standard genetic code.

revgeneticcode(*GeneticCode*) returns a structure containing the reverse mapping for an alternate genetic code.

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 quotes and is case insensitive. These property name/property value pairs are as follows:

revgeneticcode(..., 'Alphabet', *AlphabetValue*, ...) defines the nucleotide alphabet to use in the map.

revgeneticcode

revgeneticcode(..., 'ThreeLetterCodes',
ThreeLetterCodesValue, ...) returns the mapping structure with
three-letter amino acid codes as field names instead of the default
single-letter codes if ThreeLetterCodes is true.

Examples

```
moldcode = revgeneticcode(4,'Alphabet','rna');
wormcode = revgeneticcode('Flatworm Mitochondrial',...
                          'ThreeLetterCodes',true);

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'}
```

References   [1] NCBI Web page describing genetic codes:

    http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c

See Also   Bioinformatics Toolbox functions: aa2nt, aminolookup, baselookup, geneticcode, nt2aa

rmabackadj

Purpose   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, ...)

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	Property to control 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	Property to control the background noise model. Enter either true (use a truncated Gaussian distribution) or false (use a nontruncated Gaussian distribution). Default is true.
ShowplotValue	Property to control the plotting of a histogram showing the distribution of PM probe intensity values (blue) and the convoluted probability distribution function (red), with estimated parameters. 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.

Return Values

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*, ...) controls 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*, ...) controls 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. 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 *Data*.

- ..., 'Showplot', [3,5,7],...) plots the intensity values in columns 3, 5, and 7 of *Data*.

- ..., 'Showplot', 3:9,...) plots the intensity values in columns 3 to 9 of *PMData*.

Figure 1 — Histogram of log2(PM) and estimated BG density of sample: #3

Examples

1 Load a MAT file, included with Bioinformatics Toolbox, 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/repository/devel/vignette/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`, `celintensityread`, `probelibraryinfo`, `probesetlink`, `probesetlookup`, `probesetvalues`, `quantilenorm`, `rmasummary`

Purpose

Calculate gene (probe set) expression values from Affymetrix microarray probe-level data using Robust Multi-array Average (RMA) procedure

Syntax

ExpressionMatrix = rmasummary(*ProbeIndices*, *Data*)
ExpressionMatrix = rmasummary(..., '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.
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.)
OutputValue	Property to control the scale of the returned gene expression values. *OutputValue* can be:

- 'log'

- 'log2'

- 'log10'

- 'natural'

- @*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 jth probe in the probe set

$Eijn$ = residual for the jth probe on the ith chip

The RMA methods assumes $A1 + A2 + ... + AJ = 0$ for all probe sets. A robust procedure, median polish, is used to estimate 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*, ...) defines optional properties that use property

name/value pairs in any order. These property name/value pairs are as follows:

ExpressionMatrix = rmasummary(..., 'Output', *OutputValue*) controls the scale of the returned gene expression values. *OutputValue* can be:

- 'log'
- 'log2'
- 'log10'
- 'natural'
- @*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 Bioinformatics Toolbox, 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. (You will

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   `affyinvarsetnorm`, `celintensityread`, `mainvarsetnorm`, `malowess`, `manorm`, `quantilenorm`, `rmabackadj`

Purpose   Convert RNA sequence of nucleotides to DNA sequence

Syntax   *SeqDNA* = rna2dna(*SeqRNA*)

Arguments

SeqRNA	Nucleotide sequence for RNA. Enter a character string with the characters A, C, U, G, and the ambiguous nucleotide bases N, R, Y, K, M, S, W, B, D, H, and V.

Description   *SeqDNA* = rna2dna(*SeqRNA*) converts any uracil nucleotides in an RNA sequence into thymine (U-->T), and returns in the same format as DNA. For example, if the RNA sequence is an integer sequence then so is *SeqRNA*.

Example
```
rna2dna('ACGAUGAGUCAUGCUU')

ans =
ACGATGAGTCATGCTT
```

See Also   Bioinformatics Toolbox function: dna2rna

MATLAB functions: strrep, regexp

rnaconvert

Purpose	Convert secondary structure of RNA sequence between bracket and matrix notations
Syntax	*RNAStruct2* = rnaconvert(*RNAStruct*)

Arguments

RNAStruct   Secondary structure of an RNA sequence represented by either:
- Bracket notation
- Connectivity matrix

Tip Use the rnafold function to create *RNAStruct*.

Return Values

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   Bioinformatics Toolbox functions: rnafold, rnaplot

rnafold

Purpose   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, ...)
```

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).

Return Values	*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
Bioinformatics Toolbox functions: `rnaconvert`, `rnaplot`

rnaplot

Purpose	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*, ...)

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.

Return Values

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.

Format	Description
'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.

Note If you specify 'Diagram', you must also use the 'Sequence' property to provide the RNA sequence.

Format	Description
'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.
'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.

Format	Description
'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).
'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. **Note** To create a tree plot, you must have accepted a Graphviz software license (free). If you have not, you will be prompted to do so.

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).

rnaplot

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);
```

rnaplot

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 Bioinformatics Toolbox functions: rnaconvert, rnafold

samplealign

Purpose   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, ...)
```

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.

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.

samplealign

	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.

Return Values	*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.

samplealign

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 MATLAB, 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')
```

Note For examples of using function handles for the Band, Gap, and
Distance properties, see the demo Visualizing and Preprocessing
Hyphenated Mass-Spectrometry Data Sets for Metabolite and
Protein/Peptide Profiling.

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 Bioinformatics Toolbox functions: `msalign`, `msheatmap`, `mspalign`, `msppresample`, `msresample`

scfread

Purpose	Read trace data from SCF file
Syntax	*Sample* = scfread('*File*') [*Sample, Probability*] = scfread('*File*') [*Sample, Probability, Comments*] = scfread('*File*') [*A, C, T, G*] = scfread ('*File*') [*A, C, T, G, ProbA, ProbC, ProbG, ProbT*] = scfread ('*File*') [*A, C, T, G, ProbA, ProbC, ProbG, ProbT, Comments, PkIndex, Base*] = scfread ('*File*')
Arguments	*File* SCF formatted file. Enter a file name or a path and file name.

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, T, G*] = scfread ('*File*') returns the sample data for the four bases in separate variables.

[*A, C, T, G, ProbA, ProbC, ProbG, ProbT*] = scfread ('*File*') also returns the probabilities data for the four bases in separate variables.

[*A, C, T, G, 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. For more information on SCF files, see

    http://www.mrc-lmb.cam.ac.uk/pubseq/manual/formats_unix_2.html

scfread

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   Bioinformatics Toolbox functions: genbankread, traceplot

seq2regexp

Purpose   Convert sequence with ambiguous characters to regular expression

Syntax
RegExp = seq2regexp(*Seq*)
RegExp = seq2regexp(*Seq*, ...'Alphabet', *AlphabetValue*, ...)
RegExp = seq2regexp(*Seq*, ...'Ambiguous',
AmbiguousValue, ...)

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, 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

Return Values

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 , MATLAB returns
  ACGT[GTK] with the unambiguous characters G and T and the
  ambiguous character K.
- If *Seq* = 'ACGTK', and *AmbiguousValue* is false, MATLAB 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

Nucleotide Conversions (Continued)

Nucleotide Code	Nucleotide	Conversion
R	Purine	[AG]
Y	Pyrimidine	[TC]
K	Keto	[GT]
M	Amino	[AC]
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]

Amino Acid Conversion (Continued)

Amino Acid Code	Amino Acid	Conversion
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]`

Example

1 Convert a nucleotide sequence into 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

Bioinformatics Toolbox functions: restrict, seqwordcount

MATLAB functions: regexp, regexpi

seqcomplement

Purpose	Calculate complementary strand of nucleotide sequence

Syntax   *SeqC* = seqcomplement(*SeqNT*)

Arguments

SeqNT	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.

Description   *SeqC* = seqcomplement(*SeqNT*) calculates the complementary strand (A-->T, C-->G, G-->C, T-->A) of a DNA sequence and returns a sequence in the same format as SeqNT. For example, if *SeqNT* is an integer sequence then so is *SeqC*.

Example   Return the complement of a DNA nucleotide sequence.

```
s = 'ATCG';
seqcomplement(s)

ans =
TAGC
```

See Also   Bioinformatics Toolbox functions seqrcomplement, seqreverse, seqtool

Purpose   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	Scoring matrix. The default value is BLOSUM50 for amino acid sequences or NUC44 for nucleotide sequences. ScoringMatrix can also be 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

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 Bioinformatics Toolbox functions fastaread, multialignread, profalign, seqdisp, seqprofile

Purpose   Format long sequence output for easy viewing

Syntax   seqdisp(*Seq*)
                 seqdisp(..., '*PropertyName*', *PropertyValue*,...)
                 seqdisp(..., 'Row', *RowValue*)
                 seqdisp(..., 'Column', *ColumnValue*)
                 seqdisp(..., 'ShowNumbers', *ShowNumbersValue*)

Arguments

Seq	Nucleotide or amino acid sequence. Enter a character array, a FASTA file name, or a MATLAB structure with the field Sequence. Multiply aligned sequences are allowed.
	FASTA files can have the file extension fa, fasta, fas, fsa, or fst.
Row	Property to select the length of each row. Enter an integer. The default length is 60.
Column	Property to select the column width or number of symbols before displaying a space. Enter an integer. The default column width is 10.
ShowNumbers	Property to control displaying numbers at the start of each row. Enter either true (default) to show numbers or false to hide numbers.

Description seqdisp(*Seq*) displays a sequence (Seq) in rows with a default row length of 60 and a default column width of 10.

seqdisp(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

seqdisp(..., 'Row', *RowValue*) specifies the length of each row for the displayed sequence.

seqdisp(..., 'Column', *ColumnValue*) specifies the number of letters to display before adding a space. Row must be larger than and evenly divisible by Column.

seqdisp(..., 'ShowNumbers', *ShowNumbersValue*) when ShowNumbers is false, turns off the position numbers at the start of each row off.

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('AKO80777');
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

Bioinformatics Toolbox functions: multialignread, seqconsensus, seqlogo, seqprofile, seqshoworfs, seqshowwords, seqtool, getgenbank

Purpose   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 dotplot 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);

seqdotplot

```
takin = getgenbank('AB060290','Sequence',true);
seqdotplot(moufflon,takin,11,7)
```



```
Matches = seqdotplot(moufflon,takin,11,7)
Matches =
        5552
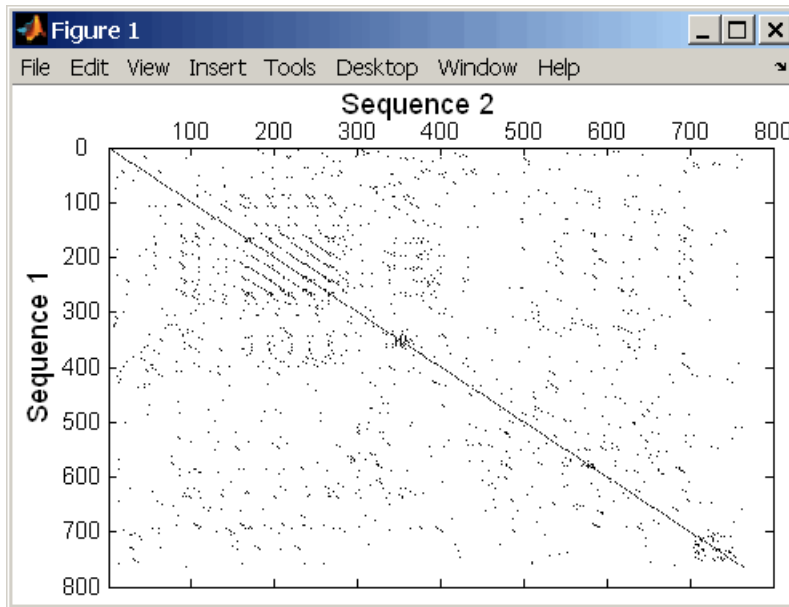
[Matches, Matrix] = seqdotplot(moufflon,takin,11,7)
```

See Also   Bioinformatics Toolbox functions nwalign, swalign

Purpose   Insert gaps into nucleotide or amino acid sequence

Syntax
NewSeq = seqinsertgaps(*Seq*, *Positions*)
NewSeq = seqinsertgaps(*Seq*, *GappedSeq*)
NewSeq = seqinsertgaps(*Seq*, *GappedSeq*, *Relationship*)

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).

Return Values

NewSeq	Sequence with gaps inserted, represented by a string specifying a nucleotide or amino acid sequence.

seqinsertgaps

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   Bioinformatics Toolbox functions: dnds, dndsml, int2aa, int2nt

seqlinkage

Purpose	Construct phylogenetic tree from pairwise distances

Syntax

Tree = seqlinkage(*Dist*)
Tree = seqlinkage(*Dist*, *Method*)
Tree = seqlinkage(*Dist*, *Method*, *Names*)

Arguments

Dist	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	Property to use alternative labels for leaf nodes. Enter a vector of structures, with the fields 'Header' or 'Name', or a cell array of strings. In both cases the number of elements you provide must comply with the number of samples used to generate the pairwise distances in *Dist*.

Description *Tree* = seqlinkage(*Dist*) returns a phylogenetic tree object from the pairwise distances, *Dist*, between the species or products. *Dist* is a matrix or vector of pairwise distances, such as returned by the seqpdist function.

Tree = seqlinkage(*Dist*, *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)

Tree = seqlinkage(*Dist*, *Method*, *Names*) passes a list of names to label the leaf nodes (for example, species or products) in a phylogenetic tree object.

Examples
```
% Load a multiple alignment of amino acids:
seqs = fastaread('pf00002.fa');
% Measure the 'Jukes-Cantor' pairwise distances:
dist = seqpdist(seqs,'method','jukes-cantor',...
                                'indels','pair');
% Build the phylogenetic tree with the single linkage
% method and pass the names of the sequences:
tree = seqlinkage(dist,'single',seqs)
view(tree)
```

See Also
Bioinformatics Toolbox functions: phytree (object constructor), phytreewrite, seqpdist, seqneighjoin

Bioinformatics Toolbox methods of phytree object: plot, view

seqlogo

Purpose   Display sequence logo for nucleotide or amino acid sequences

Syntax
```
seqlogo(Seqs)
seqlogo(Profile)
DisplayInfo = seqlogo(Seqs)
seqlogo(..., 'Displaylogo', DisplaylogoValue, ...)
seqlogo(..., 'Alphabet', AlphabetValue, ...)
seqlogo(..., 'Startat', StartatValue, ...)
seqlogo(..., 'Endat', EndatValue, ...)
seqlogo(..., 'SSCorrection', SSCorrectionValue, ...)
```

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.

Return Values

DisplayInfo	Cell array containing the symbol list in *Seqs* and the weight matrix used to graphically display the sequence logo.

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
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

DisplayInfo = seqlogo(*Seqs*) returns a cell array of unique symbols in a sequence (Seqs) and the information weight matrix used to graphically display the logo.

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   **Displaying a Sequence Logo for a Nucleotide Sequence**

1 Create a series of aligned nucleotide sequences.

```
S = {'ATTATAGCAAACTA',...
     'AACATGCCAAAGTA',...
     'ATCATGCAAAAGGA'}
```

2 Display the sequence logo.

```
seqlogo(S)
```


3 Notice that correction for small samples prevents you from seeing columns with information equal to `log2(4) = 2` bits, but you can turn this adjustment off.

```
seqlogo(S,'sscorrection',false)
```

Displaying a Sequence Logo for an Amino Acid Sequence

1 Create a series of aligned amino acid sequences.

```
S2 = {'LSGGQRQRVAIARALAL',...
      'LSGGEKQRVAIARALMN',...
```

```
'LSGGQIQRVLLARALAA',...
'LSGGERRRLEIACVLAL',...
'FSGGEKKKNELWQMLAL',...
'LSGGERRRLEIACVLAL'};
```

2 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.

seqlogo

See Also   Bioinformatics Toolbox functions: `seqconsensus`, `seqdisp`, `seqprofile`

Purpose   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)
```

See Also   MATLAB functions: `regexp`, `strmatch`

seqneighjoin

Purpose   Neighbor-joining method for phylogenetic tree reconstruction

Syntax   *Tree* = seqneighjoin(*Dist*)
                 Tree = seqneighjoin(*Dist*, *Method*)
                 Tree = seqneighjoin(*Dist*, *Method*, *Names*)
                 seqneighjoin(..., 'Reroot', *RerootValue*)

Arguments

Dist	Matrix or vector returned by the seqpdist function.
Method	Method to compute the distances between nodes. Enter 'equivar' (default), 'firstorder', or 'average'.
Names	Vector of structures with the fields 'Header', 'Name', or a cell array of strings. In all cases the number of elements must equal the number of samples used to generate the pairwise distances in Dist.

Description *Tree* = seqneighjoin(*Dist*) computes a phylogenetic tree object from *Dist*, pairwise distances between the species or products using the neighbor-joining method.

Tree = seqneighjoin(*Dist*, *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 Studier and Keppler, JMBE (1988).
`'firstorder'`	Assumes a first-order model of the variances and covariances of evolutionary distance estimates, `'a'` is adjusted at every iteration to a value between 0 and 1. Such as in Gascuel, JMBE (1997).
`'average'`	New distances are the weighted average of previous distances while the branch distances are ignored. `D(n,k) =  [ D(i,k) + D(j,k) ] /2` As in the original neighbor-joining algorithm by Saitou and Nei, JMBE (1987).

Tree = seqneighjoin(*Dist*, *Method*, *Names*) passes a list of names (*Names*) to label the leaf nodes (e.g., species or products) in the phylogenetic tree object.

seqneighjoin(..., 'Reroot', *RerootValue*), when *RerootValue* is false, excludes rerooting the resulting tree. This is useful for observing the original linkage order followed by the algorithm. By default seqneighjoin reroots the resulting tree using the midpoint method.

Examples

1 Load a multiple alignment of amino acids.

```
seqs = fastaread('pf00002.fa');
```

2 Measure the Jukes-Cantor pairwise distances.

```
dist = seqpdist(seqs,'method','jukes-cantor','indels','pair');
```

3 Build the phylogenetic using the neighbor-joining algorithm.

```
tree = seqneighjoin(dist,'equivar',seqs)
view(tree)
```

seqneighjoin

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   Bioinformatics Toolbox functions: `multialign`, `phytree` (object constructor), `seqlinkage` (alternative method to create a phylogenetic tree), `seqpdist`

Methods of phytree object: `reroot`, `view`

Purpose   Calculate pairwise distance between sequences

Syntax
D = seqpdist(*Seqs*)
D = seqpdist(*Seqs*, ...'Method', *MethodValue*, ...)
D = seqpdist(*Seqs*, ...'Indels', *IndelsValue*, ...)
D = seqpdist(*Seqs*, ...'Optargs', *OptargsValue*, ...)
D = seqpdist(*Seqs*, ...'PairwiseAlignment',
  PairwiseAlignmentValue, ...)
D = seqpdist(*Seqs*, ...'JobManager', *JobManagerValue*, ...)
D = seqpdist(*Seqs*, ...'WaitInQueue', *WaitInQueueValue*, ...)
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*, ...)

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 for calculating pairwise distances. Default is Jukes-Cantor.
IndelsValue	String that specifies how to treat sites with gaps. Default is score.

OptargsValue	String or cell array specifying 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 have the same length, `seqpdist` will assume they 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`.

JobManagerValue	A `jobmanager object`, such as returned by the Distributed Computing Toolbox function `findResource`, that represents an available distributed MATLAB resource. Specifying this property distributes pairwise alignments into a cluster of computers using Distributed Computing Toolbox. You must have Distributed Computing Toolbox to use this property.
WaitInQueueValue	Controls whether `seqpdist` waits for a distributed MATLAB resource to be available when you have set the `JobManager` property. Choices are `true` or `false` (default). You must have Distributed Computing Toolbox to use this property.
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	String specifying the scoring matrix to use for the global pairwise alignment. Choices for amino acid sequences are:

- `'PAM40'`
- `'PAM250'`
- `'DAYHOFF'`
- `'GONNET'`
- `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'`
- `'BLOSUM62'`
- `'BLOSUM100'`

Default is:

- `'NUC44'` (when *AlphabetValue* equals `'NT'`)
- `'BLOSUM50'` (when *AlphabetValue* equals `'AA'`)

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 specifying the penalty for opening a gap in the alignment. Default is 8.
ExtendedGapValue	Positive integer specifying the penalty for extending a gap. Default is equal to *GapOpenValue*.

Return Values

D		Vector containing 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. 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:

D = seqpdist(*Seqs*, ...'Method', *MethodValue*, ...) specifies a method to compute distances between every pair of sequences. 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

Methods for Nucleotides and Amino Acids (Continued)

Method	Description
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 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 form 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. You can set a by using the Optargs property. Default is 2.

You can also specify a user-defined distance function using @, for example, @distfun. The distance function must be of 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 a multiple alignment was provided 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 multiple input arguments. For example, you can 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` will assume they 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*, ...'JobManager', *JobManagerValue*, ...) distributes pairwise alignments into a cluster of computers using Distributed Computing Toolbox. *JobManagerValue* is a `jobmanager` `object` such as returned by the Distributed Computing Toolbox function `findResource`, that represents an available distributed MATLAB resource. You must have Distributed Computing Toolbox to use this property.

D = seqpdist(*Seqs*, ...'WaitInQueue', *WaitInQueueValue*, ...) controls whether seqpdist waits for a distributed MATLAB resource to be available when you have set the JobManager property. When *WaitInQueueValue* is true, seqpdist waits in the job manager queue for an available worker. When *WaitInQueueValue* is false (default) and there are no workers immediately available, seqpdist stops and displays an error message. You must have Distributed Computing Toolbox and have also set the JobManager property to use this property.

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 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 acids 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 every pair of sequences 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 every
pair of sequences, counting the gaps as point mutations.

```
dist = seqpdist(seqs,'Method','jukes-cantor',...
                'Indels','score',...
                'Scoringmatrix','pam250',...
                'PairwiseAlignment',true);
```

See Also

Bioinformatics Toolbox functions: fastaread, dnds, dndsml,
multialign, nwalign, phytree (object constructor), seqlinkage

Bioinformatics Toolbox object: phytree object

Bioinformatics Toolbox method of a phytree object: pdist

seqprofile

Purpose	Calculate sequence profile from set of multiply aligned sequences

Syntax	`Profile = seqprofile(Seqs, 'PropertyName', PropertyValue ...)`
	`[Profile, Symbols] = seqprofile(Seqs)`
	`seqprofile(..., 'Alphabet', AlphabetValue)`
	`seqprofile(..., 'Counts', CountsValue)`
	`seqprofile(..., 'Gaps', GapsValue)`
	`seqprofile(..., 'Ambiguous', AmbiguousValue),`
	`seqprofile(..., 'Limits', LimitsValue)`

Arguments

Seqs	Set of multiply aligned sequences. Enter an array of strings, cell array of strings, or an array of structures with the field `Sequence`.
Alphabet	Sequence alphabet. Enter `'NT'` (nucleotides), `'AA'` (amino acids), or `'none'`. The default alphabet is `'AA'`. When `Alphabet` is `'none'`, the symbol list is based on the observed symbols. Every character can be a symbol except for a hyphen (-) and a period (.), which are reserved for gaps.
Count	Property to control returning frequency (ratio of counts/total counts) or counts. Enter either `true` (counts) or `false` (frequency). The default value is `false`.
Gaps	Property to control counting gaps in a sequence. Enter `'all'` (counts all gaps), `'noflanks'` (counts all gaps except those at the flanks of every sequence), or `'none'`. The default value is `'none'`.

`Ambiguous`	Property to control counting ambiguous symbols. Enter `'Count'` to add partial counts to the standard symbols.
`Limits`	Property to specify using part of the sequences. Enter a `[1x2]` vector with the first position and the last position to include in the profile. The default value is `[1,SeqLength]`.

Description `Profile = seqprofile(Seqs, 'PropertyName', PropertyValue ...)` returns a matrix (`Profile`) 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 a unique symbol list (`Symbols`) where every symbol in the list corresponds to a row in the profile (`Profile`).

`seqprofile(..., 'Alphabet', AlphabetValue)` selects a nucleotide alphabet, amino acid alphabet, or no alphabet.

`seqprofile(..., 'Counts', CountsValue)` when `Counts` is `true`, returns the counts instead of the frequency.

`seqprofile(..., 'Gaps', GapsValue)` appends a row to the bottom of a profile (`Profile`) with the count for gaps.

`seqprofile(..., '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(..., 'Limits', LimitsValue)` specifies the start and end positions for the profile relative to the indices of the multiple alignment.

seqprofile

Examples

```
seqs = fastaread('pf00002.fa');
[P,S] = seqprofile(seqs,'limits',[50 60],'gaps','all')
```

See Also   Bioinformatics Toolbox functions `fastaread`, `multialignread`, `seqconsensus`, `seqdisp`, `seqlogo`

Purpose   Calculate reverse complement of nucleotide sequence

Syntax   `SeqRC = seqrcomplement(SeqNT)`

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.

Description   `seqrcomplement` calculates the reverse complementary strand of a DNA sequence.

`SeqRC = seqrcomplement(SeqNT)` calculates the reverse complementary strand 3' --> 5' (A-->T, C-->G, G-->C, T-->A) for a DNA sequence and returns a sequence in the same format as SeqNT. For example, if SeqNT is an integer sequence then so is SeqRC.

Examples   Reverse a DNA nucleotide sequence and then return its complement.

```
s = 'ATCG'
seqrcomplement(s)

ans =
CGAT
```

See Also   Bioinformatics Toolbox functions codoncount, palindromes, seqcomplement, seqreverse, seqtool

seqreverse

Purpose	Reverse letters or numbers in nucleotide sequence
Syntax	*SeqR* = seqreverse(*SeqNT*)

Arguments

SeqNT	Enter a 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.
SeqR	Returns a sequence in the same format as the nucleotide sequence. For example, if *SeqNT* is an integer sequence, then so is *SeqR*.

Description   seqreverse calculates the reverse strand of a DNA or RNA sequence.

SeqR = seqreverse(*SeqNT*) calculates the reverse strand 3' --> 5' of the nucleotide sequence.

Examples   Reverse a nucleotide sequence.

```
s = 'ATCG'
seqreverse(s)

ans =
GCTA
```

See Also   Bioinformatics Toolbox functions: seqcomplement, seqrcomplement, seqtool

MATLAB function: fliplr

Purpose	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 on page 2-6.
MinimumLengthValue	Property to set the minimum number of codons in an ORF.

seqshoworfs

AlternativeStartCodonsValue	Property to control using alternative start codons. Enter either `true` or `false`. The default value is `false`.
ColorValue	Property to select the color for highlighting the reading frame. Enter either a 1-by-3 RGB vector specifying the intensity (`0 to 255`) of the red, green, and blue components of the color, or a character from the following list: `'b'`—blue, `'g'`—green, `'r'`—red, `'c'`—cyan, `'m'`—magenta, or `'y'`—yellow.

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 COLOR should be a 1-by-6 cell array of color values. |
| *ColumnsValue* | Property to specify the number of columns in the output. |

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*, ...) selects 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 frame.

seqshoworfs(*SeqNT*, ...'Columns', *ColumnsValue*, ...) specifies how many columns per line to use in the output. The default value is 64.

Examples   Look for the open reading frames in a random nucleotide sequence.

```
s = randseq(200, 'alphabet', 'dna');
seqshoworfs(s);
```

Identify the open reading frames in a GenBank sequence.

```
HLA_DQB1 = getgenbank('NM_002123');
seqshoworfs(HLA_DQB1.Sequence);
```

See Also   Bioinformatics Toolbox functions: codoncount, cpgisland, geneticcode, seqdisp, seqshowwords, seqtool, seqwordcount

MATLAB function: regexp

seqshowwords

Purpose	Graphically display words in sequence
Syntax	seqshowwords(*Seq*, *Word*) seqshowwords(*Seq*, *Word*, ...'Color', *ColorValue*, ...) seqshowwords(*Seq*, *Word*, ...'Columns', *ColumnsValue*, ...) seqshowwords(*Seq*, *Word*, ...'Alphabet', *AlphabetValue*, ...)

Arguments

Seq	Enter either a nucleotide or amino acid sequence. You can also enter a structure with the field Sequence.
Word	Enter a short character sequence.
ColorValue	Property to select the color for highlighted characters. Enter a 1-by-3 RGB vector specifying the intensity (0 255) of the red, green, and blue components, or enter a character from the following list: 'b' – blue, 'g' – green, 'r' – red, 'c' – cyan, 'm' – magenta, or 'y' – yellow. The default color is red 'r'.
ColumnsValue	Property to specify the number of characters in a line. Default value is 64.
AlphabetValue	Property to select the alphabet. Enter 'AA' for amino acid sequences or 'NT' for nucleotide sequences. The default is 'NT'.

Description seqshowwords(*Seq*, *Word*) displays the sequence with all occurrences of a word highlighted, and returns a structure with the start and stop positions for all occurrences of the word 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. Each *PropertyName* must

be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

seqshowwords(*Seq*, *Word*, ...'Color', *ColorValue*, ...) selects the color used to highlight the words in the output display.

seqshowwords(*Seq*, *Word*, ...'Columns', *ColumnsValue*, ...) specifies how many columns per line to use in the output.

seqshowwords(*Seq*, *Word*, ...'Alphabet', *AlphabetValue*, ...) selects the alphabet for the sequence (*Seq*) and the word (*Word*).

If the search work (*Word*) contains nucleotide or amino acid symbols that represent multiple possible symbols, then seqshowwords shows all 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'.

Examples

This example shows two matches, 'TAGT' and 'TAAT', for the word 'BART'.

```
seqshowwords('GCTAGTAACGTATATATAAT','BART')

ans =
    Start: [3 17]
    Stop: [6 20]

000001 GCTAGTAACGTATATATAAT
```

seqshowwords does not highlight overlapping patterns multiple times. This example highlights two places, the first occurrence of 'TATA' and the 'TATATATA' immediately after 'CG'. The final 'TA' is not highlighted because the preceding 'TA' is part of an already matched pattern.

```
seqshowwords('GCTATAACGTATATATATA','TATA')

ans =
    Start: [3 10 14]
```

```
        Stop: [6 13 17]

000001 GCTATAACGTATATATATA
```

To highlight all multiple repeats of TA, use the regular expression `'TA(TA)*TA'`.

```
seqshowwords('GCTATAACGTATATATATA','TA(TA)*TA')

ans =
    Start: [3 10]
    Stop: [6 19]

000001 GCTATAACGTATATATATA
```

See Also   Bioinformatics Toolbox functions: `palindromes`, `cleave`, `restrict`, `seqdisp`, `seqtool`, `seqwordcount`

MATLAB functions: `strfind`, `regexp`

Purpose   Open tool to interactively explore biological sequences

Syntax   seqtool(*Seq*)
                   seqtool(..., '*PropertyName*', *PropertyValue*,...)
                   seqtool(..., 'Alphabet', *AlphabetValue*)

Arguments

Seq	Struct with a field Sequence, a character array, or a file name with an extension of .gbk, .gpt, .fasta, .fa, or .ebi

Description   seqtool(*Seq*) loads a sequence (*Seq*) into the seqtool GUI.

seqtool(..., '*PropertyName*', *PropertyValue*,...) defines optional properties using property name/value pairs.

seqtool(..., 'Alphabet', *AlphabetValue*) specifies an alphabet (*AlphabetValue*) for the sequence (*Seq*). Default is 'AA', except when all of the symbols in the sequence are A, C, G, T, and -, then *AlphabetValue* is set to 'NT'. Use 'AA' when you want to force an amino acid sequence alphabet.

Example   **1** Get a sequence from Genbank.

        S = getgenbank('M10051')

2 Open the sequence tool window with the sequence.

        seqtool(S)

seqtool

See Also   Bioinformatics Toolbox functions: aa2nt, aacount, aminolookup, basecount, baselookup, dimercount, emblread, fastaread, fastawrite, genbankread, geneticcode, genpeptread, getembl, getgenbank, getgenpept, nt2aa, proteinplot, seqcomplement, seqdisp, seqrcomplement, seqreverse, seqshoworfs, seqshowwords, seqwordcount

Purpose   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
```

seqwordcount

See Also   Bioinformatics Toolbox functions codoncount, seqshoworfs, seqshowwords, seqtool, seq2regexp

MATLAB functions strfind

Purpose   Display color-coded sequence alignment

Syntax   showalignment(*Alignment*)
                   showalignment(*Alignment*, ...'MatchColor',
                   MatchColorValue, ...)
                   showalignment(*Alignment*,
                   ...'SimilarColor' *SimilarColorValue*,
                      ...)
                   showalignment(*Alignment*, ...'StartPointers',
                      StartPointersValue, ...)
                   showalignment(*Alignment*, ...'Columns', *ColumnsValue*, ...)

Arguments	*Alignment*	Either of the following:

Either of the following:

- 3-by-N character array showing the pairwise alignment of two sequences, such as returned by the `nwalign` or `swalign` function.

- Vector or character array containing a multiple alignment, such as returned by the `multialign` function.

MatchColorValue   Color to highlight matching characters. Can be either of the following:

- 1-by-3 RGB vector

- String specifying one of the following colors:

  - `'r'` — red
  - `'g'` — green
  - `'b'` — blue
  - `'c'` — cyan
  - `'m'` — magenta
  - `'y'` — yellow

Default is red, `[255 0 0]` or `'r'`.

SimilarColorValue	Color to highlight similar characters. Can be either of the following:

- 1-by-3 RGB vector

- String specifying one of the following colors:

  - 'r' — red

  - 'g' — green

  - 'b' — blue

  - 'c' — cyan

  - 'm' — magenta

  - 'y' — yellow

Default is magenta, [255 0 255] or 'm'.

StarterPointersValue	Two-element vector that specifies the starting indices in the original sequences of a local pairwise alignment, such as the third output returned by the swalign function.
ColumnsValue	Scalar that specifies the number of characters to display in one row when displaying a pairwise alignment. Default is 64.

Description showalignment(*Alignment*) displays a color-coded alignment in a MATLAB Figure window. For pairwise sequence alignments, matching and similar characters are highlighted in red and magenta, respectively. For multiple sequence alignments, highly conserved positions are highlighted in red and conserved positions are highlighted in magenta.

showalignment(*Alignment*, ...'*PropertyName*', *PropertyValue*, ...) calls showalignment 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:

showalignment(*Alignment*, ...'MatchColor', *MatchColorValue*, ...) specifies the color to highlight matching characters in the output display. Default is red, [255 0 0] or 'r'. For example, to use cyan, enter [0 255 255] or 'c'.

showalignment(*Alignment*, ...'SimilarColor' *SimilarColorValue*, ...) specifies the color to highlight similar characters in the output display. Default is magenta, [255 0 255] or 'm'.

Note The 'StartPointers' and 'Columns' properties are available only when displaying pairwise alignments.

---

showalignment(*Alignment*, ...'StartPointers', *StartPointersValue*, ...) specifies the starting indices in the original sequences of a local pairwise alignment.

Tip You can use the third output returned by the swalign function as the *StartPointersValue*.

---

showalignment(*Alignment*, ...'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. Default is 64.

Tip To view a multiple-sequence alignment and interact with it, use the multialignviewer function.

Examples   Globally align two amino acid sequences and display their color-coded alignment.

```
[Score, Alignment] = nwalign('VSPAGMASGYD','IPGKASYD');
showalignment(Alignment);
```



```
Aligned Sequences                                          _ □ ×
🖨                                                              ↘
Identities = 6/11 (55%), Positives = 7/11 (64%)
VSPAGMASGYD
: | | | || ||
I-P-GKAS-YD
```

Read a multiple-sequence alignment file and display the color-coded alignment.

```
gag = multialignread('aagag.aln');
showalignment(gag)
```

showalignment

Tip To view a multiple-sequence alignment and interact with it, use the `multialignviewer` function.

See Also
Bioinformatics Toolbox functions: `multialign`, `multialignviewer`, `nwalign`, `swalign`

Purpose   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   Bioinformatics Toolbox functions: gethmmprof, hmmprofalign, hmmprofestimate, hmmprofgenerate, hmmprofstruct, pfamhmmread

sptread

Purpose   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 Directory.

- 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
```

sptread

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 MATLAB 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 Bioinformatics Toolbox.

```
spotStruct = sptread('spotdata.txt')
maimage(spotStruct,'Rmedian');
```

2 Alternatively, create a similar plot using more basic graphics commands.

```
Rmedian = magetfield(spotStruct,'Rmedian');
imagesc(Rmedian(spotStruct.Indices));
colormap bone
colorbar
```

See Also

Bioinformatics Toolbox functions: affyread, agferead, celintensityread, geosoftread, gprread, imageneread, maboxplot, magetfield

Purpose	Classify data using support vector machine

Syntax
   Group = svmclassify(*SVMStruct*, *Sample*)
   Group = svmclassify(*SVMStruct*, *Sample*, 'Showplot', *ShowplotValue*)

Description
   Group = svmclassify(*SVMStruct*, *Sample*) classifies each row of the data in Sample using the information in a support vector machine classifier structure SVMStruct, created using the svmtrain function. Sample must have the same number of columns as the data used to train the classifier in svmtrain. Group indicates the group to which each row of Sample has been assigned.

   Group = svmclassify(*SVMStruct*, *Sample*, 'Showplot', *ShowplotValue*) controls the plotting of the sample data in the figure created using the Showplot property with the svmtrain function.

Examples

1 Load the sample data, which includes Fisher's iris data of 5 measurements on a sample of 150 irises.

```
load fisheriris
```

2 Create data, a two-column matrix containing sepal length and sepal width measurements for 150 irises.

```
data = [meas(:,1), meas(:,2)];
```

3 From the species vector, create a new column vector, groups, to classify data into two groups: Setosa and non-Setosa.

```
groups = ismember(species,'setosa');
```

4 Randomly select training and test sets.

```
[train, test] = crossvalind('holdOut',groups);
cp = classperf(groups);
```

5 Use the `svmtrain` function to train an SVM classifier using a linear kernel function and plot the grouped data.

```
svmStruct = svmtrain(data(train,:),groups(train),'showplot',true);
```


6 Add a title to the plot, using the `KernelFunction` field from the `svmStruct` structure as the title.

```
title(sprintf('Kernel Function: %s',...
              func2str(svmStruct.KernelFunction)),...
              'interpreter','none');
```

Figure 1 — Kernel Function: linear_kernel

7 Classify the test set using a support vector machine.

```
classes = svmclassify(svmStruct,data(test,:),'showplot',true);
```

8 Evaluate the performance of the classifier.

```
classperf(cp,classes,test);
cp.CorrectRate
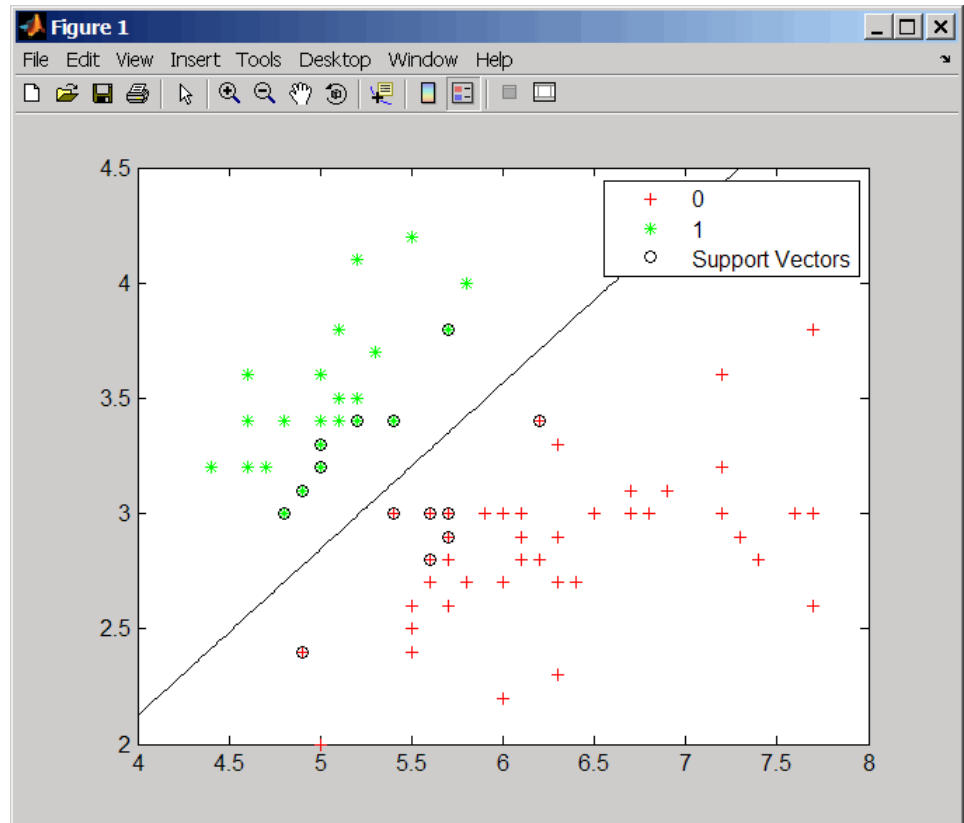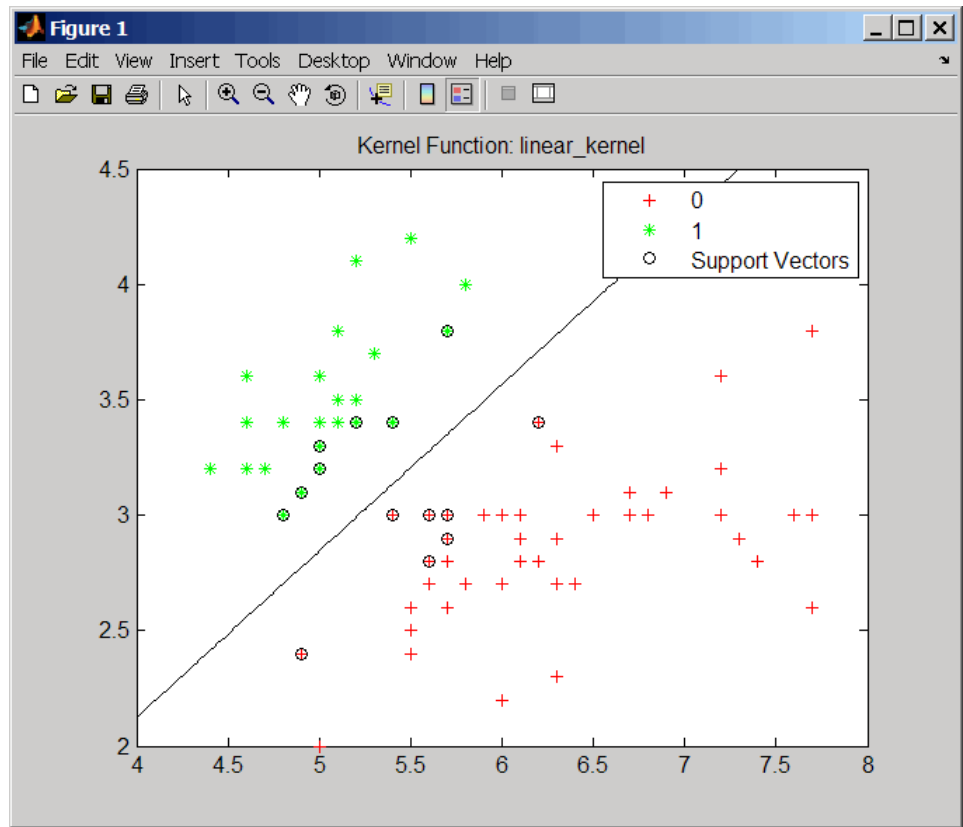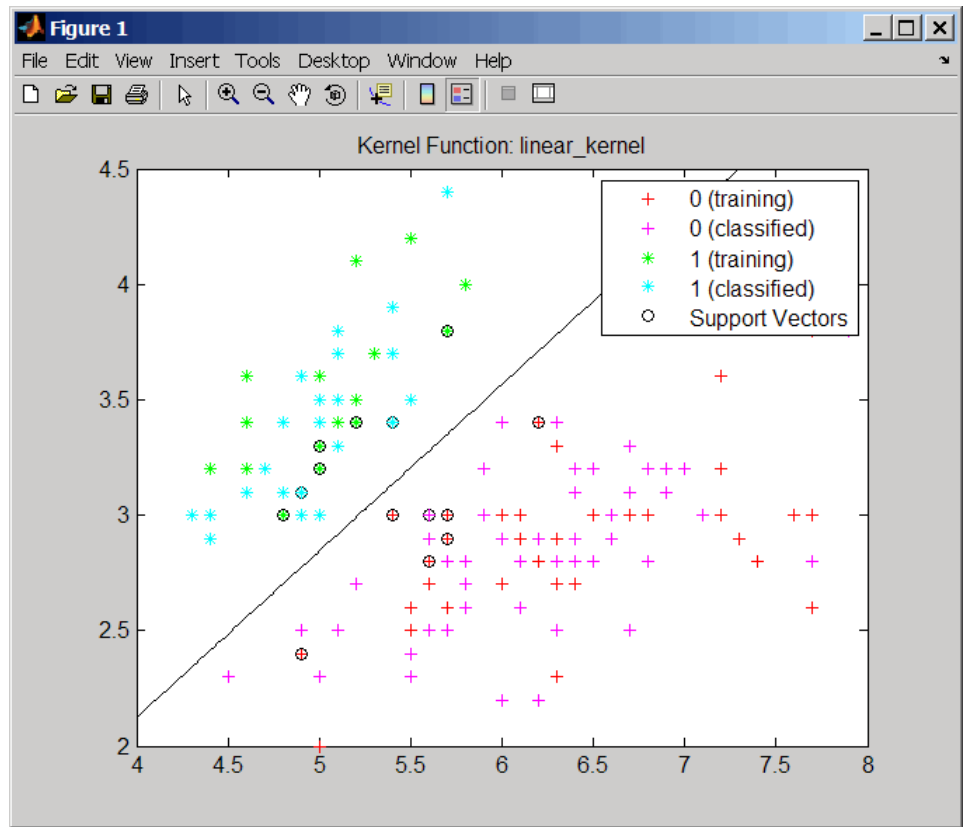
ans =

    0.9867
```

9 Use a one-norm, hard margin support vector machine classifier by changing the `boxconstraint` property.

```
figure
svmStruct = svmtrain(data(train,:),groups(train),...
                     'showplot',true,'boxconstraint',1e6);
```



```
classes = svmclassify(svmStruct,data(test,:),'showplot',true);
```

10 Evaluate the performance of the classifier.

```
classperf(cp,classes,test);
cp.CorrectRate

ans =
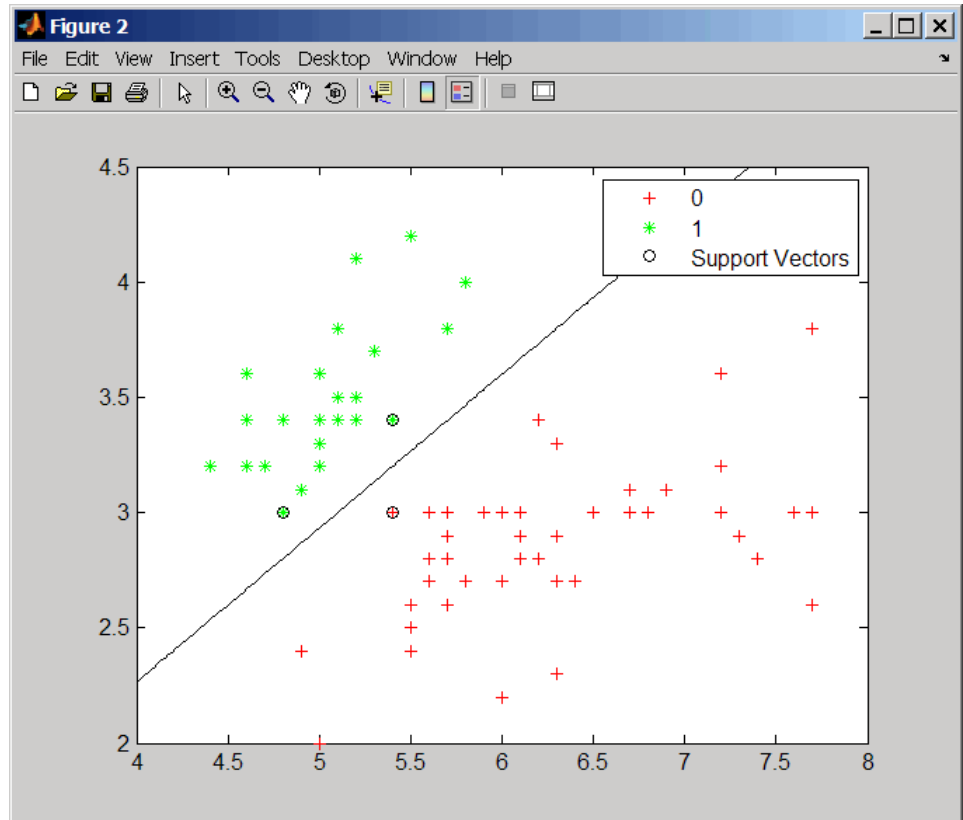
    0.9867
```

References
[1] Kecman, V., Learning and Soft Computing, MIT Press, Cambridge, MA. 2001.

[2] Suykens, J.A.K., Van Gestel, T., De Brabanter, J., De Moor, B., and Vandewalle, J., Least Squares Support Vector Machines, World Scientific, Singapore, 2002.

[3] Scholkopf, B., and Smola, A.J., Learning with Kernels, MIT Press, Cambridge, MA. 2002.

[4] Cristianini, N., and Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, First Edition (Cambridge: Cambridge University Press). http://www.support-vector.net/

See Also   Bioinformatics Toolbox functions: `classperf`, `crossvalind`, `knnclassify`, `svmtrain`

Statistics Toolbox function: `classify`

Optimization Toolbox function: `quadprog`

svmsmoset

Purpose
Create or edit Sequential Minimal Optimization (SMO) options structure

Syntax
SMO_OptsStruct = svmsmoset('*Property1Name*', *Property1Value*, '*Property2Name*', *Property2Value*, ...)
SMO_OptsStruct = svmsmoset(*OldOpts*, '*Property1Name*', *Property1Value*, '*Property2Name*', *Property2Value*, ...)
SMO_OptsStruct = svmsmoset(*OldOpts*, *NewOpts*)

Arguments

OldOpts	Structure that specifies options used by the SMO method of the svmtrain function.
NewOpts	Structure that specifies options used by the SMO method of the svmtrain function.

PropertyName	**Description of** *PropertyValue*
TolKKT	Value that specifies the tolerance with which the KKT conditions are checked. KKT conditions are Karush-Kuhn-Tucker conditions. Default is 1.0000e-003.
MaxIter	Integer that specifies the maximum number of iterations of the main loop. If this limit is exceeded before the algorithm converges, then the algorithm stops and returns an error. Default is 15000.

PropertyName	**Description of** *PropertyValue*
Display	String that specifies the level of information about the optimization iterations that is displayed as the algorithm runs. Choices are: • off — Default. Reports nothing. • iter — Reports every 500 iterations. • final — Reports only when the algorithm finishes.
KKTViolationLevel	Value that specifies the fraction of variables allowed to violate the KKT conditions. Choices are any value $\geq$ 0 and < 1. Default is 0. For example, if you set KKTViolationLevel to 0.05, then 5% of the variables are allowed to violate the KKT conditions. **Tip** Set this option to a positive value to help the algorithm converge if it is fluctuating near a good solution. For more information on KKT conditions, see Cristianini, et al. 2000.
KernelCacheLimit	Value that specifies the size of the kernel matrix cache. The algorithm keeps a matrix with up to KernelCacheLimit × KernelCacheLimit double-precision, floating-point numbers in memory. Default is 5000.

Return Values

SMO_OptsStruct Structure that specifies options used by the SMO method used by the svmtrain function.

svmsmoset

Description   *SMO_OptsStruct* = svmsmoset('*Property1Name*', *Property1Value*, '*Property2Name*', *Property2Value*, ...) creates *SMO_OptsStruct*, an SMO options structure from the specified inputs. This structure can be used as input for the svmtrain function.

SMO_OptsStruct = svmsmoset(*OldOpts*, '*Property1Name*', *Property1Value*, '*Property2Name*', *Property2Value*, ...) alters the options in *OldOpts*, an existing SMO options structure, with the specified inputs, creating a new output options structure.

SMO_OptsStruct = svmsmoset(*OldOpts*, *NewOpts*) alters the options in *OldOpts*, an existing SMO options structure, with the options specified in *NewOpts*, another SMO options structure, creating a new output options structure.

Examples   **1** Create an SMO options structure and specify the Display, MaxIter, and KernelCacheLimit properties.

```
opts = svmsmoset('Display','final','MaxIter',20000,...
                  'KernelCacheLimit',1000)

opts =

              Display: 'final'
               TolKKT: 1.0000e-003
              MaxIter: 20000
       KKTViolationLevel: 0
        KernelCacheLimit: 1000
```

2 Create an alternate SMO options structure from the previous structure. Specify different Display and KKTViolationLevel properties.

```
alt_opts = svmsmoset(opts,'Display','iter','KKTViolationLevel',.05)

alt_opts =

              Display: 'iter'
```

```
          TolKKT: 1.0000e-003
          MaxIter: 20000
KKTViolationLevel: 0.0500
 KernelCacheLimit: 1000
```

References

[1] Cristianini, N., and Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, First Edition (Cambridge: Cambridge University Press). http://www.support-vector.net/

[2] Platt, J.C. (1999). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. In Advances in Kernel Methods - Support Vector Learning, B. Scholkopf, J.C. Burges, and A.J. Smola, eds. (Cambridge MA: MIT Press), pp. 185–208.

[3] Fan, R.E., Chen, P.H., and Lin, C.J. (2005). Working Set Selection Using Second Order Information for Training SVM. Journal of Machine Learning Research *6*, 1889–1918.

[4] Bottou, L. and Lin, C.J. (2006). Support Vector Machine Solvers. http://www.csie.ntu.edu.tw/~cjlin/papers.html

See Also

Bioinformatics Toolbox functions: svmclassify, svmtrain

Optimization Toolbox function: optimset

svmtrain

Purpose   Train support vector machine classifier

Syntax
SVMStruct = svmtrain(*Training*, *Group*)
SVMStruct = svmtrain(..., 'Kernel_Function',
Kernel_FunctionValue, ...)
SVMStruct = svmtrain(..., 'RBF_Sigma', *RBFSigmaValue*, ...)
SVMStruct = svmtrain(..., 'Polyorder', *PolyorderValue*, ...)
SVMStruct = svmtrain(..., 'Mlp_Params',
Mlp_ParamsValue, ...)
SVMStruct = svmtrain(..., 'Method', *MethodValue*, ...)
SVMStruct = svmtrain(..., 'QuadProg_Opts',
   QuadProg_OptsValue, ...)
SVMStruct = svmtrain(..., 'SMO_Opts', *SMO_OptsValue*, ...)
SVMStruct = svmtrain(..., 'BoxConstraint',
   BoxConstraintValue, ...)
SVMStruct = svmtrain(..., 'Autoscale', *AutoscaleValue*, ...)
SVMStruct = svmtrain(..., 'Showplot', *ShowplotValue*, ...)

Arguments

Training	Matrix of training data, where each row corresponds to an observation or replicate, and each column corresponds to a feature or variable.
Group	Column vector, character array, or cell array of strings for classifying data in *Training* into two groups. It has the same number of elements as there are rows in *Training*. Each element specifies the group to which the corresponding row in *Training* belongs.

Kernel_FunctionValue	String or function handle specifying the kernel function that maps the training data into kernel space. Choices are:

- linear — Default. Linear kernel or dot product.

- quadratic — Quadratic kernel.

- rbf — Gaussian Radial Basis Function kernel with a default scaling factor, sigma, of 1.

- polynomial — Polynomial kernel with a default order of 3.

- mlp — Multilayer Perceptron kernel with default scale and bias parameters of [1, -1].

- @functionname — Handle to a kernel function specified using @and the functionname. For example, @kfun, or an anonymous function.

RBFSigmaValue	Positive number that specifies the scaling factor, sigma, in the radial basis function kernel. Default is 1.
PolyorderValue	Positive number that specifies the order of a polynomial kernel. Default is 3.
Mlp_ParamsValue	Two-element vector, [p1, p2], that specifies the scale and bias parameters of the multilayer perceptron (mlp) kernel. K = tanh(p1*U*V' + p2). p1 must be > 0, and p2 must be < 0. Default is [1, -1].

MethodValue	String specifying the method to find the separating hyperplane. Choices are: • QP — Quadratic Programming (requires Optimization Toolbox). The classifier is a two-norm, soft-margin support vector machine. • SMO — Sequential Minimal Optimization. The classifier is a one-norm, soft-margin support vector machine. • LS — Least-Squares. If you installed Optimization Toolbox, the QP method is the default. Otherwise, the SMO method is the default.
QuadProg_OptsValue	An options structure created by the optimset function (Optimization Toolbox). This structure specifies options used by the QP method. For more information on creating this structure, see the optimset and quadprog reference pages.
SMO_OptsValue	An options structure created by the svmsmoset function. This structure specifies options used by the SMO method. For more information on creating this structure, see the svmsmoset function.

BoxConstraintValue	Box constraints for the soft margin. Choices are:
	• Strictly positive numeric scalar.
	• Array of strictly positive values with the number of elements equal to the number of rows in the *Training* matrix.
	If *BoxConstraintValue* is a scalar, it is automatically rescaled by $N/(2*N1)$ for the data points of group one and by $N/(2*N2)$ for the data points of group two. N1 is the number of elements in group one, N2 is the number of elements in group two, and N = N1 + N2. This rescaling is done to take into account unbalanced groups, that is cases where N1 and N2 have very different values.
	If *BoxConstraintValue* is an array, then each array element is taken as a box constraint for the data point with the same index. Default is a scalar value of 1.
AutoscaleValue	Controls the shifting and scaling of data points before training. When *AutoscaleValue* is true, the columns of the input data matrix *Training* are shifted to zero mean and scaled to unit variance. Default is true.
ShowplotValue	Controls the display of a plot of the grouped data, including the separating line for the classifier, when using two-dimensional data. Choices are true or false (default).

svmtrain

Return Values

SVMStruct   Structure containing information about the trained SVM classifier, including the following fields:

- SupportVectors

- Alpha

- Bias

- KernelFunction

- KernelFunctionArgs

- GroupNames

- SupportVectorIndices

- ScaleData

- FigureHandles

Tip You can use *SVMStruct* as input to the svmclassify function, to use for classification.

Description   *SVMStruct* = svmtrain(*Training*, *Group*) trains a support vector machine (SVM) classifier using *Training*, a matrix of training data taken from two groups, specified by *Group*. svmtrain treats NaNs or empty strings in *Group* as missing values and ignores the corresponding rows of *Training*. Information about the trained SVM classifier is returned in *SVMStruct*, a structure with the following fields.

- SupportVectors

- Alpha

- Bias

- KernelFunction

- `KernelFunctionArgs`

- `GroupNames`

- `SupportVectorIndices`

- `ScaleData`

- `FigureHandles`

SVMStruct = svmtrain(*Training*, *Group*, ...'*PropertyName*', *PropertyValue*, ...) calls svmtrain 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:

SVMStruct = svmtrain(..., 'Kernel_Function', *Kernel_FunctionValue*, ...) specifies the kernel function (*Kernel_FunctionValue*) that maps the training data into kernel space. *Kernel_FunctionValue* can be one of the following strings or a function handle:

- `linear` — Default. Linear kernel or dot product.

- `quadratic` — Quadratic kernel.

- `rbf` — Gaussian Radial Basis Function kernel with a default scaling factor, sigma, of 1.

- `polynomial` — Polynomial kernel with a default order of 3.

- `mlp` — Multilayer Perceptron kernel with default scale and bias parameters of [1, -1].

- `@functionname` — Handle to a kernel function specified using @and the functionname. For example, @kfun, or an anonymous function.

A kernel function must be of the following form:

```
function K = kfun(U, V)
```

Input arguments `U` and `V` are matrices with `m` and `n` rows respectively. Return value `K` is an `m`-by-`n` matrix. If `kfun` is parameterized, you can use anonymous functions to capture the problem-dependent parameters. For example, suppose that your kernel function is:

```
function K = kfun(U,V,P1,P2)
K = tanh(P1*(U*V')+P2);
```

You can set values for `P1` and `P2` and then use an anonymous function as follows:

```
@(U,V) kfun(U,V,P1,P2)
```

For more information on the types of functions that can be used as kernel functions, see Cristianini and Shawe-Taylor, 2000.

SVMStruct = svmtrain(..., 'RBF_Sigma', *RBFSigmaValue*, ...) specifies the scaling factor, sigma, in the radial basis function kernel. *RBFSigmaValue* must be a positive number. Default is 1.

SVMStruct = svmtrain(..., 'Polyorder', *PolyorderValue*, ...) specifies the order of a polynomial kernel. *PolyorderValue* must be a positive number. Default is 3.

SVMStruct = svmtrain(..., 'Mlp_Params', *Mlp_ParamsValue*, ...) specifies the scale and bias parameters of the multilayer perceptron (mlp) kernel as a two-element vector, [p1, p2]. K = tanh(p1*U*V' + p2), p1 > 0, and p2 < 0. p1 must be > 0, and p2 must be < 0. Default is [1, -1].

SVMStruct = svmtrain(..., 'Method', *MethodValue*, ...) specifies the method to find the separating hyperplane. Choices are:

- `QP` — Quadratic Programming (requires Optimization Toolbox). The classifier is a two-norm, soft-margin support vector machine.

- `SMO` — Sequential Minimal Optimization. The classifier is a one-norm, soft-margin support vector machine.

- `LS` — Least-Squares.

If you installed Optimization Toolbox, the QP method is the default. Otherwise, the SMO method is the default.

Note If you specify the QP method, the classifier is a two-norm, soft-margin support vector machine.

SVMStruct = svmtrain(..., 'QuadProg_Opts', *QuadProg_OptsValue*, ...) specifies an options structure created by the optimset function (Optimization Toolbox). This structure specifies options used by the QP method. For more information on creating this structure, see the optimset and quadprog functions.

SVMStruct = svmtrain(..., 'SMO_Opts', *SMO_OptsValue*, ...) specifies an options structure created by svmsmoset function. This structure specifies options used by the SMO method. For more information on creating this structure, see the svmsmoset function.

SVMStruct = svmtrain(..., 'BoxConstraint', *BoxConstraintValue*, ...) specifies box constraints for the soft margin. *BoxConstraintValue* can be either of the following:

- Strictly positive numeric scalar
- Array of strictly positive values with the number of elements equal to the number of rows in the *Training* matrix

If *BoxConstraintValue* is a scalar, it is automatically rescaled by $N/(2*N1)$ for the data points of group one and by $N/(2*N2)$ for the data points of group two. N1 is the number of elements in group one, N2 is the number of elements in group two, and $N = N1 + N2$. This rescaling is done to take into account unbalanced groups, that is cases where N1 and N2 have very different values.

If *BoxConstraintValue* is an array, then each array element is taken as a box constraint for the data point with the same index.

Default is a scalar value of 1.

SVMStruct = svmtrain(..., 'Autoscale', *AutoscaleValue*, ...) controls the shifting and scaling of data points before training. When *AutoscaleValue* is true, the columns of the input data matrix *Training* are shifted to zero mean and scaled to unit variance. Default is true.

SVMStruct = svmtrain(..., 'Showplot', *ShowplotValue*, ...), controls the display of a plot of the grouped data , including the separating line for the classifier, when using two-dimensional data. Choices are true or false (default).

Memory Usage and Out of Memory Error

When you set 'Method' to 'QP', the svmtrain function operates on a data set containing N elements, it creates an (N+1)-by-(N+1) matrix to find the separating hyperplane. This matrix needs at least 8*(n+1)^2 bytes of contiguous memory. If this size of contiguous memory is not available, MATLAB displays an "out of memory" message.

When you set 'Method' to 'SMO', memory consumption is controlled by the SMO option KernelCacheLimit. For more information on the KernelCacheLimit option, see the svmsmoset function. The SMO algorithm stores only a submatrix of the kernel matrix, limited by the size specified by the KernelCacheLimit option. However, if the number of data points exceeds the size specified by the KernelCacheLimit option, the SMO algorithm slows down because it has to recalculate the kernel matrix elements.

When using svmtrain on large data sets, and you run out of memory or the optimization step is very time consuming, try either of the following:

- Use a smaller number of samples and use cross validation to test the performance of the classifier.

- Set 'Method' to 'SMO', and set the KernelCacheLimit option as large as your system permits. For information on setting the KernelCacheLimit option, see the svmsmoset function.

Tip If you set `'Method'` to `'SMO'`, setting the `'BoxConstraint'` property as small as possible will help the SMO algorithm run faster.

Examples

1 Load the sample data, which includes Fisher's iris data of 5 measurements on a sample of 150 irises.

```
load fisheriris
```

2 Create `data`, a two-column matrix containing sepal length and sepal width measurements for 150 irises.

```
data = [meas(:,1), meas(:,2)];
```

3 From the `species` vector, create a new column vector, `groups`, to classify data into two groups: Setosa and non-Setosa.

```
groups = ismember(species,'setosa');
```

4 Randomly select training and test sets.

```
[train, test] = crossvalind('holdOut',groups);
cp = classperf(groups);
```

5 Train an SVM classifier using a linear kernel function and plot the grouped data.

```
svmStruct = svmtrain(data(train,:),groups(train),'showplot',true);
```

6 Add a title to the plot, using the KernelFunction field from the svmStruct structure as the title.

```
title(sprintf('Kernel Function: %s',...
               func2str(svmStruct.KernelFunction)),...
               'interpreter','none');
```

7 Use the svmclassify function to classify the test set.

```
classes = svmclassify(svmStruct,data(test,:),'showplot',true);
```

8 Evaluate the performance of the classifier.

```
classperf(cp,classes,test);
cp.CorrectRate
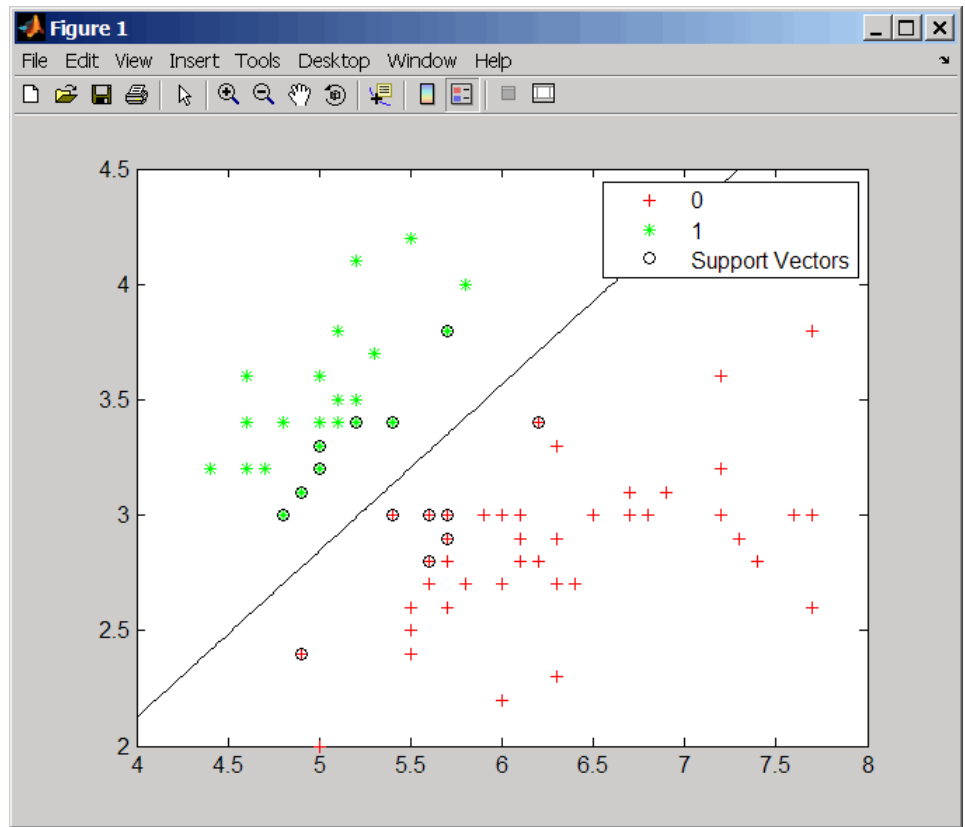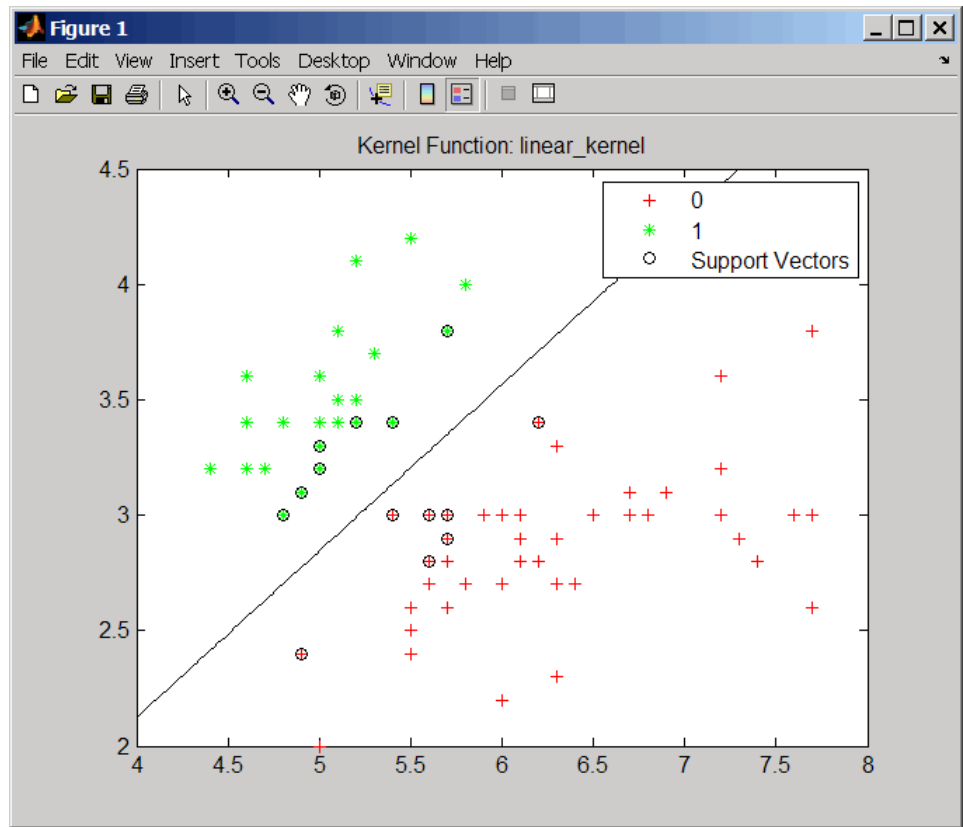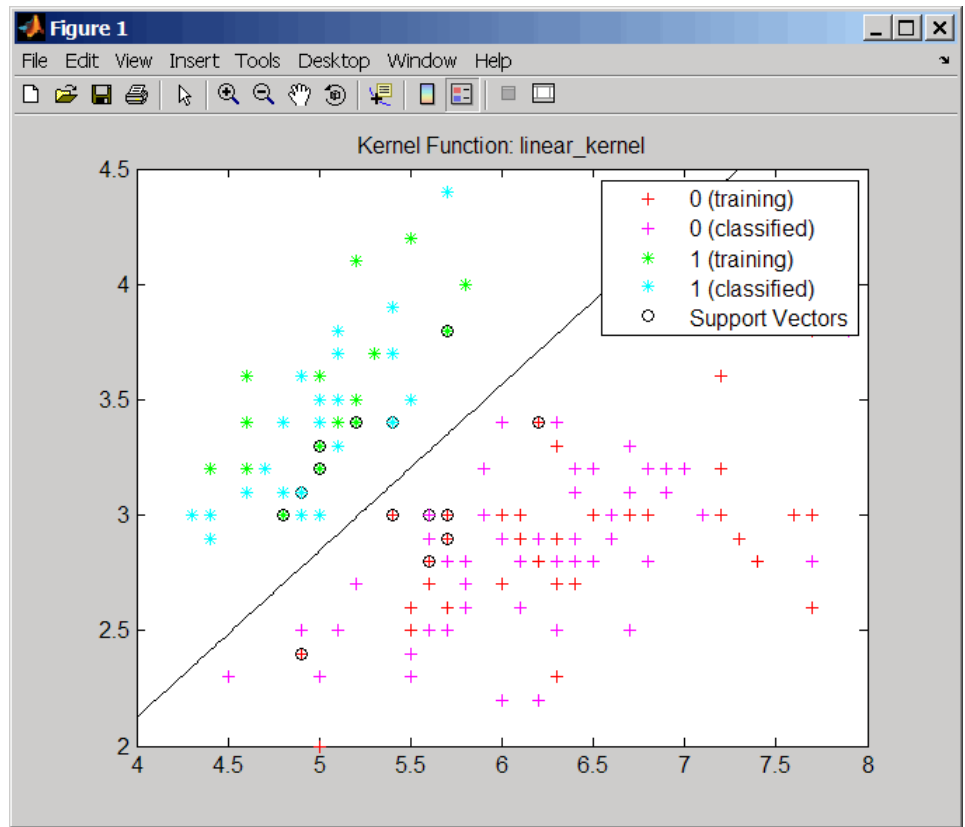
ans =

    0.9867
```

9 Use a one-norm, hard margin support vector machine classifier by changing the `boxconstraint` property.

```
figure
svmStruct = svmtrain(data(train,:),groups(train),...
                    'showplot',true,'boxconstraint',1e6);
```



```
classes = svmclassify(svmStruct,data(test,:),'showplot',true);
```

10 Evaluate the performance of the classifier.

```
classperf(cp,classes,test);
cp.CorrectRate

ans =
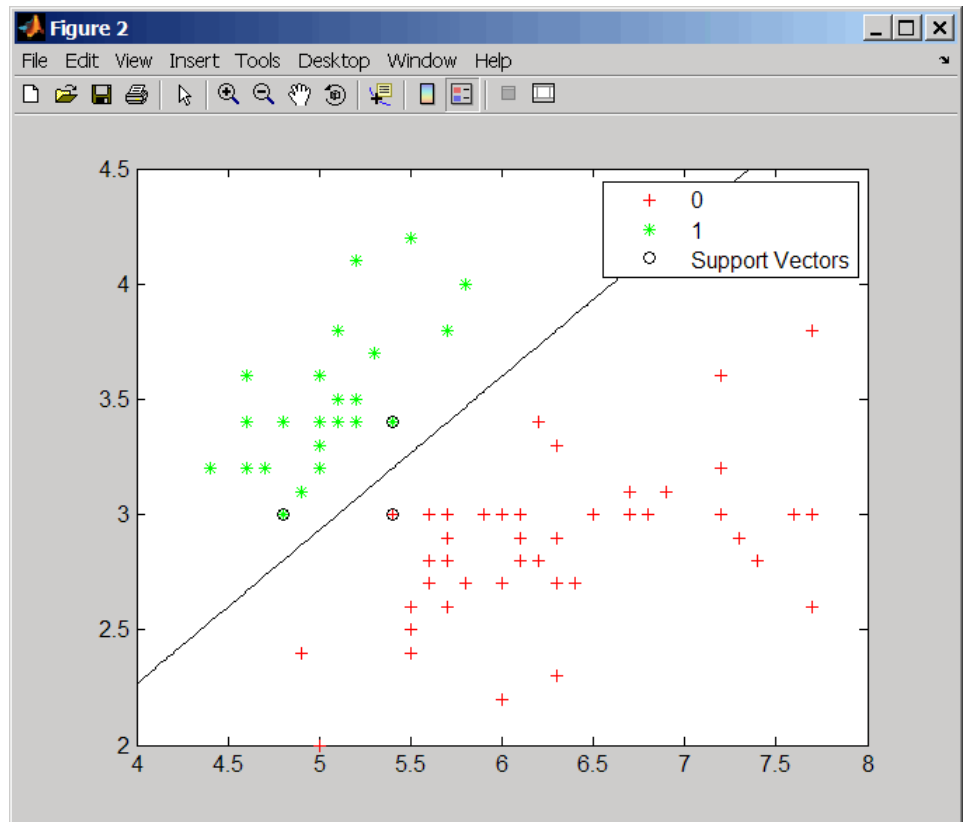
    0.9867
```

References

[1] Kecman, V. (2001). Learning and Soft Computing (Cambridge, MA: MIT Press).

[2] Suykens, J.A.K., Van Gestel, T., De Brabanter, J., De Moor, B., and Vandewalle, J. (2002). Least Squares Support Vector Machines (Singapore: World Scientific).

[3] Scholkopf, B., and Smola, A.J. (2002). Learning with Kernels (Cambridge, MA: MIT Press).

[4] Cristianini, N. and Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, First Edition (Cambridge: Cambridge University Press). http://www.support-vector.net/

See Also   Bioinformatics Toolbox functions: `knnclassify`, `svmclassify`, `svmsmoset`

Statistics Toolbox function: `classify`

Optimization Toolbox function: `quadprog`

MATLAB function: `optimset`

swalign

Purpose   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, ...)
```

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 on page 2-57 or Nucleotide Lookup on page 2-67.

AlphabetValue	String specifying the type of sequence. Choices are `'AA'` (default) or `'NT'`.

ScoringMatrixValue	String specifying the scoring matrix to use for the local alignment. Choices for amino acid sequences are:

- `'PAM40'`

- `'PAM250'`

- `'DAYHOFF'`

- `'GONNET'`

- `'BLOSUM30'` increasing by 5 up to `'BLOSUM90'`

- `'BLOSUM62'`

- `'BLOSUM100'`

Default is:

- `'BLOSUM50'` (when *AlphabetValue* equals `'AA'`)

- `'NUC44'` (when *AlphabetValue* equals `'NT'`)

Note All of the above scoring matrices have a built-in scale factor that returns *Score* in bits.

ScaleValue	Scale factor used to return *Score* in arbitrary units other than bits. Choices are any positive value. For example, if you enter `log(2)` for *ScaleValue*, then swalign returns *Score* in nats.
GapOpenValue	Penalty for opening a gap in the alignment. Choices are any positive integer. Default is `8`.

swalign

	ExtendGapValue	Penalty for extending a gap. Choices are any positive integer. Default is equal to *GapOpenValue*.
	ShowscoreValue	Controls the display of the scoring space and the winning path of the alignment. Choices are true or false (default).
Return Values	*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 the scale factor used to return *Score* in arbitrary units other than bits. Choices are any positive value.

`...` = swalign(*Seq1*,*Seq2*, `...`'GapOpen', *GapOpenValue*, `...`) specifies the penalty for opening a gap in the alignment. Choices are any positive integer. Default is 8.

`...` = swalign(*Seq1*,*Seq2*, `...`'ExtendGap', *ExtendGapValue*, `...`) specifies the penalty for extending a gap in the alignment. Choices are any positive integer. Default is equal to *GapOpenValue*.

`...` = swalign(*Seq1*,*Seq2*, `...`'Showscore', *ShowscoreValue*, `...`) controls the display of the scoring space and winning path of the alignment. Choices are true or false (default)

swalign

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 represents 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.

Tip 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. Return the optimal global alignment score in bits and the alignment character array.

```
[Score, Alignment] = swalign('VSPAGMASGYD','IPGKASYD')
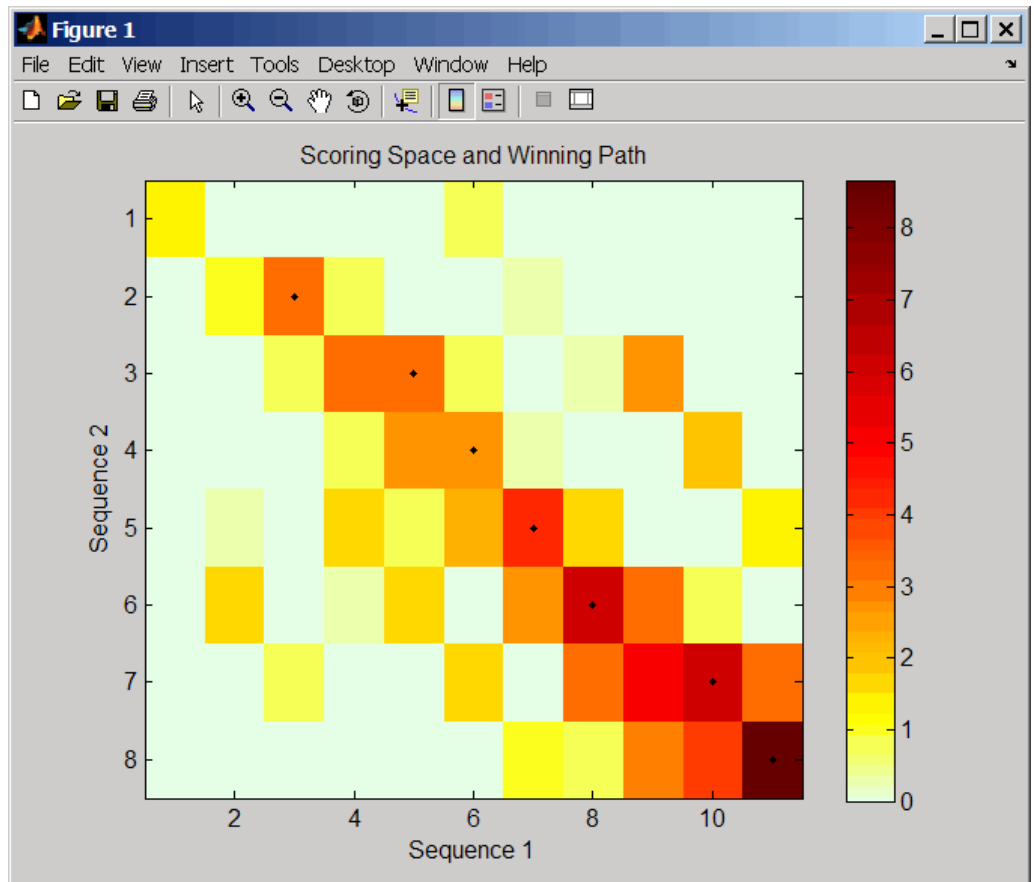
Score =

     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   Bioinformatics Toolbox functions: `blosum`, `nt2aa`, `nwalign`, `pam`, `seqdotplot`, `showalignment`

Purpose   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, T.

                traceplot(*A*, *C*, *G*, *T*) creates a trace plot from data in vectors A,
                C, G, T.

                h = traceplot(...) returns a structure with the handles of the lines
                corresponding to A, C, G, T.

Examples   tstruct = scfread('sample.scf');
                   traceplot(tstruct)

See Also   Bioinformatics Toolbox function: scfread

zonebackadj

Purpose	Perform background adjustment on Affymetrix microarray probe-level data using zone-based method

Syntax

BackAdjustedData = zonebackadj(*Data*)
[*BackAdjustedData*, *ZoneStruct*] = 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*, ...)

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. • An 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 directory. • 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).

zonebackadj

<table>
<tr><td>Return
Values</td><td>BackAdjustedData</td><td>Matrix or cell array of vectors containing background-adjusted probe intensity values.</td></tr>
<tr><td></td><td>ZoneStruct</td><td>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.</td></tr>
</table>

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 the 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.

... = 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 demo 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/products/software/specific/dtt.affx

The following example assumes that the `Ecoli-antisense-121502.CEL` file is stored on the MATLAB search path or in the current directory. 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

zonebackadj

2-882

zonebackadj

Methods — By Category

Phylogenetic Tree

Following are methods for use with a phytree object.

get (phytree)   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

Graph Visualization

Following are methods for use with 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
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 and minimum cut in biograph object
minspantree (biograph)	Find minimal spanning tree in 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

Gene Ontology

Following are methods for use with a geneont object.

getancestors (geneont)	Find terms that are ancestors of specified Gene Ontology (GO) term
getdescendants (geneont)	Find terms that are descendants of specified Gene Ontology (GO) term
getmatrix (geneont)	Convert geneont object into relationship matrix
getrelatives (geneont)	Find terms that are relatives of specified Gene Ontology (GO) term

Methods — Alphabetical List

allshortestpaths (biograph)

Purpose   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" in the Bioinformatics Toolbox documentation.

[*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 node S to node T. A 0 in this matrix indicates the source node; an Inf is an unreachable 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.

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).

allshortestpaths (biograph)

See Also Bioinformatics Toolbox functions: `biograph` (object constructor), `graphallshortestpaths`

Bioinformatics Toolbox object: `biograph object`

Bioinformatics Toolbox methods of a biograph object: `conncomp`, `isdag`, `isomorphism`, `isspantree`, `maxflow`, `minspantree`, `shortestpath`, `topoorder`, `traverse`

Purpose   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

> **Tip** For introductory information on graph theory functions, see "Graph Theory Functions" in the Bioinformatics Toolbox documentation.

conncomp (biograph)

[*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.

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   Bioinformatics Toolbox functions: `biograph` (object constructor), `graphconncomp`

Bioinformatics Toolbox object: `biograph object`

Bioinformatics Toolbox methods of a biograph object: `allshortestpaths`, `isdag`, `isomorphism`, `isspantree`, `maxflow`, `minspantree`, `shortestpath`, `topoorder`, `traverse`

dolayout (biograph)

Purpose   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

Note To use the dolayout method, you must have accepted a Graphviz software license (free). If you have not, you will be prompted to do so.

---

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 on page 5-4. For information on accessing and specifying the above properties of a biograph object, see Accessing Properties of a Biograph Object on page 5-10 and Specifying Properties of a Biograph Object on page 5-10.

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   **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)
Biograph object with 5 nodes and 9 edges.
bg.nodes(1).Position

ans =

    []
```

Nodes do not have a position yet.

2 Call the layout engine and render the graph.

```
dolayout(bg)
bg.nodes(1).Position

ans =

   112   224

view(bg)
```

3 Manually modify a node position and recalculate the paths only.

```
bg.nodes(1).Position = [150 150];
dolayout(bg, 'Pathsonly', true)
```

dolayout (biograph)

```
view(bg)
```

See Also   Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph object`

Bioinformatics Toolbox methods of a biograph object: `dolayout`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `view`

MATLAB functions: `get`, `set`

Purpose	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*).

get (phytree)

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')
```

2 Get the names of the leaves.

```
protein_names = get(tr,'LeafNames')

protein_names =

    'BAI2_HUMAN/917-1197'
    'BAI1_HUMAN/944-1191'
    'O00406/622-883'
     ...
```

See Also

Bioinformatics Toolbox

- functions — phytree (object constructor), phytreeread

- phytree object methods — getbyname, select

Purpose   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;
```

See Also   Bioinformatics Toolbox function: biograph (object constructor)

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object: dolayout, getancestors, getdescendants, getedgesbynodeid, getnodesbyid, getrelatives, view

MATLAB functions: get, set

getancestors (geneont)

Purpose	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, ...)
```

Arguments

GeneontObj	A geneont object, such as created by the geneont 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).

Return Values

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.

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 tallying counts in gene enrichment studies. For more information, see the Gene Ontology Enrichment in Microarray Data demo.

... = 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).

Examples

1 Download the current version of the Gene Ontology database from the Web into a geneont object in MATLAB.

```
GO = geneont('LIVE', true)
```

MATLAB 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 a report of the subordinate Gene Ontology terms.

```
[cm acc rels] = getmatrix(subontology);
rpt = [num2goid(acc) get(subontology.Terms, 'name')]';
disp(sprintf('%s --> %s\n',rpt{:}))

GO:0008150 --> biological_process
GO:0009636 --> response to toxin
GO:0017085 --> response to insecticide
GO:0042221 --> response to chemical stimulus
GO:0046680 --> response to DDT
GO:0050896 --> response to stimulus
```

5 View relationships of the subordinate Gene Ontology using the biograph function and methods.

```
BG = biograph(cm, get(subontology.Terms, 'name'));
view(BG)
```

See Also   Bioinformatics Toolbox functions: geneont (object constructor),
goannotread, num2goid

Bioinformatics Toolbox methods of geneont object: getdescendants,
getmatrix, getrelatives

Purpose	Branches and leaves from phytree object
Syntax	*S* = getbyname(*Tree*, *Expression*) *S* = getbyname(*Tree*, *String*, 'Exact', true)

Arguments

Tree	phytree object created by phytree function (object constructor).
Expression	Regular expression. When *Expression* is a cell array of strings, getbyname returns a matrix where every column corresponds to every query in *Expression*. For information about the symbols that you can use in a matching regular expression, see the MATLAB function regexp.
String	String or cell array of strings.

Description *S* = getbyname(*Tree*, *Expression*) returns a logical vector (*S*) of size NumNodes-by-1 with the node names of a phylogenetic tree (*Tree*) that match the regular expression (*Expression*) regardless of letter case.

S = getbyname(*Tree*, *String*, 'Exact', true) looks for exact string matches and ignores case. When *String* is a cell array of char strings, getbyname returns a vector with indices.

Examples **1** Load a phylogenetic tree created from a protein family.

```
tr = phytreeread('pf00002.tree');
```

2 Select all the 'mouse' and 'human' proteins.

```
sel = getbyname(tr,{'mouse','human'});
view(tr,any(sel,2));
```

getbyname (phytree)

See Also

Bioinformatics Toolbox

- function — `phytree` (object constructor)

- `phytree` object methods — `get`, `prune`, `select`

Purpose	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))
```

getcanonical (phytree)

```
ans =
       1
```

See Also   Bioinformatics Toolbox

- functions — `phytree` (object constructor), `phytreeread`

- `phytree` object methods — `getbyname`, `select`, `subtree`

Purpose   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;
```

See Also   Bioinformatics Toolbox function: biograph (object constructor)

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object: dolayout,
getancestors, getdescendants, getedgesbynodeid, getnodesbyid,
getrelatives, view

MATLAB functions: get, set

getdescendants (geneont)

Purpose	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, ...)
```

Arguments

GeneontObj	A geneont object, such as created by the geneont 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).

Return Values

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.

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 tallying counts in gene
enrichment studies. For more information, see the Gene Ontology
Enrichment in Microarray Data demo.

... = 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).

getdescendants (geneont)

Examples

1 Download the current version of the Gene Ontology database from the Web into a geneont object in MATLAB.

```
GO = geneont('LIVE', true)
```

MATLAB creates a geneont object and displays the number of terms in the database.

```
Gene Ontology object with 24316 Terms.
```

2 Retrieve the descendants of the mithochondrial respiratory chain GO term with an identifier of 5746.

```
descendants = getdescendants(GO,5746)

descendants =

        5746
        5747
        5749
        5750
        5751
       42652
       42653
```

3 Create a subordinate Gene Ontology.

```
subontology = GO(descendants)

Gene Ontology object with 7 Terms.
```

4 Create a report of the subordinate Gene Ontology terms.

```
[cm acc rels] = getmatrix(subontology);
rpt = [num2goid(acc) get(subontology.Terms, 'name')]';
disp(sprintf('%s --> %s\n',rpt{:}))

GO:0005746 --> mitochondrial respiratory chain
GO:0005747 --> mitochondrial respiratory chain complex I
```

```
GO:0005749 --> mitochondrial respiratory chain complex II
GO:0005750 --> mitochondrial respiratory chain complex III
GO:0005751 --> mitochondrial respiratory chain complex IV
GO:0042652 --> mitochondrial respiratory chain complex I, peripheral segment
GO:0042653 --> mitochondrial respiratory chain complex I, membrane segment
```

5 View relationships of the subordinate Gene Ontology using the
biograph function and methods.

```
BG = biograph(cm, num2goid(acc));
view(BG)
```

getdescendants (geneont)

See Also   Bioinformatics Toolbox functions: geneont (object constructor),
goannotread, num2goid

Bioinformatics Toolbox methods of geneont object: getancestors,
getmatrix, getrelatives

getedgesbynodeid (biograph)

Purpose   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.

Example   **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',[O 1 O]);
set(EdgesOut,'LineColor',[1 O O]);
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]);
```

getedgesbynodeid (biograph)

```
bg.view;
```

See Also   Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph object`

Bioinformatics Toolbox methods of a biograph object: `dolayout`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `view`

MATLAB functions: `get`, `set`

Purpose   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)
```

See Also   Bioinformatics Toolbox function: biograph (object constructor)

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object: dolayout,
getancestors, getdescendants, getedgesbynodeid, getnodesbyid,
getrelatives, view

getmatrix (geneont)

Purpose	Convert geneont object into relationship matrix
Syntax	[*Matrix, ID, Relationship*] = getmatrix(*GeneontObj*)
Arguments	
	GeneontObj geneont object created by geneont (object constructor)
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.
Examples	`GO = geneont('LIVE',true);` `[MATRIX, ID, REL] = getmatrix(GO);`
See Also	• Bioinformatics Toolbox functions — geneont (object constructor), goannotread, num2goid • Bioinformatics Toolbox object — geneont object • Bioinformatics Toolbox methods of geneont object — getancestors, getdescendants, getmatrix, getrelatives

getmatrix (phytree)

Purpose   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);
```

See Also   Bioinformatics Toolbox functions: phytree (object constructor),
phytreetool

Bioinformatics Toolbox object: phytree object

Bioinformatics Toolbox methods of phytree object: get, pdist, prune

getnewickstr (phytree)

Purpose   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.

References   Information about the Newick tree format.

http://evolution.genetics.washington.edu/phylip/newicktree.html

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)
```

See Also   Bioinformatics Toolbox

- functions — phytree (object constructor), phytreeread, phytreetool, phytreewrite, seqlinkage

- phytree object methods — get, getbyname, getcanonical

getnodesbyid (biograph)

Purpose   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.

Example   **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.

See Also   Bioinformatics Toolbox function: biograph (object constructor)

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object: dolayout, getancestors, getdescendants, getedgesbynodeid, getnodesbyid, getrelatives, view

MATLAB functions: get, set

getrelatives (biograph)

Purpose   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;
```

See Also   Bioinformatics Toolbox function: biograph (object constructor)

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object: dolayout, getancestors, getdescendants, getedgesbynodeid, getnodesbyid, getrelatives, view

MATLAB functions: get, set

Purpose   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, ...)
```

Arguments

GeneontObj	A geneont object, such as created by the geneont 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).

getrelatives (geneont)

Return Values	*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.

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 tallying counts in gene enrichment studies. For more information, see the Gene Ontology Enrichment in Microarray Data demo.

... = 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).

Examples
1 Download the current version of the Gene Ontology database from the Web into a geneont object in MATLAB.

```
GO = geneont('LIVE', true)
```

MATLAB creates a geneont object and displays the number of terms in the database.

```
Gene Ontology object with 24316 Terms.
```

2 Retrieve the immediate relatives for the mitochondrial membrane GO term with an identifier of 31966.

```
relatives = getrelatives(GO,31966,'levels',1)

relatives =

        5740
        5741
        5743
```

```
                31090
                31966
                44429
                44455
```

3 Create a subordinate Gene Ontology.

```
subontology = GO(relatives)

Gene Ontology object with 7 Terms.
```

4 Create a report of the subordinate Gene Ontology terms.

```
[cm acc rels] = getmatrix(subontology);
rpt = [num2goid(acc) get(subontology.Terms, 'name')]';
disp(sprintf('%s --> %s\n',rpt{:}))

GO:0005740 --> mitochondrial envelope
GO:0005741 --> mitochondrial outer membrane
GO:0005743 --> mitochondrial inner membrane
GO:0031090 --> organelle membrane
GO:0031966 --> mitochondrial membrane
GO:0044429 --> mitochondrial part
GO:0044455 --> mitochondrial membrane part
```

5 View relationships of the subordinate Gene Ontology using the biograph function and methods and color the mitochondrial membrane GO terms red.

```
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)
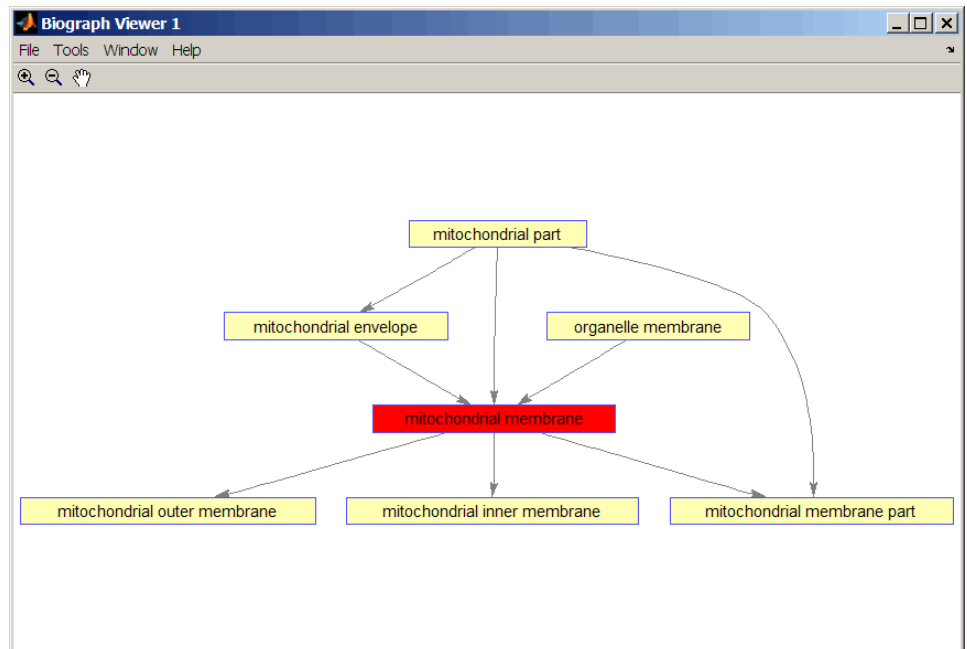
Gene Ontology object with 28 Terms.
```

8 View relationships using the biograph functions.

```
[cm acc rels] = getmatrix(subontology);
BG = biograph(cm, get(subontology.Terms, 'name'));
BG.nodes(acc==5741).Color = [1 0 0];
```

getrelatives (geneont)

```
view(BG)
```

See Also   Bioinformatics Toolbox functions: geneont (object constructor),
                  goannotread, num2goid

                  Bioinformatics Toolbox methods of geneont object: getancestors,
                  getdescendants, getmatrix

isdag (biograph)

Purpose   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" in the Bioinformatics Toolbox documentation.

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.

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   Bioinformatics Toolbox functions: biograph (object constructor),
graphisdag

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object:
allshortestpaths, conncomp, isomorphism, isspantree, maxflow,
minspantree, shortestpath, topoorder, traverse

isomorphism (biograph)

Purpose 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" in the Bioinformatics Toolbox documentation.

[*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.

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

Bioinformatics Toolbox functions: biograph (object constructor), graphisomorphism

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object: allshortestpaths, conncomp, isdag, isspantree, maxflow, minspantree, shortestpath, topoorder, traverse

isspantree (biograph)

Purpose 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" in the Bioinformatics Toolbox documentation.

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.

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 Bioinformatics Toolbox functions: biograph (object constructor), graphisspantree

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object: allshortestpaths, conncomp, isdag, isomorphism, maxflow, minspantree, shortestpath, topoorder, traverse

Purpose   Calculate maximum flow and minimum cut 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*.

maxflow (biograph)

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" in the Bioinformatics Toolbox documentation.

[*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.

[...] = 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.

maxflow (biograph)

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   Bioinformatics Toolbox functions: `biograph` (object constructor), `graphmaxflow`

Bioinformatics Toolbox object: `biograph object`

Bioinformatics Toolbox methods of a biograph object: `allshortestpaths, conncomp, isdag, isomorphism, isspantree, minspantree, shortestpath, topoorder, traverse`

Purpose	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" in the Bioinformatics Toolbox documentation.

[*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.

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   Bioinformatics Toolbox functions: biograph (object constructor), graphminspantree

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object: allshortestpaths, conncomp, isdag, isomorphism, isspantree, maxflow, shortestpath, topoorder, traverse

pdist (phytree)

Purpose Calculate pairwise patristic distances in phytree object

Syntax

```
D = pdist(Tree)
[D,C] = pdist(Tree)
pdist(..., 'PropertyName', PropertyValue,...)
pdist(..., 'Nodes', NodeValue)
pdist(... , Squareform', SquareformValue)
pdist(..., 'Criteria', CriteriaValue)
```

Arguments

Tree	Phylogenetic tree object created with the phytree constructor function.
NodeValue	Property to select the nodes. Enter either 'leaves' (default) or 'all'.
SquareformValue	Property to control creating a square matrix.

Description `D = pdist(Tree)` returns a vector (*D*) containing the patristic distances between every possible pair of leaf nodes a phylogenetic tree object (*Tree*). The patristic distances are computed by following paths through the branches of the tree and adding the patristic branch distances originally created with seqlinkage.

The output vector *D* is arranged in the order `((2,1),(3,1),...,(M,1),(3,2),...(M,3),.....(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,...)` defines optional properties using property name/value pairs.

`pdist(..., 'Nodes', NodeValue)` indicates the nodes included in the computation. When Node='leaves', the output is ordered as before, but *M* is the total number of nodes in the tree (NumLeaves+NumBranches).

pdist(... , Squareform', *SquareformValue*), when Squareform
is true, 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. C can be 'distance' (default) or 'levels'.

Examples **1** Get a phylogenetic tree from a file.

    tr = phytreeread('pf00002.tree')

2 Calculate the tree distances between pairs of leaves.

    dist = pdist(tr,'nodes','leaves','squareform',true)

See Also Bioinformatics Toolbox

- functions — phytree (object constructor), phytreeread,
  phytreetool, seqlinkage, seqpdist

plot (phytree)

Purpose   Draw phylogenetic tree

Syntax
```
plot(Tree)
plot(Tree, ActiveBranches)
plot(..., 'Type', TypeValue)
plot(...,'Orientation', OrientationValue)
plot(...,'BranchLabels', BranchLabelsValue)
plot(...,'LeafLabels', LeafLabelsValue)
plot(...,'TerminalLabels', TerminalLabelsValue)
```

Arguments

Tree	Phylogenetic tree object created with the `phytree` constructor function.
ActiveBranches	Branches veiwable in the Figure window.
TypeValue	Property to select a method for drawing a phylogenetic tree. Enter `'square'`, `'angular'`, or `'radial'`. The default value is `'square'`.
OrientationValue	Property to orient a phylogram or cladogram tree. Enter `'top'`, `'bottom'`, `'left'`, or `'right'`. The default value is `'left'`.
BranchLabelsValue	Property to control displaying branch labels. Enter either `true` or `false`. The default value is `false`.
LeafLabelsValue	Property to control displaying leaf labels. Enter either `true` or `false`. The default value is `false`.
TerminalLabels	Property to control displaying terminal labels. Enter either `true` or `false`. The default value is `false`.

Description   `plot(Tree)` draws a phylogenetic tree object into a MATLAB figure as a phylogram. The significant distances between branches and nodes

are in the horizontal direction. Vertical distances have no significance and are selected only for display purposes. Handles to graph elements are stored in the figure field UserData so that you can easily modify graphic properties.

plot(Tree, ActiveBranches) hides the nonactive branches and all of their descendants. ActiveBranches is a logical array of size numBranches x 1 indicating the active branches.

plot(..., 'Type', *TypeValue*) selects a method for drawing a phylogenetic tree.

plot(...,'Orientation', *OrientationValue*) orients a phylogenetic tree within a Figure window. The Orientation property is valid only for phylogram and cladogram trees.

plot(...,'BranchLabels', *BranchLabelsValue*) hides or displays branch labels placed next to the branch node.

plot(...,'LeafLabels', *LeafLabelsValue*) hides or displays leaf labels placed next to the leaf nodes.

plot(...,'TerminalLabels', TerminalLabelsValue) hides or displays terminal labels. Terminal labels are placed over the axis tick labels and ignored when Type= 'radial'.

H = plot(...) returns a structure with handles to the graph elements.

Examples
```
tr = phytreeread('pf00002.tree')
plot(tr,'Type','radial')
```

Graph element properties can be modified as follows:

```
h=get(gcf,'UserData')
set(h.branchNodeLabels,'FontSize',6,'Color',[.5 .5 .5])
```

See Also   Bioinformatics Toolbox

• functions — phytree (object constructor), phytreeread, phytreetool, seqlinkage

plot (phytree)

- phytree object method — view

prune (phytree)

Purpose	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 node 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 property (Mode) 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)
            % To :
```

prune (phytree)

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)
```

See Also   Bioinformatics Toolbox

- functions — phytree (object constructor), phytreetool

- phytree object methods — select, get

reorder (phytree)

Purpose   Reorder leaves of phylogenetic tree

Syntax   *Tree1Reordered* = reorder(*Tree1*, *Order*)
                 [*Tree1Reordered*, *OptimalOrder*] = reorder(*Tree1*, *Order*,
                     'Approximate', *ApproximateValue*)
                 [*Tree1Reordered*, *OptimalOrder*] = reorder(*Tree1*, *Tree2*)

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. |

Return Values | | |
|---|---|
| *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

4-67

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)
```

See Also   Bioinformatics Toolbox function: phytree (object constructor)

Bioinformatics Toolbox object: phytree object

Bioinformatics Toolbox methods of a phytree object: get, getbyname, prune

Purpose	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)
```

MATLAB draws a figure with the phylogenetic tree.

2 Place the root at `'Branch 7'`.

```
sel = getbyname(tr_1,'Branch 7');
tr_2 =  reroot(tr_1,sel)
plot(tr_2,'branchlabels',true)
```

MATLAB draws a tree 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)
```

MATLAB draws the new tree with the root moved from the center of branch 7 to branch 8.

See Also   Bioinformatics Toolbox

- functions — phytree (object constructor), seqneighjoin

- phytree object methods — get, getbyname, prune, select

Purpose   Select tree branches and leaves in phytree object

Syntax
```
S = select(Tree, N)
[S, Selleaves, Selbranches] = select(...)
select(..., 'PropertyName', PropertyValue,...)
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*,...) defines optional properties using property name/value pairs.

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. vips_human):
ind = getbyname(tr,'vips_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)
```

See Also   Bioinformatics Toolbox

- functions — phytree (object constructor), phytreetool

- phytree object methods — get, pdist, prune

shortestpath (biograph)

Purpose   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.

shortestpath (biograph)

Description

> **Tip** For introductory information on graph theory functions, see "Graph Theory Functions" in the Bioinformatics Toolbox documentation.

[*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.

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

Bioinformatics Toolbox functions: biograph (object constructor), graphshortestpath

shortestpath (biograph)

Bioinformatics Toolbox object: `biograph object`

Bioinformatics Toolbox methods of a biograph object:
`allshortestpaths, conncomp, isdag, isomorphism, isspantree, maxflow, minspantree, topoorder, traverse`

subtree (phytree)

Purpose   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 VIPS and CGRR human proteins.

```
sel = getbyname(tr,{'vips_human','cgrr_human'});
sel = any(sel,2);
tr =  subtree(tr,sel)
view(tr);
```

See Also   Bioinformatics Toolbox

- functions — phytree (object constructor)

- phytree object methods — get, getbyname, prune, select

topoorder (biograph)

Purpose   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" in the Bioinformatics Toolbox documentation.

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.

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   Bioinformatics Toolbox functions: biograph (object constructor), graphtopoorder

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object: allshortestpaths, conncomp, isdag, isomorphism, isspantree, maxflow, minspantree, shortestpath, traverse

Purpose   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" in the Bioinformatics Toolbox documentation.

traverse (biograph)

[*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.

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   Bioinformatics Toolbox functions: `biograph` (object constructor), `graphtraverse`

Bioinformatics Toolbox object: `biograph object`

Bioinformatics Toolbox methods of a biograph object: `allshortestpaths`, `conncomp`, `isdag`, `isomorphism`, `isspantree`, `maxflow`, `minspantree`, `shortestpath`, `topoorder`

view (biograph)

Purpose   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])
```

See Also   Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph object`

Bioinformatics Toolbox methods of a biograph object: `dolayout`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `view`

MATLAB functions: `get`, `set`

view (phytree)

Purpose	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 Tool 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 Tool menu bar or by using the left and right mouse buttons.

view(*Tree*, *IntNodes*) opens the Phylogenetic Tree Tool 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.

Example
```
tree = phytreeread('pf00002.tree')
view(tree)
```

See Also   Bioinformatics Toolbox functions: phytree (object constructor), phytreeread, phytreetool, seqlinkage, seqneighjoin

Bioinformatics Toolbox object: phytree object

Bioinformatics Toolbox method of phytree object: plot

Purpose   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)
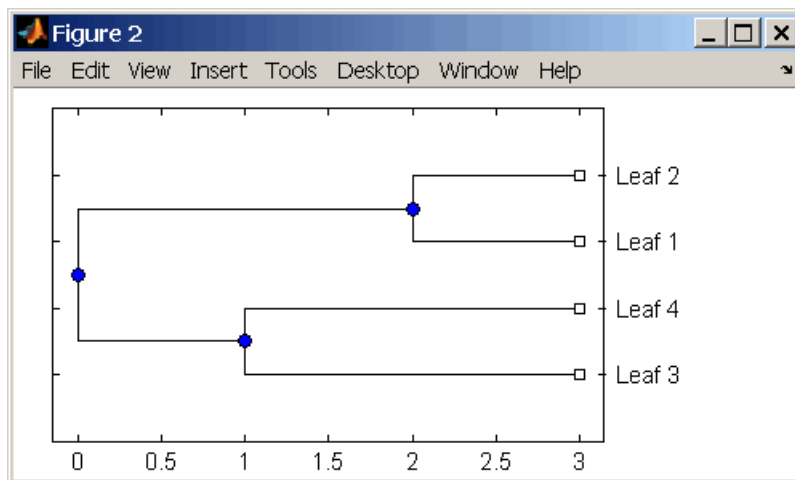
ans =

    1.0000
    1.0000
    0.8000
    0.8000
```

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   Bioinformatics Toolbox

- functions — `multialign`, `phytree` (object constructor), `profalign`, `seqlinkage`

weights (phytree)

Objects — Alphabetical List

biograph object

Purpose 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
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 and minimum cut in biograph object
minspantree (biograph)	Find minimal spanning tree in 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 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 ''. (This information is for bookkeeping purposes only.)
Label	String to label the biograph object. Default is ''. (This information is for bookkeeping purposes only.)
Description	String that describes the biograph object. Default is ''. (This information is for bookkeeping purposes only.)
LayoutType	String that specifies the algorithm for the layout engine. Choices are: • `'hierarchical'` (default) • `'equilibrium'` • `'radial'`
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.

Properties of a Biograph Object (Continued)

Property	Description
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' (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'.

Properties of a Biograph Object (Continued)

Property	Description
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 customized function to draw nodes. Default is [].

Properties of a Biograph Object (Continued)

Property	Description
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 on page 5-7.
Edges	Read-only column vector with handles to edge objects of a biograph object. The size of vector is the number of edges. For properties of edge objects, see Properties of an Edge Object on page 5-9.

Properties of a Node Object

Property	Description
ID	Read-only string defined when the biograph object is created, either by the *NodeIDs* input argument or internally by the biograph constructor function. Each node object's ID is unique and used internally to identify the node.
Label	String for labeling a node when you display a biograph object using the view method. Default is the ID property of the node object.
Description	String that describes the node. Default is ''. (This information is for bookkeeping purposes only.)
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.

Properties of a Node Object (Continued)

Property	Description
Shape	String that specifies the shape of the nodes. Choices are: • `'box'`(default) • `'ellipse'` • `'circle'` • `'rectangle'` • `'diamond'` • `'trapezium'` • `'invtrapezium'` • `'house'` • `'inverse'` • `'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`.

Properties of a Node Object (Continued)

Property	Description
TextColor	Three-element numeric vector of RGB values that specifies the color of the node labels. Default is [0, 0, 0], which defines black.
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	Read-only string defined when the biograph object is created, internally by the biograph constructor function. Each edge object's ID is unique and used internally to identify the edge.
Label	String for labeling an edge when you display a biograph object using the view method. Default is the ID property of the edge object.
Description	String that describes the edge. Default is ''. (This information is for bookkeeping purposes only.)
Weight	Value that represents the weight (cost, distance, length, or capacity) associated with the edge. Default is 1.
LineWidth	Positive number. Default is 1.

biograph object

Properties of an Edge Object (Continued)

Property	Description
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

Accessing Properties of a Biograph Object

You can access properties of a biograph object, *BGobj*, by using either of the following syntaxes:

PropertyValue = get(*BGobj*, '*PropertyName*')

PropertyValue = *BGobj.PropertyName*

Accessing Allowed Values of Biograph Object Properties

You can access allowed values for any property that has a finite set of choices by using the following syntax:

set(*BGobj*, '*PropertyName*')

Specifying Properties of a Biograph Object

You can specify properties of a biograph object, *BGobj*, by using any of the following syntaxes:

set(*BGobj*, '*PropertyName*', *PropertyValue*)

BGobj.PropertyName = *PropertyValue*

See Also

Bioinformatics Toolbox function: biograph (object constructor)

Bioinformatics Toolbox methods of a biograph object: allshortestpaths, conncomp, dolayout, getancestors,

getdescendants, getedgesbynodeid, getmatrix, getnodesbyid,
getrelatives, isdag, isomorphism, isspantree, maxflow,
minspantree, shortestpath, topoorder, traverse, view

MATLAB functions: get, set

geneont object

Purpose   Data structure containing Gene Ontology (GO) information

Description   A geneont object is a data structure containing Gene Ontology information. Gene Ontology terms can be explored and traversed through "is_a" and "part_of" relationships.

Method Summary   Following are methods of a geneont object:

getancestors (geneont)	Find terms that are ancestors of specified Gene Ontology (GO) term
getdescendants (geneont)	Find terms that are descendants of specified Gene Ontology (GO) term
getmatrix (geneont)	Convert geneont object into relationship matrix
getrelatives (geneont)	Find terms that are relatives of specified Gene Ontology (GO) term

Property Summary

Properties of a geneont Object

Property	Description
default_namespace	Read-only string containing the namespace to which terms are assigned.
format_version	Read-only string containing the version of the encoding of the OBO flat format file.

Properties of a geneont Object (Continued)

Property	Description
date	Read-only string containing the date the OBO file was last updated.
Terms	Read-only column vector with handles to term objects of a geneont object. For properties of term objects, see Properties of Terms Objects on page 5-13.

Properties of Terms Objects

Property	Description
id	Numeric value that corresponds to the GO ID of the GO term. **Tip** You can use the num2goid function to convert id to a GO ID string.
name	String representing the name of the GO term.
ontology	String limited to 'molecular function', 'biological process', or 'cellular component'.
definition	String that defines the GO term.
synonym	Numeric array containing GO IDs of GO terms that are synonyms of this GO term.
is_a	Numeric array containing GO IDs of GO terms that have an "is_a" relationship with this GO term.

geneont object

Properties of Terms Objects (Continued)

Property	Description
part_of	Numeric array containing GO IDs that of GO terms that have a "part_of" relationship with this GO term.
obsolete	Boolean value that indicates if the GO term is obsolete (1) or not obsolete (0).

See Also Bioinformatics Toolbox functions: geneont (object constructor), goannotread, num2goid

Bioinformatics Toolbox methods of geneont object: getancestors, getdescendants, getmatrix, getrelatives

Purpose   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:

get (phytree)	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

phytree object

| 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

Bioinformatics Toolbox functions: phytree (object constructor), phytreeread, phytreetool, phytreewrite, seqlinkage, seqneighjoin, seqpdist

Bioinformatics Toolbox methods of phytree object: get, getbyname, getcanonical, getmatrix, getnewickstr, pdist, plot, prune, reroot, select, subtree, view, weights

Index