Bioinformatics Toolbox

For Use with MATLAB®

- Computation
- Visualization
- Programming

Reference Guide

Version 2.1

The MathWorks

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`support@mathworks.com`	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	Mail

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

Bioinformatics Toolbox Reference Guide
© COPYRIGHT 2003 - 2005 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Revision History:

May 2005	Online only	New for Version 2.1 (Release 14SP2+)

Contents

1 Functions – Categorical List

Functions – Categorical List

This chapter is a reference for the functions in the Bioinformatics Toolbox. Functions are grouped into the following categories.

"Microarray Normalization and Filtering" (p. 1-18)	List of microarray normalization and filtering functions.
"Microarray Utility Functions" (p. 1-19)	List of microarray utility functions
"Mass Spectrometry Preprocessing and Visualization" (p. 1-20)	Preprocess raw data mass spectrometry data from instruments, and analyze spectra to identify patterns and compounds.
"Scoring Matrices" (p. 1-21)	List of scoring matrices.
"Phylogenetic Tree Tools" (p. 1-22)	List of functions for phylogenetic tree analysis.
"Phylogenetic Tree Methods" (p. 1-23)	List of methods for the phytree object.
"Graph Visualization Methods" (p. 1-24)	List of methods for the biograph object.
"Tutorials, Demos, and Examples" (p. 1-25)	Demonstrations for using the Bioinformatic Toolbox functions with real applications.

Data Formats and Databases

Use these functions to get data from Web data bases into MATLAB., use these functions to read and write to files within MATLAB using specific data formats.

blastread	Read data from NCBI BLAST report file
emblread	Read data from EMBL file
fastaread	Read data from FASTA file
fastawrite	Write to file with FASTA format
galread	Read microarray data from a GenePix array list file
genbankread	Read data from a GenBank file
genpeptread	Read data from a GenPept file
geosoftread	Read data from a Gene Expression Omnibus (GEO) SOFT file
getblast	Get 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	Get Gene Expression Omnibus (GEO) data
gethmmalignment	Retrieve multiple aligned sequences from the PFAM database
gethmmprof	Retrieve profile hidden Markov models from the PFAM database
gethmmtree	Get phylogenetic tree data from PFAM database

getpdb	Retrieve protein structure data from PDB database
getpir	Retrieve sequence data from PIR-PSD database
gprread	Read microarray data from a GenePix Results (GPR) file
imageneread	Read microarray data from an ImaGene Results file
jcampread	Read JCAMP-DX formatted files
multialignread	Read multiple sequence alignment file
pdbread	Read data from Protein Data Bank (PDB) file
pfamhmmread	Read data from a PFAM-HMM file
phytreeread	Read phylogenetic tree files
pirread	Read data from PIR file
scfread	Read trace data from SCF file
sptread	Read data from a SPOT file

Sequence Conversion

Convert nucleotide and amino acid sequences

aa2int	Convert an amino acid sequence from a letter to an integer representation
aa2nt	Convert amino acid sequence to nucleotide sequence
aminolookup	Display amino acid codes, integers, abbreviations, names, and codons
baselookup	Display nucleotide codes, integers, names, and abbreviations
dna2rna	Convert DNA sequence to RNA sequence
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
seq2regexp	Convert sequence with ambiguous characters to regular expression
seqcomplement	Calculate complementary strand of nucleotide sequence

seqrcomplement	Calculate reverse complement of a nucleotide sequence
seqreverse	Reverse the letters or numbers in a nucleotide sequence

Sequence Statistics

List of sequence statistics functions

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

Sequence Utilities

List of sequence utilities functions.

aminolookup	Display amino acid codes, integers, abbreviations, names, and codons
baselookup	Display nucleotide codes, integers, names, and abbreviations
blastncbi	Generate a remote BLAST request
cleave	Cleave amino acid sequence with enzyme
geneticcode	Return nucleotide codon to amino acid mapping
joinseq	Join two sequences to produce the shortest supersequence
palindromes	Find palindromes in a sequence
pdbdistplot	Visualize intermolecular distances in PDB file
pdbplot	Plot 3D protein structure
proteinplot	Display characteristics for amino acid sequences
ramachandran	Draw Ramachandran plot for PDB data
randseq	Generate random sequence from finite alphabet
rebasecuts	Find restriction enzymes that cut a protein sequence
restrict	Split nucleotide sequence at specified restriction site
revgeneticcode	Get the reverse mapping for a genetic code
seqconsensus	Calculate a consensus sequence

seqdisp	Format long sequence output for easy viewing
seqlogo	Display sequence logo for nucleotide and amino acid sequences
seqmatch	Find matches for every string in a library
seqprofile	Calculate a sequence profile from a set of multiply aligned sequences
seqshoworfs	Display open reading frames in a sequence
seqtool	Open interactive tool to explore biological sequences

Pairwise Sequence Alignment

List of pairwise sequence alignment functions

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

Multiple Sequence Alignment

List of multiple sequence alignment functions.

fastaread	Read data from FASTA file
multialign	Align multiple sequences using progressive method.
multialignread	Read multiple sequence alignment file
profalign	Align two profiles using Needleman-Wunsch global alignment
showalignment	Display a sequence alignment with color

Statistical Learning

List of statistical functions

classperf	Evaluated the performance of a classifier
crossvalind	Generate cross-validation indices
knnclassify	Classify data using the nearest–neighbor method
knnimpute	Impute missing data using the nearest-neighbor method
randfeatures	Generate a randomized subset of features
rankfeatures	Rank key features by class separability criteria
svmclassify	Classify data using a support vector machine
svmtrain	Train a support vector machine classifier

Protein Analysis

List of protein analysis functions.

aacount	Count the amino acids in a sequence
aminolookup	Display amino acid codes, integers, abbreviations, names, and codons
atomiccomp	Calculate atomic composition of a protein
cleave	Cleave amino acid sequence with enzyme
isoelectric	Estimate isoelectric point for amino acid sequence
molweight	Calculate molecular weight of amino acid sequence
pdbdistplot	Visualize intermolecular distances in PDB file
pdbplot	Plot 3D protein structure
proteinplot	Display characteristics for amino acid sequences
ramachandran	Draw Ramachandran plot for PDB data
rebasecuts	Find restriction enzymes that cut a protein sequence

Trace Tools

List of functions for analysis of nucleotide traces.

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

Profile Hidden Markov Models

List of Hidden Markov Model functions

gethmmalignment	Retrieve multiple aligned sequences from the PFAM database
gethmmprof	Retrieve profile hidden Markov models from the PFAM database
gethmmtree	Get phylogenetic tree data from PFAM database
hmmprofalign	Align a query sequence to a profile using hidden Markov model based alignment
hmmprofestimate	Estimate profile HMM parameters using pseudocounts
hmmprofgenerate	Generate a random sequence drawn from the profile HMM
hmmprofmerge	Concatenate the prealigned strings of several sequences to a profile HMM
hmmprofstruct	Create a profile HMM structure
pfamhmmread	Read data from a PFAM-HMM file
showhmmprof	Plot an Hidden Markov Model (HMM) profile

Microarray File Formats

List of microarray file format functions

affyread	Read microarray data from Affymetrix GeneChip file
galread	Read microarray data from a GenePix array list file
geosoftread	Read data from a Gene Expression Omnibus (GEO) SOFT file
getgeodata	Get Gene Expression Omnibus (GEO) data
gprread	Read microarray data from a GenePix Results (GPR) file
imageneread	Read microarray data from an ImaGene Results file
sptread	Read data from a SPOT file

Microarray Visualization

List of microarray visualization functions

clustergram	Create dendrogram and heat map
clustergram	Create dendrogram and heat map
maboxplot	Display a box plot for microarray data
maimage	Display a spatial image for microarray data
mairplot	Display intensity versus ratio scatter plot for microarray signals
maloglog	Create a loglog plot of microarray data
mapcaplot	Create a Principal Component plot of expression profile data
redgreencmap	Display a red and green colormap

Microarray Normalization and Filtering

List of microarray normalization and filtering functions.

exprprofrange	Calculate range of gene expression profiles
exprprofvar	Calculate variance of gene expression profiles
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
malowess	Smooth microarray data using the Lowess method
manorm	Normalize microarray data
quantilenorm	performs quantile normalization over multiple arrays

Microarray Utility Functions

List of microarray utility functions.

probelibraryinfo	Extract probe set library information for probe results
probesetlink	Link to NetAffx Web site
probesetlookup	Look up gene name for probe set
probesetplot	Plots values for Affymetrix CHP file probe set
probesetvalues	Extract probe set values from probe results

Mass Spectrometry Preprocessing and Visualization

List of mass spectrometry functions to preprocess raw data from instruments and analysis functions to identify compounds.

jcampread	Read JCAMP-DX formatted files
msalign	Align peaks in mass spectrum to reference peaks
msbackadj	Correct the baseline of a mass spectrum
msheatmap	Display color image for set of spectra
mslowess	Smooth mass spectrum using nonparametric method
msnorm	Normalize set of mass spectra
msresample	Resample a mass spectrometry signal
mssgolay	Smooth mass spectrum with least-squares polynomial
msviewer	Explore MS spectrum or set of spectra with GUI

Scoring Matrices

List of scoring matrices.

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

Phylogenetic Tree Tools

List of functions for phylogenetic tree analysis.

dnds	Estimate synonymous and nonsynonymous substitution rates
dndsml	Estimate synonymous-nonsynonymous substitution rates by the maximum likelihood method
gethmmtree	Get phylogenetic tree data from PFAM database
phytreeread	Read phylogenetic tree files
phytreetool	View, edit, and explore phylogenetic tree data
phytreewrite	Write phylogenetic tree object to Newick formatted file
seqlinkage	Construct phylogenetic tree from pairwise distances
seqneighjoin	Neighbor-joining method for phylogenetic tree reconstruction
seqpdist	Calculate pairwise distance between sequences

Phylogenetic Tree Methods

List of methods for the phytree object.

get (phytree)	Get information about a phylogenetic tree object
getbyname (phytree)	Select branches and leaves from a phytree object
getcanonical (phytree)	Calculate the canonical form of a phylogenetic tree
getnewickstr (phytree)	Create Newick formatted string
pdist (phytree)	Calculate pairwise patristic distances in a phytree object
phytree (phytree)	Object constructor for a phylogenetic tree object
plot (phytree)	Draw a phylogenetic tree
prune (phytree)	Remove branch nodes from phylogenetic tree
reroot (phytree)	Change the root of a phylogenetic tree
select (phytree)	Select tree branches and leaves in phytree object
subtree (phytree)	Extract a subtree
view (phytree)	View phylogenetic tree
weights (phytree)	Calculate weights for a phylogenetic tree

Graph Visualization Methods

List of methods for the biograph object.

biograph (biograph)	Create biograph object
dolayout (biograph)	Calculate node positions and edge trajectories
getancestors (biograph)	Find ancestors in a biograph object
getdescendants (biograph)	Find descendants in a biograph object
getedgesbynodeid (biograph)	Get handles to edges in graph
getnodesbyid (biograph)	Get handles to nodes
getrelatives (biograph)	Find relatives in a biograph object
view (biograph)	Draw figure from biograph object

Tutorials, Demos, and Examples

Sequence analysis

- seqstatsdemo — Sequence statistics tutorial example
- aligndemo — Basic sequence alignment tutorial
- alignsigdemo — How to estimate the significance of sequence alignments
- alignscoringdemo — Tutorial showing the use of scoring matrices

Hidden Markov Model profiles

- hmmprofdemo — HMM profile alignment tutorial example

Microarray analysis

- mousedemo — Microarray normalization and visualization example
- yeastdemo — Microarray data analysis example
- biclusterdemo — Clustergram functionality examples

Phylogenetic Analysis

- primatesdemo — Building a phylogenetic tree for the hominidae species
- hivdemo — Analyzing the origin of the HIV with phylogenetic trees

External software interface

- bioperldemo — Calling Bioperl functions from within MATLAB
- biojavademo — Calling BioJava functions from within MATLAB

External web database interface

- biowebservicedemo — How to use a Simple Object Access Protocol (SOAP) based web service from within MATLAB

2

Functions — Alphabetical List

aa2int

Purpose

Convert an amino acid sequence from a letter to an integer representation

Syntax

```
SeqInt = aa2int(SeqChar)
```

Arguments

SeqChar	Amino acid sequence represented with letters. Enter a character string with characters from the table Mapping Amino Acid Letters to Integers (unknown characters are mapped to `0`). Integers are arbitrarily assigned to IUB/IUPAC letters. You can also enter a structure with a field `Sequence`.
SeqInt	Amino acid sequence represented with numbers.

Mapping Amino Acid Letters to Integers

Amino Acid	**Code**	**Amino Acid**	**Code**
Alanine	`A1`	Phenylalanine	`F14`
Arginine	`R2`	Proline	`P15`
Asparagine	`N3`	Serine	`S—16`
Aspartic acid (Aspartate)	`D4`	Threonine	`T—17`
Cysteine	`C5`	Tryptophan	`W18`
Glutamine	`Q6`	Tyrosine	`Y19`
Glutamic acid (Glutamate)	`E7`	Valine	`V20`
Glycine	`G8`	Aspartic acid or Asparagine	`B21`
Histidine	`H9`	Glutamic acid or glutamine	`Z22`
Isoleucine	`I10`	Unknown or any amino acid	`X23`

Amino Acid	Code	Amino Acid	Code
Leucine	L11	Translation stop	*24
Lysine	K12	Gap of indeterminate length	- 25
Methionine	M13	Any character or symbol not in table	?0

Description

SeqInt = aa2int(*SeqChar*) converts a character string of amino acids (*SeqChar*) to a 1-by-N array of integers (*SeqInt*) using the table Mapping Amino Acid Letter to Integers.

Examples

Convert an amino acid sequence of letters to a vector of integers.

```
SeqInt = aa2int('MATLAB')

SeqInt =
  13   1   17   11   1   21
```

Convert a random amino acid sequence of letters to integers.

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

SeqChar =
   dwcztecakfuecvifchds

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

aa2nt

Purpose

Convert amino acid sequence to nucleotide sequence

Syntax

```
SeqNT = aa2nt(SeqAA)
aa2nt(..., 'PropertyName', PropertyValue,...)
aa2nt(..., 'GeneticCode', GeneticCodeValue)
aa2nt(..., '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	Property to select a genetic code. Enter a code number or code name from the table Genetic Code below. If you use a code name, you can truncate the name to the first two characters of the name.
AlphabetValue	Property to select a nucleotide alphabet. Enter either `'DNA'` or `'RNA'`. The default value is `'DNA'`, which uses the symbols `A`, `C`, `T`, `G`. The value `'RNA'` uses the symbols `A`, `C`, `U`, `G`.

Genetic Code

Code Number	Code Name	Code Number	Code Name
1	Standard	12	Alternative Yeast Nuclear
2	Vertebrate Mitochondrial	13	Ascidian Mitochondrial
3	Yeast Mitochondrial	14	Flatworm Mitochondrial

Code Number	Code Name	Code Number	Code Name
4	Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma /Spiroplasma	15	Blepharisma Nuclear
5	Invertebrate Mitochondrial	16	Chlorophycean Mitochondrial
6	Ciliate, Dasycladacean, and Hexamita Nuclear	21	Trematode Mitochondrial
9	Echinoderm Mitochondrial	22	Scenedesmus Obliquus Mitochondrial
10	Euplotid Nuclear	23	Thraustochytrium Mitochondrial
11	Bacterial and Plant Plastid		

Description

`SeqNT = aa2nt(SeqAA)` converts an amino acid sequence (*SeqAA*) to a nucleotide sequence (*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 then one possible nucleotide codon, this function selects randomly 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.

`aa2nt(..., 'PropertyName', PropertyValue,...)` defines optional properties using property name/value pairs.

`aa2nt(..., 'GeneticCode', GeneticCodeValue)` selects a genetic code (*GeneticCodeValue*) to use when converting an amino acid sequence (*SeqAA*) to a nucleotide sequence (*SeqNT*).

`aa2nt(..., 'Alphabet' AlphabetValue)` selects a nucleotide alphabet (*AlphabetValue*).

Standard Genetic Code

Amino Acid		**Amino Acid**	
Alanine (A)	`GCT, GCC, GCA, GCG`	Phenylalanine (F)	`TTT, TTC`
Arginine (R)	`CGT, CGC, CGA, CGG, AGA, AGG`	Proline (P)	`CCT, CCC, CCA, CCG`
Asparagine (N)	`ATT, AAC`	Serine (S)	`TCT, TCC, TCA,TCG, AGT, AGC`
Aspartic acid (Aspartate, D)	`GAT, GAC`	Threonine (T)	`ACT, ACC, ACA, ACG`
Cysteine (C)	`TGT, TGC`	Tryptophan (W)	`TGG`
Glutamine (Q)	`CAA, CAG`	Tyrosine (Y)	`TAT, TAC`
Glutamic acid (Glutamate, E)	`GAA, GAG`	Valine (V)	`GTT, GTC, GTA, GTG`
Glycine (G)	`GGT, GGC, GGA, GGG`	Aspartic acid or Asparagine	`B`—random codon from `D` and `N`

Amino Acid		**Amino Acid**	
Histidine (H)	`CAT, CAC`	Glutamic acid or Glutamine	`Z`—random codon from `E` and `Q`
Isoleucine (I)	`ATT, ATC, ATA`	Unknown or any amino acid	`Xrandom codon`
Leucine (L)	`TTA, TTG, CTT, CTC, CTA, CTG`	Translation stop (*)	`TAA, TAG, TGA`
Lysine (K)	`AAA, AAG`	Gap of indeterminate length (-)	`---`
Methionine (M)	`ATG`	Any character or any symbol not in table (?)	`???`

Examples

1 Convert a 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 characters `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 the amino acids in a 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 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`.
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'`. The default value is `'bundle'`.
StructureValue	Property to control blocking the unknown characters warning and not counting 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, adds 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

Read microarray data from Affymetrix GeneChip file

Syntax

```
AFFYData = affyread(File)
AFFYData = affyread(File, LibraryDir)
```

Arguments

File	Enter a filename, or a path and filename supported by your computer. Supported file formats are `DAT`, `EXP`, `CEL`, `CHP` and, `CDF`. If the file cannot be located on the Web, it needs to be stored locally.
LibraryDir	Enter the path and directory where the library file (`CDF`) is stored.

Description

`AFFYData = affyread(File)` reads an Affymetrix data file (`File`) and creates a MATLAB structure (`AFFYDdata`).

`AFFYData = affyread(File, LibraryDir)` specifies the directory where the library files (`CDF`) are stored.

Note: The function `affyread` only works on PC supported platforms.

GeneChip and Affymetrix are registered trademarks of Affymetrix, Inc.

See Also

Bioinformatics Toolbox functions `gprread`, `probelibraryinfo`, `probesetlink`, `probesetlookup`, `probesetplot`, `probesetvalues`, `sptread`

aminolookup

Purpose

Display amino acid codes, integers, abbreviations, names, and codons

Syntax

```
aminolookup(SeqAA)
aminolookup(..., 'PropertyName', PropertyValue,...)
aminolookup('Code', CodeValue)
aminolookup('Integer', IntegerValue)
aminolookup('Abbreviation', AbbreviationValue)
aminolookup('Name', NameValue)
```

Arguments

SeqAA	Amino acid sequence. Enter a character string of single-letter codes or three-letter abbreviations from the Amino Acid Lookup Table below.
CodeValue	Amino acid single-letter code. Enter a single character from the Amino Acid Lookup Table below.
IntegerValue	
AbbreviationValue	Amino acid three-letter abbreviation. Enter a three-letter abbreviation from the Amino Acid Lookup Table below.
NameValue	Amino acid name. Enter an amino acid name from the Amino Acid Lookup Table below.

Amino Acid Lookup Table

Code	Integer	Abbreviation	Name	Codons
A	1	Ala	Alanine	GCU GCC GCA GCG
R	2	Arg	Arginine	CGU CGC CGA CGG AGA AGG

Code	Integer	Abbreviation	Name	Codons
N	3	Asn	Asparagine	AAU AAC
D	4	Asp	Aspartic acid (Aspartate)	GAU GAC
C	5	Cys	Cysteine	UGU UGC
Q	6	Gln	Glutamine	CAA CAG
E	7	Glu	Glutamic acid (Glutamate)	GAA GAG
G	8	Gly	Glycine	GGU GGC GGA GGG
H	9	His	Histidine	CAU CAC
I	10	Ile	Isoleucine	AUU AUC AUA
L	11	Leu	Leucine	UUA UUG CUU CUC CUA CUG
K	12	Lys	Lysine	AAA AAG
M	13	Met	Methionine	AUG
F	14	Phe	Phenylalanine	UUU UUC
P	15	Pro	Proline	CCU CCC CCA CCG
S	16	Ser	Serine	UCU UCC UCA UCG AGU AGC
T	17	Thr	Threonine	ACU ACC ACA ACG
W	18	Trp	Tryptophan	UGG
Y	19	Tyr	Tyrosine	UAU UAC
V	20	Val	Valine	GUU GUC GUA GUG

Code	Integer	Abbreviation	Name	Codons
B	21	Asx	Aspartic acid or Asparagine	AAU AAC GAU GAC
Z	22	Glx	Glutamic acid or Glutamine	CAA CAG GAA GAG
X	23	Xaa	Any amino acid	All codons
*	24	END	Termination (translation stop)	UAA UAG UGA
-	25	GAP	Gap of unknown length	- - -
?	0	???	Unknown amino acid	

Description

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

`aminolookup(`*`SeqAA`*`)` converts between amino acid three-letter abbreviations and one-letter codes. If the input is a character string of three-letter abbreviations, then the output is a character string with the corresponding one-letter codes. If the input is a character string of single-letter codes, then the output is a character string of three-letter codes.

If you enter one of the ambiguous characters `B`, `Z`, `X`, this function displays the abbreviation for the ambiguous amino acid character.

```
aminolookup('abc')

ans=
AlaAsxCys
```

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

`aminolookup('Code', CodeValue)` displays the corresponding amino acid three-letter abbreviation and name.

`aminolookup('Integer', IntegerValue)` displays the corresponding amino acid single-letter code and name.

`aminolookup('Abbreviation', AbbreviationValue)` displays the corresponding amino acid single-letter code and name.

`aminolookup('Name', NameValue)` displays the corresponding single-letter amino acid code and three-letter abbreviation.

Examples

1. Display the single-letter code and three-letter abbreviation for proline.

```
aminolookup('Name','proline')

ans =
P  Pro
```

2. Convert a single-letter amino acid sequence to a three-letter sequence.

```
aminolookup('MWKQAEDIRDIYDF')

ans =
MetTrpLysGlnAlaGluAspIleArgAspIleTyrAspPhe
```

3. Convert a three-letter amino acid sequence to a single-letter sequence.

```
aminolookup('MetTrpLysGlnAlaGluAspIleArgAspIleTyrAspPhe')

ans =
MWKQAEDIRDIYDF
```

4. Display the single-letter code, three-letter abbreviation, and name for an integer.

```
aminolookup('integer', 1)

ans =
A  Ala Alanine
```

See Also Bioinformatics Toolbox functions `aa2int`, `aacount`, `geneticcode`, `int2aa`, `nt2aa`, `revgeneticcode`

Purpose Calculate atomic composition of a protein

Syntax `Atoms = 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 `Atoms = 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 (`Atoms`) with fields `C`, `H`, `N`, `O`, and `S`.

Examples Get an amino acid sequence from the Protein Sequence Database (`PIR-PSD`) and count the atoms in the sequence.

```
pirdata = getpir('cchu','SequenceOnly',true);
mwcchu = atomiccomp(pirdata)

mwcchu =
    C: 526
    H: 845
    N: 143
    O: 149
    S: 6

mwcchu.C

ans=
      526
```

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

basecount

Purpose

Count nucleotides in a sequence

Syntax

```
Bases = 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	Property to select a type of plot. Enter either 'pie' or 'bar'.
OthersValue	Property to control counting ambiguous characters individually. Enter either full' or 'bundle'. Default is 'bundle'.

Description

Bases = 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 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`), 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 `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 4 nucleotides, and only if there are ambiguous and unknown characters, add an `Others` field with the counts.
- `basecount(SeqNT, 'Others', 'full')` — Display 4 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 4 nucleotides and always display an `Others` field. If there are ambiguous and unknown characters, adds counts to the `Others` field otherwise display `0`.

- `basecount(SeqNT, 'Others', 'full', 'Structure', 'full')` — Display 4 nucleotides, 11 ambiguous nucleotides, gaps, and `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`

baselookup

Purpose Display nucleotide codes, integers, names, and abbreviations

Syntax

```
baselookup(..., 'PropertyName', PropertyValue,...)
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 Nucleotide Lookup Table below. `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 Nucleotide Lookup Table below. `Code` can also be a cell array or a two-dimensional character array.
IntegerValue	Nucleotide integer. Enter an integer from the Nucleotide Lookup Table below. Integers are arbitrarily assigned to IUB/IUPAC letters.
NameValue	Nucleotide name. Enter a nucleotide name from the Nucleotide Lookup Table below. *NameValue* can also be a single name, a cell array, or a two-dimensional character array.

Nucleotide Lookup Table

Code	Integer	Base Name	Meaning	Complement
A	1	**A**denine	A	T
C	2	**C**ytosine	C	G
G	3	**G**uanine	G	C
T	4	**T**hymine	T	A
U	4	**U**racil	U	A
R	5	(Pu**R**ine)	G|A	Y
Y	6	(P**Y**rimidine)	T|C	R
K	7	(**K**eto)	G|T	M
M	8	(A**M**ino)	A|C	K
S	9	**S**trong interaction (3 H bonds)	G|C	S
W	10	**W**eak interaction (2 H bonds)	A|T	W
B	11	Not-A (B follows A)	G|T|C	V
D	12	Not-C (D follows C)	G|A|T	H
H	13	Not-G (H follows G)	A|T|C	D
V	14	Not-T (or U) (V follows U)	G|A|C	B
N,X	15	A**N**y nucleotide	G|A|T|C	N
-	16	Gap of indeterminate length	Gap	-

Description

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

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

`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` , `geneiccode`, `nt2aa` , `nt2int` , `revgeneticode`, `seqtool`

Purpose

Create biograph object

Syntax

```
BGobj = biograph(CMatrix)
BGobj = biograph(CMatrix, NodeIDs)
```

Arguments

`CMatrix`	Connection matrix. Enter a square matrix that is full or sparse. For a square matrix the number of rows is equal to the number of nodes. A value of `1` indicates a connection to a node while a `0` indicates no connection.
`NodeIds`	Node identification strings. Enter a cell array of strings with the same number of strings as the number of rows/columns in the connection matrix (`CMatrix`). Default values are the row/column numbers.

Description

`BGobj = biograph(CMatrix)` creates a graph 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.

A biograph (`BGobj`) has two properties (`Nodes`, `Edges`) that have their own properties.

`BGobj = biograph(CMatrix, NodeIDs)` specifies the node identification strings (`NodeIDs`).

Access properties of a biograph object with `BGobj.`*`propertyname`*, `BGobj.`*`propertyname.propertyname`*, or with the `get` and `set` commands.

Properties for the Object Biograph

Biograph Property	Description
`ID`	Enter a character string.
`Label`	Enter a character string.
`Description`	Description of the graph. Enter text.
`LayoutType`	Algorithm for the layout engine. Enter `'hierarchical'`(default), `'equilibrium'`, `'radial'`.
`EdgeType`	Enter `'straight'`, `'curved'`(default), `'segmented'`. Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Graphs with `LayoutType` equal to `'equilibrium'` or `'Radial`' cannot produce curved or segmented edges.
`Scale`	Property to post-scale the node coordinates. Enter a positive number.
`LayoutScale`	Property to scale the size of the nodes before calling the layout engine. Enter a positive number.
`ShowArrows`	Property to control showing arrows with the edges. Enter either `'on'` (default) or `'off`'.
`NodeAutoSize`	Property to control precalculating the node size before calling the layout engine. Enter either `'on'` or `'off'`.
`NodeCallback`	User callback for all nodes. Enter the name of a function or a function handle. Default is `'display'`.

Biograph Property	Description
`EdgeCallback`	User callback for all edges. Enter the name of a function or function handle. Default is `'display'`.
`Nodes`	Column vector with handles to nodes. Size of vector is `NumberOfNodes x 1`. For properties of the Nodes property, see the table below.
`Edges`	Column vector with handles to edges. Size of vector is `NumberOfEdges x 1`. For properties of the Edges property, see the table below.

Properties of the Nodes Property

`ID`	Character string defined when the biograph object is created. Node IDs must be unique. Read-only.
`Label`	User defined label for a node on a graph. Enter a character string. The default value is the `ID` property.
`Description`	Description of the node. Enter text.
`Position`	Two element numeric vector of `x` and `y` coordinates computed by the layout engine. The default is `[]`. For example, `[150 150]`.
`Shape`	Enter `'box'`(default), `'ellipse'`, `'circle'`, `'rect'`,`'rectangle'`,`'diamond'`, `'trapezium'`, `'house'`, `'invtrapezium'`, `'inverse'`, `'parallelogram'`.

`Size`	Two element numeric vector calculated before calling the layout engine using the actual font size and shape of the node. The default value is `[10 10]`.
`Color`	RGB three element numeric vector. Default is `[1 1 0.7]`.
`LineWidth`	Positive number. Default is `1`.
`LineColor`	RGB three element numeric vector. Default is `[0.3 0.3 1]`.
`FontSize`	Positive number. Default is `8` pts.
`TextColor`	RGB three element numeric vector. Default is `[0 0 0]`.

Properties of the Edge Property

`ID`	Character string defined when the biograph object is created. Edge `ID`s must be unique. Read-only.
`Label`	Label for a node on a graph. Enter a string.
`Description`	Description for a node. Enter a text.
`LineWidth`	Positive number. Default is `1`.
`LineColor`	RGB three element numeric vector. Default is `[0.5 0.5 0.5]`.

Method Summary

biograph (biograph)	Create biograph object
dolayout (biograph)	Calculate node positions and edge trajectories
getancestors (biograph)	Find ancestors in a biograph object

getdescendants (biograph)	Find descendants in a biograph object
getedgesbynodeid (biograph)	Get handles to edges in graph
getnodesbyid (biograph)	Get handles to nodes
getrelatives (biograph)	Find relatives in a biograph object
view (biograph)	Draw figure from biograph object

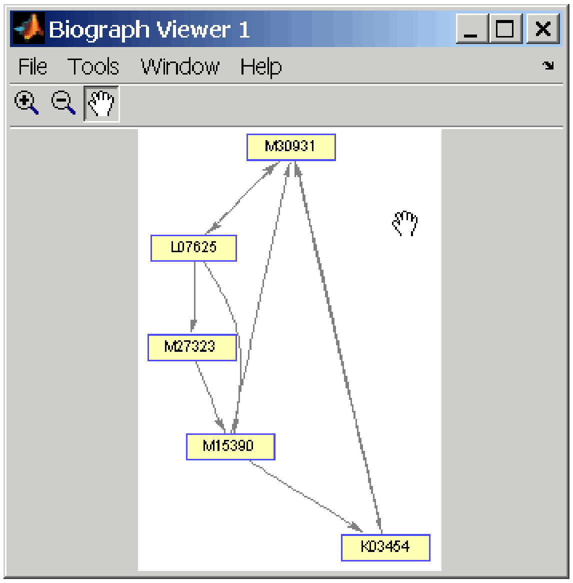
Example

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];
bg1 = biograph(cm)
get(bg1.nodes,'ID')

ans =
    'Node 1'
    'Node 2'
    'Node 3'
    'Node 4'
    'Node 5'
```

2 Create a `biograph` object and assign the node IDs.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
ids = {'M30931','L07625','K03454','M27323','M15390'};
bg2 = biograph(cm,ids);
get(bg2.nodes,'ID');

view(bg2);
```



In `bg1.Node, the properties ID` and `Label` are set to the same value. However, you can only modify the **Label** field. `Node.ID` is used internally to identify the nodes.

See Also

Bioinformatics Toolbox methods for the object biograph, `biograph`, `dolayout`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `view`

MATLAB functions `get`, `set`

Purpose

Generate a remote BLAST request

Syntax

```
blastncbi(Seq, Program, 'PropertyName', PropertyValue...)
RID = blastncbi(Seq, Program)
[RID, RTOE]= blastncbi(Seq, Program)

blastncbi(..., 'Database', DatabaseValue)
blastncbi(..., 'Descriptions', DescriptionsValue)
blastncbi(..., 'Alignments', AlignmentsValue)
blastncbi(..., 'Filter', FilterValue)
blastncbi(..., 'Expect', ExpectValue)
blastncbi(..., 'Word', WordValue)
blastncbi(..., 'Matrix', MatrixValue)
blastncbi(..., 'Gapopen', GapopenValue)
blastncbi(..., 'ExtendGap', ExtendGapValue)
blastncbi(..., 'Inclusion', InclusionValue)
blastncbi(..., 'Pct', PctValue)
```

Arguments

`Seq`	Nucleotide or amino acid sequence. Enter a GenBank or RefSeq accession number, GI, FASTA file, URL, string, character array, or a MATLAB structure that contains the field `Sequence`. You can also enter a structure with the field `Sequence`.
`Program`	BLAST program. Enter `'blastn'`, `'blastp'`, `'pciblast'`, `'blastx'`, `'tblastn'`, `'tblastx'`, or `'megablast'`.

Database	Property to select a database. Compatible databases depend upon the type of sequence submitted and program selected. The nonredundant database, `'nr'`, is the default value for both nucleotide and amino acid sequences. For nucleotide sequences, enter `'nr'`, `'est'`, `'est_human'`, `'est_mouse'`, `'est_others'`, `'gss'`, `'htgs'`, `'pat'`, `'pdb'`, `'month'`, `'alu_repeats'`, `'dbsts'`, `'chromosome'`, or `'wgs'`. The default value is ’`nr'`. For amino acid sequences, enter `'nr'`, `'swissprot'`, `'pat'`, `'pdb'`, or `'month'`. The default value is `'nr'`.
`Description`	Property to specify the number of short descriptions. The default value is normally `100`, and for `Program = pciblast`, the default value is `500`.
`Alignment`	Property to specify the number of sequences to report high-scoring segment pairs (HSP). The default value is normally `100`, and for `Program = pciblast`, the default value is `500`.
`Filter`	Property to select a filter. Enter `'L'` (low-complexity), `'R'` (human repeats), `'m'` (mask for lookup table), or `'lcase'` (to turn on the lowercase mask). The default value is `'L'`.
`Expect`	Property to select the statistical significance threshold. Enter a real number. The default value is `10`.
`Word`	Property to select a word length. For amino acid sequences, `Word` can be `2` or `3` (`3` is the default value), and for nucleotide sequences, Word can be `7`, `11`, or `15` (`11` is the default value). If `Program = 'MegaBlast'`, `Word` can be `11`, `12`, `16`, `20`, `24`, `28`, `32`, `48`, or `64`, with a default value of `28`

`Matrix`	Property to select a substitution matrix for amino acid sequences. Enter `'PAM30'`, `'PAM70'`, `'BLOSUM80'`, `'BLOSUM62'`, or `'BLOSUM45'`. The default value is `'BLOSUM62'`.
`Inclusion`	Property for PCI-BLAST searches to define the statistical significance threshold. The default value is `0.005`.
`Pct`	Property to select the percent identity. Enter `None`, `99`, `98`, `95`, `90`, `85`, `80`, `75`, or `60`. Match and mismatch scores are automatically selected. The default value is `99` (`99`, `1`, `-3`)

Description

The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of interesting protein and nucleotide sequences against known structures in existing online databases.

`blastncbi(Seq, Program)` sends a BLAST request against a sequence (`Seq`) to NCBI using a specified program (`Program`).

- With no output arguments, `blastncbi` returns a command window link to the actual NCBI report.
- A call with one output argument returns the Report ID (RID).
- A call with two output arguments returns both the RID and the Request Time Of Execution (`RTOE`, an estimate of the time until completion).

`blastncbi` uses the NCBI default values for the optional arguments: `'nr'` for the database, `'L'` for the filter, and `'10'` for the expectation threshold. The default values for the remaining optional arguments depend on which program is used. For help in selecting an appropriate BLAST program, visit

```
http://www.ncbi.nlm.nih.gov/BLAST/producttable.shtml
```

Information for all of the optional parameters can be found at

```
http://www.ncbi.nlm.nih.gov/blast/html/blastcgihelp.html
```

`blastncbi(..., 'Database', DatabaseValue)` selects a database for the alignment search.

`blastncbi(..., 'Descriptions', DescriptionsValue)`, when the function is called without output arguments, specifies the numbers of short descriptions returned to the quantity specified.

`blastncbi(..., 'Alignments', AlignmentsValue)`, when the function is called without output arguments, specifies the number of sequences for which high-scoring segment pairs (HSPs) are reported.

`blastncbi(..., 'Filter', FilterValue)` selects the filter to applied to the query sequence.

`blastncbi(...  , 'Expect', ExpectValue)` provides a statistical significance threshold for matches against database sequences. 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(..., 'Word', WordValue)` selects a word size for amino acid sequences.

`blastncbi(..., 'Matrix', MatrixValue)` selects the substitution matrix for amino acid sequences only. This matrix assigns the score for a possible alignment of two amino acid residues.

`blastncbi(..., 'GapOpen', GapOpenValue)` selects a gap penalty for amino acid sequences. Allowable values for a gap penalty vary with the selected substitution matrix. For information about allowed gap penalties for matrixes other then the `BLOSUM62` matrix, see

```
http://www.ncbi.nlm.nih.gov/blast/html/blastcgihelp.html
```

`blastncbi(...  , 'ExtendGap', ExtendGapValue)` defines the penalty for extending a gap greater than one space.

`blastncbi(..., 'Inclusion', InclusionValue)` for PSI-BLAST only, defines the statistical significance threshold (`InclusionValue`) for including a sequence in the Position Specific Score Matrix (PSSm) created by PSI-BLAST for the subsequent iteration. The default value is `0.005`.

`blastncbi(..., 'Pct', *PctValue*)`, when *`ProgramValue`* is `'Megablast'`, selects the percent identity and the corresponding match and mismatch score for matching existing sequences in a public database.

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 BLASTREAD to parse the report and
% load it into a MATLAB structure.
blastread(RID)
```

See Also

Bioinformatics Toolbox function `blastread`, `getblast`

blastread

Purpose

Read data from NCBI BLAST report file

Syntax

`Data = blastread(File)`

Arguments

`File`	NCBI BLAST formatted report file. Enter a filename, a path and filename, or a URL pointing to a file. *`File`* can also be a MATLAB character array that contains the text for a NCBI BLAST report.

Description

BLAST (**B**asic **L**ocal **A**lignment **S**earch **T**ool) reports offer a fast and powerful comparative analysis of interesting protein and nucleotide sequences against known structures in existing online databases. BLAST reports can be lengthy, and parsing the data from the various formats can be cumbersome.

`Data = blastread(File)` reads a BLAST report from an NCBI formatted file (*`File`*) and returns a data structure (*`Data`*) containing fields corresponding to the BLAST keywords. `blastread` parses the basic BLAST reports `BLASTN`, `BLASTP`, `BLASTX`, `TBLASTN`, and `TBLASTX`.

`Data` contains the following fields:

```
RID
Algorithm
Query
Database
Hits.Name
Hits.Length
Hits.HSP.Score
Hits.HSP.Expect
Hits.HSP.Identities
Hits.HSP.Positives  (peptide sequences)
Hits.HSP.Gaps
Hits.HSP.Frame      (translated searches)
Hits.HSP.Strand     (nucleotide sequences)
Hits.HSP.Alignment (3xn: Query- R1, Alignment- R2, Subject-R3)
Hits.HSPs.QueryIndices
Hits.HSPs.SubjectIndices
Statistics
```

References

For more information about reading and interpreting BLAST reports, see

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

Examples

1 Create a BLAST request with a GenPept accession number.

```
RID = blastncbi('AAA59174', 'blastp', 'expect', 1e-10)
```

2 pass the RID to `getblast` to download the report and save % it to a text file.

```
getblast(RID, 'ToFile' ,'AAA59174_BLAST.rpt')
```

3 Using the saved file, read the results into a MATLAB structure.

```
results = blastread('AAA59174_BLAST.rpt')
```

See Also

Bioinformatics Toolbox functions `blastncbi`, `getblast`

blosum

Purpose

Return a BLOSUM scoring matrix

Syntax

```
Matrix = blosum(Identity,
                'PropertyName', PropertyValue...)
[Matrix, Matrixinfo] = blosum(N)

blosum(..., 'Extended', ExtendedValue)
blosum(..., 'Order', OrderValue)
```

Arguments

`Identity`	Percent identity level. Enter `values from 30` to `90` in increments of `5`, enter `62`, or enter `100`.
`Extended`	Property to control the listing of extended amino acid codes. Enter either `true` or `false`. The default value is `true`.
`Order`	Property to specify the order amino acids are listed in the matrix. Enter a character string of legal amino acid characters. The length is `20` or `24` characters.

Description

`Matrix = blosum(`*`Identity`*`, '`*`PropertyName`*`', `*`PropertyValue`*`...)` returns a BLOSUM (**Blo**cks **Sub**stitution **M**atrix) 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 *
```

`blosum(..., 'Extended', `*`ExtendedValue`*`)` if `Extended` is `false`, this function returns the scoring matrix for the standard `20` amino acids. Ordering of the output when `Extended` is `false` is

```
A R N D C Q E G H I L K M F P S T W Y V
```

`blosum(..., 'Order', *OrderValue*)` returns a BLOSUM matrix ordered by an amino acid sequence (*`OrderString`*).

`[B, MatrixInfo] = blosum(Identity)` returns a structure of information about a BLOSUM matrix with the fields `Name`, `Scale`, `Entropy`, `ExpectedScore`, `HighestScore`, `LowestScore`, and `Order`.

Examples

Return a BLOSUM matrix with a value 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 `blosum62`, `dayhoff`, `gonnet`, `nwalign`, `pam`, `swalign`

classperf

Purpose

Evaluated the performance of a classifier

Syntax

```
classperf
cp = classperf(groundtruth)
classperf(cp, classout)
classperf(cp, classout, testidx)
cp = classperf(groundtruth, classout,...)
cp = classperf(...,'positive', p, 'negative', n)
```

Description

`classperf` provides an interface to keep track of the performance during the validation of classifiers. `classperf` creates and updates a classifier performance (CP) object 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', p, 'negative', n) 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). p defaults to the first class returned by grp2idx(groundtruth), while n 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 p and n 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 `knnclassify`, `svmclassifycrossvalind`

Statistical Toolbox functions `grp2idx`, `classify`

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 set the probability that a cleavage site will be cleaved. Enter a value from `0` to `1`. The default value is `1`.

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

Example

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`, `seqshowwords`

MATLAB function `regexp`

Purpose

Create dendrogram and heat map

Syntax

```
clustergram(Data)
clustergram(..., 'PropertyName', PropertyValue,...)
clustergram(..., 'RowLabels', RowLabelsValue)
clustergram(..., 'ColumnLabels', ColumnLabelsValue)
clustergram(..., 'Pdist', PdistValue)
clustergram(..., 'Linkage', LinkageValue)
clustergram(..., 'Dendrogram', DendrogramValue)
clustergram(..., 'ColorMap', ColorMapValue)
clustergram(..., 'SymmetricRange', SymmetricRangeValue)
clustergram(..., 'Dimension', DimensionValue)
clustergram(..., 'Ratio', RatioValue)
```

Arguments

Data	Matrix where each row corresponds to a gene. Each column is the result from one experiment.
RowLabelsValue	Property to label the rows in `Data.ColLabels` Enter a cell array of text strings.
ColumnLabelsValue	Property to label the columns in `Data`. For example, you can enter the names of the genes. Enter a cell array of text strings.
PdistValue	Property to select the distance metric and pass arguments to the function `pdist`. The default distance metric for a `clustergram` is `'correlation'`.
LinkageValue	Property to select the linkage method and pass arguments to the function `linkage`. The default linkage method is `'average'`
DendrogramValue	Property to pass arguments to the function `dendrogram`.

ColorMapValue	Property to select a colormap. Enter the name or function handle of a function that returns a colormap, or an `M`-by-`3` array containing RGB values. The default value is `REDGREENCMAP`.
`SymmetricRangValue`	Property to force the color range to be symmetric around zero. Enter either `true` or `false`. The default value is `true`.
`DimensionValue`	Property to select either a one-dimensional or two-dimensional clustergram. Enter either `1` or `2`. The default value is `1`.
`RatioValue`	Property to specify the ratio of the space that the dendrogram(s) uses.

Description

`clustergram(Data)` creates a dendrogram and heat map from gene expression data (*`Data`*) using hierarchical clustering with correlation as the distance metric and using average linkage to generate the hierarchical tree. The clustering is performed on the rows of data (*`Data`*). The rows of are typically genes and the columns are the results from different microarrays. To cluster the columns instead of the rows, transpose the data using the transpose (`'`) operator.

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

`clustergram(..., 'RowLabels',` *`RowLabelsValue`*`)` uses the contents of a cell array (*`RowLabelsValue`*) as labels for the rows in *`Data`*.

`clustergram(..., 'ColumnLabels',` *`ColumnLabelsValue`*`)` uses the contents of a cell array (*`ColumnLabelsValue`*) as labels for the columns in *`Data`*.

`clustergram(..., 'Pdist',` *`PdistValue`*`)` sets the distance metric the function `pdist` uses to calculate the pairwise distances between observations. If the distance metric requires extra arguments, then pass the arguments as a cell array. For example, to use the Minkowski distance with exponent `P` you the help for the Statistical Toolbox function `pdist`.

`clustergram(..., 'Linkage', *LinkageValue*)` selects the linkage method the function `linkage` uses to create the hierarchical cluster tree. For more information about the available options, see the help for the Statistical Toolbox function `linkage`.

`clustergram(..., 'Dendrogram', *DendrogramValue*)` passes arguments the function `dendrogram` uses to create a dendrogram. `Dendrogram` should be a cell array of parameter name/value pairs that can be passed to `dendrogram`. For more information about the available options, see the help for the Statistical Toolbox function `dendrogram`.

`clustergram(..., 'ColorMap', *ColorMapValue*)` specifies the colormap for the figure containing the clustergram. This controls the colors used to display the heat map.

`clustergram(..., 'SymmetricRange', *SymmetricRangeValue*)`, when `SymmetricRange` is `false`, disables the default behavior of forcing the color scale of the heat map to be symmetric about zero.

`clustergram(..., 'Dimension', *DimensionValue*)` specifies whether to create a one-dimensional or two-dimensional clustergram. 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(..., 'Ratio', *RatioValue*)` specifies the ratio of the space that the dendrogram(s) uses, relative to the size of the heat map, in the `X` and `Y` directions. If `Ratio` is a single scalar value, it is used as the ratio for both directions. If `Ratio` 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. The default ratio is `1/5`.

Hold the mouse button down over the image to see the exact values at a particular point.

Example

1. Load filtered yeast data.

```
clustergram(yeastvalues);
```

2. Add labels.

```
clustergram(yeastvalues,'ROWLABELS',genes,'COLUMNLABELS',times);
```

3. Change the clustering parameters.

```
clustergram(yeastvalues,'PDIST','euclidean','LINKAGE','complete');
```

4. Change the dendrogram color parameter.

```
clustergram(yeastvalues,'ROWLABELS',genes,'DENDROGRAM',{'color',5});
```

See Also

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

Purpose

Create dendrogram and heat map

Syntax

```
clustergram(Data)
clustergram(..., 'PropertyName', PropertyValue,...)
clustergram(..., 'RowLabels', RowLabelsValue)
clustergram(..., 'ColumnLabels', ColumnLabelsValue)
clustergram(..., 'Pdist', PdistValue)
clustergram(..., 'Linkage', LinkageValue)
clustergram(..., 'Dendrogram', DendrogramValue)
clustergram(..., 'ColorMap', ColorMapValue)
clustergram(..., 'SymmetricRange', SymmetricRangeValue)
clustergram(..., 'Dimension', DimensionValue)
clustergram(..., 'Ratio', RatioValue)
```

Arguments

Data	Matrix where each row corresponds to a gene. Each column is the result from one experiment.
RowLabelsValue	Property to label the rows in `Data.ColLabels` Enter a cell array of text strings.
ColumnLabelsValue	Property to label the columns in `Data`. For example, you can enter the names of the genes. Enter a cell array of text strings.
PdistValue	Property to select the distance metric and pass arguments to the function `pdist`. The default distance metric for a `clustergram` is `'correlation'`.
LinkageValue	Property to select the linkage method and pass arguments to the function `linkage`. The default linkage method is `'average'`
DendrogramValue	Property to pass arguments to the function `dendrogram`.

ColorMapValue	Property to select a colormap. Enter the name or function handle of a function that returns a colormap, or an `M`-by-`3` array containing RGB values. The default value is `REDGREENCMAP`.
`SymmetricRangValue`	Property to force the color range to be symmetric around zero. Enter either `true` or `false`. The default value is `true`.
`DimensionValue`	Property to select either a one-dimensional or two-dimensional clustergram. Enter either `1` or `2`. The default value is `1`.
`RatioValue`	Property to specify the ratio of the space that the dendrogram(s) uses.

Description

`clustergram(Data)` creates a dendrogram and heat map from gene expression data (*`Data`*) using hierarchical clustering with correlation as the distance metric and using average linkage to generate the hierarchical tree. The clustering is performed on the rows of data (*`Data`*). The rows of are typically genes and the columns are the results from different microarrays. To cluster the columns instead of the rows, transpose the data using the transpose (`'`) operator.

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

`clustergram(..., 'RowLabels',` *`RowLabelsValue`*`)` uses the contents of a cell array (*`RowLabelsValue`*) as labels for the rows in *`Data`*.

`clustergram(..., 'ColumnLabels',` *`ColumnLabelsValue`*`)` uses the contents of a cell array (*`ColumnLabelsValue`*) as labels for the columns in *`Data`*.

`clustergram(..., 'Pdist',` *`PdistValue`*`)` sets the distance metric the function `pdist` uses to calculate the pairwise distances between observations. If the distance metric requires extra arguments, then pass the arguments as a cell array. For example, to use the Minkowski distance with exponent `P` you the help for the Statistical Toolbox function `pdist`.

`clustergram(..., 'Linkage', *LinkageValue*)` selects the linkage method the function `linkage` uses to create the hierarchical cluster tree. For more information about the available options, see the help for the Statistical Toolbox function `linkage`.

`clustergram(..., 'Dendrogram', *DendrogramValue*)` passes arguments the function `dendrogram` uses to create a dendrogram. `Dendrogram` should be a cell array of parameter name/value pairs that can be passed to `dendrogram`. For more information about the available options, see the help for the Statistical Toolbox function `dendrogram`.

`clustergram(..., 'ColorMap', *ColorMapValue*)` specifies the colormap for the figure containing the clustergram. This controls the colors used to display the heat map.

`clustergram(..., 'SymmetricRange', *SymmetricRangeValue*)`, when `SymmetricRange` is `false`, disables the default behavior of forcing the color scale of the heat map to be symmetric about zero.

`clustergram(..., 'Dimension', *DimensionValue*)` specifies whether to create a one-dimensional or two-dimensional clustergram. 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(..., 'Ratio', *RatioValue*)` specifies the ratio of the space that the dendrogram(s) uses, relative to the size of the heat map, in the `X` and `Y` directions. If `Ratio` is a single scalar value, it is used as the ratio for both directions. If `Ratio` 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. The default ratio is `1/5`.

Hold the mouse button down over the image to see the exact values at a particular point.

Example

1 Load filtered yeast data.

```
clustergram(yeastvalues);
```

2 Add labels.

```
clustergram(yeastvalues,'ROWLABELS',genes,'COLUMNLABELS',times);
```

3 Change the clustering parameters.

```
clustergram(yeastvalues,'PDIST','euclidean','LINKAGE','complete');
```

4 Change the dendrogram color parameter.

```
clustergram(yeastvalues,'ROWLABELS',genes,'DENDROGRAM',{'color',5});
```

See Also

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

Purpose

Calculate codon frequency for each amino acid in a 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 Genetic Code on page 2-4.

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

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

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

Example

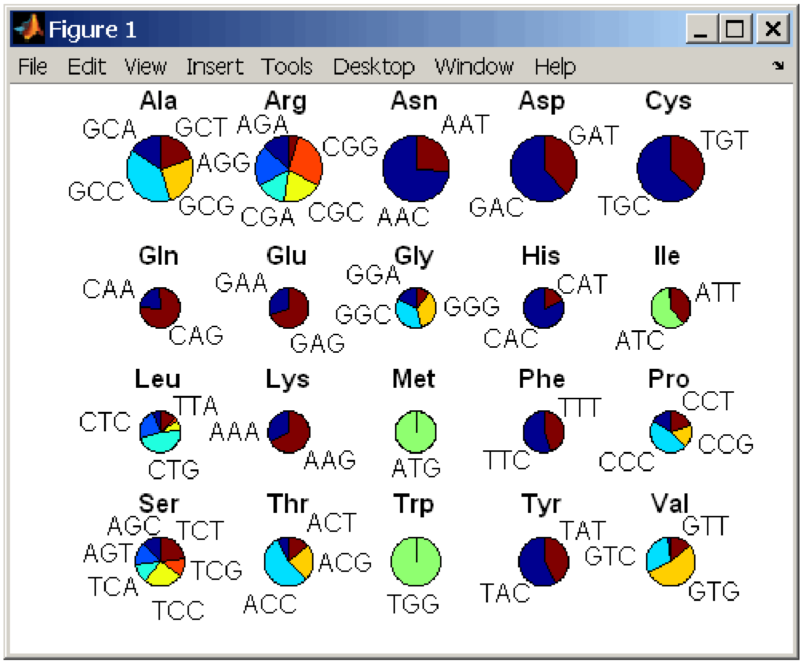
1. Import a nucleotide sequence from GenBank to MATLAB. For example, get the DNA sequence that codes for a human insulin receptor.

```
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.



See Also

Bioinformatics Toolbox functions `aminolookup`, `codoncount`, `geneticcode`, `nt2aa`

codoncount

Purpose

Count codons in nucleotide sequence

Syntax

```
Codons = codoncount(SeqNT,
                    'PropertyName', PropertyValue...)
[Codons, CodonArray] = codoncount(SeqNT)

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`.
`Frame`	Property to select a reading frame. Enter `1`, `2`, or `3`. Default value is `1`.
`Reverse`	Property to control returning the complement sequence. Enter `true` or `false`. Default value is `false`.
`Figure`	Property to control plotting a heat map. Enter either `true` or `false`. Default value is `false`.

Description

`Codons = codoncount(SeqNT, 'PropertyName',PropertyValue...)` counts the number of codon 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.
  ```

- 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 4x4x4 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(...,'Frame', *FrameValue*) counts the codons in a specific reading frame.

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

codoncount(..., 'Figure', *FigureValue*), when Figure 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);
```

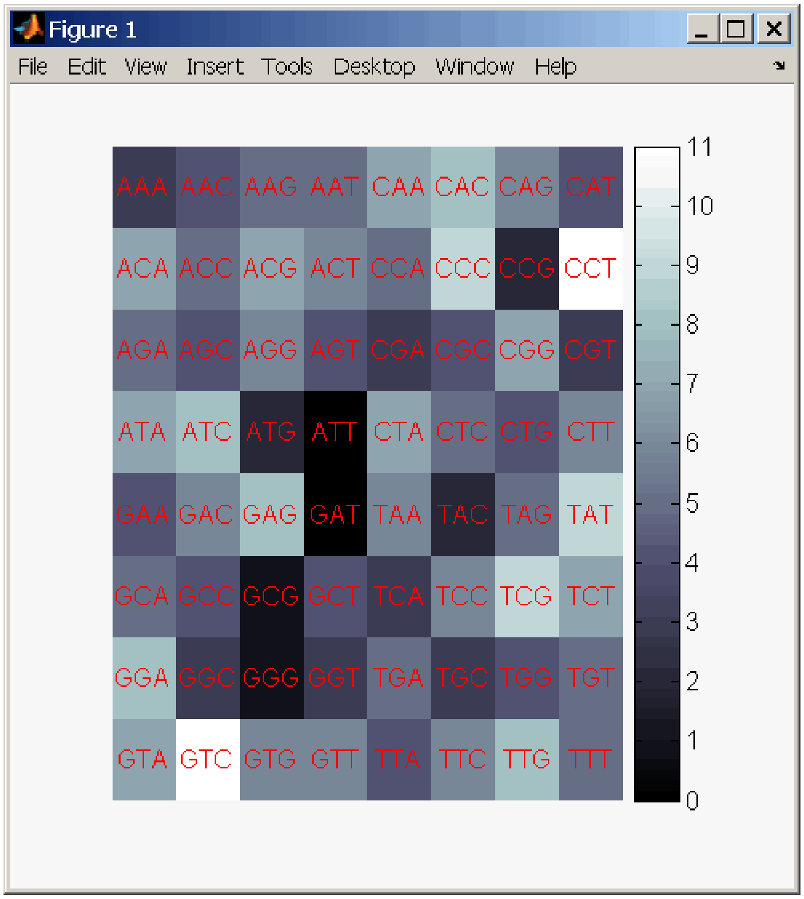


Figure 1
File Edit View Insert Tools Desktop Window Help
AAA AAC AAG AAT CAA CAC CAG CAT
ACA ACC ACG ACT CCA CCC CCG CCT
AGA AGC AGG AGT CGA CGC CGG CGT
ATA ATC ATG ATT CTA CTC CTG CTT
GAA GAC GAG GAT TAA TAC TAG TAT
GCA GCC GCG GCT TCA TCC TCG TCT
GGA GGC GGG GGT TGA TGC TGG TGT
GTA GTC GTG GTT TTA TTC TTG TTT
11
10
9
8
7
6
5
4
3
2
1
0

See Also

Bioinformatics Toolbox functions `aacount` , `basecount`, `baselookup`, `codonbias`, `dimercount`, `nmercount`, `ntdensity`, `seqrcomplement`, `seqwordcount`

cpgisland

Purpose

Locate CpG islands in a 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 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.

Example

1. Import a nucleotide sequence from GenBank. For example, get a sequence from Homo Sapiens chromosome 12.

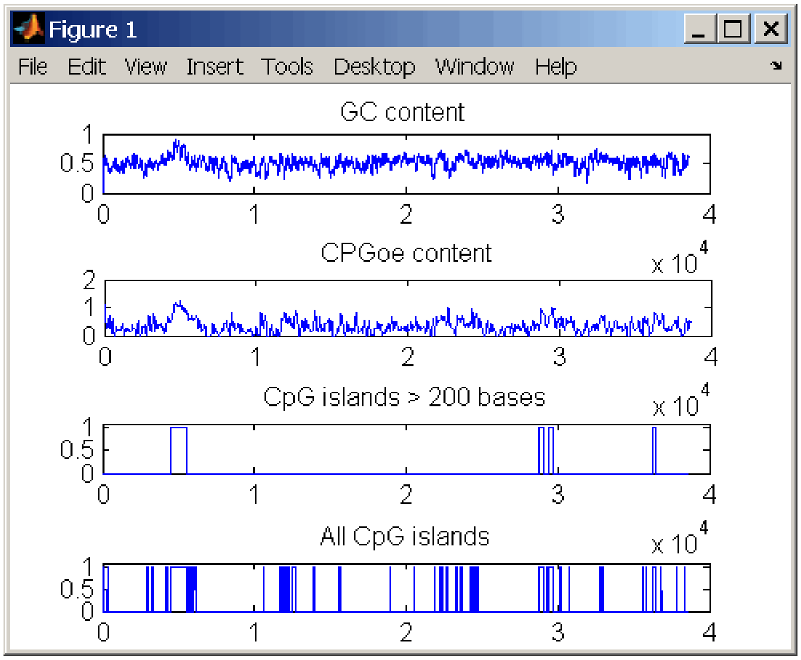
   ```
   S = getgenbank('AC156455');
   ```

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`

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.

Example 1

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`

Statistical Toolbox functions `classify`, `grp2idx`, ,

dayhoff

Purpose Return a Dayhoff scoring matrix

Syntax `ScoringMatrix = dayhoff`

Description `PAM250` type scoring matrix. 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 a sequence

Syntax

```
Dimers = dimercount(SeqNT,
                    'PropertyName', PropertyValue...)
[Dimers, Percent] = dimercount(SeqNT)

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.`
`ChartStyle`	Property to select the type of plot. Enter `'pie'` or `'bar'`.

Description

`Dimers = dimercount(SeqNT, 'PropertyName', PropertyValue...)` 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.

```
    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(..., 'Chart', ChartStyle)` creates a chart showing the relative proportions of the dimers. Valid styles are `'Pie'` and `'Bar'`.

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`

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 Mapping Nucleotide Letters to Integers on page 2-247. 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

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 Mapping Nucleotide Letters to Integers on page 2-247. 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

Calculate node positions and edge trajectories

Syntax

```
dolayout(BGobj, 'Propertyname', Propertyvalue...)
dolayout(..., 'OnlyPaths', OnlyPathsValue)
```

Arguments

`BGobj`	Biograph object.
`OnlyPaths`	Property to control the calculation of node position and edge paths. Enter `'true'` to calculate only the edge paths.

Description

`dolayout(BGobj, 'Propertyname', Propertyvalue...)` 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 following biograph object properties interact with the layout engine:

- `LayoutType` — Selects 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` — When `NodeAutoSize` is `'on'`, the layout engine uses the node properties `FontSize`, `Shape`, and `LayoutScale` to precalculate the actual size of every node. When `NodeAutoSize` is `'off'`, the layout engine uses the node property `Size`.

`dolayout(..., 'OnlyPaths', OnlyPathsValue)`, when `OnlyPaths` is `'true'`, leaves the nodes at their current positions and calculates new curves for the edges.

Example

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)
bg.nodes(1).Position
```

Nodes do not have a position yet.

2 Call the layout engine and render the graph.

```
dolayout(bg)
bg.nodes(1).Position
view(bg)
```

3 Manually modify a node position and recalculate the paths.

```
bg.nodes(1).Position = [150 150];
dolayout(bg, 'Onlypaths', true)
view(bg)
```

See Also

Bioinformatics Toolbox methods for the biograph object, `biograph`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `view`

MATLAB functions `get`, `set`

Purpose

Estimate synonymous and nonsynonymous substitution rates

Syntax

```
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2)
dnds(..., 'PropertyName', PropertyValue,...)
dnds(..., 'GeneticCode', GeneticCodeValue)
dnds(..., 'Method', MethodValue)
```

Arguments

SeqNT1, *SeqNT2*	Nucleotide sequences. Enter a character string or a structure with the field `Sequence`.
GeneticCodeValue	Property to select a genetic code. Enter a code number or code name from the table Genetic Code on page 2-4. If you use a code name, you can truncate the name to the first two characters of the name.
MethodValue	Property to select the method for calculating substitution rates. Enter `'NG'`, `'LWL'`, or `'PBL'`.

Description

`[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2)` estimates the synonymous and nonsynonymous substitution rate per site between two homologous nucleotide sequences (*SeqNT1*, *SeqNT2*) by comparing codons using the Nei-Gojobori method. This function returns the nonsynonymous substitution rate (*Dn*), the synonymous substitution rate (*Ds*), the variance for the nonsynonymous substitution rate (*Vardn*), and the variance for the synonymous substitutions per site (*Vards*). Any codons that include gaps are excluded from calculation. This analysis considers the number of codons in the shortest sequence.

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

`dnds(..., 'GeneticCode', GeneticCodeValue)` calculates synonymous and nonsynonymous substitution rates using the specified genetic code. The default is `'Standard'` or 1.

`dnds(..., 'Method', MethodValue)` allows you to calculate synonymous and nonsynonymous substitution rates using the following approaches:

`'NG'` — uses the Nei-Gojobori method '86 (default)

`'LWL'` — uses the Li-Wu-Luo method '85

`'PBL'` — uses the Pamilo-Bianchi-Li method '93

References

[1] Li W, Wu C, Luo C, "A new method for estimating synonymous and aonsynonymous rates of nucleotide substitution considering the relative likelihood of nucleotide and codon changes" in Molecular Biology Evolution, (1984) 2(2):150-174.

[2] Nei M, Gojobori T, "Simple methods for estimating the numbers of synonymous and nonsynonymous nucleotide substitutions" in Molecular Biological Evolution, (1986) 3(5):418-426.

[3] Nei M, Jin L, "Variances of the average numbers of nucleotide substitutions within and between populations" in Molecular Biology Evolution, (1989) 6(3):290-300.

[4] Nei M, Kumar S, Molecular Evolution and Phylogenetics. Oxford University Press. Chapter 4. (2000).

[5] Pamilo P, Bianchi N, "Evolution of the Zfx And Zfy genes: rates and interdependence between the genes" in Molecular Biology Evolution, (1993) 10(2): 271-281.

Examples

1 Get two sequences from Genbank for the human immunodeficiency virus.

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

2 Pairwise align the sequences using the Needleman-Wunsch algorithm.

```
[sc,al]= nwalign(gag1,gag2,'alpha','nt');
```

3 Calculate synonymous and nonsynonymous substitution rates.

```
[dn ds vardn vards] = dnds(al(1,:), al(3,:))

dn =
    0.0240
ds =
    0.0739
vardn =
    2.2745e-005
vards =
    2.6447e-004
```

See Also

Bioinformatics Toolbox functions `dndsml`, `geneticcode`, `nt2aa`, `seqpdist`

dndsml

Purpose

Estimate synonymous-nonsynonymous substitution rates by the maximum likelihood method

Syntax

```
[Dn, Ds, Like] = dndsml(SeqNT1, SeqNT2)
dndsml(..., 'PropertyName', PropertyValue,...)
dndsml(..., 'GeneticCode', GeneticCodeValue)
```

Arguments

SeqNT1, *SeqNT2*	Nucleotide sequences. Enter a character string or a structure with the field `Sequence`.
GeneticCodeValue	Property to select a genetic code. Enter a code number or code name from the table Genetic Code on page 2-4. If you use a code name, you can truncate the name to the first two characters of the name.

Description

`[Dn, Ds, Like] = dndsml(SeqNT1, SeqNT2)` estimates synonymous and nonsynonymous substitution rates between two homologous sequences (*SeqNT1*, *SeqNT2*) by the maximum likelihood method. `dndsml` returns the nonsynonymous substitution rate (*Dn*), the synonymous substitution rate (*Ds*), and the likelihood of this estimate (*Like*). The maximum likelihood method is best suited for sequences larger than `100` bases. Gaps are ignored in this analysis. This analysis considers the number of codons in the shortest sequence.

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

`dndsml(..., 'GeneticCode', GeneticCodeValue)` calculates synonymous and nonsynonymous substitution rates using the specified genetic code. The default is `'Standard'` or 1.

References

[1] Tamura, Nei, Molecular Biology and Evolution, (1993)

[2] Yang, Nielsen, Molecular Biology and Evolution, (2000)

Examples

1 Get two sequences from Genbank for the human immunodeficiency virus.

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

2 Pairwise align the sequences using the Needleman-Wunsch algorithm.

```
 [sc,al]= nwalign(gag1,gag2,'alpha','nt');
```

3 Calculate synonymous and nonsynonymous substitution rates.

```
[dn ds like] = dndsml(al(1,:), al(3,:))

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

See Also

Bioinformatics Toolbox functions `dnds`, `geneticcode`, `nt2aa`, `seqpdist`

emblread

Purpose

Read data from EMBL file

Syntax

```
EMBLData = emblread('File',
                    'PropertyName', PropertyValue)

emblread(..., 'SequenceOnly', SequenceOnlyValue)
```

Arguments

`File`	EMBL formatted file (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *`File`* can also be a MATLAB character array that contains the text for a filename.
`SequenceOnly`	Property to control reading only the sequence. Enter `true`.
`EMBLData`	MATLAB structure with fields corresponding to EMBL data.
`EMBLSeq`	MATLAB character string without metadata for the sequence.

Description

`EMBLData = emblread('`*`File`*`', '`*`PropertyName`*`',` *`PropertyValue...`*`)` 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` for the `137.0` version contains the following fields:

```
Comments
Identification
Accession
SequenceVersion
Datecreated
Dateupdated
Description
Keyword
```

```
OrganismSpecies
OorganismClassification
Organelle
Reference.Number
Reference.Comment
Reference.Position
Reference{#}.MedLine
Referemce{#}.PubMed
Reference.Authors
Reference.Title
Reference.Location
DatabaseCrossReference
Feature
Basecount
Sequence
```

`Seq = emblread('`*`File`*`', 'SequenceOnly', `*`SequenceOnlyValue`*`)`, when `SequenceOnly` 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`, `featuresparse`,`genbankread`, `getembl`, `seqtool`

Purpose

Calculate range of gene expression profiles

Syntax

```
exprprofrange(Data, 'PropertyName', PropertyValue...)
[Range, LogRange] = exprprofrange(Data)

exprprofrange(..., 'ShowHist', ShowHistValue)
```

Arguments

`Data`	Matrix where each row corresponds to a gene.
`ShowHist`	Property to control displaying a histogram with range data. Enter either `true` (include range data) or `false`. The default value is `false`.

Description

`exprprofrange(Data, 'PropertyName', PropertyValue...)` calculates the range of each expression profile in a data set (`Data`).

`[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(..., 'ShowHist', ShowHistValue)`, when `ShowHist` is `true`, displays a histogram of the range data.

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 function `exprprofvar`, `generangefilter`

Purpose

Calculate variance of gene expression profiles

Syntax

`exprprofvar(Data, 'PropertyName', PropertyValue...)`

`exprprofvar(..., 'ShowHist', ShowHistValue)`

Arguments

`Data`	Matrix where each row corresponds to a gene.
`ShowHist`	Property to control the display of a histogram with variance data. Enter `true`.

Description

`exprprofvar(Data, 'PropertyName', PropertyValue...)` 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(..., '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')
multialignread(..., 'PropertyName', PropertyValue,...)
multialignread(..., 'IgnoreGaps', IgnoreGapsValue)
```

Arguments

`File`	FASTA formatted file (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *`File`* can also be a MATLAB character array that contains the text for a filename.
`IgnoreGapsValue`	Property to control removing gap symbols.
`FASTAData`	MATLAB structure with the fields `Header` and `Sequence`.

Description

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

```
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.

`multialignread(..., 'PropertyName', PropertyValue,...)` defines optional properties. 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).

`multialignread(..., 'IgnoreGaps', IgnoreGapsValue)`, when *`IgnoreGapsValue`* is `true`, removes any gap symbol (`'-'` or `'.'`) from the sequences. Default is `false`.

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 function `emblread`, `fastawrite`, `genbankread`, `genpeptread`, `multialignread`, `seqprofile`, `seqtool`

fastawrite

Purpose

Write to file with FASTA format

Syntax

```
fastawrite('File', Data)
fastawrite('File', Header, Sequence)
```

Arguments

File	Enter either a filename or a path and filename supported by your operating system. (ASCII text file).
`Data`	Enter a character string with a FASTA format, a sequence object, a structure containing the fields `Sequence` and `Header`, or a GenBank/GenPept structure.
`Header`	Information about the sequence.
`Sequence`	Nucleotide or amino acid sequence using the standard IUB/IUPAC codes. For a list of valid characters, see Mapping Amino Acid Letters to Integers on page 2-2 and Mapping Nucleotide Letters to Integers on page 2-247.

Description

`fastawrite('File', Data)` writes the contents of `Data` to a file with a FASTA format.

`fastawrite('File', Header, Sequence)` writes header and sequence information to a file with a FASTA format.

Examples

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

%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 function `fastaread`, `seqtool`

Purpose

Read microarray data from a GenePix array list file

Syntax

```
GALData = galread('File')
```

Arguments

File	GenePix Array List formatted file (GAL). Enter a filename, or enter a path and filename.

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:

```
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.axon.com/GN_GenePix_File_Formats.html#gal
```

For a list of supported file format versions, see

```
http://www.axon.com/gn_GPR_Format_History.html
```

GenePix is a registered trademark of Axon Instruments, Inc.

See Also

Bioinformatics Toolbox functions `affyread`, `geosoftread`, `gprread`, `imageneread`, `sptread`

Purpose

Read data from a GenBank file

Syntax

```
GenBankData = genbankread('File')
```

Arguments

`File`	GenBank formatted file (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *`File`* can also be a MATLAB character array that contains the text of a GenBank formatted file.
`GenBankData`	MATLAB structure with fields corresponding to GenBank data.

Discussion

`genbankread` reads data from a GenBank formatted file into a MATLAB structure.

`GenBankData = genbankread('`*`File`*`')` reads in a GenBank formatted file (*`File`*) and creates a structure (`Data`) 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.

`GenBankData` contains the following fields:

```
LocusName
LocusSequenceLength
LocusMoleculeType
LocusGenBankDivision
LocusModificationDate
Definition
Accession
Version
GI
Keywords
Segment
Source
SourceOrganism
Reference.Number
Reference.Authors
Reference.Title
Reference.Journal
Reference.MedLine
Reference.PubMed
Reference.Remark
Comment
Features
BaseCount
Sequence
```

Examples

Get sequence information for the gene HEXA, store in a file, and then read back into MATLAB.

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

See Also

Bioinformatics Toolbox functions `emblread`, `getgenbank`, `fastaread`, `genpeptread`, `getgenbank`, `scfread`, `seqtool`, `featuresparse`

Purpose

Remove genes with low entropy expression values

Syntax

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

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 same number of rows as `Data`. Each row contains the name or ID of the gene in the data set.
`Percentile`	Property to specify a percentile below which gene data is removed. Enter a value from `0` to `100`.

Description

`Mask = geneentropyfilter(Data, 'PropertyName', PropertyValue...)` 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`), where `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)`.

`geneentropyfilter(..., 'Percentile', PercentileValue)` removes from `Data` gene expression profiles with entropy values less than the percentile `Percentile`.

Reference

Kohane, I.S., Kho, A.T., Butte, A.J., *Microarrays for an Integrative Genomics*, MIT Press, 2003.

Examples

```
load yeastdata
[fyeastvalues, fgenes] = geneentropyfilter(yeastvalues,genes);
```

See Also

Bioinformatics Toolbox functions `exprprofrange`, `exprprofvar`, `genelowvalfilter`, `generangefilter`, `genevarfilter`

Purpose

Remove gene profiles with low absolute values

Syntax

```
Mask = genelowvalfilter(Data,
                           'PropertyName', PropertyValue...)
[Mask, FData] = genelowvalfilter(Data)
[Mask, FData, FNames] = genelowvalfilter(Data, Names)

genelowvalfilter(..., 'Percentile', PercentileValue)
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.
`Percentile`	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.
`AnyVal`	Property to select the minimum or maximum absolute value for comparison with `AbsValue`. If `AnyValValue` is `true`, selects the minimum absolute value. If `AnyVal` 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, '`*`PropertyName`*`',` *`PropertyValue...`*`)` identifies gene expression profiles in `Data` with all absolute values less than the `10`th 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`). `FData` can also be created 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`. `FNames` can also be created using `FNames = Names(I)`.

`genelowvalfilter(..., 'Percentile', `*`PercentileValue`*`)` removes from `Data` gene expression profiles with all absolute values less than the 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 `AbsVal`.

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

Reference

Kohane, I.S., Kho, A.T., Butte, A.J., *Microarrays for an Integrative Genomics*, MIT Press, 2003.

Examples

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

See Also

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

Purpose

Remove gene profiles with small profile ranges

Syntax

```
Mask = generangefilter(Data,
                         'PropertyName', PropertyValue...)
[Mask, FData] generangefilter(Data)
[Mask, FData, FNames] = generangefilter(Data, Names)

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 same number of rows as `Data`. Each row contains the name or ID of the gene in the data set.
`Percentile`	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.
`LOGPercentile`	Property to specify the LOG of a percentile.
`LOGValue`	Property to specify the LOG of an absolute value.

Description

`Mask = generangefilter(Data, 'PropertyName', PropertyValue...)` calculates the range for each gene expression profile in `Data`, and then identifies the expression profiles with ranges less than the `10`th percentile.

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

`[Maks, FData] = generangefilter(Data)` returns a filtered data matrix (`FData`). `FData` can alos be created using `FData = Data(find(I),:)`.

`[Maks, FData, FNames] = generangefilter(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`. `FNames` can also be created using `FNames = Names(I)`.

`generangefilter(..., 'Percentile',` *`PercentileValue`*`)` removes from `Data` gene expression profiles with ranges less than the percentile `Percentile`.

`generangefilter(..., 'AbsValue',` *`AbsValueValue`*`)` removes from `Data` gene expression profiles with ranges less than `AbsValue`.

`generangefilter(..., 'LOGPercentile',` *`LOGPercentileValue`*`)` filters genes with profile ranges in the lowest `LOGPercentile` percent of the log range.

`generangefilter(..., 'LOGValue',` *`LOGValueValue`*`)` filters genes with profile log ranges lower than `LOGValue`.

Reference

Kohane, I.S., Kho, A.T., Butte, A.J., *`Microarrays for an Integrative Genomics`*, MIT Press, 2003.

Examples

```
load yeastdata
[mask, fyeastvalues, fgenes] = generangefilter(yeastvalues,genes);
```

See Also

Bioinformatics Toolbox functions `exprprofrange`, `exprprofvargeneentropyfilter`, `genelowvalfilter`, `genevarfilter`

Purpose

Return nucleotide codon to amino acid mapping

Syntax

```
Map = geneticcode(GeneticCode)
geneticcode(GeneticCode)
```

Arguments

GeneticCode	Enter a code number or code name from the table Genetic Code below. 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`

Code Number	Code Name
14	Flatworm Mitochondrial
15	Blepharisma Nuclear
16	Chlorophycean Mitochondrial
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 the `transl_table` (code) number from the NCBI Genetics Web page (`http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c`) or one of the supported names in the genetic code table above.

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,
                        'PropertyName', PropertyValue...)
[Mask, FData] = genevarfilter(Data)
[Mask, FData, FNames] = genevarfilter(Data, Names)

genevarfilter(..., 'Percentile', PercentileValue)
genevarfilter(..., 'AbsValue', AbsValueValue)
```

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 same number of rows as `Data`. Each row contains the name or ID of the gene in the data set.
`Percentile`	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, 'PropertyName', PropertyValue...)` calculates the variance for each gene expression profile in `Data` and then identifies the expression profiles with a variance less than the `10`th percentile.

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

`[Mask, FData] = genevarfilter(Data)` returns the filtered data matrix `FData`. `FData` can also be created 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(..., 'Percentile', *PercentileValue*)` removes from `Data` gene expression profiles with a variance less than the percentile `Percentile`.

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

Reference

Kohane, I.S., Kho, A.T., Butte, A.J., *Microarrays for an Integrative Genomics*, MIT Press, 2003.

Examples

```
load yeastdata
[fyeastvalues, fgenes] = genevarfilter(yeastvalues,genes);
```

See Also

Bioinformatics Toolbox functions `exprprofrange`, `exprprofvar`, `generangefilter`, ,`geneentropyfilter`, `genelowvalfilter`

Purpose

Read data from a GenPept file

Syntax

```
GenPeptData = genpeptread('File')
```

Arguments

`File`	GenPept formatted file (ASCII text file). Enter a filename, a path and filename, 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 recently changed the name of their protein search engine from GenPept to Entrez Protein. However, the function names in the Bioinformatics Toolbox (getgenpept, 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:

```
LocusName
LocusSequenceLength
LocusMoleculeType
LocusGenBankDivision
LocusModificationDate
Definition
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`, `pirread`, `featuresparse`, `seqtool`

Purpose

Read data from a Gene Expression Omnibus (GEO) SOFT file

Syntax

```
GEOSOFTData = geosoftread('File')
```

Arguments

`File`	Gene Expression Omnibus (GEO) formatted file (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *`File`* can also be a MATLAB character array that contains the text of a GEO file.

Description

`geosoftread` reads data from a Gene Expression Omnibus (GEO) SOFT formatted file (*`File`*), and creates a MATLAB structure (*`GEOSOFTdata`*) with the following fields:

```
Scope
Accession
Header
ColumnDescriptions
ColumnNames
Data
```

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

Examples

Get data from the GEO Web site and save it to a file.

```
geodata = getgeodata('GSM3258','ToFile','GSM3258.txt');
```

Use `geosoftread` to access a local copy from disk instead of accessing it from the GEO Web site.

```
geodata = geosoftread('GSM3258.txt')
```

See Also

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

get (phytree)

Purpose Get information about a phylogenetic tree object

Syntax

```
[Value1, Value2,...] = get(Tree, Name1,Name2,...)
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, Name1,Name2,...)` returns the specified properties from a phytree object (`Tree`).

The valid choices for `'Name'` are

`'Pointers'`	Branch to leaf/branch connectivity list
`'Distances'`	Edge length for every leaf/branch
`'NumLeaves'`	Number of leaves
`'NumBranches'`	Number of branches
`'NumNodes'`	Number of nodes (`NumLeaves + Numbranches`)
`'LeafNames'`	Names of the leaves
`'BranchNames'`	Names of the branches
`'NodeNames'`	Names of all the nodes

`get(`*`Tree`*`)` displays all property names and their current values for a phytree object (*`Tree`*).

`V`` = get(`*`Tree`*`)` returns a structure where each field name is the name of a property of a phytree object (*`Tree`*) and each field contains the value of that property.

Examples

1 Read in a phylogenetic tree from a file.

```
tr = phytreeread('pf00002.tree')
```

2 Get the names of the leafs.

```
protein_names = get(tr,'LeafNames')

protein_names =

    'BAI2_HUMAN/917-1197'
    'BAI1_HUMAN/944-1191'
    'O00406/622-883'
     ...
```

See Also

Bioinformatics Toolbox functions `phytree`, `phytreeread`, and phytree object methods `select`, `getbyname`

Purpose Find ancestors in a 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`).

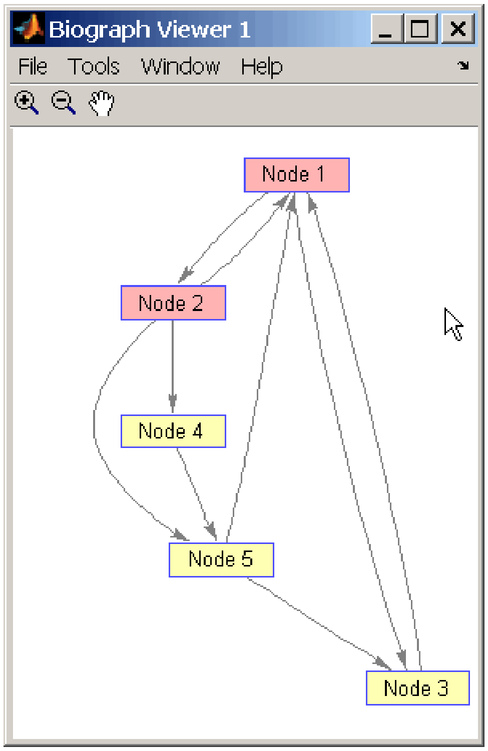
Examples

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)
```

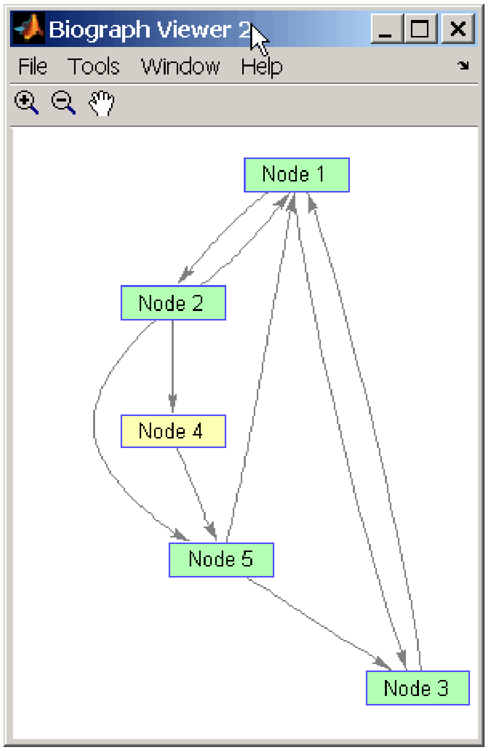
2 Find one generation of ancestors for node 2.

```
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 methods for the biograph object, `biograph`, `dolayout`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `view`

Purpose

Get BLAST report from NCBI Web site

Syntax

```
Data = getblast(RID)
getblast(..., 'PropertyName', PropertyValue,...)
getblast(..., 'Descriptions', DescriptionsValue)
getblast(..., 'Alignments', AlignmentsValue)
getblast(..., 'ToFile', ToFileValue)
getblast(..., 'FileFormat', FileFormatValue)
```

Arguments

RID	BLAST Request ID (*RID*) from the function `blastncbi`.
DescriptionsValue	Property to specify the number of descriptions in a report.
AlignmentsValue	Property to select the number of alignments in a report. Enter values from `1` to `100`. The default value is `50`.
ToFileValue	Property to enter a filename for saving report data.
FileFormatValue	Property to select the format of the file named in `ToFileValue`. Enter either `'TEXT'` or 'HTML'.The default value is `'TEXT'`.

Description

BLAST (**B**asic **L**ocal **A**lignment **S**earch **T**ool) reports offer a fast and powerful comparative analysis of interesting protein and nucleotide sequences against known structures in existing online databases. `getblast` parses NCBI BLAST reports, including BLASTN, BLASTP, BLASTX, TBLASTN, TBLASTX and psi-BLAST.

`Data = getblast(`*`RID`*`)` reads a BLAST Request ID (*RID*) and returns the report data in a structure (*Data*). The NCBI Request ID (*RID*) must be a recently generated report because NCBI purges reports after 24 hours.

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

`getblast(..., 'Descriptions', DescriptionsValue)` includes the specified number of descriptions (`DescriptionsValue`) in the report.

`getblast(..., 'Alignments', AlignmentsValue)` includes the specified number of alignments in the report.

`getblast(..., 'ToFile', ToFileValue)` saves the data returned from the NCBI BLAST report to a file (`ToFileValue`). The default format for the file is text, but you can specify HTML with the property `FileFormat`.

`getblast(..., 'FileFormat', FileFormatValue)` returns the report in the specified format (`FileFormatValue`).

For more information about reading and interpreting BLAST reports, see

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

Example

1 Run a BLAST search with an NCBI accession number.

```
RID = blastncbi('AAA59174','blastp','expect',1e-10)
```

2 Pass the RID to GETBLAST to parse the report, load it into a MATLAB structure, and save a copy as a text file.

```
report = getblast(RID,'TOFILE','Report.txt')
```

See Also

Bioinformatics Toolbox functions `blastncbi`, `blastread`

getbyname (phytree)

Purpose

Select branches and leaves from a phytree object

Syntax

```
S = getbyname(Tree, Expression)
S = getbyname(Tree, String,
'Exact', true)
```

Arguments

Tree	Phytree object created with the function `phytree`.
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	Char string or cell array of char 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));
```

See Also

Bioinformatics Toolbox phytree and `phytree` object methods prune, select, get

getcanonical (phytree)

Purpose

Calculate the canonical form of a phylogenetic tree

Syntax

```
Pointers = getcanonical(Tree)
[Pointers, Distances, Names] = getcanonical(Tree)
```

Arguments

Tree	Phytree object created with the function `phytree`.

Description

`Pointers` `= getcanonical(`*`Tree`*`)` returns the pointers for the canonical form of a phylogenetic tree (*`Tree`*). In a canonical tree the leaves are ordered alphabetically and the branches are ordered first by their width and then alphabetically by their first element. A canonical tree is isomorphic to all the trees with the same skeleton independently of the order of their leaves and branches.

`[`*`Pointers, Distances, Names`*`] = getcanonical(`*`Tree`*`)` returns, in addition to the pointers described above, the reordered distances (*`Distances`*) and node names (*`Names`*).

Examples

1 Create two phylogenetic trees with the same skeleton but slightly different distances.

```
b = [1 2; 3 4; 5 6; 7 8;9 10];
tr_1 = phytree(b,[.1 .2 .3 .3 .4 ]');
tr_2 = phytree(b,[.2 .1 .2 .3 .4 ]');
```

2 Plot the trees.

```
plot(tr_1)
plot(tr_2)
```

3 Check whether the trees have an isomorphic construction.

```
isequal(getcanonical(tr_1),getcanonical(tr_2))
```

```
ans =
     1
```

See Also

Bioinformatics Toolbox functions `phytreeread`, and phytree object methods `getbyname`, `select`, `subtree`, `phytree`.

getdescendants (biograph)

Purpose

Find descendants in a 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`).

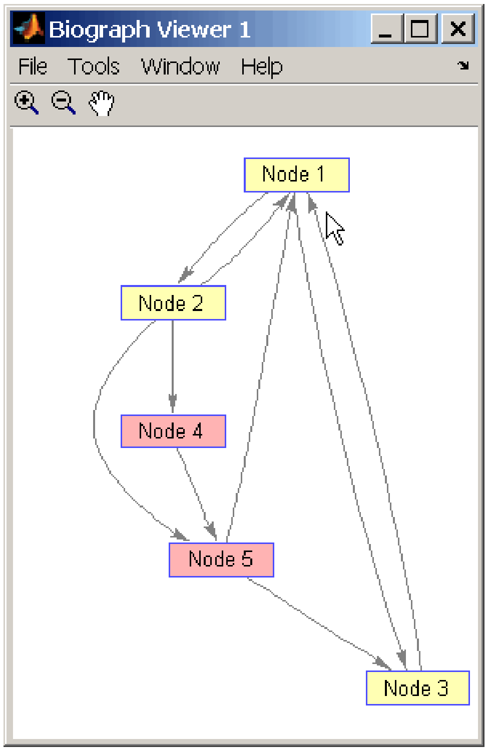
Examples

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)
```

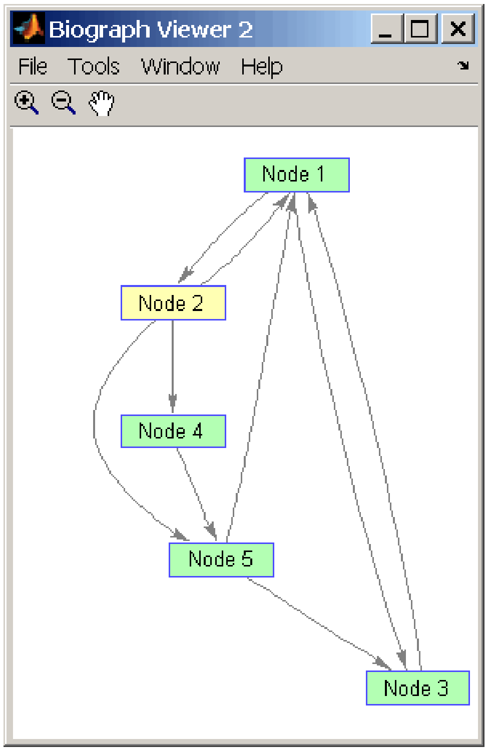
2 Find one generation of descendants for node 4.

```
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 methods for the biograph object, `biograph`, `dolayout`, `getancestors`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `view`

MATLAB functions `get`, `set`

Purpose

Get handles to edges in graph

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 edge handles that connect the specified source nodes (`SourceIDs`) to the specified sink nodes (`SinkIDs`).

Example

1 Create a biograph object for the Hominidae family.

```
species = {'Homosapiens','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 `Homosapiens` node.

```
EdgesIn = getedgesbynodeid(bg,[],'Homo');
EdgesOut = getedgesbynodeid(bg,'Homo');
set(EdgesIn,'LineColor',[0 1 0]);
set(EdgesOut,'LineColor',[1 0 0]);
bg.view;
```

3 Find all edges that connect members of the Cercopithecidae family to members of the Hominidae family.

```
Cercopithecidae = {'Macaca','Baboon'};
Hominidae = {'Homo','Pan','Gorilla','Pongo'};
edgesSel = getedgesbynodeid(bg,Cercopithecidae,Hominidae);
set(bg.edges,'LineColor',[.5 .5 .5]);
set(edgesSel,'LineColor',[0 0 1]);
```

```
bg.view;
```

See Also

Bioinformatics Toolbox methods for the biograph object, `biograph`, `dolayout`, `getancestors`, `getdescendants`, `getnodesbyid`, `getrelatives`, `view`

MATLAB functions `get`, `set`

Purpose

Retrieve sequence information from EMBL database

Syntax

```
Data = getembl('AccessionNumber',
               'PropertyName', PropertyValue...)

getembl(..., 'ToFile', ToFileValue)
getembl(..., 'SequenceOnly', SequenceOnlyValue)
```

Arguments

AccessionNumber	Unique identifier for a sequence record. Enter a unique combination of letters and numbers
`ToFile`	Property to specify the location and filename for saving data. Enter either a filename or a path and filename supported by your system (ASCII text file).
`SequenceOnly`	Property to control getting a sequence without the metadata. Enter `true` or `false`.

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`*`', '`*`PropertyName`*`',` *`PropertyValue`*`...)` searches for the accession number in the EMBL database (`http://www.ebi.ac.uk/embl`) and returns a MATLAB structure containing the following fields:

```
Comments
Identification
Accession
SequenceVersion
DateCreated
DateUpdated
```

```
Description
Keyword
OrganismSpecies
OrganismClassification
Organelle
Reference
DatabaseCrossReference
Feature
BaseCount
Sequence
```

`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 `SequenceOnly` is `true`, returns only 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 filename is given without a path, the file is stored in the current directory.

```
Seq = getembl('X00558','ToFile','c:\project\rat_protein.txt')
```

Retrieve only the sequence for the rat liver apolipoprotein.

```
Seq = getembl('X00558','SequenceOnly',true)
```

See Also

Bioinformatics Toolbox functions `emblread`, `getgenbank`, `getgenpept`, `getpdb`, `getpir`, `seqtool`

Purpose

Retrieve sequence information from GenBank database

Syntax

```
Data = getgenbank('AccessionNumber',
                  '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.
`ToFile`	Property to specify the location and filename for saving data. Enter either a filename or a path and filename supported by your system (ASCII text file).
`FileFormat`	Property to select the format for the file specified with the property ToFileValue. Enter either `'GenBank'` or `'FASTA'`.
`SequenceOnly`	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`*`', '`*`PropertyName`*`',` *`PropertyValue`*`...)` 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(..., '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.

getgenbank(...) 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.

Examples

Retrieve the sequence from chromosome 19 that codes for the human insulin receptor and store it in structure S.

```
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'
                 Keywords: 'insulin receptor; tyrosine kinase.'
                  Segment: []
                   Source: 'Homo sapiens (human)'
           SourceOrganism: [3x65 char]
                Reference: {[1x1 struct]}
                  Comment: [14x67 char]
                 Features: [51x74 char]
                      CDS: [139 4287]
                 Sequence: [1x4723 char]
                SearchURL: [1x105 char]
              RetrieveURL: [1x95 char]
```

See Also

Bioinformatics Toolbox functions featuresparse,genbankread, getembl, getgenpept, getpdb, getpir, seqtool

Purpose

Retrieve sequence information from GenPept database

Syntax

```
Data = getgenpept('AccessionNumber',
                  'PropertyName', PropertyValue...)

getgenpept(..., 'ToFile', ToFileValue)
getgenpept(..., 'SequenceOnly', SequenceOnlyValue)
```

Arguments

AccessionNumber	Unique identifier for a sequence record. Enter a combination of letters and numbers.
`ToFile`	Property to specify the location and filename for saving data. Enter either a filename or a path and filename supported by your system (ASCII text file).
`FileFormat`	Property to select the format for the file specified with the property *ToFileValue*. Enter either `'GenBank'` or `'FASTA'`.
`SequenceOnly`	Property to control getting the sequence only. Enter either `true` or `false`.

Description

`getgenpept` retrieves a protein (amino acid) sequence and sequence information from the database `GenPept`. 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 recently changed the name of their protein search engine from GenPept to Entrez Protein. However, the function names in the Bioinformatics Toolbox (getgenpept, 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`*`', '`*`PropertyName`*`',`*`PropertyValue...`*`)` 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(..., '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 `SequenceOnly` is `true`. When the properties `SequenceOnly` and `ToFile` are used together, the output file is in the FASTA format.

`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.

Examples

Retrieve the sequence for the human insulin receptor and store it in structure `Seq`.

```
Seq = getgenpept('AAA59174')
```

See Also

Bioinformatics Toolbox functions `featuresparse`,`genpeptread`, `getembl`, `getgenbank`, `getpdb`, `getpir`

getgeodata

Purpose

Get Gene Expression Omnibus (GEO) data

Syntax

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

getgeodata(..., 'ToFile', ToFileValue)
```

Arguments

AccessionNumber	Unique identifier for a sequence record. Enter a combination of letters and numbers.
`ToFile`	Property to specify the location and filename for saving data. Enter either a filename, or a path and filename supported by your system (ASCII text file).

Description

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

```
Scope
Accession
Header
ColumnDescriptions
ColumnNames
Data
```

`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`.

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`

gethmmalignment

Purpose Retrieve multiple aligned sequences from the PFAM database

Syntax ‘

```
AlignData = gethmmalignment('PFAMKey',
                             'PropertyName', PropertyValue...)

gethmmalignment(..., 'ToFile', ToFileValue)
gethmmalignment(..., 'Type', TypeValue)
```

Arguments

PFAMKey	Unique identifier for a sequence record. Enter a unique combination of letters and numbers.
ToFile	Property to specify the location and filename for saving data. Enter either a filename, or a path and filename supported by your system (ASCII text file).
Type	Property to select the set of alignments returned. Enter either `'seed'` or `'full'`.

Description `AlignData = gethmmalignment('`*`PFAMKey`*`', '`*`PropertyName`*`',`*`PropertyValue`*`...)` retrieves multiple aligned sequences from a profile hidden Markov model stored in the PFAM database and returns a MATLAB structure containing the following fields:

```
Header
Sequence
```

`gethmmalignment(..., 'ToFile',` *`ToFileValue`*`)` saves the data returned from the PFAM database to a file. Read a FASTA formatted file with PFAM data back into MATLAB using the function `fastaread`.

`gethmmalignment(..., 'Type',` *`TypeValue`*`)` returns only the alignments used to generate the HMM model if `Type='seed'`, and if

`Type='full'`, returns all alignments that fit the model. Default is `'full'`.

Examples

Retrieve a multiple alignment of the sequences used to train the HMM profile model for global alignment to the 7 transmembrane receptor protein in the secretin family (`PFAMKey = PF00002`).

```
pfamalign = gethmmalignment(2,'Type','seed')
```

or

```
pfamalign = gethmmalignment('PF00002','Type','seed')
```

See Also

Bioinformatics Toolbox function `fastaread`, `gethmmprof`, `gethmmtree`, `pfamhmmread`, `multialignread`

gethmmprof

Purpose

Retrieve profile hidden Markov models from the PFAM database

Syntax

```
Model = gethmmprof('AccessionNumber',
                   'PropertyName', PropertyValue...)

gethmmprof(..., 'ToFile', ToFileValue)
gethmmprof(..., 'Mode', ModeValue)
```

Arguments

AccessionNumber	Unique identifier for a sequence record. Enter a unique combination of letters and numbers.
`ToFile`	Property to specify the location and filename for saving data. Enter either a filename or a path and filename supported by your system (ASCII text file).
`Mode`	Property to select returning the global or local alignment mode. Enter either `'ls'` for the global alignment mode or `'fs'` for the local alignment mode. Default value is `'ls'`.

Description

`Model = gethmmprof('`*`AccessionNumber`*`',` `'`*`PropertyName`*`',`*`PropertyValue...`*`)` searches for the PFAM family accession number in the PFAM database and returns a MATLAB structure containing the following fields:

```
Name
PfamAccessionNumber
ModelDescription
ModelLength
Alphabet
MatchEmission
InsertEmission
NullEmission
BeginX
MatchX
```

```
InsertX
DeleteX
FlankingInsertX
```

`gethmmprof(..., 'ToFile', ToFileValue)` saves data returned from the PFAM database in a file (`ToFileValue`). Read an `hmmprof` formatted file back into MATLAB using the function `pfamhmmread`.

`gethmmprof(..., 'Mode', ModeValue)` selects either the global alignment model or the local alignment model.

Examples

Retrieve a HMM profile model for global alignment to the 7-transmembrane receptor protein in the secretin family. (`PFAM key = PF00002`)

```
hmmmodel  = gethmmprof(2)
```

or

```
hmmmodel  = gethmmprof('PF00002')
```

See Also

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

Purpose

Get phylogenetic tree data from PFAM database

Syntax

```
Tree = gethmmtree(AccessionNumber)

Tree = gethmmtree(...,'ToFile',ToFileValue)
Tree = gethmmtree(...,'Type', TypeValue)
```

Arguments

AccessionNumber	Accession number in the PFAM database.
ToFile	Property to specify the location and filename for saving data. Enter either a filename or a path and filename supported by your system (ASCII text file).
Type	Property to control which alignments are included in the tree. Enter either `'seed'` or `'full'`. The default value is `'full'`.

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(...,'ToFile',` *`ToFileValue`*`)` saves the data returned from the PFAM database in the file `ToFileValue`.

`Tree = gethmmtree(...,'Type',` *`TypeValue`*`)`, when `Type` is `'seed'`, returns a tree with only the alignments used to generate the HMM model. When `Type` is `'full'`, 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`

getnewickstr (phytree)

Purpose

Create Newick formatted string

Syntax

```
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

`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 `phytreeread`, `phytreetool`, `phytreewrite`, `seqlinkage`, and phytree object methods `get`, `getbyname`, `getcanonical`, `phytree`.

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 node handles for the specified nodes (`NodeIDs`).

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 methods for the biograph object, `biograph`, `dolayout`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getrelatives`, `view`

MATLAB functions `get`, `set`

Purpose

Retrieve protein structure data from PDB database

Syntax

```
Data = getpdb('PDBid',
              'PropertyName', PropertyValue...)

getpdb(..., 'ToFile', ToFileValue)
getpdb(..., 'MirrorSite', MirrorSiteValue)
```

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.
`ToFile`	Property to specify the location and filename for saving data. Enter either a filename or a path and filename supported by your system (ASCII text file).
`MirrorSite`	Property to select Web site. Enter either `http://rutgers.rcsb.org/pdb` to use the Rutgers University Web site, or enter `http://nist.rcsb.org/pdb` for the National Institute of Standards and Technology site.

Description

`getpdb` retrieves sequence information from the Protein Data Bank. This database contains 3-D biological macromolecular structure data.

`Data = getpdb('`*`PDBid`*`', '`*`PropertyName`*`',`*`PropertyValue...`*`)` searches for the ID in the PDB database and returns a MATLAB structure containing the following fields:

```
Header
Title
Compound
Source
Keywords
```

```
ExperimentData
Authors
Journal
Remark1
Remark2
Remark3
Sequence
HeterogenName
HeterogenSynonym
Formula
Site
Atom
RevisionDate
Superseded
Remark4
Remark5
Heterogen
Helix
Turn
Cryst1
OriginX
Scale
Terminal
HeterogenAtom
Connectivity
```

`getpdb(..., 'ToFile', ToFileValue)` saves the data returned from the database to a file. Read a PDB formatted file back into MATLAB using the function `pdbread`.

`getpdb(...,'MirrorSite', MirrorSiteValue)` allows you to choose a mirror site for the PDB database. The default site is the San Diego Supercomputer Center, `http://www.rcsb.org/pdb`. See `http://www.rcsb.org/pdb/mirrors.html` for a full list of PDB mirror sites.

Examples

Retrieve the structure information for the electron transport (heme protein) with PDB ID `5CYT`.

```
pdbstruct = getpdb('5CYT')
```

See Also

Bioinformatics Toolbox functions `getembl`, `getgenbank`, `getgenpept`, `getpir`, `pdbdistplot`, `pdbplot`, `pdbread`

getpir

Purpose

Retrieve sequence data from PIR-PSD database

Syntax

```
Data = getpir('AccessionNumber',
              'PropertyName', PropertyValue...)

getpir(..., 'ToFile', ToFileValue)
getpir(..., 'SequenceOnly', SequenceOnlyValue)
```

Arguments

AccessionNumber	Unique identifier for a sequence record. Enter a unique combination of letters and numbers.
`ToFile`	Property to specify the location and filename for saving data. Enter either a filename or a path and filename supported by your system.
`SequenceOnly`	Property to control getting the sequence only. Enter either `true` or `false`.

Description

`Data = getpir('`*`AccessionNumber`*`',` `'`*`PropertyName`*`',`*`PropertyValue`*`...)` searches for the accession number in the PIR-PSD database, and returns a MATLAB structure containing the following fields:

```
Entry
EntryType
Title
Organism
Date
Accessions
Reference
Genetics
Classification
Keywords
Feature
Summary
Sequence
```

`getpir(..., 'ToFile', ToFileValue)` saves the data retrieved from the PIR-PSD database in a file. Read a PIR-PSD formatted file back into MATLAB using the function `pirread`.

`getpir(..., 'SequenceOnly', SequenceOnlyValue)` returns only the sequence information for the protein as a string if `SequenceOnly` is `true`.

The Protein Sequence Database (PIR-PSD) is maintained by the Protein Information Resource (PIR) division of the National Biomedical Research Foundation (NBRF), which is affiliated with Georgetown University Medical Center.

Examples

Return a structure, `pirdata`, that holds the result of a query into the PIR-PSD database using `'cchu'` as the search string.

```
pirdata = getpir('cchu')

pirdata =
             Entry: 'CCHU'
         EntryType: 'complete'
             Title: 'cytochrome c [validated] - human'
          Organism: [1x1 struct]
              Date: [1x1 struct]
        Accessions: 'A31764; A05676; I55192; A00001'
         Reference: {[1x1 struct]  [1x1 struct]  [1x1 struct]
                     [1x1 struct]}
          Genetics: {[1x1 struct]}
    Classification: [1x1 struct]
          Keywords: [1x157 char]
           Feature: {1x5 cell}
           Summary: [1x1 struct]
          Sequence: [1x105 char]
```

Return a string, `pirdata`, that holds the sequence information for the query `'cchu'` in the PIR-PSD database.

```
pirseq = getpir('cchu','SequenceOnly',true)
```

Return a structure, pirdata, that holds the result of a query into the PIR database using 'cchu' as the search string. It also creates a text file, cchu.pir, in the current folder that holds the data retrieved from the PIR database. Note that the entire data retrieved from the database is stored in *ToFileValue* even if SequenceOnly is true.

```
pirdata = getpir('cchu', 'ToFile','cchu.pir')
```

See Also

Bioinformatics Toolbox functions getembl, getgenbank, getgenpept, getpdb, pirread

Purpose

Find relatives in a 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 methods for the biograph object, `biograph`, `dolayout`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `view`

MATLAB functions `get`, `set`

gonnet

Purpose

Return a Gonnet scoring matrix

Syntax

```
gonnet
```

Description

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, Benner S; "Exhaustive matching of the entire protein sequence database" in Science; Vol. 256, pp. 1443-1445; June 1992.

See Also

Bioinformatics Toolbox functions `blosum`,`dayhoff`, `pam`

Purpose

Read microarray data from a GenePix Results (GPR) file

Syntax

```
GPRData = gprread('File',
                  'PropertyName', PropertyValue...)

gprread(..., 'CleanColNames', CleanColNameValue)
```

Arguments

File	GenePix Results formatted file (file extension `GPR`). Enter a filename or a path and filename.
`CleanColNames`	Property to control creating column names that MATLAB can use as variable names.

Description

`GPRData = gprread('File', 'PropertyName', PropertyValue...)` reads GenePix results data from *File* and creates a MATLAB structure `GPRData` with the following fields:

```
Header
Data
Blocks
Columns
Rows
Names
IDs
ColumnNames
Indices
Shape
```

`gprread(..., 'CleanColNames', CleanColNamesValue)`. A `GPR` file may contain column names with spaces and some characters that MATLAB cannot use in MATLAB variable names. If `CleanColNames` is `true`, `gprread` returns `ColumnNames` that are valid MATLAB variable names and names that you can use in functions. By default, `CleanColNames` is `false` and `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.axon.com/GN_GenePix_File_Formats.html
```

For a list of supported file format versions, see

```
http://www.axon.com/gn_GPR_Format_History.html
```

Sample data can be found at the following Web address. Save this file to your working directory to run the example below.

```
http://www.axon.com/genomics/Demo.gpr
```

GenePix is a registered trademark of Axon Instruments, Inc.

Examples

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

% Alternatively, create a similar plot using
% more basic graphics commands.

f635Col = find(strcmp(gprStruct.ColumnNames,'F635 Median'));
F635Median = gprStruct.Data(:,f635Col);
 imagesc(F635Median(gprStruct.Indices));
colormap bone
colorbar
```

See Also

Bioinformatics Toolbox functions `affyread`, `galread`, `geosoftread`, `imageneread`, `sptread`

Purpose

Align a query sequence to a profile using hidden Markov model based alignment

Syntax

```
Alignment = hmmprofalign(Model, Seq,
                              'PropertyName', PropertyValue...)
[Alignment, Score] = hmmprofalign(Model, Seq)

hmmprofalign(..., 'ShowScore', ShowScoreValue)
hmmprofalign(..., 'Flanks', FlanksValue)
hmmprofalign(..., 'ScoreFlanks', ScoreFlanksValue)
hmmprofalign(..., 'ScoreNullTransitions',
ScoreNullTransValue)
```

Arguments

`Model`	Hidden Markov model created with the function `hmmprofstruc`.
`Seq`	Amino acid or nucleotide sequence. You can also enter a structure with the field `Sequence`.
`ShowScore`	Property to control displaying the scoring space and the winning path. Enter either `true` or `falase`. The default value is `false`.
`Flanks`	Property to control including the symbols generated by the FLANKING INSERT states in the output sequence. Enter either `true` or `false`. The default value is `false`.
`ScoreFlanks`	Property to control including the transition probabilities for the flanking states in the raw score. Enter either `true` or `false`. Default value is `false`.
`ScoreNullTrans`	Property to control adjusting the raw score using the null model for transitions (`Model.NullX`). Enter either `true` or `false`. The default value is `false`.

Description

`Alignment = hmmprofalign(Model, Seq, 'PropertyName', PropertyValue...)` 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(..., 'ShowScore', ShowScoreValue)`,when `ShowScore` is `true`, displays the scoring space and the winning path.

`hmmprofalign(..., 'Flanks', FlanksValue)`, when `Flanks` is `true`, includes the symbols generated by the FLANKING INSERT states in the output sequence.

`hmmprofalign(..., 'ScoreFlanks', ScoreFlanksValue)`, when `ScoreFlanks` is true, includes the transition probabilities for the flanking states in the raw score.

`hmmprofalign(..., 'ScoreNullTransitions', ScoreNullTransitionValue)`, when `ScoreNullTransitions` is `true`, adjusts the raw score using the null model for transitions (`Model.NullX`).

Note Multiple 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`, `hmmprofmerge`, `hmmprofstruct`, `pfamhmmread`, `showhmmprof`, `multialign`, `profalign`

hmmprofestimate

Purpose

Estimate profile HMM parameters using pseudocounts

Syntax

```
hmmprofestimate(Model, MultipleAlignment,
                'PropertyName', PropertyValue...)

hmmprofestimate(..., 'A', AValue)
hmmprofestimate(..., 'Ax', AxValue)
hmmprofestimate(..., 'BE', BEValue)
hmmprofestimate(..., 'BDx', BDxValue)
```

Arguments

`Model`	Hidden Markov model created with the function `hmmprofstruc`.
`MultipleAlignment`	Array of sequences. `Sequences` can also be a structured array with the aligned sequences in a field `Aligned` or `Sequences`, and the optional names in a field `Header` or `Name`.
`A`	Property to set the pseudocount weight A. Default value is `20`.
`Ax`	Property to set the pseudocount weight Ax. Default value is 20.
`BE`	Property to set the background symbol emission probabilities. Default values are taken from `Model.NullEmission`.
`BMx`	Property to set the background transition probabilities from any MATCH state (`[M->M M->I M->D]`). Default values are taken from `hmmprofstruct`.
`BDx`	Property to set the background transition probabilities from any DELETE state (`[D->M D->D]`). Default values are taken from `hmmprofstruct`.

Description

`hmmprofestimate(Model, MultipleAlignment, 'PropertyName', PropertyValue...)` returns a structure with the fields containing the updated estimated parameters of a profile HMM. Symbol emission and state transition probabilities are estimated using the real counts and weighted pseudocounts obtained with the background probabilities. Default weight is `A=20`, the default background symbol emission for match and insert states is taken from `Model.NullEmission`, and the default background transition probabilities are the same as default transition probabilities returned by `hmmprofstruct`.

Model Construction: Multiple aligned sequences should contain uppercase letters and dashes indicating the model MATCH and DELETE states agreeing with `Model.ModelLength`. If model state annotation is missing, but `MultipleAlignment` is space aligned, then a "maximum entropy" criteria is used to select `Model.ModelLength` states.

Note: Insert and flank insert transition probabilities are not estimated, but can be modified afterwards using `hmmprofstruct`.

`hmmprofestimate(..., 'A', AValue)` sets the pseudocount weight `A = Avalue` when estimating the symbol emission probabilities. Default value is `20`.

`hmmprofestimate(...,'Ax', AxValue)` sets the pseudocount weight `Ax = Axvalue` when estimating the transition probabilities. Default value is 20.

`hmmprofestimate(...,'BE', BEValue)` sets the background symbol emission probabilities. Default values are taken from `Model.NullEmission`.

`hmmprofestimate(...,'BMx', BMxValue)` sets the background transition probabilities from any MATCH state (`[M->M M->I M->D]`). Default values are taken from `hmmprofstruct`.

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

See Also

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

Purpose Generate a random sequence drawn from the profile HMM

Syntax

```
Sequence = hmmprofgenerate(Model,
                           'PropertyName', PropertyValue....)
[Sequence, Profptr] = hmmprofgenerage(Model)

hmmprofgenerate(..., 'Align', AlignValue)
hmmprofgenerate(..., 'Flanks', FlanksValue)
hmmprofgenerate(..., 'Signature', SignatureValue)
```

Arguments

`Model`	Hidden Markov model created with the function `hmmprofstruc`.
`Align`	Property to control using uppercase letters for matches and lowercase letters for inserted letters. Enter either `true` or `false`. The default value is `false`.
`Flanks`	Property to control including the symbols generated by the FLANKING INSERT states in the output sequence. Enter either `true` or `false`. The default value is `false`.
`Signature`	Property to control returning the most likely path and symbols. Enter either `true` or `false`. Default value is `false`.

Description

`Seq = hmmprofgenerate(Model, 'PropertyName', PropertyValue...)` returns a string (`Seq`) 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] = hmmprofgenerage(Model)` returns a vector of the same length as the profile model pointing to the respective states in the output sequence. Null pointers (0) mean that such states do not exist in the output sequence, either because they are never touched (i.e., jumps from the BEGIN state to MATCH states or from MATCH states to the END state), or because DELETE states are not in the output sequence (not aligned output; see below).

`hmmprofgenerate(..., '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 `Align` is `false`, the output is a sequence of uppercase symbols. The default value is `true`.

`hmmprofgenerate(..., 'Flanks', *FlanksValue*)` if `Flanks` is `true`, the output sequence includes the symbols generated by the FLANKING INSERT states. The default value is `false`.

`hmmprofgenerate(..., 'Signature', *SignatureValue*)` if `Signature` 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 the prealigned strings of several sequences to a profile HMM

Syntax

```
A = 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 a HMM model profile. The output is aligned corresponding to the HMM states.

- Match states — Uppercase letters
- Insert states — Lowercase letters or asterisks (*)
- Delete states — Dashes

Periods (.) are added at positions corresponding to inserts in other sequences. The input sequences must have the same number of profile states, that is, the joint count of capital letters and dashes must be the same.

`hmmprofmerge(Sequences, Names)` labels the sequences with `Names`.

`hmmprofmerge(Sequences, Names, Scores)` sorts the displayed sequences using `Scores`.

Examples

```
load('hmm_model_examples','model_7tm_2')  %load model
load('hmm_model_examples','sequences')  %load sequences

for ind =1:length(sequences)
    [scores(ind),sequences(ind).Aligned] =...
        hmmprofalign(model_7tm_2,sequences(ind).Sequence);
    end
hmmprofmerge(sequences, scores)
```

See Also

Bioinformatics Toolbox functions `hmmprofalign`, `hmmprofstruct`

Purpose

Create a profile 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 hmmprofstruc.
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`'.

`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]`.

`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)`.

`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].`

`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)

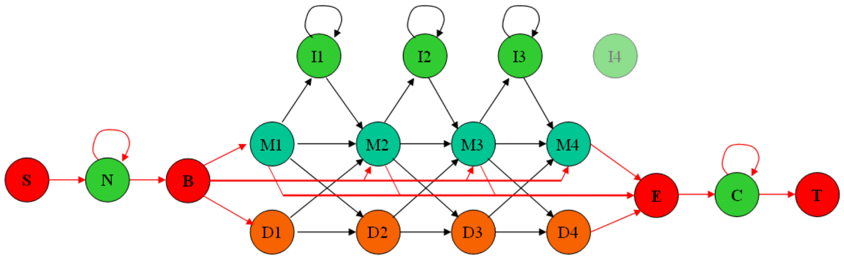
Name	Model Name
`IDNumber`	Identification Number
`Description`	Short description of the model

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 figure shown below is a state diagram for a HMM profile of length 4. Insert, match, and delete states are in the regular part (middle section).

- 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 know as a silent state because it does not emit any symbol),
- Insert state represents the excess of one or more symbols in the target sequence that are not included in the profile.

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 `gethmmprof`, `hmmprofalign`, `hmmprofestimate`, `hmmprofgenerate`, `hmmprofmerge`, `pfamhmmread`, `showhmmprof`, `aacount`, `basecount`

Purpose

Read microarray data from an ImaGene Results file

Syntax

```
GPRData = gprread('File',
                  'PropertyName', PropertyValue...)

gprread(..., 'CleanColNames', CleanColNamesValue)
```

Arguments

File	ImaGene Results formatted file Enter a filename or a path and filename.
`CleanColName`	Property to control creating column names that MATLAB can use as variable names.

Description

`imagedata = imagegeenread(`*`File`*`, '`*`PropertyName`*`',` *`PropertyValue`*`...)` reads ImaGene results data from *`File`* and creates a MATLAB structure `imagedata` containing the following fields:

```
HeaderAA
Data
Blocks
Rows
Columns
Fields
IDs
ColumnNames
Indices
Shape
```

`imageneread(..., 'CleanColNames',` *`CleanColNamesValue`*`)`. An ImaGene file may contain column names with spaces and some characters that MATLAB cannot use in MATLAB variable names. If `CleanColNames` is `true`, `imagene` returns `ColumnNames` that are valid MATLAB variable names and names that you can use in functions. By default, `CleanColNames` is `false` and `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

```
% Read in a sample ImaGene file and plot the Signal Mean
cy3Data = imageneread('cy3.txt');
maimage(cy3Data,'Signal Mean');

% Read in the Cy5 channel and create a loglog plot of Signal Median
cy5Data = imageneread('cy5.txt');
sigMedianCol = find(strcmp('Signal Median',cy3Data.ColumnNames));
cy3Median = cy3Data.Data(:,sigMedianCol);
cy5Median = cy5Data.Data(:,sigMedianCol);
maloglog(cy3Median,cy5Median,'title','Signal Median');
```

See Also

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

Purpose

Convert amino acid sequence from integer to letter representation

Syntax

```
SeqChar = int2aa(SeqInt,
                    'PropertyName', PropertyValue...)

int2aa(..., 'Case', CaseValue)
```

Arguments

SeqInt	Amino acid sequence represented with integers. Enter a vector of integers from the table Mapping Amino Acid 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.
`Case`	Property to select the case of the returned character string. Enter either `'upper'` or `'lower'`. Default is `'upper'`.

Mapping Amino Acid Integers to Letters

Amino Acid	Code	Amino Acid	Code	Amino Acid	
Alanine	`A1`	Isoleucine	`I10`	Tyrosine	`Y19`
Arginine	`R2`	Leucine	`L11`	Valine	`V20`
Asparagine	`N3`	Lysine	`K12`	Aspartic acid or Asparagine	`B21`
Aspartic acid (aspartate)	`D4`	Methionine	`M13`	Glutamic acid or Glutamine	`Z22`
Cystine	`C5`	Phenylalanine	`F14`	Any amino acid	`X23`

Amino Acid	Code	Amino Acid	Code	Amino Acid	
Glutamine	Q6	Proline	P15	Translation stop	*24
Glutamic acid (glutamate)	E7	Serine	S16	Gap of indeterminate length	- 25
Glycine	G8	Threonine	T17	Unknown or any integer not in table	?0
Histidine	H9	Tryptophan	W18		

Description

`SeqChar = int2aa(SeqInt, 'PropertyName', PropertyValue...)` converts a 1-by-N array of integers to a character string using the table Mapping Amino Acid Interger sot Letters above.

`int2aa(..., 'Case', CaseValue)` sets the output case of the nucleotide string. Default is uppercase.

Examples

```
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

```
SeqChar = int2nt(SeqInt,
                 'PropertyName', PropertyValue...)

int2nt(..., 'Alphabet', AlphabetValue)
int2nt(..., 'Unknown', UnknownValue)
int2nt(..., 'Case', CaseValue)
```

Arguments

`SeqInt`	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.
`Alphabet`	Property to select the nucleotide alphabet. Enter either `'DNA'` or `'RNA'`.
`Unknown`	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`, `T`, `C`, `G` or the ambiguous nucleotide characters `N`, `R`, `Y`, `K`, `M`, `S`, `W`, `B`, `D`, `H`, or `V`. The default character is `*`.
`Case`	Property to select the letter case for the nucleotide sequence. Enter either `'upper'` or `'lower'`. The default value is `'lower'`.

Mapping Nucleotide Integers to Letters

Base	Code	Base	Code	Base	Code
Adenosine	1—A	T, C (pyrimidine)	6—Y	A, T, G (not C)	12—D
Cytidine	2—C	G, T (keto)	7—K	A, T, C (not G)	13—H
Guanine	3—G	A, C (amino)	8—M	A, G, C (not T)	14—V
Thymidine	4—T	G, C (strong)	9—S	A, T, G, C (any)	15—N
Uridine (if 'Alphabet' = 'RNA'	4—U	A, T (weak)	10—W	Gap of indeterminate length	16 — -
A, G (purine)	5—R	T, G, C (not A)	11—B	Unknown (default)	0 and 17—*

Description

`int2nt(SeqNT, 'PropertyName', PropertyValue...)` converts a `1`-by-`N` array of integers to a character string using the table Mapping Nucleotide Letters to Integers above.

`int2nt(..., 'Alphabet', AlphabetValue)` defines the nucleotide alphabet to use. The default value is `'DNA'`, which uses the symbols `A`, `T`, `C`, and `G`. If `Alphabet` is set to `'RNA'`, the symbols `A`, `C`, `U`, `G` are used instead.

`int2nt(..., 'Unknown', UnknownValue)` defines the character to represent an unknown nucleotide base. The default character is `'*'`.

`int2nt(..., 'Case', CaseValue)` sets the output case of the nucleotide string. The default is uppercase.

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 function `aa2int`,, `int2aa`, `nt2int`

Purpose

Estimate isoelectric point for amino acid sequence

Syntax

```
pI = isoelectric(SeqAA,)
                  'PropertyName', PropertyValue...)
[pI Charge] = isoelectric(SeqAA)

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]`.
`PKVals`	Property to provide alternative pK values.
`Charge`	Property to select a specific pH for estimating charge. Enter a number between `0` and `14`. The default value is `7.2`.
`Chart`	Property to control plotting a graph of charge versus pH. Enter `true` or `false`.

Description

`isoelectric` estimates the isoelectric point (the pH at which the protein has a net charge of zero) for an amino acid sequence and it estimates the charge for a given pH (default is pH 7.2). The estimates 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.
  ```

`pI = isoelectric(Seq_AA, 'PropertyName', PropertyValue...)` returns the isoelectric constant (`pI`) for an amino acid sequence.

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

Example

```
% Get a sequence from PDB and estimate the isoelectric point.
pdbSeq = getpdb('1CIV', 'SequenceOnly', true)
% then 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 calculates
% its charge at pH 7.3 (typical blood pH)
gpSeq = getgenpept('AAB39602')
[pI Charge] = isoelectric(gpSeq, 'Charge', 7.38)
```

See Also

Bioinformatics functions `aacount`, `msalign`

Purpose

Read JCAMP-DX formatted files

Syntax

`JCAMPData = jcampread(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.org/index.html
```

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

```
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:

```
XData
YData
XUnits
YUnits
Notes
```

`File` is a JCAMP-DX formatted file (ASCII text file). Enter a filename, a path and filename, 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.

Examples

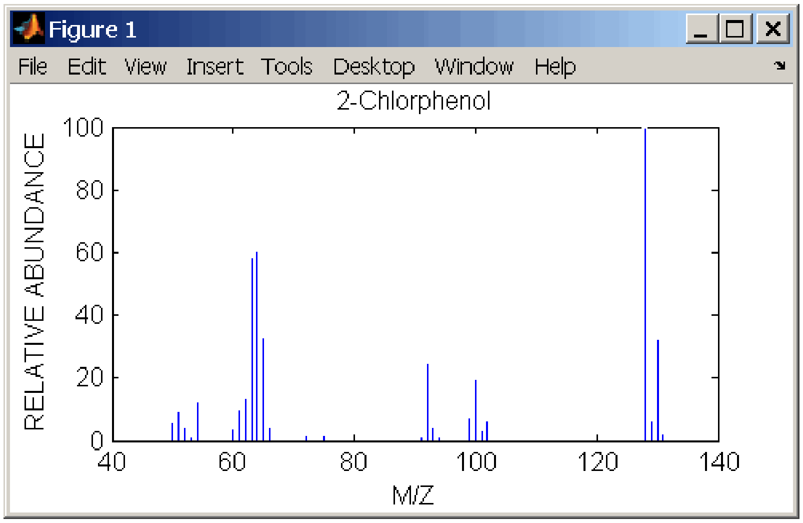
1 Download test data in the file `isa_ms1.dx` from

```
http://www.jcamp.org/testdata.html/testdata.zip
```

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`

Purpose

Join two sequences to produce the shortest supersequence

Syntax

```
SeqNT3 = joinseq(SeqNT1, SeqNT2)
```

Arguments

`SeqNT1, SeqNT2`	Nucleotide sequences.

Description

`joinseq(SeqNT1, SeqNT2)` creates a new sequence that is the shortest supersequence of `Seq1` and `Seq2`. 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`, `paren`, `strcat`, `strfind`

knnclassify

Purpose

Classify data using the 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)
```

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 `NaN`s 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. The default is `1`.

`Class = knnclassify(Sample, Training, Group, k, distance)` enables you to specify the distance metric. The choices for `distance` are

`'euclidean'`	Euclidean distance — the 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. The choices for `rule` are

`'nearest'`	Majority rule with nearest point tie-break — the 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.

Examples

Example 1

The following example classifies the rows of the matrix `sample`:

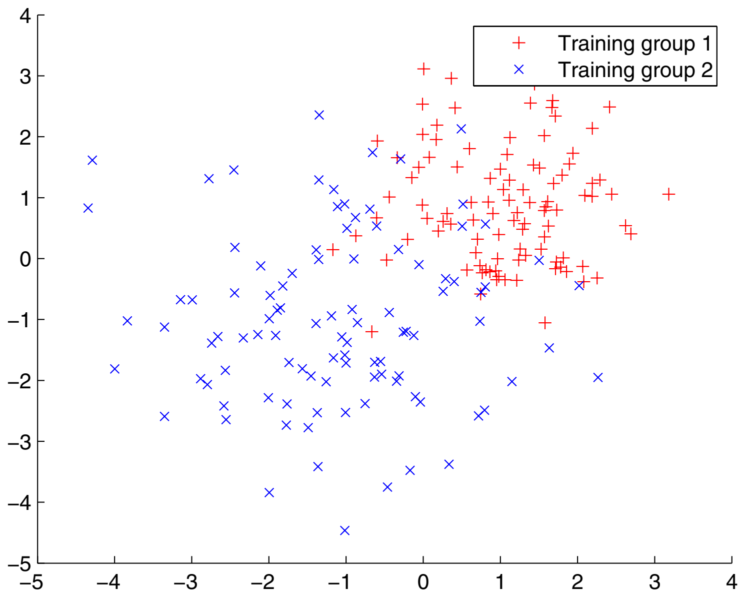
```
sample = [.9 .8;.1 .3;.2 .6]

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`.

Example 2

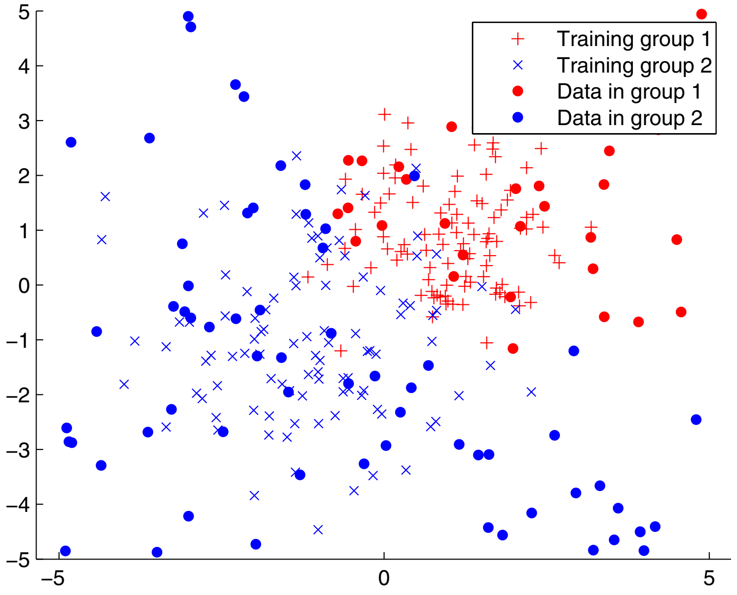
The following example classifies each row of the data in `sample` into one of the two groups in `training`. The following commands create the matrix `training` and the grouping variable `group`, and plot the rows of `training` in two groups.

```
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.

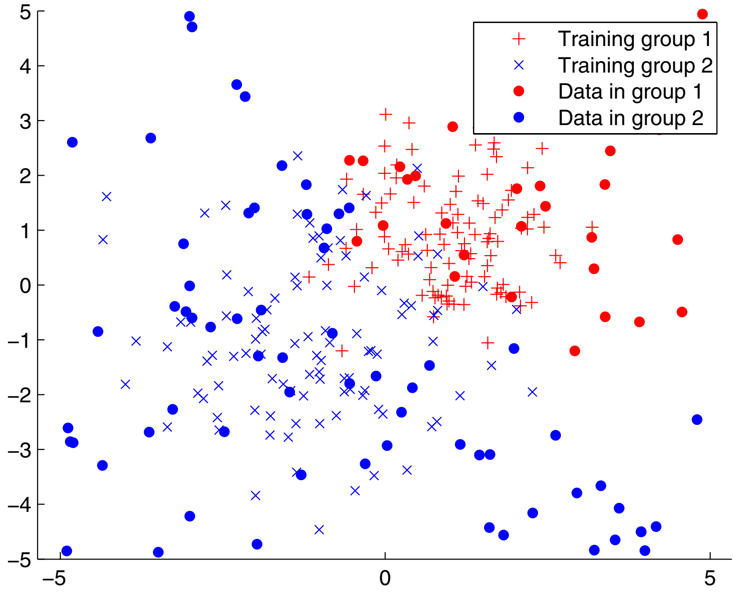
```
sample = unifrnd(-5, 5, 100, 2);
% Classify the sample using the nearest neighbor classification
c = knnclassify(sample, training, group);
gscatter(sample(:,1),sample(:,2),c,'mc'); hold on;
legend('Training group 1','Training group 2', ...
       'Data in group 1','Data in group 2');
hold off;
```



Example 3

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 g
```



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.

See Also

Bioinformatics Toolbox functions ,`knnimputeclassperf` ,`crossvalind` ,`svmclassify` `svmtrain`

Statistical Toolbox functions `classify`

References

[1] Mitchell, Tom,*Machine Learning*, McGraw-Hill, 1997.

knnimpute

Purpose

Impute missing data using the nearest-neighbor method

Syntax

```
knnimpute(Data)
knnimpute(Data, k)
knnimpute(..., 'distance', distfun)
knnimpute(..., 'distargs', args)
knnimpute(...,'weights',w)
knnimpute(...,'median',true)
```

Description

`knnimpute(Data)`replaces `NaN`s 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 `NaN`s 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(..., 'distance', distfun)` computes nearest-neighbor columns using the distance metric `distfun`. The choices for `distfun` are

`'euclidean'`	Euclidean distance — the 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', args)` passes the arguments `args` to the function `distfun`. `args` can be a single value or a cell array of values.

`knnimpute(...,'weights',w)` enables you to specify the weights used in the weighted mean calculation. `w` should be a vector of length `k`.

`knnimpute(...,'median',true)` uses the median of the `k` nearest neighbors instead of the weighted mean.

Examples

Example 1

```
A = [1 2 5;4 5 7;NaN -1 8;7 6 0]

A =

     1     2     5
     4     5     7
   NaN    -1     8
     7     6     0
```

Note that `A(3,1) = NaN`. Because column 2 is the closest column to column 1 in Euclidean distance, `knnimpute` imputes the `(3,1)` entry of column 1 to be the corresponding entry of column 2, which is `-1`.

```
knnimpute(A)

ans =
     1     2     5
     4     5     7
    -1    -1     8
     7     6     0
```

Example 2

The following example loads the data set `yeastdata` and imputes missing values in the array `yeastvalues`.

```
load yeastdata
% Remove data for empty spots
emptySpots = strcmp('EMPTY',genes);
yeastvalues(emptySpots,:) = [];
genes(emptySpots) = [];
% Impute missing values
imputedValues = knnimpute(yeastvalues);
```

See Also

`isnan`, `knnclassify`, `nanmean`, `nanmedian`, `pdist`

References

[1] Speed, T., *Statistical Analysis of Gene Expression Microarray Data*, Chapman & Hall, 2003.

Purpose

Display a box plot for microarray data

Syntax

```
maboxplot(Data, 'PropertyName', PropertyValue...)
maboxplot(Data, ColumnName)
maboxplot(MasStruct, FieldName)

maboxplot(..., 'Title', TitleValue)
maboxplot(..., 'Notch', NotchValue)
maboxplot(..., 'Symbol', SymbolValue)
maboxplot(..., 'Orientation', OrientationValue)
maboxplot(..., 'WhiskerLength', WhiskerLengthValue)

H = maboxplot(...)
[H, HLines] = maboxplot(...)
```

Description

`maboxplot(Data, 'PropertyName', PropertyValue...)` displays a box plot of the values in the columns of `Data`. `Data` can be a numeric array or a structure containing a field called `Data`.

`maboxplot(Data,ColumnName)` labels the box plot column names. For microarray data structures that are block based, `maboxplot` creates a box plot of a given field for each block.

`maboxplot(MasStruct, FieldName)` displays a box plot of field `FieldName` for each block in microarray data structure `MasStruct`.

`maboxplot(..., 'Title', TitleValue)` allows you to specify the title of the plot. The default `Title` is `FieldName`.

`maboxplot(..., 'Notch', NotchValue)` if `Notch` is `true`, draws notched boxes. The default is `false` to show square boxes.

`maboxplot(..., 'Symbol', SymbolValue)` allows you to specify the symbol used for outlier values. The default `Symbol` is `'+'`.

`maboxplot(..., 'Orientation', OrientationValue)` allows you to specify the orientation of the box plot. The choices are `'Vertical'` and `'Horizontal'`. The default is `'Vertical'`.

`maboxplot(..., 'WhiskerLength', WhiskerLengthValue)` allows you to specify the whisker length for the box plot. *`WhiskerLengthValue`* defines the maximum length of the whiskers as a function of the interquartile range (IQR) (default = `1.5`). The whisker extends to the most extreme data value within `WhiskerLength*IQR` of the box. If `WhiskerLength = 0`, then `maboxplot` displays all data values outside the box, using the plotting symbol `Symbol`.

`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.

Examples

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

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

% 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 `maboxplot`, `maimage`, `mairplot`, `maloglog`, , `manorm`

Statistics Toolbox function `boxplot`

Purpose

Display a spatial image for microarray data

Syntax

```
maimage(X, FieldName, 'PropertyName', PropertyValue...)

maimage(..., 'Title', TitleValue)
maimage(..., 'ColorBar', ColorBarValue)
maimage(..., 'HandleGraphicsPropertyName' PropertyValue)
H = maimage(...)
[H, HLines] = maimage(...)
```

Description

`maimage(X, FieldName, 'PropertyName', PropertyValue...)` displays an image of field `FieldName` from microarray data structure `X`. Microarray data can be GenPix Results (GPR) format.

`maimage(..., 'Title', TitleValue)` allows you to specify the title of the plot. The default title is `FieldName`.

`maimage(..., 'ColorBar', ColorBarValue)` if *`ColorBarValue`* is `true`, a colorbar is shown. If *`ColorBarValue`* is `false`, no colorbar is shown. The default is for the colorbar to be shown.

- ColorBarValue — Property to control displaying the colorbar in a figure window. Enter either `true` or `false`. The default value is `false`.

`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')`.

`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.

Examples

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

See Also Bioinformatics Toolbox functions `imagesc`, `maboxplotmairplot`, `maloglog` , `malowess`.

Purpose

Display intensity versus ratio scatter plot for microarray signals

Syntax

```
mairplot(X, Y, 'PropertyName', PropertyValue...)

mairplot(..., 'FactorLines', FactorLinesValue)
mairplot(..., 'Title', TitleValue)
mairplot(..., 'Labels', LabelsValue)
mairmage(..., 'HandleGraphicsPropertyName' PropertyValue)
[Intensity, Ratio] = mairplot(...)
[Intensity, Ratio, H] = mairplot(...)
```

Arguments

`X, Y`	Gene expression data.
`FactorLines`	Property to specify a factor of change.
`Title`	Property to specify a title for the plot.
`Labels`	Property to specify labels for the plot.
`HandleGraphics`	Property to pass optional property name/value pairs from Handle Graphics.

Description

`mairplot(X, Y, 'PropertyName', PropertyValue...)` creates an intensity versus ratio scatter plot of `X` versus `Y`.

`mairplot(..., 'FactorLines', FactorLinesValue)` adds lines showing a factor of `N` change.

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

`mairplot(..., 'Labels', LabelsValue)` allows you to specify 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.

`maimage(..., 'HandleGraphicsPropertyName' PropertyValue)` allows you to pass optional Handle Graphics property name/property value pairs to the function.

`[Intensity, Ratio] = mairplot(...)` returns the intensity and ratio values.

`[Intensity, Ratio, H] = mairplot(...)` returns the handle of the plot.

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
cy3data = maStruct.Data(:,36);
cy5data = maStruct.Data(:,37);
positiveVals = (cy3data>0) & (cy5data>0);
cy3data(~positiveVals) = [];
cy5data(~positiveVals) = [];
mairplot(cy3data,cy5data,'title','R vs G')
figure
names = maStruct.Names(positiveVals);
mairplot(cy3data,cy5data,'FactorLines',2,...
         'Labels',maStruct.Names)
```

See Also

Bioinformatics Toolbox functions `maboxplot`, `maloglog`, , `maimage`, `manorm`

Purpose

Create a loglog plot of microarray data

Syntax

```
maloglog(X, Y, 'PropertyName', PropertyValue...)

maloglog(..., 'FactorLines', FactorLinesValue)
maloglog(..., 'Title', TitleValue)
maloglog(..., 'Labels', LablesValues)
maloglog(..., 'HandleGraphicName', HGValue)
H = maloglog(...)
```

Description

`maloglog(X, Y, 'PropertyName', PropertyValue...)` creates a loglog scatter plot of `X` versus `Y`.

`maloglog(..., 'FactorLines', N)` adds lines 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 point on 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.

`H = maloglog(...)` returns the handle to the plot.

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
Red = maStruct.Data(:,4);
Green = maStruct.Data(:,13);
maloglog(Red, Green, 'title', 'Red versus Green')
figure
maloglog(Red, Green, 'FactorLines', 2,...
        'Labels', maStruct.Names)
```

See Also

Bioinformatics Toolbox functions `maboxplot`, `mairplot` , `maimage`, `mairplot`, `malowess`, `manorm`

MATLAB function `loglog`

Purpose

Smooth microarray data using the Lowess method

Syntax

```
YSmooth = malowess(X, Y, 'PropertyName', PropertyValue...)

malowess(..., 'Order', OrderValue)
malowess(..., 'Robust', RobustValue)
malowess(..., 'Span', SpanValue)
```

Arguments

`X, Y`	Scatter data.
`Order`	Property to select the order of the algorithm. Enter either `1` (linear fit) or `2` (quadratic fit). The default order is `1`.
`Rubust`	Property to select a robust fit. Enter either `true` or `false`.
`Span`	Property to specify the window size. The default value is `0.05` (5% of total points in `X`)

Description

`YSmooth = malowess(X, Y, 'PropertyName', PropertyValue...)` smooths scatter data (`X, Y`) using the Lowess smoothing method. The default window size is 5% of the length of `X`.

`malowess(..., 'Order', OrderValue)` chooses the order of the algorithm. Note that the MATLAB 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`*.

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
cy3data = maStruct.Data(:,4);
```

```
cy5data = maStruct.Data(:,13);
[x,y] = mairplot(cy3data, cy5data);
drawnow
ysmooth = malowess(x,y);
hold on;
plot(x,ysmooth,'rx');
ynorm = y - ysmooth;
```

See Also

Bioinformatics Toolbox functions `maboxplot`, `maimagemairplot`, `maloglog`, `manorm`, `quantilenorm`

Statistics Toolbox `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 = maStruct.Data(:,4);
Green = maStruct.Data(:,13);
% 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:MaloglogZeroValues');
maloglog(normRedBs,normGreenBs,'title',...
'Normalized Background-Subtracted Median Values',...
               'factorlines',true)
warning(w);
```

See Also

Bioinformatics Toolbox functions `maboxplot`, `mairplot`, `maloglog`, `malowess`, `quantilenorm`

mapcaplot

Purpose

Create a Principal Component plot of expression profile data

Syntax

```
mapcaplot(Data)
mapcaplot(Data,Label)
```

Arguments

`Data`	Microarray data
`Label`	Data point labels.

Description

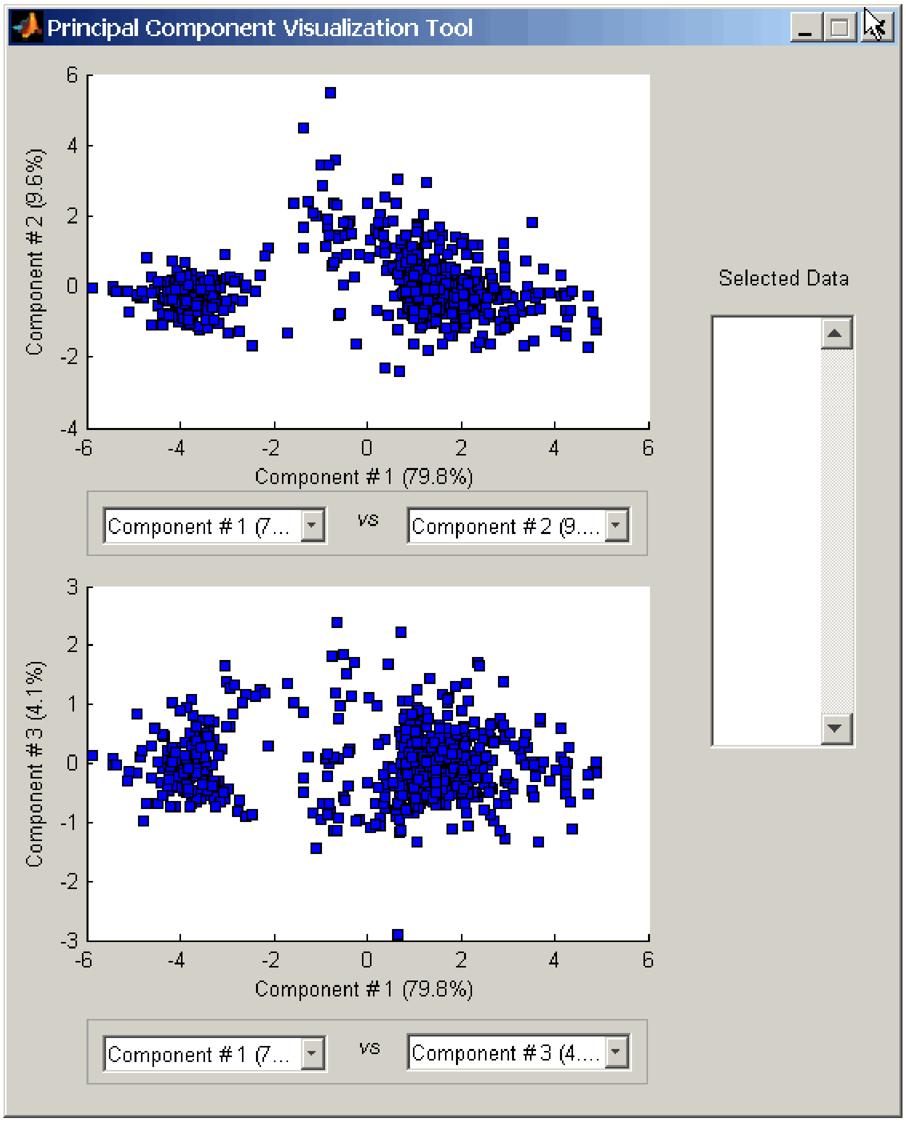
`mapcaplot(Data)` creates 2D scatter plots of principal components of the array `DATA`. The principal components used for the `x` and `y` data are selected from popup menus, below each scatter plot.

Once the principal components have been plotted, a region can be selected in either axes with the mouse. This will highlight the points in the selected region, and the corresponding points in the other axes. This will also display a list of the row numbers of the selected points in the list box. Selecting an entry in the list box will display a label with the row number in each axes, at the corresponding point. Clicking on a point in the scatter plot will display a label with its row number until the mouse is released.

`mapcaplot(Data,Label)` uses the elements of the cell array of strings `Label`, instead of the row numbers, to label the data points.

Examples

```
load filteredyeastdata
mapcaplot(yeastvalues,genes)
```


Principal Component Visualization Tool
Component #2 (9.6%)
Component #1 (79.8%)
Component #1 (7...
vs
Component #2 (9....
Component #3 (4.1%)
Component #1 (79.8%)
Component #1 (7...
vs
Component #3 (4....
Selected Data

See Also

Bioinformatics Toolbox function

Statistical Toolbox function `princomp`

Purpose

Align peaks in mass spectrum to reference peaks

Syntax

```
YOut = msalign(MZ, Y, R, 'PropertyName', PropertyValue...)
msalign(..., 'Weights', WeightsValue)
msalign(..., 'Range', RangeValue)
msalign(..., 'WidthOfPulses', WidthOfPulsesValue)
msalign(..., 'WindowSizeRatio', WindowSizeRatioValue)
msalign(..., 'Iterations', IterationsValue)
msalign(..., 'GridSteps', GridStepsValue)
msalign(..., 'SearchSpace', SearchSpaceValue)
[YOut, ROut] = msalign(..., 'Group', GroupValue),
msalign(..., '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.
`R`	Reference mass vector with a list of known masses in the sample spectrum.

Description

`YOut = msalign(MZ, Y, R, 'PropertyName', PropertyValue...)` aligns a raw mass spectrum (`Y`) by scaling and shifting the mass/charge scale (`MZ`) so that the cross-correlation between the spectrum (`Y`) and a synthetic spectrum is maximum. A synthetic spectrum is built with Gaussian pulses centered at the masses specified by the reference mass vector (`R`). Once the new mass/charge scale is determined, a new spectrum (`YOut`) is calculated by piecewise cubic interpolating and shifting the new spectrum from the original mass/charge vector (`MZ`). This method preserves the shape of the peaks.

`msalign` uses an iterative grid search until it finds the best scale and shift factors for every spectrum.

Note The algorithm works best with three to five marker masses that you know will appear in the spectrum. If you use a single marker mass (a single internal standard), there is a possibility of picking a peak between the marker and sample peak for that marker as `msalign` scales and shifts the `MZ` vector. If you only require to shift the `MZ` vector, you may prefer to use`YOut = interp1(MZ, MZ-(MarkerMass-PeakPosition, Y)`.

`msalign(..., 'Weights',` *`WeightsValue`*`)`specifies the relative weights for every mass in the reference mass vector (`R`). The size of the weight vector (*`WeightsValue`*) must be the same as the reference mass vector (R). The default value is `ones(size(R))` with a range of `0` to`1`, but you can use any range. If you have a small number of reference masses, you might want to change the weights.

`msalign(..., 'Range',` *`RangeValue`*`)`specifies the lower and upper bound for the allowable range in m/z units to shift any of the mass peaks. The default value is `[-100 100]`. Use these values to tune the robustness of the algorithm. Ideally, you should only try to correct small shifts by keeping the bounds small.

Note You can try to correct larger shifts by increasing the bounds, but you might also pick the wrong peaks to be aligned.

`msalign(..., 'WidthOfPulses',` *`WidthOfPulsesValue`*`)` specifies the width (*`WidthOfPulsesValue`*) in m/z units for all the Gaussian pulses used to build the correlating synthetic spectrum. *`WidthOfPulsesValue`* is at the point where the Gaussian pulse reaches `60.65`% of its maximum. The default value 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 errors out.

Note Tuning the spread of the Gaussian pulses controls a tradeoff between robustness (wider pulses) and precision (narrower pulses), but the spread is unrelated to the shape of the observed peaks in the spectrum.

`msalign(..., 'WindowSizeRatio',` *`WindowSizeRatioValue`*`)` specifies a scaling value that determines the size of the window around every alignment peak. The synthetic spectrum is correlated to the sample spectrum only within these regions, which saves computation time. Size of the window is given by `Width * WindowSizeRatio` in m/z units. The default value 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. The default value is `5`.

`msalign(..., 'GridSteps',` *`GridStepsValue`*`)` specifies the number of steps for the search grid. For example, at every iteration the search area is divided by `GridStepValue^2`. The default value is `20`.

`msalign(..., 'SearchSpace',` *`SearchSpaceValue`*`)` specifies the type of search space. Enter either `'regular'`(evenly spaced lattice) or `'latin'` (random latin hypercube with `Grid^2` samples). The default value is `'regular'`.

`[YOut, ROut] = msalign(..., 'Group',` *`GroupValue`*`)`, when *`GroupValue`* is `true` and `Y` contains more than one spectrum, updates the original peak locations so that the actual movement of the peaks is minimized. `ROut` contains the reference masses with the updated ion peak locations. Use this property when you are uncertain about the values for the reference masses. The default value is `false`.

`msalign(..., 'ShowPlot',` *`ShowPlotValue`*`)` plots the original and the aligned spectrum over the reference masses (`R`). When `msalign` is called without output arguments, the spectra are plotted unless

ShowPlotValue is `false`. When *ShowPlotValues* is `true`, only the first spectrum in `Y` is plotted. The default value is `false`.

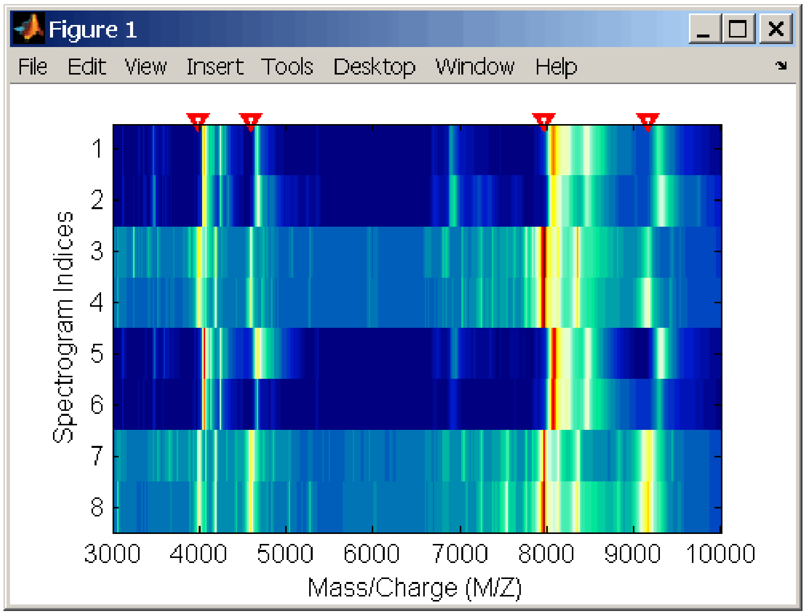
Example 1

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];
```

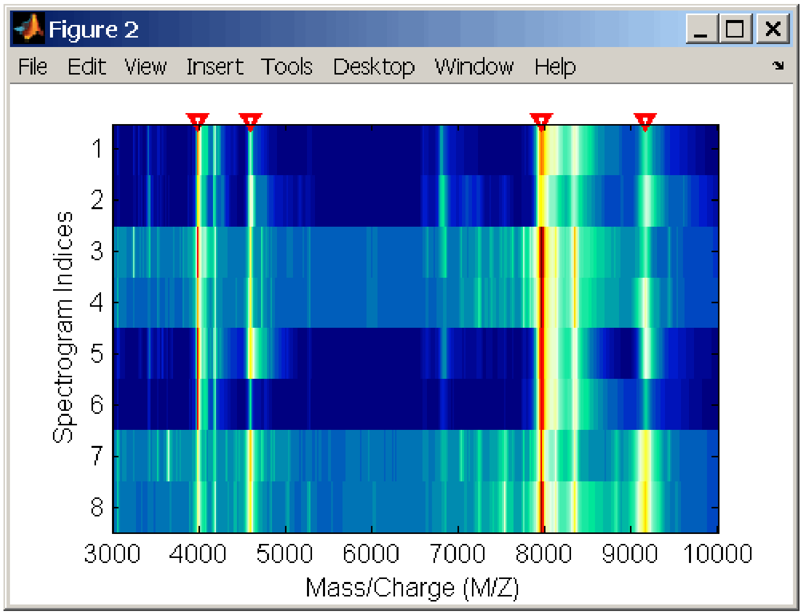
2 Display a color image of the mass spectra before alignment.

```
msheatmap(MZ_lo_res,Y_lo_res,'markers',R,'limit',[3000 10000])
title('before alignment')
```



3 Align spectra with reference masses and display a color image of mass spectra after alignment.

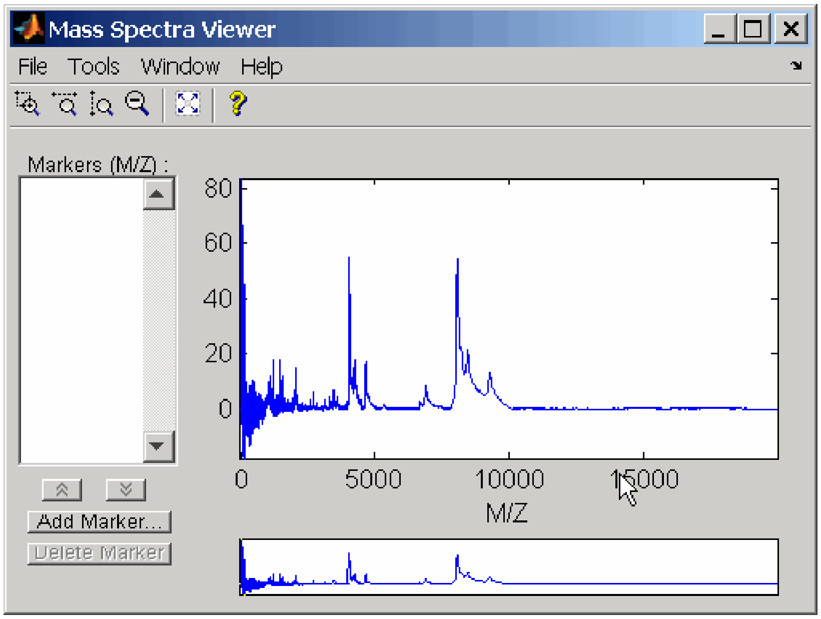
```
YA = msalign(MZ_lo_res,Y_lo_res,R,'weights',W);
msheatmap(MZ_lo_res,YA,'markers',R,'limit',[3000 10000])
title('after alignment')
```



Example 2

1 Align a spectrum with a single reference peak. 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)
```

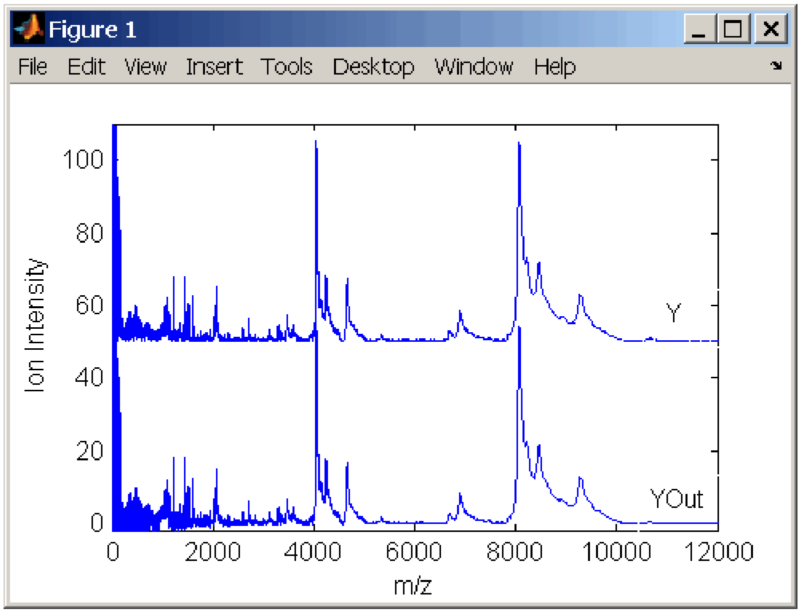


2 Select a reference peak by zooming and right-clicking a peak.

3 Shift a spectrum by the difference between the known reference mass (RP) and the experimental mass (SP).

```
RP = 4000;
SP = 4050.33;
YOut = interp1(MZ, MZ-(RP-SP, Y);
```

The plot below shows the original spectrum on top and the shifted spectrum on the bottom.



See Also

Bioinformatic Toolbox functions `msbackadj`, `msheatmap`, `mslowess`, `msnorm`, `msresample`, `mssgolay`, `msviewer`

msbackadj

Purpose

Correct the baseline of a mass spectrum

Syntax

```
Yout = msback(MZ, Y)
msbackadj(..., 'PropertyName', PropertyValue,...)
msbackadj(..., 'WindowSize', WindowSizeValue)
msbackadj(..., 'StepSize', StepSizeValue)
msbackadj(..., 'RegressionMethod', RegressionMethodValue)
msbackadj(..., 'EstimationMethod', EstimationMethodValue)
msbackadj(..., 'SmoothMethod', SmoothMethodValue)
msbackadj(..., 'QuantileValue', QuantileValueValue)
msbackadj(..., 'PreserveHeights', PreserveHeightsValue)
msbackadj(..., '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 = msback(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`)

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

`msbackadj(..., '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 towards the end of the spectrum, you may want to use variable parameters.

`msbackadj(..., '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.

`msbackadj(..., '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'`.

`msbackadj(..., '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.

`msbackadj(..., '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 value is `'none'`.

`msbackadj(..., 'QuantileValue', QuantileValueValue)` specifies the quantile value. The default value is `0.10`.

`msbackadj(..., '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.

`msbackadj(..., '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`.

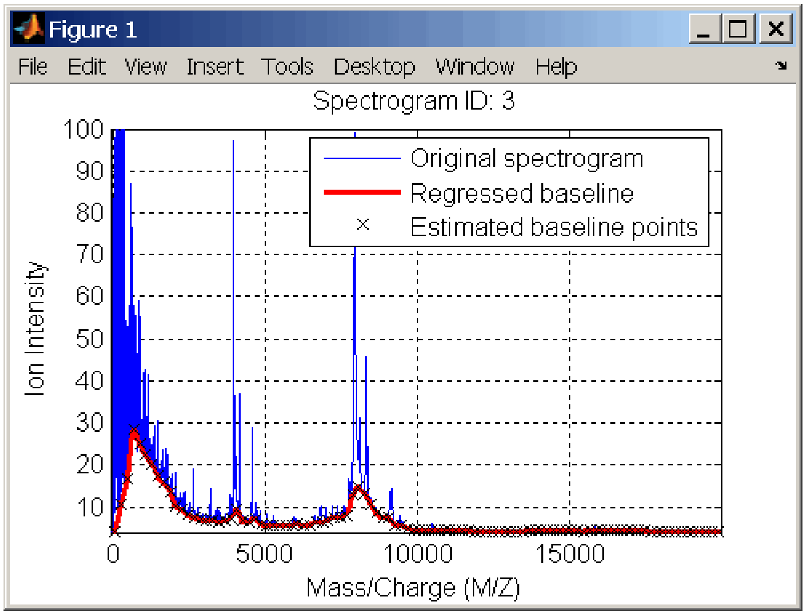
Example

1 Load sample data.

```
load sample_lo_res
```

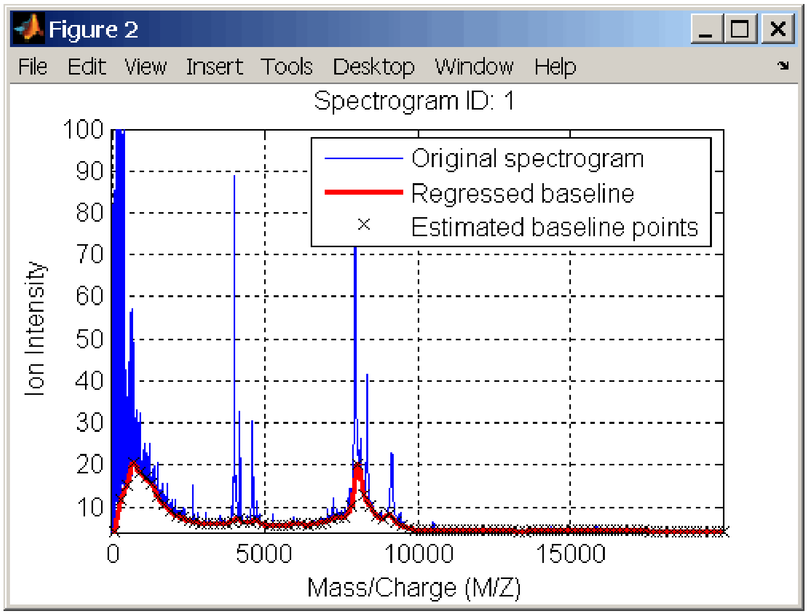
2 Adjust the baseline for a group of spectra and show only the third spectrum and its estimated background.

```
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

The Bioinformatic Toolbox functions `msalign`, `mslowess`, `msheatmap`, `msnorm`, `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 The MATLAB 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);
```

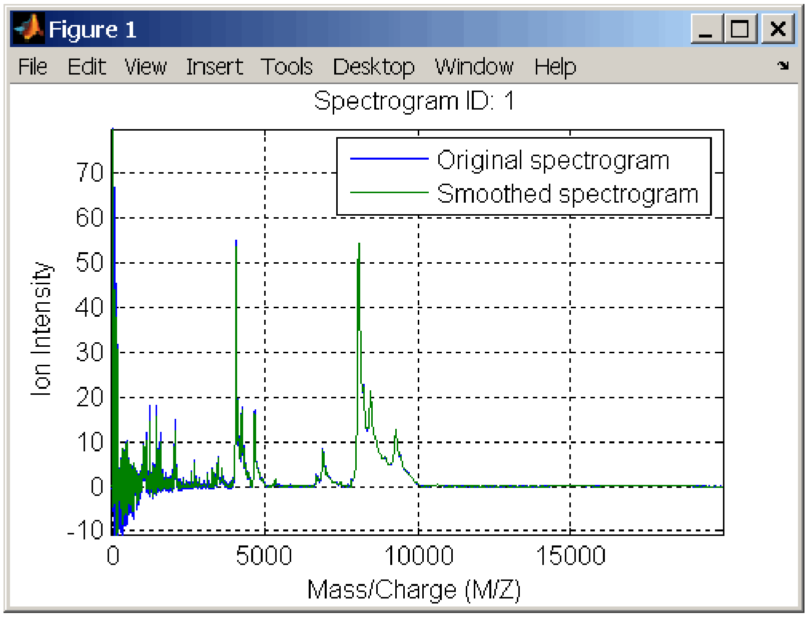
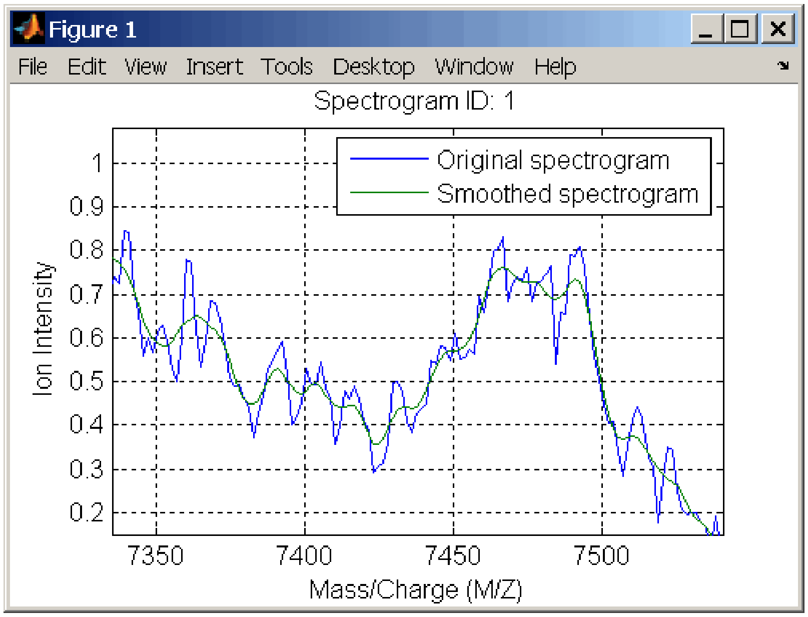


Figure 1
File Edit View Insert Tools Desktop Window Help
Spectrogram ID: 1
Original spectrogram
Smoothed spectrogram
Ion Intensity
Mass/Charge (M/Z)
70
60
50
40
30
20
10
0
-10
0
5000
10000
15000



See Also

Bioinformatic Toolbox functions `msalign`, `msbackadj`, `msheatmap`, `msheatmap`,`msnorm`, `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.

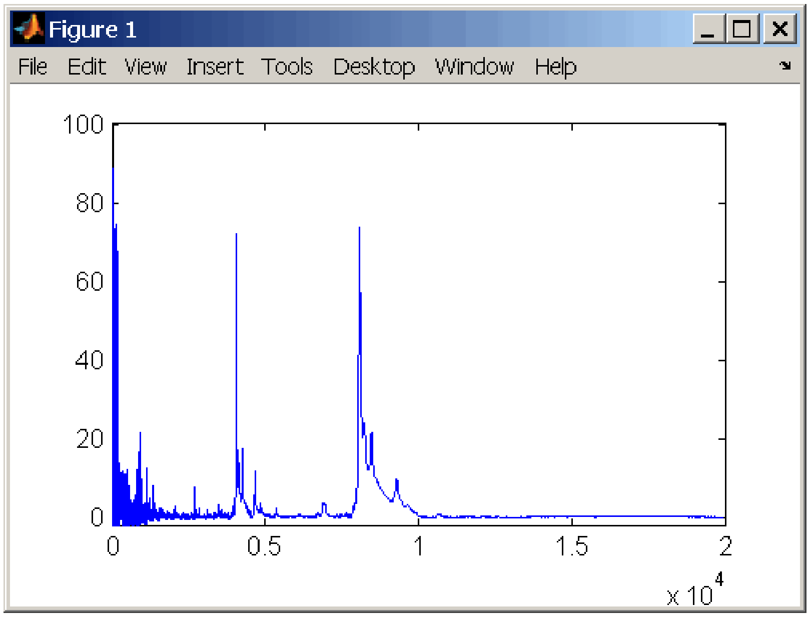
`msnorm(..., 'Method', MethodValue)` selects a method for normalizing the AUC of every spectrum. Enter either `'Median'` (default) or `'Mean'`.

`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

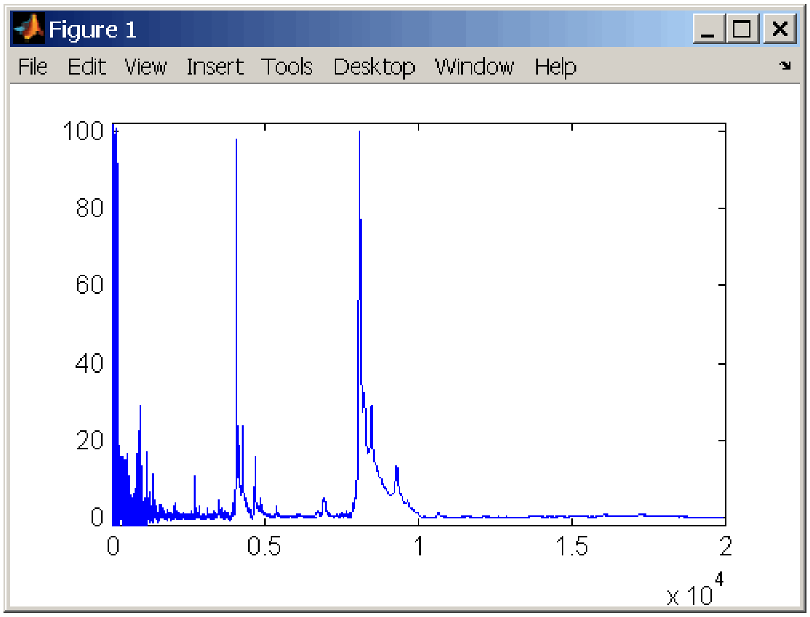
1 Load sample data and plot one of the spectra.

```
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

Bioinformatic Toolbox functions `msalign`, `msbackadj`, `msheatmap`, `mslowess`, `msresample`, `mssgolay`, `msviewer`

msheatmap

Purpose

Display color image for set of spectra

Syntax

```
msheatmap(MZ, Y, 'PropertyName', PropertyValue...)
msheatmap(..., 'Markers', MarkersValue)
msheatmap(..., 'Limits', LimitsValues)
msheatmap(..., '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

`msheatmap(MZ, Y, 'PropertyName', PropertyValue...)` shows a heatmap image of the spectra in `Y`.

`msheatmap(..., 'Markers', MarkersValue)` specifies a list of markers with positions marked along the top axis. The default value is `[]`.

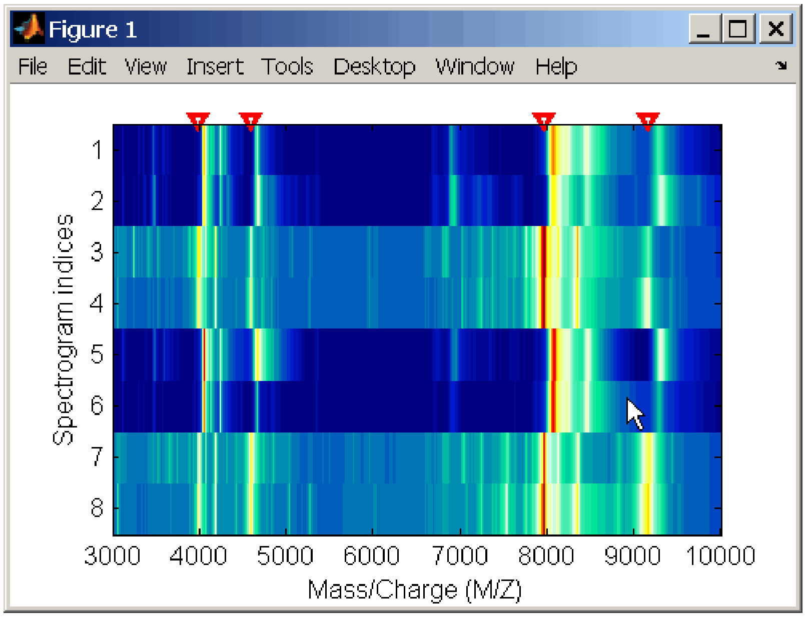
`msheatmap(..., 'Limits', LimitsValues)` specifies a `[2x1]` vector with the mass/charge range for the heatmap image.

`msheatmap(..., 'Group', GroupValue)` specifies the class label for every spectrum used to group the rows of the heatmap image. *`GroupValue`* can be a numeric vector or a cell array of strings with the same number of elements as there are spectra in `Y`.

Examples

1 Load sample data.

```
load sample_lo_res
M = [3991.4 4598 7964 9160];
msheatmap(MZ_lo_res,Y_lo_res,'markers',M,'limit',[3000 10000])
```



2 Plot heatmap.

```
msheatmap(MZ_lo_res,Y_lo_res,'markers',M,'group',[1 1 2 2 1 1 2 2])
```

See Also

The Bioinformatic Toolbox functions `msalign`, `msbackadj`, `mslowess`, `msnorm`, `msresample`, `mssgolay`, `msviewer`

msresample

Purpose

Resample a 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

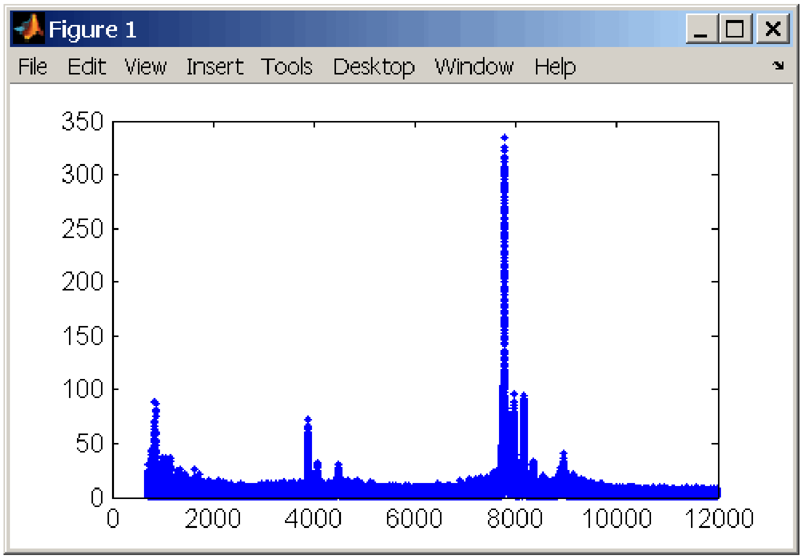
```
load sample_hi_res;
```

```
mz = MZ_hi_res;
y = Y_hi_res;
```

2 Plot original data to a lower resolution.

```
plot(mz, y, '.')
```

MATLAB draws a figure.



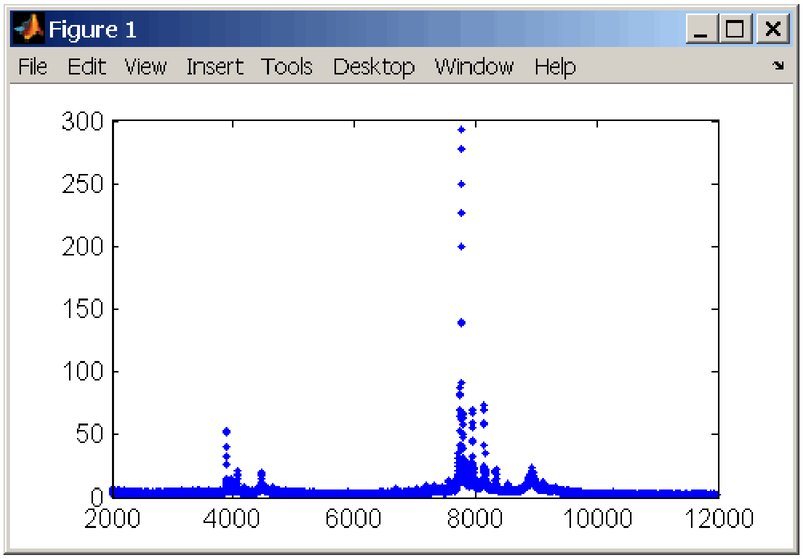
3 Resample data

```
[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

The Bioinformatic Toolbox functions `msalign`, `msbackadj`, `msheatmap`, `mslowess`, `msnorm`, , `mssgolay`, `msviewer`

Purpose

Smooth mass spectrum with least-squares polynomial

Syntax

```
Yout = mssgolay(MZ,Y, '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, '`*`PropertyName`*`', `*`PropertyValue...`*`)` 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(..., '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 incremented 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

Bioinformatic Toolbox functions `msalign`, `msbackadj`, `msheatmap`, `mslowess`, `msnorm`, , `msresample`, `msviewer`

msviewer

Purpose

Explore MS spectrum or set of spectra with GUI

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

Bioinformatic Toolbox functions `msalign`, `msbackadj`, `mslowess`, `msnorm`, `msheatmap`, `msresample`, `mssgolay`

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 table Amino Acid Lookup Table on page 2-14. 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

Get the protein sequence for `cytochrome c` and determine its molecular weight.

```
pirdata = getpir('cchu','SequenceOnly',true)
mwcchu = molweight(pirdata)


mwcchu =
  1.1749e+004
```

See Also

Bioinformatics Toolbox functions `aacount`, `atomiccomp`, `isoelectric`, `proteinplot`

multialign

Purpose

Align multiple sequences using progressive method.

Syntax

```
SeqsMultiAligned = multialign(Seqs)
SeqsMultiAligned = multialign(Seqs, Tree)
multialign(..., 'PropertyName', PropertyValue,...)
multialign(..., 'Weights', WeightsValue)
multialign(..., 'SmiInterp', SimiInterpValue)
multialign(..., 'GapOpen', GapOpenValue)
multialign(..., 'ExtendedGap', ExtendedGapValue)
multialign(..., 'DelayCutoff', DelayCutoffValue)
multialign(..., 'Verbose', VerboseValue)
multialign(..., 'OldGapAdjust', OldGapAdjustValue)
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 that all of them share the same scale. ???LUCIO: how can the default scoring matrix for AA be a series and not a single matrix???
GapOpenValue	Scalar or a function specified using `@`. `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`). Defaults value is `@(sm,sx,len1,len2) 2*sm`.
ExtendedGapValue	Scalar or a function specified using `@`. 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 value is `@(sm,sx,len1,len2) sm/20`.
DelayCutoffValue	The default is unity where sequences with the closest sequence farther than the median distance are delayed.

Description

SeqsMultiAligned = `multialign(`*Seqs*`)` performs a progressive multiple alignment for a set of sequences (*Seqs*). Pariwise 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’ — 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(..., 'SmiInterp', *SimiInterpValue*)`, when *`SimiInterpValue`* 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. Default is true.

`multialign(..., 'GapOpen', GapOpenValue)` specifies the initial penalty for opening a gap.

`multialign(..., 'ExtendedGap', ExtendedGapValue)` 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(..., 'Verbose', VerboseValue)`, when `VerboseValue` is `true`, turns on verbosity. Default value is `false`.

The remaining input optional arguments are analogous to the function profalign and are used through every step of the progressive alignment of profiles.

`multialign(..., 'OldGapAdjust', OldGapAdjustValue)`. Default is `true`.

`multialign(..., 'TerminalGapAdjust', TerminalGapAdjustValue)`, Default is false.

???DOC: r14sp2+ maybe, r14sp3, sometime look at profalign arguments and add redundant info here???

Example

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)
```

3 Score the progressive alignment with the PAM family.

```
ma = multialign(p53,tree,'ScoringMatrix',{'pam150','pam200','pam250
showalignment(ma)
```

4 Promote terminations with gaps in the alignment.

```
seqs = {'CACGTAACATCTC','ACGACGTAACATCTTCT','AAACGTAACATCTCGC'};
multialign(seqs,'terminalGapAdjust',true)
```

See Also

Bioinformatics Toolbox functions `hmmprofalign`, `multialignread`, `nwalign`, `profalign`, `seqprofile`, `seqconsensus`, `seqneighjoin` `showalignment`.

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 filename, a path and filename, 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`.

Example

1 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`, `gethmmalignmet`, `seqdisp`, `multialign`, `seqconsensus`, `seqprofile`.

Purpose

Count the number of n-mers in a 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, 'PropertyName', PropertyValue)

nt2aa(..., 'Frame', FrameValue)
nt2aa(..., 'GeneticCode', GeneticCodeValue)
nt2aa(..., 'AlternativeStartCodons', AlternativeValue)
```

Arguments

`SeqNT`	DNA nucleotide sequence. Enter a character string with only the characters `A`, `T`, `C`, and `G`. You cannot use the character `U`, ambiguous characters, or a hyphen. You can also enter a structure with the field `Sequence`.
`FrameValue`	Property to select a frame. Enter `1`, `2`, `3`, or `'ALL'`. The default value is `1`.
`GeneticCodeValue`	Property to select a genetic code. Enter a code number or code name from the table Genetic Code on page 2-244. If you use a code name, you can truncate the name to the first two characters of the name.
`AlternativeValue`	Property to control the use of alternative codons. Enter either `true` or `false`. The default value is `true`.

Genetic Code

Code Number	Code Name
1	`Standard`
2	`Vertebrate Mitochondrial`
3	`Yeast Mitochondrial`

Code Number	Code Name
4	`Mold`, `Protozoan`, and `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

`SeqAA = nt2aa(SeqNT, '`*`PropertyName`*`', `*`PropertyValue`*`)` converts a nucleotide sequence to an amino acid sequence using the standard genetic code.

`nt2aa(..., 'Frame', `*`FrameValue`*`)` converts a nucleotide sequence for a specific reading frame to an amino acid sequence. If *`FrameValue`* equals `'ALL'`, then the three reading frames are converted and the output is a 3-by-1 cell array.

`nt2aa(..., 'GeneticCode', `*`GeneticCodeValue`*`)` converts a nucleotide sequence to an amino acid sequence using a specific genetic code.

`nt2aa(..., 'AlternativeStartCodons', `*`AlternativeValue`*`)` controls the use of alternative start codons. By default, `AlternativeStartCodons` is set to `true`, and if the first codon of a sequence corresponds to a known alternative start codon, the codon is translated to methionine.

If this option is set to `false`, then alternative start codons at the start of a sequence are translated to their corresponding amino acids for the genetic code that you use, 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

Convert the gene ND1 on the human mitochondria genome.

```
mitochondria = getgenbank('NC_001807','SequenceOnly',true)
gene = mitochondria (3308;4264)
protein1 = nt2aa(gene,'GeneticCode', 2)
protein2 = getgenpept('NP_536843',SequenceOnly',true)
```

Convert the gene ND2 on the human mitochondria genome. In this case, the first codon is `att`, which is converted to M, while the following `att` codons are converted to `I`. If you set `'AlternativeStartCodons'` to `false`, then the first codon `att` is converted to `I`.

```
mitochondria = getgenbank('NC_001807','SequenceOnly',true)
gene = mitochondria (3371:4264)
protein1 = nt2aa(gene,'GeneticCcode',2)
protein2 = getgenpept('NP_536844', 'SequenceOnly',true)
```

See Also

Bioinformatics Toolbox functions `aa2int`, `baselookup`, `geneticcode`, `revgeneticcode`, `aminolookup`, `baselookup`, `codonbias`, `dnds`, `dndsml`, `seqtool`

Purpose

Convert nucleotide sequence from letter to integer representation

Syntax

```
SeqInt = nt2int(SeqChar, 'PropertyName', PropertyValue)

nt2int(..., 'Unknown', UnknownValue)
nt2int(..., 'ACGTOnly', ACGTOnlyValue)
```

Arguments

`SeqNT`	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 value is `0`.
`ACGTOnlyValue`	Property to control the use of ambiguous nucleotides. Enter either `true` or `false`. Default value is `false`.

Mapping Nucleotide Letters to Integers

Base	Code	Base	Code	Base	Code
Adenosine	`A—1`	`T, C` (pyrimidine)	`Y—6`	`A, T, G (not C)`	`D—12`
Cytidine	`C—2`	`G`, `T` (keto)	`K—7`	`A, T, C (not G)`	`H—13`
Guanine	`G—3`	`A`, `C` (amino)	`M—8`	`A, G, C (not T)`	V—`14`

Base	Code	Base	Code	Base	Code
Thymidine	T—4	G, C (strong)	S—9	A, T, G, C (any)	N—15
Uridine	U—4	A, T (weak)	W—10	Gap of indeterminate length	- —16
A, G (purine)	R—5	T, G, C (not A)	B—11	Unknown (default)	*—0 and ≥17

Description

`nt2int(SeqNT, 'PropertyName', PropertyValue)` converts a character string of nucleotides to a 1-by-N array of integers using the table Mapping Nucleotide Letters to Integers above. Unknown characters (characters not in the table) are mapped to 0. Gaps represented with hyphens are mapped to 16.

`nt2int(SeqNT,'Unknown',UnknownValue)` defines the number used to represent unknown nucleotides. The default value is 0.

`nt2int(SeqNT,'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 function aa2int, baselookup, int2aa, int2nt

Purpose

Plot the density of nucleotides along a sequence

Syntax

```
ntdensity(SeqNT, 'PropertyName', PropertyValue)

ntdenstiy(..., 'Window', WindowValue)
[Density, HighCG] = ntdensity(..., 'CGThreshold',
CGThresholdValue)
```

Description

`ntdensity(SeqNT)` plots the density of nucleotides `A`, `T`, `C`, `G` in sequence `SeqNT`.

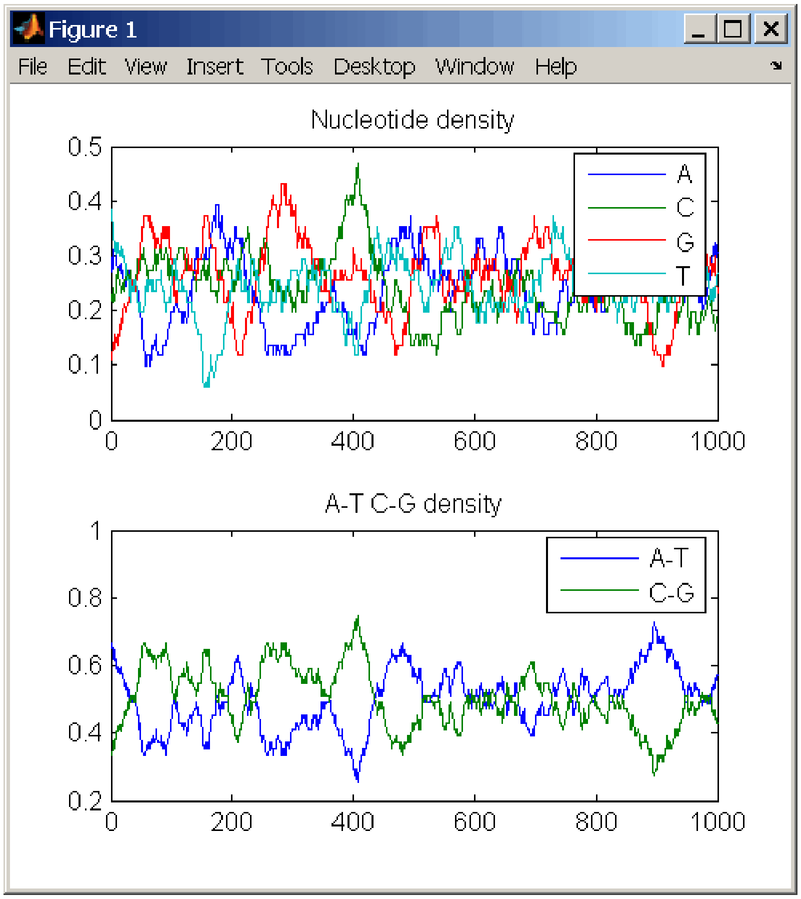
`Denstity = ntdensity(SeqNT, 'PropertyName', PropertyValue)` returns a MATLAB structure with the density of nucleotides `A`, `C`, `G`, and `T`.

`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 a NUC44 scoring matrix for nucleotide sequences

Syntax

```
ScoringMatrix = nuc44
```

Description

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`

`[Matrix, MatrixInfo] = nuc44` returns the structure of information about the matrix with `Name` and `Order`.

nwalign

Purpose

Globally align two sequences using the Needleman-Wunsch algorithm

Syntax

```
nwalign(Seq1, Seq2,
        'PropertyName', PropertyValue...)
[Score, Alignment] =nwalign(Seq1, Seq2)
[Score, Alignment, Start] = nwalign(Seq1, Seq2)

nwalign(..., 'ScoringMatrix', ScoringMatrixValue)
swalign(..., 'Scale', ScaleValue)
nwalign(..., 'GapOpen', GapOpenValue)
nwalign(..., 'ExtendGap', ExtendGapValue)
nwalign(..., 'Alphabet', AlphabetVlaue)
swalign(..., 'Showscore', ShowscoreValue)
```

Arguments

`Seq1, Seq2`	Nucleotide or amino acid sequences. Enter a character string or a structure with the field `Sequence`.
`Alphabet`	Property to select the type of sequence. Value is either`'AA'` or `'NT'`. The default value is `'AA'`.
`ScoringMatrix`	Enter the name of a scoring matrix. Values are `'PAM40`', `'PAM250'`, `DAYHOFF`, `GONNET`, `'BLOSUM30'` increasing by 5 to `'BLOSUM90'`, `'BLOSUM62'`, or `'BLOSUM100'`. The default value when `AlphabetValue = 'aa'` is `'BLOSUM50'`, while the default value when `AlphabetValue = 'nt'` is `nuc44`.
`Scale`	Property to specify a scaling factor for a scoring matrix.
`GapOpen`	Property to specify the penalty for opening a gap. The default value is `8`.

`ExtendedGap`	Property to specify the penalty for extending a gap. If `ExtendGap` is not specified, then the default value is equal to `GapOpen`.
`Showscore`	Property to control displaying the scoring space and the winning path. Enter either `true` or `false`. The default value is `false`.

Description

`nwalign(Seq1, Seq2, '`*`PropertyName`*`', `*`PropertyValue`*`...)` returns the alignment score in bits for the optimal alignment. The scale factor used to calculate the score is provided by the scoring matrix information. If this is not defined, then `nwalign` returns the raw score.

`[Score, Alignment] = nwalign(Seq1, Seq2)` returns a string showing an optimal global alignment for the sequences. Amino acids that match are indicated with the symbol `|`, while related amino acids (nonmatches with a positive scoring matrix value) are indicated with the symbol `:`. Units for `Score` are bits.

`[Score, Alignment, Start] = nwalign(Seq1, Seq2)` returns a `2x1` vector with the starting point indices indicating the starting point of the alignment in the two sequences. Note: This output is for consistency with `swalign`, but because this is a global alignment, the starting position is always `[1;1]`.

`nwalign(..., 'Alphabet', `*`AlphabetValue`*`)` selects the amino acid or nucleotide alphabet for sequences.

`nwalign(..., 'ScoringMatrix', `*`ScoringMatirxValue`*`)` selects the scoring matrix to use for the alignment.

`swalign(..., 'Scale', `*`ScaleValue`*`)` specifies the scale factor of the scoring matrix to return the score using arbitrary units. If the scoring matrix also provides a scale factor, then both are used.

`nwalign(..., 'GapOpen', `*`GapOpenValue`*`)` specifies the penalty for opening a gap in the alignment.

`nwalign(..., 'ExtendGap', *ExtendGapValue*)` specifies the penalty for extending a gap in the alignment. If `ExtendGap` is not specified, then extensions to gaps are scored with the same value as `GapOpen`.

`nwalign(..., 'Showscore', *ShowscoreValue*)` displays the scoring space and the winning path.

Examples

Globally align two amino acid sequences.

```
[Score, Alignment] = nwalign('VSPAGMASGYD','IPGKASYD')

Score =
    7.3333

Alignment =
VSPAGMASGYD
: | | || ||
I-P-GKAS-YD
```

Select scoring matrix and gap penalty.

```
[Score, Alignment] = nwalign('IGRHRYHIGG','SRYIGRG',...
                              'scoringmatrix','pam250',...
                              'gapopen',5)

Score =
    2.3333
Alignment =

IGRHRYHIG-G
 :  || || |
-S--RY-IGRG
```

See Also

Bioinformatics Toolbox functions `blosum`, `multialign`, `nt2aa`, `pam`, `profalign`, `seqdotplot`,`showalignment`, `swalign`

Purpose

Find palindromes in a sequence

Syntax

```
[Position, Length] = palindromes(SeqNT,
                                'PropertyName',
PropertyValue)
[Postion, 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`

Purpose

Return a PAM scoring matrix

Syntax

```
ScoringMatrix = pam(N, 'PropertyName', PropertyValue)
[ScoringMatirx, MatrixInfo] = pam(N)

ScoringMatrix = pam(..., 'Extended', ExtendedValue)
ScoringMatrix = pam(..., '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.
`Extended`	Property to add ambiguous characters to the scoring matrix. Enter either `true` or `false`. Default is `false`.
`Order`	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, 'PropertyName', *PropertyValue*)` 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`.

`B = pam(..., '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.

`B = pam(..., 'Order', '*OrderString*')` 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

Get the PAM matrix with N = 50.

```
PAM50 = pam(50)

PAM250 = pam(250,'Order','CSTPAGNDEQHRKMILVFYW')
```

See Also

Bioinformatics Toolbox functions blosum, dayhoff, gonnet, nwalign, swalign, pam250

Purpose

Visualize intermolecular distances in 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`, `pdbread`, `pdbplot`, `pdbread`, `proteinplot`, `ramachandran`

Purpose

Plot 3D protein structure

Syntax

```
pdbplot(PDBid, 'PropertyName', PropertyValue ...)
pdbplot(..., 'Plotmode', PlotmodeValue)
pdbplot(..., 'Colormode', ColormodeValue)
pdbplot(..., 'Showlabel', ShowlabelValue)
FigureHandle = pdbplot(...)
www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=808
```

Arguments

`PDBid`	PDBID can also be the name of a PDB structure or a file containing a PDB structure.
`Plotmode`	Property to select display backbone and side chains. Enter either `'backbone'` or `'mainchain'`. The default value is `'backbone'` for the alpha carbon backbone.
`Colormode`	Property to select the color of atoms or folding patters. Enter `'atom'`, `'chain'`, or `'secondary'`. The default is `'chain'`.

Description

`pdbplot(PDBid, '`*`PropertyName`*`', `*`PropertyValue`*` ...)` retrieves 3D information from the Web for a protein (*`PDBid`*), and plots the backbone structure. Information for the protein is in the Protein Data Bank (PDB) database.

`pdbplot(..., 'Plotmode', `*`PlotmodeValue`*`)` selects a plot with only the alpha-carbon backbone or a plot with amino acid side-chains.

`pdbplot(..., 'Colormode', `*`ColormodeValue`*`)` selects the colors for a plot.

- If `Colormode` is `'atom'` and `Plotmode` is `'mainchain'`, atoms and connections are colored green for carbon, blue for nitrogen, and red for oxygen.
- The `Colormode` is ”chain’, the entire structure is one color.

- If `Colormode` is `'secondary'`, alpha helix patterns are colored yellow, sheets are blue, turns are gray and, non alpha helix are cyan.

`pdbplot(..., 'Showlabel', ShowlabelValue)` when `Showlabel` is `true`, displays the labels that represent each amino acid name and sequence number in the protein. The default is false.

`FigureHandle = pdbplot(...)` returns the handle for the PDB plot figure.

For more on viewing PDB molecules in MATLAB, see the molecule viewer in MATLAB Central
`www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=808`

Examples

Plot the 3D backbone structure for the protein Insulin-Like-Growth-Factor-1. The identification number for this protein in the PDB database is `1B9G`.

1. In the MATLAB Command Window, type

```
pdbplot('1B9G')
```

A figure window opens with the 3D structure for this protein. The figure title displays the identification number `PDB Plot  1B9G` while the bottom of the figure shows the protein title or compound name `Title: INSULIN-LIKE-GROWTH-FACTOR-1`.

3. Rotate, translate, and zoom the structure with the MATLAB camera toolbar.

4. From **File** menu, select

- **Save to Figure file** — Saves the plot to a MATLAB figure file
- **Print** - Prints the plot
- **Close** - Closes the current PDB plot figure window
- **Close All** - Closes all the opened PDB plot figure windows

5. Select the different view options from the **View** menu or navigation tool on the right side of the figure.

Select an Plot option button:

- **Backbone** - Plots c- alpha trace
- **Main Chain** - Plots main chain

Select a Color check box:

- **Atoms** - Color atoms based on predefined color code: Red = oxygen, Green = carbon, Blue = nitrogen
- **Secondary** - Color secondary structures based on predefined color code: yellow = a-helix, blue = beta-strand, gray = turn, cyan = helix (non-alpha), green = all other structures

Select the Show check box:

- **Labels** - Show amino acid sequence labels

6. From the **Help** menu, **Help** or **Demos** for Bioinformatics toolbox.

See Also

Bioinformatics Toolbox functions `getpdb`, `pdbdistplot`, `pdbread`, `proteinplot`, `ramachandran`

pdbread

Purpose

Read data from Protein Data Bank (PDB) file

Syntax

```
PDBData = pdbread('File')
```

Arguments

File	Protein Data Bank (PDB) formatted file (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text for a PDB file.

Description

The Protein Data Bank (PDB) is an archive of experimentally determined three-dimensional protein structures. `pdbread` reads data from a PDB formatted file into MATLAB.

`PDBData = pdbread('File')` reads the data in PDB formatted text file *File* and stores the data in the MATLAB structure `PDBData`.

The data stored in each record of the PDB file is converted, where appropriate, to a MATLAB structure. For example, the ATOM records in a PDB file are converted to an array of structures with the following fields: `AtomSerNo`, `AtomName`, `altLoc`, `resName`, `chainID`, `resSeq`, `iCode`, `X`, `Y`, `Z`, `occupancy`, `tempFactor`, `segID`, `element`, and `charge`.

The sequence information from the PDB file is stored in the `Sequence` field of `PDBData`. The sequence information is itself a structure with the fields `NumOfResidues`, `ChainID`, `ResidueNames`, and `Sequence`. The field `ResidueNames` contains the three-letter codes for the sequence residues. The field `Sequence` contains the single-letter codes for the sequence. If the sequence has modified residues, then the `ResidueNames` might not correspond to the standard three-letter amino acid codes, in which case the field `Sequence` will contain a `?` in the position corresponding to the modified residue.

For more information about the PDB format, see

```
http://www.rcsb.org/pdb/docs/format/pdbguide2.2/
guide2.2_frame.html
```

Examples

Get information for the human hemoglobin protein with number `1A00` from the Protein Data Bank, store information in the file `collagen.pdb`, and then read the file back into MATLAB.

```
getpdb( '1A00','ToFile', 'collagen.pdb')
pdbdata = pdbread('collagen.pdb')
```

See Also

Bioinformatics Toolbox functions `genpeptread`, `getpdb`, `pdbplot`, `pdbdistplot`, `pirread`

pdist (phytree)

Purpose

Calculate pairwise patristic distances in a 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 function `phytree (phytree)`.
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 `I`th and `J`th 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 `I`th and the `J`th 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 function `phytree`, `phytreetool`, `seqpdist`, `seqlinkage`

pfamhmmread

Purpose

Read data from a PFAM-HMM file

Syntax

```
Data = pfamhmmread('File')
```

Arguments

File	PFAM-HMM formatted file. Enter a filename, a path and filename, 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-HHM 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.

`pfammread` 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?id=7tm_2'])
```

See Also

Bioinformatics Toolbox functions `gethmmalignment`, `gethmmprof`, `hmmprofalign`, `hmmprofstruct`, `showhmmprof`

Purpose

Object constructor for a phylogenetic tree object

Syntax

```
Tree = phytree(B)
Tree = phytree(B, D)
Tree = phytree(B, C)
Tree = phytree(BC)
Tree = phytree(..., N)
```

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.
`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 branch or leaves, and distances of branches.
`N`	Cell array with the names of leafs and branches.

Description

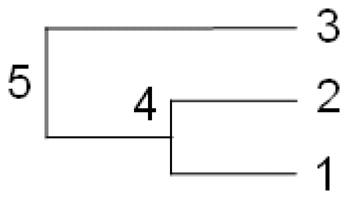
`Tree = phythree(B)` creates an ultrametric phylogenetic tree object.

`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 leave 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`.

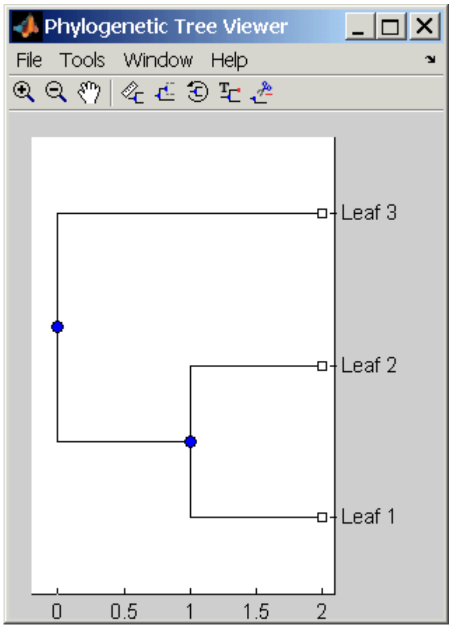
Branches are defined in chronological order (for example, `B(i,:) > NUMLEAVES + i`). As a consequence, the first row can only have pointers to leaves, and the last row must represent the root branch. Parent-child distances are set to `1`, unless the child is a leaf and to satisfy the ultrametric condition of the tree its distance is increased.

Given a tree with 3 leafs and 2 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 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 branch distances defined by `C`. C is a numeric array of size `[NUMBRANCHES X 1]` with the coordinates of every branch node. In ultrametric trees all the leaves are at the same location (for example, 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.

Method Summary

get (phytree)	Get information about a phylogenetic tree object
getbyname (phytree)	Select branches and leaves from a phytree object
getcanonical (phytree)	Calculate the canonical form of a phylogenetic tree
getnewickstr (phytree)	Create Newick formatted string

pdist (phytree)	Calculate pairwise patristic distances in a phytree object
phytree (phytree)	Object constructor for a phylogenetic tree object
plot (phytree)	Draw a phylogenetic tree
prune (phytree)	Remove branch nodes from phylogenetic tree
reroot (phytree)	Change the root of a phylogenetic tree
select (phytree)	Select tree branches and leaves in phytree object
subtree (phytree)	Extract a subtree
view (phytree)	View phylogenetic tree
weights (phytree)	Calculate weights for a phylogenetic tree

Examples

Create phylogenetic tree for a set of multiply aligned sequences.

```
Sequences = multialignread('aagag.aln')
distances = seqpdist(Sequences)
tree = seqlinkage(distances)
phytreetool(tree)
```

See Also

Bioinformatics Toolbox functions `dnds`, `phytreetool`, `phytreewrite`, `seqlinkage`, `seqneighjoin`, `seqpdist`, and the phytree object methods `get (phytree)`, `select (phytree)`

Purpose

Read phylogenetic tree files

Syntax

`Tree = phytreeread(File)`

Arguments

`File`	Newick formatted tree files (ASCII text file). Enter a filename, a path and filename, 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 (phytree)`.

Description

`Tree = phytreeread(Filename)` 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 `gethmmtree`, `phytreetool`, `phytreewrite` and the `phytree` object method `phytree (phytree)`

Purpose

View, edit, and explore phylogenetic tree data

Syntax

```
phytreetool(Tree)
phytreetool(File)
```

Arguments

Tree	`Phytree` object created with the function `phytree (phytree)` or `dnds`.
File	Newick or ClustalW tree formatted file (ASCII text file) with phylogenetic tree data. Enter a filename, a path and filename, 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 `dnds`, `phytreewrite` and the phytree object methods `phytree (phytree)`, `plot (phytree)`, `view (phytree)`

Purpose

Write phylogenetic tree object to Newick formatted file

Syntax

```
phytreewrite('File', Tree)
phytreewrite(Tree)
```

Arguments

File	Newick formatted file. Enter either a filename or a path and filename supported by your operating system (ASCII text file).
Tree	Phylogenetic tree object. `Tree` must be an object created with either the function `phytree (phytree)` or imported using the function `dnds`.

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 filename.

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 `dnds`, `phytreetool`, `seqlinkage`, and the phytree object methods `phytree (phytree)`, getnewickstr

Purpose

Read data from PIR file

Syntax

```
PIRData = pirread('File')
pirread('String')
```

Arguments

`File`	Protein Information Resource (PIR-PSD) formatted file (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *`File`* can also be a MATLAB character array that contains the text for a PIR-PSD file.
`String`	Character string with PIR data.

Description

`PIRData = pirread('`*`File`*`')` reads data from a Protein Information Resource (PIR-PSD) formatted file *`File`* and creates a MATLAB structure `PIRData` with the following fields:

```
Entry
EntryType
Title
Organism
Date
Accessions
Reference
Genetics
Classification
Keywords
Feature
Summary
Sequence: [1x105 char]
```

`pirread('`*`String`*`')` attempts to retrieve PIR data from the string `String`.

For more information on the PIR-PSD database, see

```
http://pir.georgetown.edu
```

Examples

Get protein information for cytochrome C from the PIR-PSD database, save the information in the file `cchu.txt`, and then read the information back into MATLAB.

```
getpir('cchu', 'ToFile', 'cchu.txt')
pirdata = pirread('cchu.txt')
```

See Also

Bioinformatics Toolbox functions `genpeptread`, `getpir`, `pdbread`

Purpose

Draw a 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`	`phytree` object created with the function `phytree (phytree)`
`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`.

plot (phytree)

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 `dnds`, `phytreetool`, `seqlinkage`

`phytree` object methods `phytree (phytree)`, `view (phytree)`

Purpose

Extract probe set library information for probe results

Syntax

```
ProbeInfo = probelibraryinfo(CELStruct, CDFStruct)
```

Description

`ProbeInfo = probelibraryinfo(CELStruct, CDFStruct)`creates a table of information linking the probe data in a `CEL` file structure with probe set information from a `CDF` file structure.

`ProbeInfo` is a matrix with three columns and the same number of rows as the probes field of the `CELStruct`. The first column is the probe set `ID` number to which the corresponding probe belongs. The second column contains the probe pair number and the third column indicates if the probe is a perfect match (`1`) or mismatch (`-1`) probe. Probes that do not correspond to a probe set in the CDF library file have probe set `ID` equal to `0`.

Note: Affymetrix probe pair indexing is `0` based while MATLAB indexing is `1` based. The output from `probelibraryinfo` is `1` based.

Examples

1 Get the file Drosophila-121502.cel from

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

2 Read the data into MATLAB.

```
celStruct = affyread('Drosophila-121502.cel');
cdfStruct = affyread('D:\Affymetrix\LibFiles\...
                     DrosGenome1\DrosGenome1.CDF');
```

3 Extract probe set library information.

```
probeinfo = probelibraryinfo(celStruct,cdfStruct);
```

4 Find out which probeset the 1104th probe belongs to

```
cdfStruct.ProbeSets(probeinfo(1104,1)).Name
```

See Also

Bioinformatics Toolbox functions `affyread`, `probesetlink`, `probesetlookup`, `probesetvalues`

probesetlink

Purpose

Link to NetAffx Web site

Syntax

```
probesetlink(AFFYStruct, ID)
URL = probesetlink(AFFYStruct, ID)
probesetlink(..., 'Source', SourceValue)
probesetlink(..., 'Browser', BrowserValue)
URL = probesetlink(..., 'NoDisplay', NoDisplayValue)
```

Description

`probesetlink(AFFYStruct, ID)` displays information from the NetAffx Web site about probe set `ID` from the `CHP` or `CDF` structure `AFFYStruct`. `ID`can be the index of the probe set or the probe set name.

`URL = probesetlink(AFFYStruct, ID)` returns the URL for the information.

`probesetlink(..., 'Source', *SourceValue*)` when `Source` is `true`, links to the data source (e.g. GenBank, Flybase) for the probe set.

`probesetlink(..., 'Browser', *BrowserValue*)` when `Browser` is `true`, displays the information in the system Web browser.

`URL = probesetlink(..., 'NoDisplay', *NoDisplayValue*)` when `NoDisplay` is `true`, returns the URL but does not open a browser.

Note: NetAffx Web site requires you to register and provide a user name and password.

Examples

1 Get the file Drosophila-121502.chp from

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

2 Read the data into MATLAB.

```
chpStruct = affyread('Drosophila-121502.chp',...
                     'D:\Affymetrix\LibFiles\DrosGenome1')
```

3 Displays information from the NetAffx Web site.

```
probesetlink(chpStruct,'AFFX-YEL018w/_at');
```

See Also

Bioinformatics Toolbox functions `affyread`, `probesetlookup`, `probesetplot`, `probelibraryinfo`, `probesetvalues`

probesetlookup

Purpose

Look up gene name for probe set

Syntax

```
probesetlookup(AFFYStruct, ID)
probesetlookup(AFFYStruct, Name)
[Name, NDX, Description, Source, SourceURL] = probesetlookup(...)
```

Description

`probesetlookup(AFFYStruct, ID)` returns the gene name for a probe set `ID` from a `CHP` or `CDF` structure (`AFFYStruct`).

`probesetlookup(AFFYStruct, Name)` returns the probe set ID for a gene name (`Name`) from a `CHP` or `CDF` structure (`AFFYStruct`).

`[Name, NDX, Description, Source, SourceURL] = probesetlookup(...)` returns the name, index into the `CHP` or `CDF` struct, , description, source, and source URL and for the probe set.

Examples

1 Get the file Drosophila-121502.chp from

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

2 Read the data into MATLAB.

```
chpStruct = affyread('Drosophila-121502.chp',...
                     'D:\Affymetrix\LibFiles\DrosGenome1')
```

3 Get the gene name.

```
probesetlookup(chpStruct,'AFFX-YEL018w/_at')
```

See Also

Bioinformatics Toolbox functions `affyread`, `probesetlink`, `probesetplot`, `probelibraryinfo`

Purpose

Plots values for Affymetrix CHP file probe set

Syntax

```
probesetplot(CHPStruct, ID, 'PropertyName', PropertyValue)
probesetplot(..., 'GeneName', GeneNameValue)
probesetplot(..., 'Field', FieldValue)
probesetplot(..., 'ShowStats',ShowStatsValue)
```

Description

`probesetplot(CHPStruct, ID, 'PropertyName', PropertyValue)` plots the PM and MM intensity values for probe set `ID`. `CHPStruct` is a structure created from an Affymetrix `CHP` file. `ID` can be the index of the probe set or the probe set name. Note: the probe set numbers for a `CHP` file use `0` based indexing while MATLAB uses `1` based indexing. `CHPStruct.ProbeSets(1)` has ProbeSetNumber `0`.

`probesetplot(..., 'GeneName', GeneNameValue)` when `GeneName` is `true`, uses the gene name, rather than the probeset name for the title.

`probesetplot(..., 'Field', FieldValue)` shows the data for a field (`FieldValue`). Valid fieldnames are: `Background`, `Intensity`, `StdDev`, `Pixels`, and `Outlier`.

`probesetplot(..., 'ShowStats',ShowStatsValue)` when `ShowStats` is `true`, adds mean and standard deviation lines to the plot.

Examples

1 Get the file Drosophila-121502.chp from

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

2 Read the data into MATLAB.

```
chpStruct = affyread('Drosophila-121502.chp',...
                     'D:\Affymetrix\LibFiles\DrosGenome1')
```

3 Plots PM and MM intensity values.

```
probesetplot(chpStruct,'AFFX-YEL018w/_at','showstats',true);
```

See Also

Bioinformatics Toolbox functions `affyread`, `probesetlink`, `probesetlookup`

Purpose Extract probe set values from probe results

Syntax

```
PSValues = probesetvalues(CELStruct, CDFStruct, PS)
```

Description

`PSValues = probesetvalues(CELStruct, CDFStruct, PS)` creates a table of values for a probe set (`PS`) from the probe data in a `CEL` file structure (`CELStruct`). `PS` is a probe set index or probe set name from the CDF library file structure (`CDFStruct`). `PSValues` is a matrix with `18` columns and one row for each probe pair in the probe set. The columns correspond to the fields in a CHP probe set data structure:

```
'ProbeSetNumber'
     'ProbePairNumber'
     'UseProbePair'
     'Background'
     'PMPosX'
     'PMPosY'
     'PMIntensity'
     'PMStdDev'
     'PMPixels'
     'PMOutlier'
     'PMMasked'
     'MMPosX'
     'MMPosY'
     'MMIntensity'
     'MMStdDev'
     'MMPixels'
     'MMOutlier'
     'MMMasked'
```

There are some minor differences between the output of this function and the data in a CHP file. The PM and MM Intensity values in the CHP file are normalized by the Affymetrix software. This function returns the raw intensity values. The `'UseProbePair'` and `'Background'` fields are only returned by this function for compatibility with the CHP probe set data structure and are always set to zero.

Examples

1 Get the file Drosophila-121502.cel from

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

2 Read the data into MATLAB.

```
celStruct = affyread('Drosophila-121502.cel');
cdfStruct = affyread('D:\Affymetrix\LibFiles\DrosGenome1\...
                         DrosGenome1.CDF');
```

3 Get the values for probe set 147439_at.

```
psvals = probesetvalues(celStruct,cdfStruct,'147439_at')
```

See Also

Bioinformatics Toolbox functions `affyread`, `probelibraryinfo`, `probesetlink`, `probesetlookup`

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 `ExtendedGap` 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

Display 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` 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 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.

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`, `msalign`, `pdbdistplot`, `pdbplot`, `seqtool`.

MATLAB function `plotyy`

prune (phytree)

Purpose

Remove branch nodes from phylogenetic tree

Syntax

```
T2 = prune(T1, Nodes)
T2 = prune(T1, Nodes, 'Mode','Exclusive')
```

Arguments

`T1`	Phylogenetic tree object. See `phytree (phytree)`.
`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 :
```

Remove all the 'mouse' proteins use

```
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 function `phytree (phytree)`, `phytreetool`, and methods for the `phytree` object select and get

quantilenorm

Purpose

performs quantile normalization over multiple arrays

Syntax

Description

NORMDATA = QUANTILENORM(DATA), where the columns of DATA correspond to separate chips, normalizes the distributions of the values in each column. Note that if DATA contains NaN values, then NORMDATA will also contain NaNs 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`.

Purpose

Draw Ramachandran plot for PDB data

Syntax

```
ramachandran('PDBid')
ramachandran('File')
ramachandran(PDBData)
Angles = ramachandran(...)
[Angles, Handle] = ramachandran(...)
```

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.
File	Protein Data Bank (PDB) formatted file (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *`File`* can also be a MATLAB character array that contains the text for a PDB file.
`PDBData`	MATLAB structure with PDB formatted data.

Description

`ramachandran` generates a plot of the torsion angle PHI (torsion angle between the `'C-N-CA-C'` atoms) and the torsion angle PSI (torsion angle between the `'N-CA-C-N'` atoms) of the protein sequence.

`ramachandran(PDBid)` generates the Ramachandran plot for the protein with PDB code `ID`.

`ramachandran('File')` generates the Ramachandran plot for protein stored in the PDB file *`File`*.

`ramachandran(PDBData)` generates the Ramachandran plot for the protein stored in the structure `PDBData`, where `PDBData` is a MATLAB structure obtained by using `pdbread` or `getpdb`.

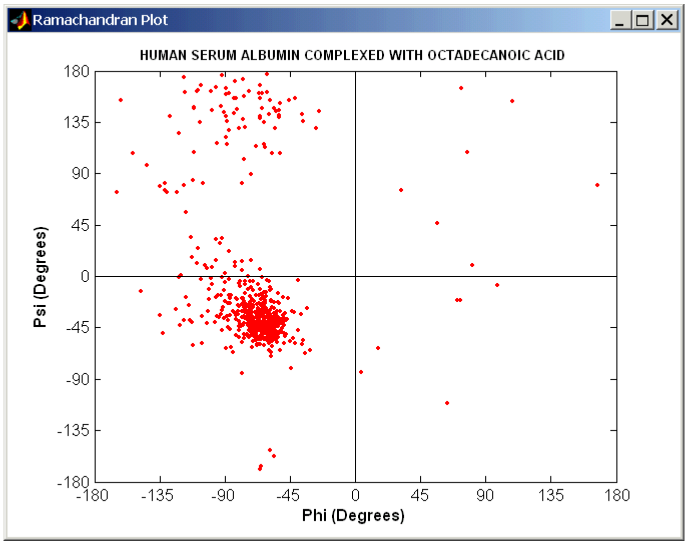
`Angles = ramachandran(...)` returns an array of the torsion angles PHI, PSI, and OMEGA for the residue sequence.

`[Angles, Handle] = ramachandran(...)` returns a handle to the plot.

Examples

Generate the Ramachandran plot for the human serum albumin complexed with octadecanoic acid.

```
ramachandran('1E7I')
```



See Also

Bioinformatics Toolbox functions `getpdb`, `pdbdistplot`, `pdbread`, `pdbplot`

Purpose

Generate a 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`, `rankfeatures`, `svmclassify`

Statistical Toolbox function `classify`

randseq

Purpose

Generate random sequence from finite alphabet

Syntax

```
Seq = randseq(Length, 'PropertyName', PropertyValue)

randseq(..., 'Alphabet', AlphabetValue)
randseq(..., 'Weights', WeightsValue)
randseq(..., 'FromStructure', FromStructureValue)
randseq(..., 'Case',CaseValue)
randseq(..., 'DataType', DataTypeValue)
```

Arguments

`Length`	
`AlphabetValue`	Property to select the alphabet for the sequence. Enter `'dna'`, `'rna'`, or `'amino'`. The default value is `'dna'`.
`WeightsValue`	Property to specify a weighted random sequence.
`FromStructureValue`	Property to specify a weighted random sequence using output structures from the functions `basecount`, `dimercount`, `codoncount`, or `aacount`.
`CaseValue`	Property to select the case of letters in a sequence when `Alphabet` is `'char'`. Values are`'upper'` or `'lower'`. The default value is `'upper'`.
`DataTypeValue`	Property to select the data type for a sequence. Values are `'char'` for letter sequences, and `'uint8'` or `'double'` for numeric sequences. Creates a sequence as an array of *`DataType`*. The default data type is `'char'`.

Description

`randseq(...,'Alphabet', AlphabetValue)` generates a sequence from a specific alphabet.

`randseq(..., 'Weights', WeightsValue)` creates a weighted random sequence where the `i`th 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 `i`th element of the nucleotide alphabet is given by `int2nt(i)`, and the `i`th element of the amino acid alphabet is given by `int2aa(i)`.

`randseq(..., 'FromStructure', FromStructureValue)` creates a weighted random sequence with weights given by the output structure from `basecount`, `dimercount`, `codoncount`, or `aacount`.

`randseq(..., 'Case', CaseValue)` specifies the case for a letter sequence.

`randseq(...,'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`,

Purpose

Rank key features by class separability criteria

Syntax

```
[IDX, Z] = rankfeatures(X, Group)
rankfeatures(..., 'PropertyName', PropertyValue,...)
rankfeatures(..., 'Criterion', CriterionValue)
rankfeatures(..., 'CCWeighting', ALPHA)
rankfeatures(..., 'NWeighting', BETA)
rankfeatures(..., 'NumberOfIndices', N)
rankfeatures(..., '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`.

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

`rankfeatures(..., 'Criterion', CriterionValue)`sets the criterion used to assess the significance of every feature for separating two labeled groups. Options 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 under the empirical receiver operating characteristic (ROC) curve
`'wilcoxon'`	Absolute value of the u-statistic of a two-sample unpaired Wilcoxon test, also known as Mann-Whitney

Notes: 1) `'ttest'`, `'entropy'`, and `'brattacharyya'` assume normal distributed classes while `'roc'` and `'wilcoxon'` are nonparametric tests. 2) All tests are feature independent.

`rankfeatures(..., '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.

`rankfeatures(..., '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.

`rankfeatures(..., '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.

`rankfeatures(..., '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. 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))`

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
```

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
```

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')
```

See Also

Statistical Toolbox functions `classify`, `classperf`, `crossvalind`, `randfeatures`, `svmclassify`

Purpose

Find restriction enzymes that cut a protein sequence

Syntax

```
[Enzymes, Sites] = rebasecuts(SeqNT)
rebasecuts(SeqNT, Group)
rebasecuts(SeqNT, [Q, R])
rebasecuts(SeqNT, S)
```

Arguments

SeqNT	Amino acid 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 an amino acid 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
```

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`, `seq2regexp`, `seqshowwords`, `restrict`

MATLAB function `regexp`

Purpose

Display a red and green colormap

Syntax

```
redgreencmap(Length)
```

Arguments

`Length`	Length of the colormap. Enter either `256` or `64`. The default value is the length of the colormap of the current figure.

Description

`redgreencmap(Length)` returns an `M`-by-3 matrix containing a red and green colormap. Low values are bright green, values in the center of the map are black, and high values are red.

`redgreencmap`, by itself, is the same length as the current colormap.

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

MATLAB functions `colormap`, `colormapeditor`

reroot (phytree)

Purpose

Change the root of a phylogenetic tree

Syntax

```
Tree2 = reroot(Tree1)
Tree2 = reroot(Tree1, Node)
Tree2 = reroot(Tree1, Node, Distance)
```

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.

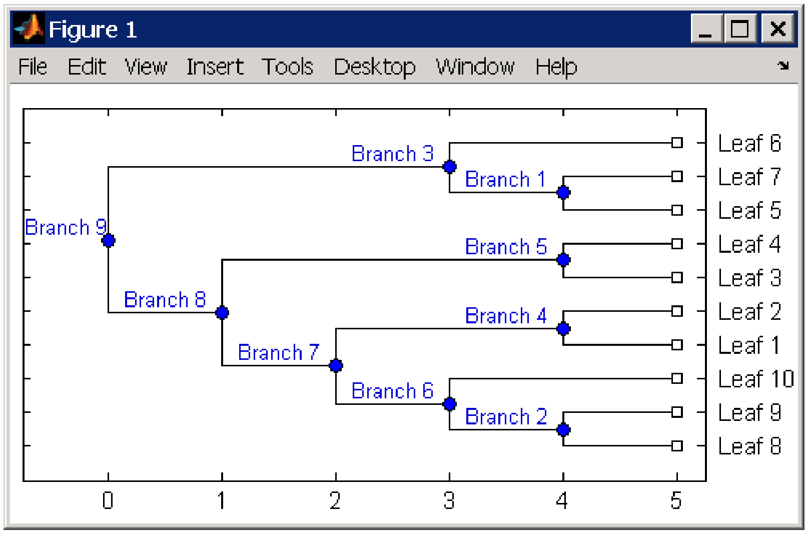
`Tree2 = reroot(Tree1, Node, Distance)` changes the root of a phylogenetic tree (*`Tree1`*) to a new root at a given distance (*`Distance`*) from the reference branch node (*`Node`*) toward the original root of the tree. Note: The new branch representing the root in the new tree (`Tree2`) is labeled `'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)
```

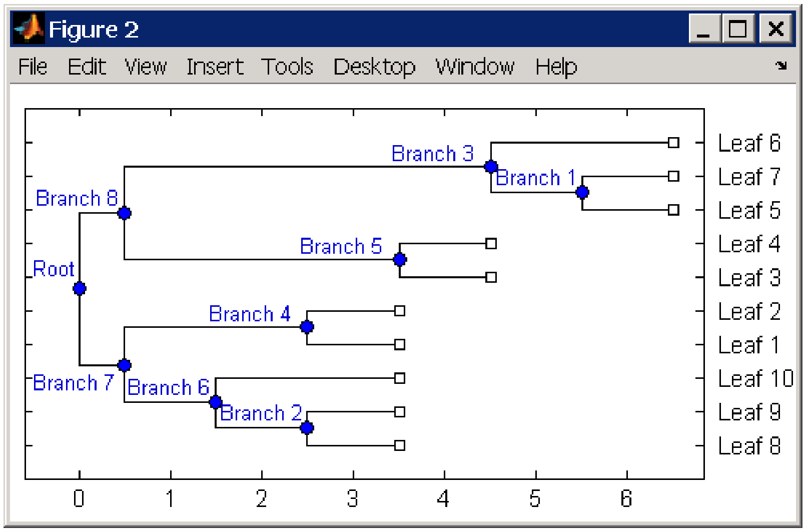
MATLAB draws a figure with the phylogenetic tree.



2 Place the root at `'Branch 7'`.

```
sel = getbyname(tr_1,'Branch 7');
tr_2 =  reroot(tr_1,sel)
plot(tr_2,'branchlabels',true)
```

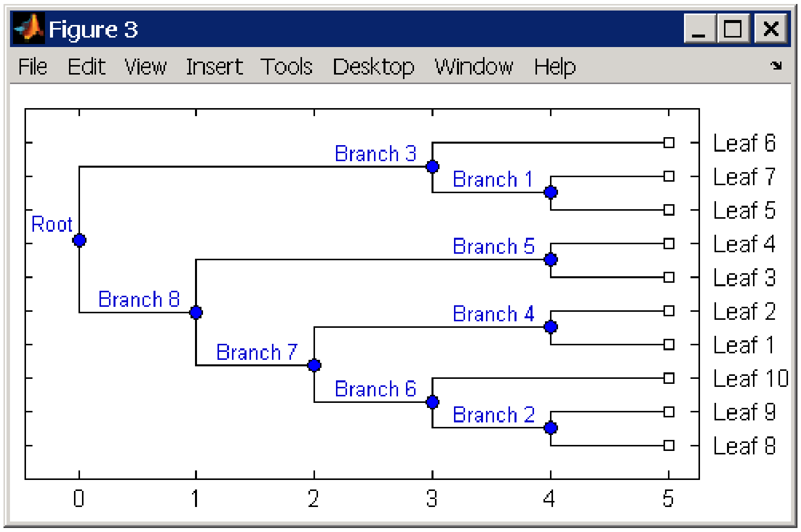
MATLAB draws a tree with the root moved to the center of branch 7.



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

Phytree methods `phytree`, `get`, `getbyname`, `prune`, `select`, `seqneighjoin`

Purpose

Split nucleotide sequence at specified restriction site

Syntax

```
Fragments = restrict(SeqNT, Enzyme)
Fragments = restrict(SeqNT, Pattern, Position)
[Fragments, CuttingSites] = restrict(...)
[Fragments, CuttingSites, Lengths] = restrict(...)
restrict(..., 'PropertyName', PropertyValue,...)
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 SEQ sequence into fragments at the restriction sites of restriction enzyme (*`Enzyme`*). The return values are stored in a cell array of sequences.

`Fragments = restrict(SeqNT, Pattern, Position)` cuts a sequence (*SeqNT*) into fragments at specified 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(..., 'PropertyName', PropertyValue,...)` defines optional properties using property name/value pairs.

`restrict(..., 'PartialDigest', PartialDigestValue)` simulates a partial digest where each restriction site in the sequence has a probability `PartilDigest` of being cut.

REBASE, the restriction enzyme database, is a collection of information about restriction enzymes and related proteins. Search REBASE for the name of a restriction enzyme at

```
http://rebase.neb.com/rebase/rebase.html
```

For more information on REBASE, go to

```
http://rebase.neb.com/rebase/rebase.html
```

Example

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 function `cleave`, `seq2regexp`, `seqshowwords`, `rebasecuts`

MATLAB function `regexp`

revgeneticcode

Purpose

Get the reverse mapping for a genetic code

Syntax

```
map = revgeneticcode
revgeneticcode(GeneticCode,
               'PropertyName', PropertyValue ...)

revgeneticcode(..., 'Alphabet' AlphabetValue)
revgeneticcode(..., 'ThreeLetterCodes', CodesValue)
```

Arguments

`GeneticCode`	Enter a code number or code name from the table Genetic Code on page 2-320. 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'`.
`CodesValue`	Property to select one- or three-letter amino acid codes. Enter `true` for three-letter code or `false` for one-letter code.

Genetic Code

Code Number	Code Name
1	`Standard`
2	`Vertebrate Mitochondrial`
3	`Yeast Mitochondrial`

Code Number	Code Name
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

`map = revgeneticcode` returns a structure containing the reverse mapping for the standard genetic code.

`revgeneticcode(`*`GeneticCode`*`, '`*`PropertyName`*`',` *`PropertyValue`* `...)` returns a structure of the inverse mapping for alternate genetic codes.

`revgeneticcode(..., 'Alphabet' A`*`lphabetValue`*`)` defines the nucleotide alphabet to use in the map.

`revgeneticcode(..., 'ThreeLetterCodes',` *`CodesValue`*`)` 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',...
                          'ThreeLetterCode',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'}
    Starts: {'TAA'  'TAG'  'TGA'}
```

See Also

Bioinformatics Toolbox functions `aa2nt`, `aminolookup`, `baselookup`, `geneticcode`, `nt2aa`

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`.

Examples

```
rna2dna('ACGAUGAGUCAUGCUU')

ans =
ACGATGAGTCATGCTT
```

See Also

Bioinformatics Toolbox function `dna2rna`

MATLAB functions `strrep`, `regexp`

scfread

Purpose

Read trace data from SCF file

Syntax

```
[Sample, Probability, Comments] = scfread('File')
[A,C,T,G, ProbA, ProbC, ProbG, ProbT,
Comments] = scfread ('File')
```

Arguments

`File`	SCF formatted file. Enter a filename or a path and filename.

Description

`scfread` reads data from a SCF formatted file into a MATLAB structure.

`[Sample, Probability, Comments] = scfread('`*`File`*`')` reads an SCF formatted file and returns the sample data in the structure `Sample`, with fields `A`, `C`, `T`, `G`, probability data in the structure `Probability`, and comment information from the file in `Comments`.

`[A,C,T,G, ProbA, ProbC, ProbG, ProbT, Comments] = scfread ('`*`File`*`')` reads an SCF formatted file and returns the sample data and probabilities for nucleotides 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
```

Examples

Examples of SCF files can be found at

```
ftp://ftp.ncbi.nih.gov/pub/TraceDB/example/
```

Unzip the file `bcm-example.tgz` with SCF files to your MATLAB working directory.

```
[Sample, Probability, Comments] = scfread('HCIUP1D61207.scf')

Sample =
    A: [10827x1 double]
    C: [10827x1 double]
    G: [10827x1 double]
    T: [10827x1 double]

Probability =
    prob_A: [742x1 double]
    prob_C: [742x1 double]
    prob_G: [742x1 double]
    prob_T: [742x1 double]

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`

select (phytree)

Purpose

Select tree branches and leaves in phytree object

Syntax

```
S = select(T)
S = select(T, N)
[S, Selleaves, Selbranches] = select(...)

S = select(..., 'Reference', ReferenceValue)
S = select(..., 'Criteria', CriteriaValue)
S = select(..., 'Threshold', ThresholdValue)
S = select(..., 'Exclude', ExcludeValue)
S = select(..., 'Propagate', PropagateValue)
```

Arguments

Tree	Phylogenetic tree created with the function `phytree (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 = 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.

`S = 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.

`S = select(..., 'Threshold', ThresholdValue)` selects all the nodes where closeness is less than or equal to the threshold value `V`. Notice that 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`.

`S = select(..., 'Exclude', ExcludeValue)` sets a postfilter that excludes all the branch nodes from `S` when `E='branches'` or all the leaf nodes when `E='leaves'`. The default is `'none'`.

`S = 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.

`[S, Selleaves, Selbranches] = select(...)` returns two additional logical vectors, one for the selected leaves and one for the selected branches.

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

The Bioinformatics Toolbox functions `phytree (phytree)`, `phytreetool`

`phytree` object methods `get`, `pdist`, `prune`.

Purpose

Convert sequence with ambiguous characters to regular expression

Syntax

`seq2regexp(Seq)`

Arguments

`Seq`	Nucleotide or amino acid sequence.

Nucleotide Conversions

Nucleotide Letter	Nucleotide	Nucleotide Letter	Nucleotide
`A—A`	Adenosine	`S—[GC]`	(Strong)
`C—C`	Cytosine	`W—[AT]`	(Weak)
`G—G`	Guanine	`B—[GTC]`	
`T—T`	Thymidine	`D—[GAT]`	
`U—U`	Uridine	`H—[ACT]`	
`R—[GA]`	(Purine)	`V—[GCA]`	
`Y—[TC]`	(Pyrimidine)	`N—[AGCT]`	Any nucleotide
`K—[GT]`	(Keto)	`- — -`	Gap of indeterminate length
`M—[AC]`	(Amino)	`?—?`	Unknown

Amino Acid Conversion

Amino Acid Letter	Description
`B—[DN]`	Aspartic acid or asparagine

Amino Acid Letter	Description
`Z—[EQ]`	Glutamic acid or glutamine
`X—[ARNDCQEGHILKMFPSTWYV]`	Any amino acid

Description

`seq2regexp(Seq)` converts ambiguous nucleotide or amino acid symbols in a sequence into a regular expression format using IUB/IUPAC codes.

Examples

Convert a nucleotide sequence into a regular expression.

```
r = seq2regexp('ACWTMAN')

r =
AC[AT]T[AC]A[AGCT]
```

See Also

Bioinformatics Toolbox functions , `seqwordcount`

MATLAB functions `regexp`, `regexpi`

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`.

Examples

Return the complement of a DNA nucleotide sequence.

```
s = 'ATCG';
seqcomplement(s)

ans =
TAGC
```

See Also

Bioinformatics Toolbox functions `seqrcomplement`, `seqreverse`, `seqtool`

seqconsensus

Purpose

Calculate a 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`, , `profalign`,`seqdisp`, `seqlogo`, `seqprofile`

Purpose Format long sequence output for easy viewing

Syntax

```
seqdisp(Seq, 'PropertyName', PropertyValue ...)

seqdisp(..., 'Row', RowValue)
seqdisp(..., 'Column', ColumnValue)
seqdisp(..., 'ShowNumbers', ShownumbersValue)
```

Arguments

`Seq`	Nucleotide or amino acid sequence. Enter a character array, a FASTA filename, 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` or `false`. The default value is `true` to show numbers.

Description

`seqdisp(Seq, '`*`PropertyName`*`', `*`PropertyValue`*` ...)` displays a sequence (`Seq`) in rows with a default row length of 60 and a default column width of `10`.

`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('AK080777');
seqdisp(mouseHEXA, 'Row', 50, 'Column', 10)
```

Create and save a FASTA file with two sequences, and then display it.

```
hdr = ['Sequence A'; 'Sequence B'];
seq = ['TAGCTGRCCAAGGCCAAGCGAGCTTN';'ATCGACYGGTTCCGGTTCGCTCGAAN']
fastawrite('local.fa', hdr, seq);
seqdisp('local.fa', 'ShowNumbers', false')

ans =
>Sequence A
 1  TAGCTGRCCA AGGCCAAGCG AGCTTN
>Sequence B
 1  ATCGACYGGT TCCGGTTCGC TCGAAN
```

See Also

Bioinformatics Toolbox function `multialignread`, `seqconsensus`, `seqlogo`, `seqprofile`, `seqshoworfs`, `seqshowwords`, `seqtoolgetgenbank`

Purpose

Create dot plot of two sequences

Syntax

```
seqdotplot(Seq1,Seq2)
seqdotplot(Seq1,Seq2, Window, Number)
```

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`.

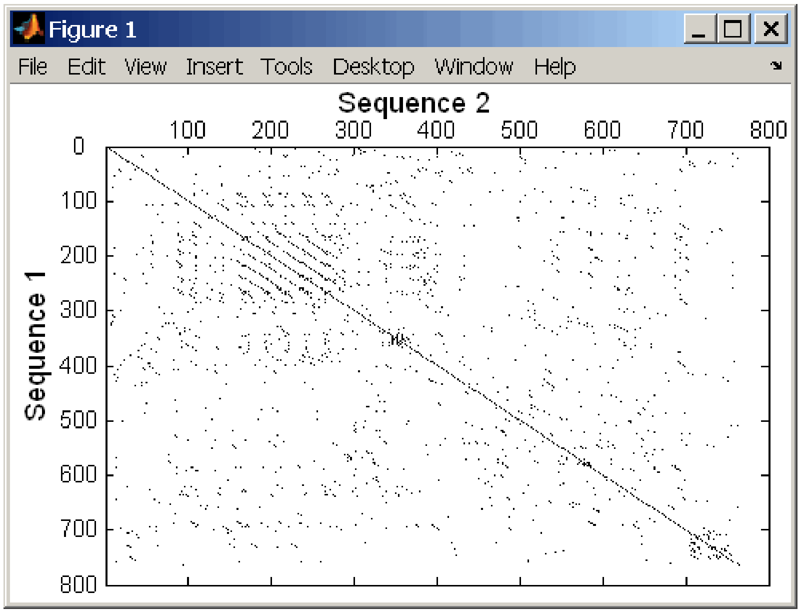
`Matches = seqdotplot(...)` returns the number of dots in the dot plot matrix.

[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);
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

Construct phylogenetic tree from pairwise distances

Syntax

```
Tree = seqlinkage(Dist)
Tree = seqlinkage(Dist, Method)
Tree = seqlinkage(Dist, Method, Names)
```

Arguments

`Dist`	Pairwise distances generated from the function `seqpdist`.
`Method`	Property to select a distance method. Enter a method from the table below.
`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) such as is generated by the function `seqpdist`.

`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

The Bioinformatics Toolbox functions `phytree (phytree)`, `phytreewrite`, `seqpdist`, `seqneighjoin`

`phytree` object methods `plot` and `view`

seqlogo

Purpose Display sequence logo for nucleotide and amino acid sequences

Syntax

```
seqlogo(Seqs)
seqlogo(Profile)
DiplayInfo = seqlogo(Seqs)
DisplayInfo = seqlogo(..., 'Displaylogo', DisplaylogoValue).
seqlogo(..., 'Alphabet', AlphabetValue)
seqlogo(..., 'Startat', StartatValue)
seqlogo(..., 'Endat', EndatValue)
seqlogo(..., 'SSCorrection', SSCorrectionValue).
```

Arguments

`Seqs`	Set of pairwise or 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`.
`Displaylogo`	Property to control drawing a sequence logo. Enter either `true` or `false`.

Description

`seqlogo(`*`Seqs`*`)` displays a sequence logo for a set of aligned sequences (`Seqs`). 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.

`seqlogo(`*`Profile`*`)` displays a sequence logo for a sequence profile (P) retruned by the function seqprofile.

`Profile`	For amino acids, frequency distribution matrix of size `[20 x sequence length]`. For nucleotides, matrix of size `[4 x sequence length]` using the DNA alphabet. If gaps were included, `Profile` may have `21` (or `5`) rows , but `seqlogo` ignores gaps.

The alphabet for nucleic acids is colored as follows

`A`	Green
`C`	Blue
`G`	Yellow
`T`, `U`	Red

The alphabet for proteins is colored according to chemical property as follows

`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

Ambiguous symbols not in the list above are added to the logo and colored purple.

`DiplayInfo = seqlogo(Seqs)`returns a cell array of unique symbols in a sequence (`Seqs`) and the information weight matrix used for graphically displaying the logo.

`DisplayInfo = seqlogo(..., 'Displaylogo', `*`DisplaylogoValue`*`).` when `Displaylogo` is `false`, returns display information, but does not draw the sequence logo.

`seqlogo(..., 'Alphabet', `*`AlphabetValue`*`)` selects the alphabet for nucleotide sequences (`'NT'`) or amino acid sequences (`'AA'`). The default is `'NT'`. If you provide amino acid sequences to `seqlogo`, you must select `'AA'` for the `Alphabet`.

`seqlogo(..., 'Startat', `*`StartatValue`*`)` specifies the starting position for the sequences (`Seqs`). The default starting position is `1`.

`seqlogo(..., 'Endat', `*`EndatValue`*`)` specifies the ending position for the sequences (`Seqs`). The default ending position is the maximum length of the sequences (`Seqs`).

`seqlogo(..., 'SSCorrection', SSCorrectionValue).` when `SSCorrection` is `false`, no estimation is made for the number of bits. A simple calculation of bits tends to overestimate the conservation at a particular location. To compensate for this overestimation, when `SSCorrection` is `true`, a rough estimate is applied as an approximate correction. This correction works better when the number of sequences is greater than 50. The default is `true`.

Reference

Schneider, T.D., Stephens, R.M., "Sequence Logos: A new way to display consensus sequences," Nucleic Acids Research, Vol. 18, pp. 6097-6100, 1990.

Examples

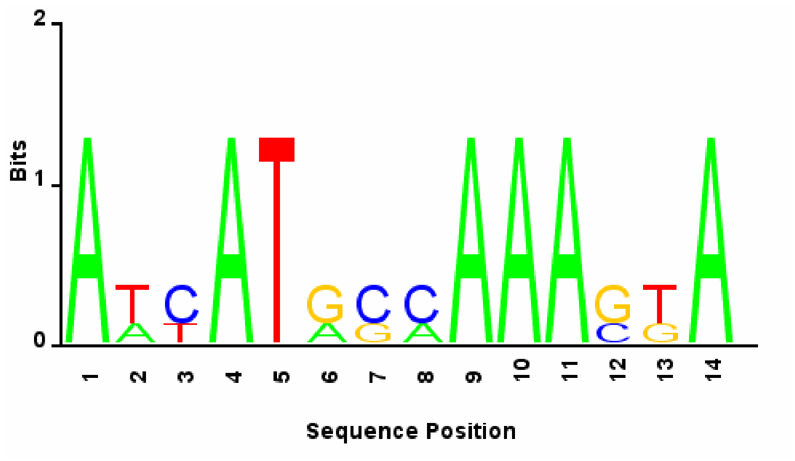
1 Get a series of aligned sequences.

```
S = {'ATTATAGCAAACTA',...
     'AACATGCCAAAGTA',...
     'ATCATGCAAAAGGA'}
```

2 Display the sequence logo.

```
seqlogo(S)
```

MATLAB draws a figure.



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)
```

???LI: you have to explain this in more detail to me someday???

See Also

Bioinformatics Toolbox functions `seqconsensus`, `seqdisp`, `seqprofile`

Purpose

Find matches for every string in a 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 `strmatch`, `regexpi`

Purpose

Neighbor-joining method for phylogenetic tree reconstruction

Syntax

```
Tree = seqneighjoin(Dist)
Tree = seqneighjoin(Dist, Method)
Tree = seqneighjoin(Dist, Method, Names)
seqneighjoin(..., 'PropertyName', PropertyValue,...)
seqneighjoin(..., 'Reroot', RerootValue)
```

Arguments

`Dist`	Matrix or vector returned by the function seqpdist
`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 pairwise distances (*`Dist`*) between the species or products using the neighbor-joining method.

`Tree` `= seqneighjoin(`*`Dist`*`,` *`Method`*`)` selects a method (*`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).

The following table describes the values for *`Method`*.

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

`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.

References

[1] [1] Saitou N, Nei M, “The neighbor-joining method: a new method for reconstructing phylogenetic trees” in Mol Biol Evol. (1987) 4(4):406-25.

[2] [2] Gascuel O, “BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data” in Mol. Biol. Evol. (1997) 14:685-695.

[3] [3] Studier JA, Keppler KJ, “A note on the neighbor-joining algorithm of Saitou and Nei” in Mol Biol Evol. (1988) 5(6):729-31.

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)
   ```

See Also

Bioinformatics Toolbox function `multialign`.

Phylogenetic tree tool functions `seqlinkage` (alternative method to create a phylogenetic tree) and `seqpdist`.

Phylogenetic tree object methods `phytree` (object constructor), `reroot`, and `view`

seqpdist

Purpose

Calculate pairwise distance between sequences

Syntax

```
D = seqpdist(Seqs,'PropertyName', PropertyValue ...)

seqpdist(..., 'Method', MethodValue)
seqpdist(..., 'Indels', IndelsValue)
seqpdist(..., 'Optargs', OptargsValue)
seqpdist(..., 'PairwiseAlignment',PairwiseAlignmentValue)
seqpdist(..., 'Squareform', SquareformValue)
seqpdist(..., 'Alphabet', AlphabetValue)

seqpdist(..., 'ScoringMatrix', ScoringMatrixValue)
seqpdist(..., 'Scale', ScaleValue
seqpdist(..., 'GapOpen', GapOpenValue)
seqpdist(..., 'ExtendGap', ExtendGapValue)
```

Arguments

`Seqs`	Cell array with nucleotide or amino acid sequences.
`Method`	Property to select the method for calculating pairwise distances.
`Indels`	Property to indicate treatment of gaps.
`Optargs`	Property to pass required arguments by the distance method selected with the property `Method`.
`PairwiseAlignment`	Property to force pairwise alignment.
`SquareForm`	Property to control formatting the output as a square or triangular matrix.
`Alphabet`	Property to select an alphabet. Enter either `'NT'` for nucleotides or `'AA'` for amino acids.
`ScoringMatrix`	Property to select a scoring matrix for pairwise alignment.

`Scale`	Property to select a scale factor for the scoring matrix.
`GapOpen`	Property to select a gap penalty.
`ExtendedGap`	Property to select a penalty for extending a gap.

Description

`D = seqpdist(Seqs, 'PropertyName', PropertyValue ...)` returns a vector `D` containing biological distances between each pair of sequences stored in the `M` elements of the cell `Seqs`.

`D` is an 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)`. `Seqs` can also be a vector of structures with the field `Sequence` or a matrix of `chars`.

`seqpdist(..., 'Method', MethodValue)` selects a method (`MethodValue`) to compute distances between every pair of sequences.

Distances defined for both nucleotides and amino acids:

`'p-distance'`	Proportion of sites at which the two sequences are different. `p` —> `1` for poorly related and `p` —> `0` for similar sequences.
`'Jukes-Cantor'` (default)	Maximum likelihood estimate of the number of substitutions between two sequences. For NT `d = -3/4 log(1p * 4/3)` AA `d = -19/20 log(1p * 20/19)`
`'alignment-score'`	Distance (`d`) between two sequences (1 and 2) is computed from the pairwise alignment score (`s`) as follows:

	`d(1,2) = (1-s(1,2)/s(1,1)) * (1-s(1,2)/s(2,2))` 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 `s(x,y)>s(x,x)`, then `d(x,y)=0`.

Distances defined only for nucleotides and no scoring of gaps:

`'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]`.
`'Kimura'`	Considers separately the transitional and transversion nucleotide substitution.
`'Tamura'`	Considers separately the transitional and transversion nucleotide substitution and the GC content. GC content can be computed from the input sequences or given by setting `'OPTARGS'`.

`'Hasegawa'`	Considers separately the transitional and transversional nucleotide substitution and the background nucleotide frequencies. Background frequencies can be computed from the input sequences or given by setting `'OPTARGS'` to `[gA gC gG gT]`.
`'Nei-Tamura'`	Considers separately the transitional substitution between purines, the transitional substitution between pyramidines and the transversional substitution and the background nucleotide frequencies. Background frequencies can be computed from the input sequences or given by setting `'OPTARGS'` to `[gA gC gG gT]`.

Distances defined only for amino acids and no scoring of gaps:

`'Poisson'`	Asumes 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'`. `'a'` can be set by `'OPTARGS'`. The default value is `2`.

A user defined distance function can also be specified using @, for example, @distfun, the distance function must be of the form:

```
function D = distfun(S1, S2, OPTARGS)
```

Taking as arguments two same-length sequences (NT or AA) plus zero or more additional problem-dependent arguments in `OPTARGS`, and returning a scalar that represents the distance between `S1` and `S2`.

`seqpdist(..., 'Indels', IndelsValue)` indicates how to treat sites with gaps. Options 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 at the input Seqs.

`seqpdist(..., 'Optargs', OptargsValue)` some distance methods require or accept optional arguments. Use a cell array to pass more than one input argument (for example, The nucleotide frequencies in the Tajima-Nei distance function can be specified instead of computing them from the input sequences).

`seqpdist(..., 'PairwiseAlignment', PairwiseAlignmentValue)`, when `PairwiseAlignment` is `true`, ignores multiple alignment of the input sequences (if any) and forces a pairwise alignment of input sequences. If the input sequences are not prealigned, this flag is set automatically. Pairwise alignment can be slow for a large number of sequences. The default value is `false`.

`seqpdist(..., 'Squareform', SquareformValue)`, when `SquareForm` is `true`, converts the output into a square formatted matrix so the `D(I,J)` denotes the distance between the `I`th and `J`th sequences. The output matrix is symmetric and has a zero diagonal. Setting the property `Squareform` to true is the same as using the function `squareform` in the Statistical Toolbox.

`seqpdist(..., 'Alphabet', AlphabetValue)` specifies whether the sequences are amino acids (`'AA'`) or nucleotides (`'NT'`). The default value is `'AA'`.

The remaining input properties are analogous to the function `nwalign` and are used when the property `PairwiseAlignment = true` or the property `Method = 'alignment-score'`. For more information about these properties, see `nwalign`.

`seqpdist(..., 'ScoringMatrix', *ScoringMatrixValue*)` specifies the scoring matrix to be used for the alignment. The default value is `BLOSUM50` for AA and `NUC44` for NT.

`seqpdist(..., 'Scale', *ScaleValue*)` indicates the scale factor of the scoring matrix to return the score using arbitrary units. If the scoring matrix info also provides a scale factor, then both are used.

`seqpdist(..., GapOpen', *GapOpenValue*)` specifies the penalty for opening a gap in the alignment. The default gap open penalty is 8.

`seqpdist(..., 'ExtendGap', *ExtendGapValue*)` specifies the penalty for extending a gap in the alignment. If `ExtendGap` is not specified, then extensions to gaps are scored with the same value as `GapOpen`.

Examples

```
% Load a multiple alignment of amino acids:
seqs = fastaread('pf00002.fa');

% For every possible pair of sequences in the multiple
% alignment removes sites with gaps and scores with the
% substitution matrix PAM250:

dist = seqpdist(seqs,'method','alignment-score',...
                'indels','pairwise-delete',...
                'scoringmatrix','pam250')

% To 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)

% To 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`, `seqlinkage`, and `phytree` methods `phytree`, `pdist`

Purpose

Calculate a sequence profile from a 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.

Examples

```
seqs = fastaread('pf00002.fa');
[P,S] = seqprofile(seqs,'limits',[50 60],'gaps','all')
```

See Also

Bioinformatics Toolbox functions `fastaread`, , , `seqdisp`, `seqlogo`

Purpose

Calculate reverse complement of a 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`

Purpose

Reverse the letters or numbers in a 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`

seqshoworfs

Purpose

Display open reading frames in a sequence

Syntax

```
seqshoworfs(SeqNT, 'PropertyName', PropertyValue)

seqshoworfs(..., 'Frames', FramesValue)
seqshoworfs(..., 'GeneticCode', GeneticCodeValue)
seqshoworfs(..., 'MinimumLength', MinimumLengthValue)
seqshoworfs(..., 'AlternativeStartCodons', StartCodonsValue)
seqshoworfs(..., 'Color', ColorValue)
seqshoworfs(..., '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 `geneticcode`.
`MinimumLengthValue`	Property to set the minimum number of codons in an ORF.
`StartCodonsValue`	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(..., '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(..., 'GeneticCode', *GeneticCodeValue*)` specifies the genetic code to use for finding open reading frames.

`seqshoworfs(..., 'MinimumLength', *MinimumLengthValue*)` sets the minimum number of codons for an ORF to be considered valid. The default value is `10`.

`seqshoworfs(..., 'AlternativeStartCodons', *StartCodonsValue*)` 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 of alternative start codons, see

```
http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/
wprintgc.cgi?mode=t#SG1
```

`seqshoworfs(..., '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(..., '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`, `geneticcode`, `seqdisp`,`seqshowwords`, `seqwordcount`, `cpgisland`, `seqtool`

MATLAB function `regexp`

Purpose

Graphically display the words in a sequence

Syntax

```
seqshowwords(Seq, Word)
seqshowwords(..., 'PropertyName', PropertyValue,...)
seqshowwords(..., 'Color', ColorValue)
seqshowwords(..., 'Columns', ColumnsValue)
seqshowwords(..., '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 (`0255`) 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(..., '`*`PropertyName`*`', `*`PropertyValue`*`,...)` defines optional properties using property name/value pairs.

`seqshowwords(..., 'Color', `*`ColorValue`*`)` selects the color used to highlight the words in the output display.

seqshowwords(..., 'Columns', *ColumnsValue*) specifies how many columns per line to use in the output.

seqshowwords(..., '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,, seqdisp, seqtool, seqwordcount

MATLAB functions strfind, regexp

seqtool

Purpose Open interactive tool to explore biological sequences

Syntax

```
seqtool(Seq)
seqtool(..., 'PropertyName', PropertyValue,...)
seqtool(..., 'Alphabet', AlphabetValue)
```

Arguments

Seq	Struct with a field Sequence, a character array, or a filename 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`*). The default value is `'AMINO'`except when all of the symbols in the sequence are in `A`, `C`, `G`, `T`, or `-`, then *`AlphabetValue`* is set to `'NT'`. Use 'AMINO' when you want to force an amino acid sequence alphabet.

Example

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

See Also

`aa2nt`, `aacount`, `aminolookup`, `basecount`, `baselookup`, `dimercount`, `emblread`, `fastaread`, `fastawrite`, `genbankread`, `geneticcode`, `genpeptread`, `getembl`, `getgenbank`, `getgenpept`, `nt2aa`, `proteinplot`, `seqcomplement`, `seqdisp`, `seqrcomplement`, `seqreverse`, `seqshoworfs`, `seqshowwords`, `seqwordcount`

Purpose

Count the number of occurrences of a word in a sequence

Syntax

```
seqwordcount(Seq, Word)
```

Arguments

`Seq`	Enter a nucleotide or amino acid sequence of characters. You can also enter a structure with the field `Sequence`.
`Word`	Enter a short sequence of characters.

Description

`seqwordcount(Seq, Word)` counts the number of times that a word appears in a sequence, and then returns the number of occurrences of that word.

If `Word` contains nucleotide or amino acid symbols that represent multiple possible symbols (ambiguous characters), then `seqwordcount` counts all matches. For example, the symbol `R` represents either `G` or `A` (purines). For another example, if `word` equals `'ART'`, then `seqwordcount` counts occurrences of both `'AAT'` and `'AGT'`.

Examples

`seqwordcount` does not count overlapping patterns multiple times. In the following example, `seqwordcount` reports three matches. `TATATATA` is counted as two distinct matches, not three overlapping occurrences.

```
seqwordcount('GCTATAACGTATATATAT','TATA')

ans =
     3
```

The following example reports two matches (`'TAGT'` and `'TAAT'`). `B` is the ambiguous code for `G`, `T`, or `C`, while `R` is an ambiguous code for `G` and `A`.

```
seqwordcount('GCTAGTAACGTATATATAAT','BART')

ans =
     2
```

See Also

Bioinformatics Toolbox functions `codoncount`, `seqshoworfs`, `seqshowwords`, `seqtool`, `seq2regexp`

MATLAB functions `strfind`

Purpose

Display a sequence alignment with color

Syntax

```
showalignment(Alignment)
showalignment(..., 'PropertyName', PropertyValue,...)
showalignment(..., 'MatchColor', MatchColorValue)
showalignment(..., 'SimilarColor' SimilarColorValue)
showalignment(..., 'StartPointers', StartPointersValue)
showalignment(..., 'Columns', ColumnsValue)
```

Arguments

Alignment	For pairwise alignments, matches and similar residues are highlighted and *Alignment* is the output from one of the functions `nwalign` or `swalign`. For multiple sequence alignment highly conserved columns are highlighted and *Alignment* is the output from the function `multialign`.
MatchColorValue	Property to select the color to highlight matching characters. Enter a `1`-by-`N` RGB vector specifying the intensity (`0` to `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'`.
SimilarColorValue	Property to select the color to highlight similar characters. Enter a `1`-by-`3` RGB vector or color character. The default color is magenta.

StarterPointersValue	Property to specify the starting indices of the aligned sequences. `StartPointers` is the two element vector returned as the third output of the function `swalign`.
ColumnsValue	Property to specify the number of characters in a line. Enter the number of characters to display in one row. The default value is `64`.

Description

`showalignment(`*`Alignment`*`)` displays an alignment in a MATLAB figure window.

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

`showalignment(..., 'MatchColor', `*`MatchColorValue`*`)` selects the color to highlight the matches in the output display. The default color is red. For example, to use cyan, enter `'c'` or `[0 255 255]`.

`showalignment(..., 'SimilarColor' `*`SimilarColorValue`*`)` selects the color to highlight similar residues that are not exact matches. The default color is magenta.

The following options are only available when showing pairwise alignments:

`showalignment(..., 'StartPointers', `*`StartPointersValue`*`)` specifies the starting indices in the original sequences of a local alignment.

`showalignment(..., 'Columns', `*`ColumnsValue`*`)` specifies how many columns per line to use in the output, and labels the start of each row with the sequence positions.

Examples

Enter two amino acid sequences and show their alignment.

```
[Score, Alignment] = nwalign('VSPAGMASGYD','IPGKASYD');
showalignment(Alignment);
```

```
Identities = 6/11 (55%), Positives = 7/11 (64%)
VSPAGMASGYD
: | | || ||
I-P-GKAS-YD
```

Enter a multiplyaligned set of sequences and show their alignment.

```
gag = multialignread('aagag.aln');
showalignment(gag)
```

See Also Bioinformatics Toolbox functions `nwalign`, `swalign`

showhmmprof

Purpose

Plot an Hidden Markov Model (HMM) profile

Syntax

```
showhmmprof(Model)
showhmmprof(..., 'PropertyName', PropertyValue,...)
showhmmprof(..., 'Scale', ScaleValue)
showhmmprof(..., 'Order', OrderValue)
```

Arguments

Model	Hidden Markov model created with the functions `gethmmprof` and `pfamhmmread` functions.
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 alphabet. Enter a character string with the

Description

`showhmmprof(`*`Model`*`)` plots a profile hidden Markov model described by the structure `Model`.

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

`showhmmprof(..., '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(..., '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`

Purpose

Read data from a SPOT file

Syntax

```
SPOTData = sptread('File',
                   'PropertyName', PropertyValue)

sptread(..., 'CleanColNames, 'CleanColNamesValues')
```

Arguments

File	SPOT formatted file (ASCII text file). Enter a filename, a path and filename, or URL pointing to a file. *File* can also be a MATLAB character array that contains the text for a SPOT file.
CleanColNamesValue	Property to control using valid MATLAB variable names.

Description

`SPOTData = sptread('`*`File`*`')` reads a SPOT formatted file and creates a MATLAB structure `SPOTData` containing the following fields:

```
Header
Data
Blocks
Columns
Rows
IDs
ColumnNames
Indices
Shape
```

`sptread(..., 'CleanColNames, `*`CleanColNamesValue`*`)` The column names in the SPOT 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

```
% Read in a sample SPOT file and plot the median foreground
% intensity for the 635 nm channel.
spotStruct = sptread('spotdata.txt')
maimage(spotStruct,'Rmedian');

% Alternatively, create a similar plot using
% more basic graphics commands.

rmedCol = find(strcmp(spotStruct.ColumnNames,'Rmedian'));
Rmedian = spotStruct.Data(:,rmedCol);
imagesc(Rmedian(spotStruct.Indices));
colormap bone
colorbar
```

See Also

Bioinformatics Toolbox functions affyread, geosoftread, imageneread, maboxplot, gprread

subtree (phytree)

Purpose

Extract a 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 methods for a phylogenetic tree object `phytree`, `prune`, `select`, `get`, `getbyname`

Purpose

Classify data using a support vector machine

Syntax

```
Group = svmclassify(SVMStruct, Sample, 'PropertyName',
PropertyValue...)
svmclassify(..., 'Showplot', ShowplotValue)
```

Description

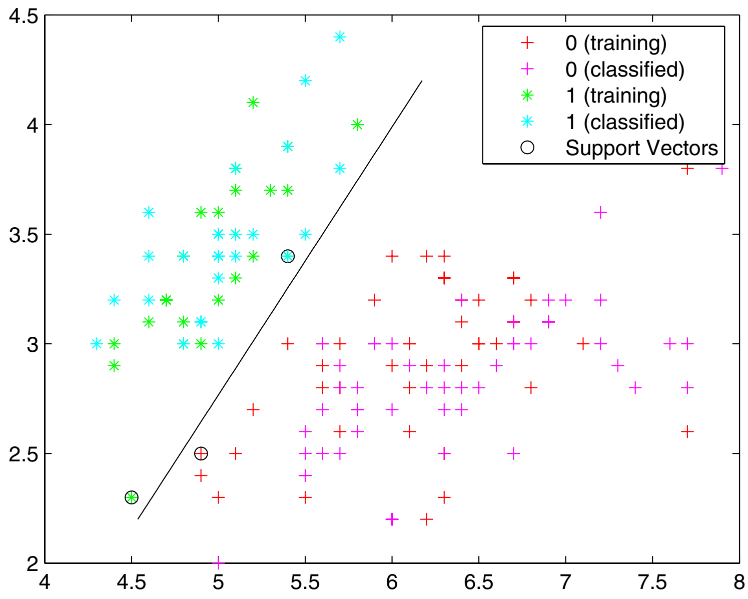
`Group = svmclassify(SVMStruct, Sample, 'PropertyName', PropertyValue...)` classifies each row of the data in `Sample` using the information in a support vector machine classifier structure `SVMStruct`, created using `svmtrain`. `Sample` must have the same number of columns as the data used to train the classifier in `svmtrain`. `Group` indicates the group to which each row of `Sample` has been assigned.

`svmclassify(..., 'Showplot', ShowplotValue)` when `Showplot` is `true`, plots thesample data on the figure created using the `showplot` option in `svmtrain`.

Example

```
% Load the data and select features for classification
load fisheriris
data = [meas(:,1), meas(:,2)];
% Extract the Setosa class
groups = ismember(species,'setosa');
% Randomly select training and test sets
[train, test] = crossvalind('holdOut',groups);
cp = classperf(groups);
% Use a linear support vector machine classifier
svmStruct = svmtrain(data(train,:),groups(train),'showplot',true);
classes = svmclassify(svmStruct,data(test,:),'showplot',true);
% See how well the classifier performed
classperf(cp,classes,test);
cp.CorrectRate
```

This displays the following plot:



See Also

Bioinformatics Toolbox functions `knnclassify`, `classperf`, `crossvalind`, `svmtrain`

Statistical Toolbox functions `classify`, `quadprog`,

Purpose

Train a support vector machine classifier

Syntax

```
SVMStruct = svmtrain(Training, group)
svmtrain(...,'kernel_function',kfun)
svmtrain(..., 'polyorder', order)
svmtrain(...,'mlp_params',[p1 p2])
svmtrain(...,'method',method)
svmtrain(..., 'quadprog_opts',options)
svmtrain(..., 'showplot',true)
```

Description

`SVMStruct = svmtrain(Training, group)` trains a support vector machine classifier using data `Training` taken from two groups given by `group`. `SVMStruct` contains information about the trained classifier that is used by `svmclassify` for classification. `group` is a column vector of values, of the same length as `Training`, that defines two groups. Each element of `group` specifies the group to which the corresponding row of `Training` belongs. `group` can be a numeric vector, a string array, or a cell array of strings. `svmtrain` treats `NaN`s or empty strings in `group` as missing values, and ignores the corresponding rows of `Training`.

`svmtrain(...,'kernel_function',kfun)` specifies the kernel function `kfun` that is used to map the training data into kernel space. The default kernel function is the dot product. `kfun` can be one of the following strings or a function handle:

`'linear'`	Linear kernel or dot product
`'quadratic'`	Quadratic kernel
`'polynomial'`	Polynomial kernel (default order 3)
`'rbf'`	Gaussian radial basis function kernel
`'mlp'`	Multilayer perceptron kernel (default scale 1)
Function handle	A handle to a kernel function specified using `@`, for example `@kfun`, or an anonymous function

A kernel function must be of the form

```
function K = kfun(U, V)
```

The returned value `K` is a matrix of size m-by-n, where `U` and `V` have m and n rows respectively. 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)
```

`svmtrain(..., 'polyorder', order)`enables you to specify the order of a polynomial kernel. The default order is `3`.

`svmtrain(...,'mlp_params',[p1 p2])`enables you to specify the parameters of the multilayer perceptron (mlp) kernel. The mlp kernel requires two parameters, `p1` and `p2`, where K = tanh(p1*U*V' + p2), `p1` > 0, and `p2` < 0. Default values are `p1` = 1 and `p2` = -1.

`svmtrain(...,'method',method)`enables you to specify the method used to find the separating hyperplane. The options are

`'QP'`	Quadratic programming (requires the Optimization Toolbox)
`'LS'`	Least-squares method

Note If you installed the Optimization Toolbox, the `'QP'` method is the default. If not, the only available method is `'LS'`.

`svmtrain(..., 'quadprog_opts',options)`enables you to pass an options structure, created using `optimset`, to the Optimization Toolbox function `quadprog` when using the `'QP'` method. See the `optimset` reference page for more details.

`svmtrain(..., 'showplot',true)`, when used with two-dimensional data, creates a plot of the grouped data and plots the separating line for the classifier.

Example

Load sample data

```
load fisheriris
X = [meas(:,1), meas(:,2)];
```

Extract the Setosa class

```
groups = ismember(species,'setosa');
```

Randomly select training and test sets

```
[train, test] = crossvalind('holdOut',groups);
```

Use a linear support vector machine classifier

```
svmStruct = svmtrain(X(train,:),groups(train),'showplot',true);
classes = svmclassify(svmStruct,X(test,:),'showplot',true);
```

See Also

Bioinformatics Toolbox `knnclassify`, `svmclassify`,

Statistical Toolbox functions `classify`,

Optimization Toolbox Function `quadprog`

swalign

Purpose

Locally align two sequences using the Smith-Waterman algorithm

Syntax

```
swalign(Seq1, Seq2)
[Score, Alignment] = swalign(Seq1, Seq2)
[Score, Alignment, Start] = swalign(Seq1, Seq2)
swalign(..., 'PropertyName', PropertyValue,...)
swalign(..., 'Alphabet', AlphabetValue)
swalign(..., 'ScoringMatrix', ScoringMatrixValue)
swalign(..., 'Scale', ScaleValue)
swalign(..., 'GapOpen', GapOpenValue)
swalign(..., 'ExtendGap', ExtendGapValue)
swalign(..., 'Showscore', ShowscoreValue)
```

Arguments

Seq1, *Seq2*	Nucleotide or amino acid sequences. Enter a character string or vector of integers. You can also enter a structure with the field `Sequence`.
AlphabetValue	Property to select an amino acid or nucleotide sequences. Enter either `'AA'` or `'NT'`. The default value is `'AA'`.
ScoringMatrixValue	Property to select the scoring matrix. Enter the name of a scoring matrix. Values are `'PAM40`', `'PAM250'`, `DAYHOFF`, `GONNET`, `'BLOSUM30'` increasing by `5` to `'BLOSUM90'`, or `'BLOSUM62'`, or `'BLOSUM100'`. The default value when `AlphabetValue = 'aa'` is `'BLOSUM50'`, while the default value when `AlphabeValue = 'nt'` is `nuc44`.
ScaleValue	Property to specify a scaling factor for a scoring matrix.
GapOpenValue	Property to specify the gap open penalty. Enter an integer for the gap penalty. Default value is `8`.

ExtendGapValue	Property to specify the extended gap open penalty. Enter an integer for the extended gap penalty. The default value equals the `GapOpen` value.
ShowscoreValue	Property to control displaying the scoring space and the winning path. Enter either `true` or `false`. The default value is `false`.

Description

`swalign(`*`Seq1`*`, `*`Seq2`*`)` returns the alignment score in bits for the optimal alignment. The scale factor used to calculate the score is provided by the scoring matrix. If this is not defined, then `swalign` returns the raw score.

`[`*`Score, Alignment`*`] = swalign(`*`Seq1`*`, `*`Seq2`*`)` returns a `3-by-n` character array showing the two sequences and the local alignment between them. Amino acids that match are indicated with the symbol `|`, while related amino acids (nonmatches with a positive scoring matrix value) are indicated with the symbol `:`.

`[`*`Score, Alignment, Start`*`] = swalign(`*`Seq1`*`, `*`Seq2`*`)` returns a 2-by-1 vector with the starting point indices where the alignment begins for each sequence.

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

`swalign(..., 'Alphabet', `*`AlphabetValue`*`)` specifies whether the sequences are amino acids (`'AA'`) or nucleotides (`'NT'`). The default value is `'AA'`.

`swalign(..., 'ScoringMatrix', `*`ScoringMatrixValue`*`)` specifies the scoring matrix to use for the alignment. The default is `'blosum50'` for `Alphabet = 'AA'` or `'NUC44'` for `Alphabet = NT`.

`swalign(..., 'Scale', `*`ScaleValue`*`)` indicates the scale factor of the scoring matrix to return the score using arbitrary units. If the scoring matrix also provides a scale factor, then both are used.

`swalign(..., 'GapOpen', GapOpenValue)` specifies the penalty for opening a gap in the alignment. The default gap open penalty is `8`.

`swalign(..., 'ExtendGap', ExtendGapValue)` specifies the penalty for extending a gap in the alignment. If `ExtendGap` is not specified, then extensions to gaps are scored with the same value as `GapOpen`.

`swalign(..., 'Showscore', ShowscoreValue)` displays the scoring space and the winning path.

Scores are 'raw' scores which mean the final score is an accumulation of using the scoring matrix values at each position of the alignment. Accumulation means that it is the sum of the amino acid matches (including the gap penalties). If the provided scoring matrix (or the one used by default) has a Scale entry, then the score is returned in 'bits'.

References

[1] Durbin R. Eddy S, Krogh A, Mitchison G; Biological Sequence Analysis. Cambridge University Press, 1998.

[2] Smith T, Waterman M.; "Identification of common molecular subsequences" in Journal Molecular Biology; 1981, Vol. 147, pp. 195-197.

Examples

Return the score in bits and the local alignment using the default `ScoringMatrix` (`'BLOSUM50'`) and default values for the `GapOpen` and `ExtendGap` values.

```
[Score, Alignment] = swalign('VSPAGMASGYD','IPGKASYD')

Score =
     8.6667

Alignment =
PAGMASGYD
| | || ||
P-GKAS-YD
```

Align two amino sequences using a specified scoring matrix (`'pam250'`) and a gap open penalty of `5`.

```
[Score, Alignment] = swalign('HEAGAWGHEE','PAWHEAE',...
                             'ScoringMatrix', 'pam250',...
                             'GapOpen',5)

Score =
     8
Alignment =
GAWGHE
:|| ||
PAW-HE
```

Align two amino sequences and return the `Score` in nat units (nats).

```
[Score, Alignment] = swalign('HEAGAWGHEE','PAWHEAE',...
                             'Scale',log(2))

Score =
    6.4694
Alignment =
AWGHE
|| ||
AW-HE
```

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`

Purpose

Draw figure from biograph object

Syntax

```
view(BGobj)
BGobjHandle = view(BGobj)
```

Arguments

`BGobj`	Biograph object.

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 methods for the biograph object, `biograph`, `dolayout`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`

view (phytree)

Purpose

View phylogenetic tree

Syntax

```
view(Tree)
view(Tree, IntNodes)
```

Arguments

`Tree`	`phytree` object created with `phytree (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

```
tr = phytreeread('pf00002.tree')
view(tree)
```

See Also

Bioinformatics Toolbox functions `dnds`, `phytreetool`, `seqlinkage`, `seqneighjoin`

`phytree` object methods `phytree (phytree)`, `plot (phytree)`

weights (phytree)

Purpose

Calculate weights for a phylogenetic tree

Syntax

```
W = weights(Tree)
```

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.

References

[1] Thompson J.D, Higgins D.G, Gibson T.G, Nucleic Acids Res, (1994) 22(22):4673-4680.

[2] Henikoff S, Henikoff J.G, JMB, (1994) 243(4):574-578.

See Also

`multialign`, `phytree`, `profalign`, `seqlinkage`

Properties — Alphabetical List

Index