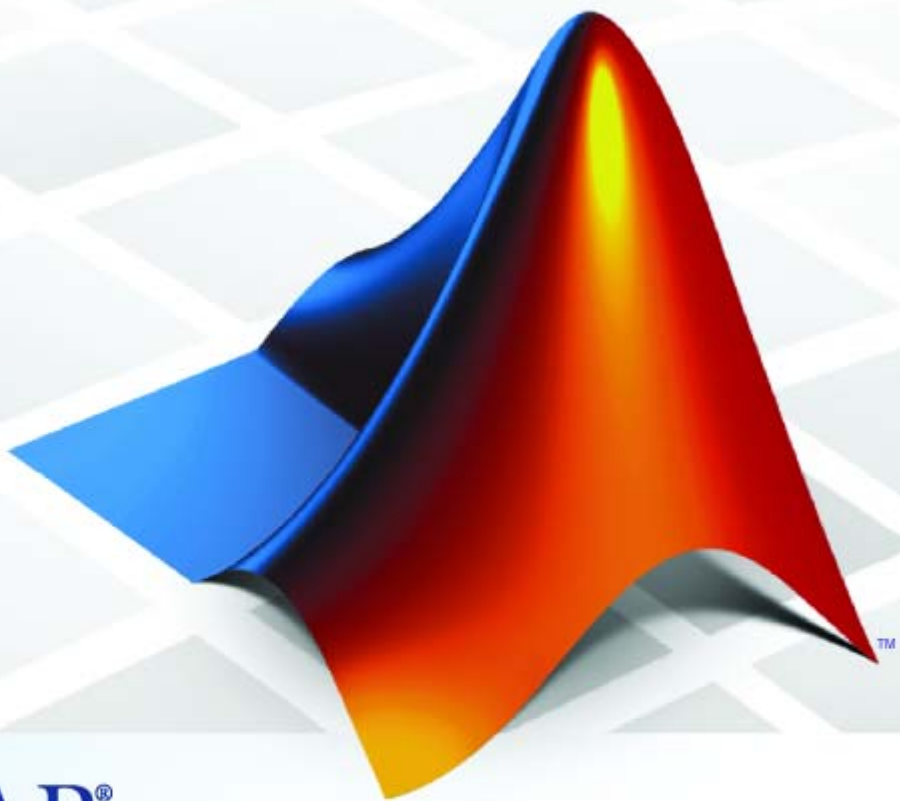


Bioinformatics Toolbox™ 3

Reference



MATLAB®

How to Contact The MathWorks



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

Bioinformatics Toolbox™ Reference

© COPYRIGHT 2003–2008 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.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Revision History

May 2005	Online only	New for Version 2.1 (Release 14SP2+)
September 2005	Online only	Revised for Version 2.1.1 (Release 14SP3)
November 2005	Online only	Revised for Version 2.2 (Release 14SP3+)
March 2006	Online only	Revised for Version 2.2.1 (Release 2006a)
May 2006	Online only	Revised for Version 2.3 (Release 2006a+)
September 2006	Online only	Revised for Version 2.4 (Release 2006b)
March 2007	Online only	Revised for Version 2.5 (Release 2007a)
April 2007	Online only	Revised for Version 2.6 (Release 2007a+)
September 2007	Online only	Revised for Version 3.0 (Release 2007b)
March 2008	Online only	Revised for Version 3.1 (Release 2008a)
October 2008	Online only	Revised for Version 3.2 (Release 2008b)

Function Reference

1

Constructor	1-3
Data Formats and Databases	1-4
Trace Tools	1-6
Sequence Conversion	1-7
Sequence Utilities	1-8
Sequence Statistics	1-9
Sequence Visualization	1-10
Pairwise Sequence Alignment	1-11
Multiple Sequence Alignment	1-11
Scoring Matrices	1-12
Phylogenetic Tree Tools	1-12
Graph Theory	1-13
Gene Ontology	1-14
Protein Analysis	1-14
Profile Hidden Markov Models	1-15

Microarray File Formats	1-16
Microarray Utilities	1-17
Microarray Data Analysis and Visualization	1-18
Microarray Normalization and Filtering	1-19
Statistical Learning	1-21
Mass Spectrometry	1-21

Functions — Alphabetical List

2

Method Reference

3

Phylogenetic Tree	3-1
Graph Visualization	3-2
Gene Ontology	3-3
Clustergram	3-4
DataMatrix	3-4

Methods — Alphabetical List

4

Object Reference

5

Index

Function Reference

Constructor (p. 1-3)	Create objects.
Data Formats and Databases (p. 1-4)	Read data into the MATLAB® software from Web databases; read and write to files using specific sequence data formats
Trace Tools (p. 1-6)	Read data from SCF file and draw nucleotide trace plots
Sequence Conversion (p. 1-7)	Convert nucleotide and amino acid sequences between character and integer formats, reverse and complement order of nucleotide bases, and translate nucleotides codons to amino acids
Sequence Utilities (p. 1-8)	Calculate consensus sequence from set of multiply aligned sequences, run BLAST search from MATLAB environment, and search sequences using regular expressions
Sequence Statistics (p. 1-9)	Determine base counts, nucleotide density, codon bias, and CpG islands; search for words and identify open reading frames (ORFs)
Sequence Visualization (p. 1-10)	Visualize sequence data
Pairwise Sequence Alignment (p. 1-11)	Compare nucleotide or amino acid sequences using pairwise sequence alignment functions

Multiple Sequence Alignment (p. 1-11)	Compare sets of nucleotide or amino acid sequences; progressively align sequences using phylogenetic tree for guidance
Scoring Matrices (p. 1-12)	Standard scoring matrices such as PAM and BLOSUM families of matrices that alignment functions use
Phylogenetic Tree Tools (p. 1-12)	Read phylogenetic tree files, calculate pairwise distances between sequences, and build a phylogenetic tree
Graph Theory (p. 1-13)	Apply basic graph theory algorithms to sparse matrices
Gene Ontology (p. 1-14)	Read Gene Ontology formatted files
Protein Analysis (p. 1-14)	Determine protein characteristics and simulate enzyme cleavage reactions
Profile Hidden Markov Models (p. 1-15)	Get profile hidden Markov model data from the PFAM database or create your own profiles from set of sequences
Microarray File Formats (p. 1-16)	Read data from common microarray file formats including Affymetrix [®] GeneChip [®] , ImaGene [®] results, and SPOT files; read GenePix [®] GPR and GAL files
Microarray Utilities (p. 1-17)	Using Affymetrix and GeneChip data sets, get library information for probe, gene information from probe set, and probe set values from CEL and CDF information; show probe set information from the NetAffx [™] Web site and plot probe set values

Microarray Data Analysis and Visualization (p. 1-18)	Analyze and visualize microarray data with t tests, spatial plots, box plots, loglog plots, and intensity-ratio plots
Microarray Normalization and Filtering (p. 1-19)	Normalize microarray data with lowess and mean normalization functions; filter raw data for cleanup before analysis
Statistical Learning (p. 1-21)	Classify and identify features in data sets, set up cross-validation experiments, and compare different classification methods
Mass Spectrometry (p. 1-21)	Read data from common mass spectrometry file formats, preprocess raw mass spectrometry data from instruments, and analyze spectra to identify patterns and compounds

Constructor

<code>biograph</code>	Create <code>biograph</code> object
<code>clustergram</code>	Compute hierarchical clustering, display dendrogram and heat map, and create <code>clustergram</code> object
<code>DataMatrix</code>	Create <code>DataMatrix</code> object
<code>geneont</code>	Create <code>geneont</code> object
<code>phytree</code>	Create <code>phytree</code> object

Data Formats and Databases

<code>affygcrrma</code>	Perform GC Robust Multi-array Average (GCRMA) procedure on Affymetrix microarray probe-level data
<code>affyprobeseqread</code>	Read data file containing probe sequence information for Affymetrix GeneChip array
<code>affyread</code>	Read microarray data from Affymetrix GeneChip file
<code>affyrma</code>	Perform Robust Multi-array Average (RMA) procedure on Affymetrix microarray probe-level data
<code>affysnpannotread</code>	Read Affymetrix Mapping DNA array data from CSV-format annotation file
<code>agferead</code>	Read Agilent® Feature Extraction Software file
<code>blastformat</code>	Create local BLAST database
<code>blastread</code>	Read data from NCBI BLAST report file
<code>blastreadlocal</code>	Read data from local BLAST report
<code>celintensityread</code>	Read probe intensities from Affymetrix CEL files
<code>cytobandread</code>	Read cytogenetic banding information
<code>emblread</code>	Read data from EMBL file
<code>fastaread</code>	Read data from FASTA file
<code>fastawrite</code>	Write to file using FASTA format
<code>galread</code>	Read microarray data from GenePix array list file

genbankread	Read data from GenBank® file
genpeptread	Read data from GenPept file
geoseriesread	Read Gene Expression Omnibus (GEO) Series (GSE) format data
geosoftread	Read Gene Expression Omnibus (GEO) SOFT format data
getblast	Retrieve BLAST report from NCBI Web site
getembl	Retrieve sequence information from EMBL database
getgenbank	Retrieve sequence information from GenBank database
getgenpept	Retrieve sequence information from GenPept database
getgeodata	Retrieve Gene Expression Omnibus (GEO) format data
gethmmalignment	Retrieve multiple sequence alignment associated with hidden Markov model (HMM) profile from PFAM database
gethmmprof	Retrieve hidden Markov model (HMM) profile from PFAM database
gethmmtree	Retrieve phylogenetic tree data from PFAM database
getpdb	Retrieve protein structure data from Protein Data Bank (PDB) database
goannotread	Read annotations from Gene Ontology annotated file
gprread	Read microarray data from GenePix Results (GPR) file
ilmnbsread	Read gene expression data exported from Illumina® BeadStudio™ software

<code>imageneread</code>	Read microarray data from ImaGene Results file
<code>jcampread</code>	Read JCAMP-DX-formatted files
<code>multialignread</code>	Read multiple sequence alignment file
<code>multialignwrite</code>	Write multiple alignment to file using ClustalW ALN format
<code>mzcdfread</code>	Read mass spectrometry data from netCDF file
<code>mzxmlread</code>	Read data from mzXML file
<code>pdbread</code>	Read data from Protein Data Bank (PDB) file
<code>pdbwrite</code>	Write to file using Protein Data Bank (PDB) format
<code>pfamhmmread</code>	Read data from PFAM HMM-formatted file
<code>phytreeread</code>	Read phylogenetic tree file
<code>phytreewrite</code>	Write phylogenetic tree object to Newick-formatted file
<code>scfread</code>	Read trace data from SCF file
<code>sptread</code>	Read data from SPOT file

Trace Tools

<code>scfread</code>	Read trace data from SCF file
<code>traceplot</code>	Draw nucleotide trace plots

Sequence Conversion

aa2int	Convert amino acid sequence from letter to integer representation
aa2nt	Convert amino acid sequence to nucleotide sequence
aminolookup	Find amino acid codes, integers, abbreviations, names, and codons
baselookup	Find nucleotide codes, integers, names, and complements
dna2rna	Convert DNA sequence to RNA sequence
int2aa	Convert amino acid sequence from integer to letter representation
int2nt	Convert nucleotide sequence from integer to letter representation
nt2aa	Convert nucleotide sequence to amino acid sequence
nt2int	Convert nucleotide sequence from letter to integer representation
rna2dna	Convert RNA sequence of nucleotides to DNA sequence
rnaconvert	Convert secondary structure of RNA sequence between bracket and matrix notations
seq2regexp	Convert sequence with ambiguous characters to regular expression
seqcomplement	Calculate complementary strand of nucleotide sequence
seqrcomplement	Calculate reverse complement of nucleotide sequence
seqreverse	Reverse letters or numbers in nucleotide sequence

Sequence Utilities

<code>aminolookup</code>	Find amino acid codes, integers, abbreviations, names, and codons
<code>baselookup</code>	Find nucleotide codes, integers, names, and complements
<code>blastlocal</code>	Perform search on local BLAST database to create BLAST report
<code>blastncbi</code>	Create remote NCBI BLAST report request ID or link to NCBI BLAST report
<code>cleave</code>	Cleave amino acid sequence with enzyme
<code>cleavelookup</code>	Find cleavage rule for enzyme or compound
<code>featuresparse</code>	Parse features from GenBank, GenPept, or EMBL data
<code>geneticcode</code>	Return nucleotide codon to amino acid mapping for genetic code
<code>joinseq</code>	Join two sequences to produce shortest supersequence
<code>oligoprop</code>	Calculate sequence properties of DNA oligonucleotide
<code>palindromes</code>	Find palindromes in sequence
<code>pdbdistplot</code>	Visualize intermolecular distances in Protein Data Bank (PDB) file
<code>proteinplot</code>	Characteristics for amino acid sequences
<code>proteinpropplot</code>	Plot properties of amino acid sequence
<code>ramachandran</code>	Draw Ramachandran plot for Protein Data Bank (PDB) data

<code>randseq</code>	Generate random sequence from finite alphabet
<code>rebasecuts</code>	Find restriction enzymes that cut nucleotide sequence
<code>restrict</code>	Split nucleotide sequence at restriction site
<code>revgeneticcode</code>	Return reverse mapping (amino acid to nucleotide codon) for genetic code
<code>rnafold</code>	Predict minimum free-energy secondary structure of RNA sequence
<code>seqconsensus</code>	Calculate consensus sequence
<code>seqdisp</code>	Format long sequence output for easy viewing
<code>seqinsertgaps</code>	Insert gaps into nucleotide or amino acid sequence
<code>seqlogo</code>	Display sequence logo for nucleotide or amino acid sequences
<code>seqmatch</code>	Find matches for every string in library
<code>seqprofile</code>	Calculate sequence profile from set of multiply aligned sequences
<code>seqshoworfs</code>	Display open reading frames in sequence

Sequence Statistics

<code>aaaccount</code>	Count amino acids in sequence
<code>aminolookup</code>	Find amino acid codes, integers, abbreviations, names, and codons
<code>basecount</code>	Count nucleotides in sequence

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

Sequence Visualization

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

Pairwise Sequence Alignment

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

Multiple Sequence Alignment

<code>fastaread</code>	Read data from FASTA file
<code>multialign</code>	Align multiple sequences using progressive method
<code>multialignread</code>	Read multiple sequence alignment file
<code>multialignviewer</code>	Open viewer for multiple sequence alignments
<code>multialignwrite</code>	Write multiple alignment to file using ClustalW ALN format
<code>profalign</code>	Align two profiles using Needleman-Wunsch global alignment
<code>seqpdist</code>	Calculate pairwise distance between sequences
<code>showalignment</code>	Display color-coded sequence alignment

Scoring Matrices

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

Phylogenetic Tree Tools

<code>dnds</code>	Estimate synonymous and nonsynonymous substitution rates
<code>dndsm1</code>	Estimate synonymous and nonsynonymous substitution rates using maximum likelihood method
<code>gethmmtree</code>	Retrieve phylogenetic tree data from PFAM database
<code>phytreeread</code>	Read phylogenetic tree file
<code>phytreetool</code>	View, edit, and explore phylogenetic tree data
<code>phytreewrite</code>	Write phylogenetic tree object to Newick-formatted file
<code>seqinsertgaps</code>	Insert gaps into nucleotide or amino acid sequence
<code>seqlinkage</code>	Construct phylogenetic tree from pairwise distances

seqneighjoin	Neighbor-joining method for phylogenetic tree reconstruction
seqpdist	Calculate pairwise distance between sequences

Graph Theory

graphallshortestpaths	Find all shortest paths in graph
graphconncomp	Find strongly or weakly connected components in graph
graphisdag	Test for cycles in directed graph
graphisomorphism	Find isomorphism between two graphs
graphisspantree	Determine if tree is spanning tree
graphmaxflow	Calculate maximum flow in directed graph
graphminspantree	Find minimal spanning tree in graph
graphpred2path	Convert predecessor indices to paths
graphshortestpath	Solve shortest path problem in graph
graphtopoorder	Perform topological sort of directed acyclic graph
graphtraverse	Traverse graph by following adjacent nodes

Gene Ontology

goannotread	Read annotations from Gene Ontology annotated file
num2goid	Convert numbers to Gene Ontology IDs

Protein Analysis

aacount	Count amino acids in sequence
aminolookup	Find amino acid codes, integers, abbreviations, names, and codons
atomiccomp	Calculate atomic composition of protein
cleave	Cleave amino acid sequence with enzyme
cleavelookup	Find cleavage rule for enzyme or compound
evalrasmolscript	Send RasMol script commands to Molecule Viewer window
isoelectric	Estimate isoelectric point for amino acid sequence
molviewer	Display and manipulate 3-D molecule structure
molweight	Calculate molecular weight of amino acid sequence
pdbdistplot	Visualize intermolecular distances in Protein Data Bank (PDB) file
pdbsuperpose	Superpose 3-D structures of two proteins

<code>pdbtransform</code>	Apply linear transformation to 3-D structure of molecule
<code>proteinplot</code>	Characteristics for amino acid sequences
<code>proteinpropplot</code>	Plot properties of amino acid sequence
<code>ramachandran</code>	Draw Ramachandran plot for Protein Data Bank (PDB) data

Profile Hidden Markov Models

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

<code>pfamhmmread</code>	Read data from PFAM HMM-formatted file
<code>showhmmprof</code>	Plot hidden Markov model (HMM) profile

Microarray File Formats

<code>affygcрма</code>	Perform GC Robust Multi-array Average (GCRMA) procedure on Affymetrix microarray probe-level data
<code>affyprobeseqread</code>	Read data file containing probe sequence information for Affymetrix GeneChip array
<code>affyread</code>	Read microarray data from Affymetrix GeneChip file
<code>affyrma</code>	Perform Robust Multi-array Average (RMA) procedure on Affymetrix microarray probe-level data
<code>affysnannotread</code>	Read Affymetrix Mapping DNA array data from CSV-format annotation file
<code>agferead</code>	Read Agilent Feature Extraction Software file
<code>celintensityread</code>	Read probe intensities from Affymetrix CEL files
<code>galread</code>	Read microarray data from GenePix array list file
<code>geoseriesread</code>	Read Gene Expression Omnibus (GEO) Series (GSE) format data
<code>geosoftread</code>	Read Gene Expression Omnibus (GEO) SOFT format data

getgeodata	Retrieve Gene Expression Omnibus (GEO) format data
gprread	Read microarray data from GenePix Results (GPR) file
ilmnbsread	Read gene expression data exported from Illumina BeadStudio software
imageneread	Read microarray data from ImaGene Results file
sptread	Read data from SPOT file

Microarray Utilities

affysnpintensitysplit	Split Affymetrix SNP probe intensity information for alleles A and B
affysnpquartets	Create table of SNP probe quartet results for Affymetrix probe set
ilmnbslookup	Look up Illumina BeadStudio target (probe) sequence and annotation information
magetfield	Extract data from microarray structure
probelibraryinfo	Create table of probe set library information
probesetlink	Display probe set information on NetAffx Web site
probesetlookup	Look up information for Affymetrix probe set
probesetplot	Plot Affymetrix probe set intensity values
probesetvalues	Create table of Affymetrix probe set intensity values

Microarray Data Analysis and Visualization

<code>cghcbs</code>	Perform circular binary segmentation (CBS) on array-based comparative genomic hybridization (aCGH) data
<code>cghfreqplot</code>	Display frequency of DNA copy number alterations across multiple samples
<code>chromosomeplot</code>	Plot chromosome ideogram with G-banding pattern
<code>clustergram</code>	Compute hierarchical clustering, display dendrogram and heat map, and create <code>clustergram</code> object
<code>maboxplot</code>	Create box plot for microarray data
<code>mafdr</code>	Estimate false discovery rate (FDR) of differentially expressed genes from two experimental conditions or phenotypes
<code>maimage</code>	Spatial image for microarray data
<code>mairplot</code>	Create intensity versus ratio scatter plot of microarray data
<code>maloglog</code>	Create loglog plot of microarray data
<code>mapcaplot</code>	Create Principal Component Analysis (PCA) plot of microarray data
<code>mattest</code>	Perform two-sample t-test to evaluate differential expression of genes from two experimental conditions or phenotypes
<code>mavolcanoplot</code>	Create significance versus gene expression ratio (fold change) scatter plot of microarray data

redbluecmap	Create red and blue colormap
redgreencmap	Create red and green colormap

Microarray Normalization and Filtering

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

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

Statistical Learning

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

Mass Spectrometry

<code>jcampread</code>	Read JCAMP-DX-formatted files
<code>msalign</code>	Align peaks in mass spectrum to reference peaks
<code>msbackadj</code>	Correct baseline of mass spectrum

<code>msdotplot</code>	Plot set of peak lists from LC/MS or GC/MS data set
<code>msheatmap</code>	Create pseudocolor image of set of mass spectra
<code>mslowess</code>	Smooth mass spectrum using nonparametric method
<code>msnorm</code>	Normalize set of mass spectra
<code>mspalign</code>	Align mass spectra from multiple peak lists from LC/MS or GC/MS data set
<code>mspeaks</code>	Convert raw mass spectrometry data to peak list (centroided data)
<code>msppresample</code>	Resample mass spectrometry signal while preserving peaks
<code>msresample</code>	Resample mass spectrometry signal
<code>mssgolay</code>	Smooth mass spectrum with least-squares polynomial
<code>msviewer</code>	Explore mass spectrum or set of mass spectra
<code>mzcdf2peaks</code>	Convert mzCDF structure to peak list
<code>mzcdfinfo</code>	Return information about netCDF file containing mass spectrometry data
<code>mzcdfread</code>	Read mass spectrometry data from netCDF file
<code>mzxml2peaks</code>	Convert mzXML structure to peak list
<code>mzxmlinfo</code>	Return information about mzXML file

mzxmlread

Read data from mzXML file

samplealign

Align two data sets containing
sequential observations by
introducing gaps

Functions — Alphabetical List

aa2int

Purpose Convert amino acid sequence from letter to integer representation

Syntax `SeqInt = aa2int(SeqChar)`

Arguments `SeqChar` One of the following:

- String of single-letter codes specifying an amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers on page 2-2. Unknown characters are mapped to 0. Integers are arbitrarily assigned to IUB/IUPAC letters.
- MATLAB structure containing a `Sequence` field that contains an amino acid sequence, such as returned by `fastaread`, `getgenpept`, `genpeptread`, `getpdb`, or `pdbread`.

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

Description `SeqInt = aa2int(SeqChar)` converts `SeqChar`, a character string of single-letter codes specifying an amino acid sequence, to `SeqInt`, a row vector of integers specifying the same amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers on page 2-2.

Mapping Amino Acid Letter Codes to Integers

Amino Acid	Code	Integer
Alanine	A	1
Arginine	R	2
Asparagine	N	3

Mapping Amino Acid Letter Codes to Integers (Continued)

Amino Acid	Code	Integer
Aspartic acid (Aspartate)	D	4
Cysteine	C	5
Glutamine	Q	6
Glutamic acid (Glutamate)	E	7
Glycine	G	8
Histidine	H	9
Isoleucine	I	10
Leucine	L	11
Lysine	K	12
Methionine	M	13
Phenylalanine	F	14
Proline	P	15
Serine	S	16
Threonine	T	17
Tryptophan	W	18
Tyrosine	Y	19
Valine	V	20
Asparagine or Aspartic acid (Aspartate)	B	21
Glutamine or Glutamic acid (Glutamate)	Z	22
Unknown amino acid (any amino acid)	X	23
Translation stop	*	24

Mapping Amino Acid Letter Codes to Integers (Continued)

Amino Acid	Code	Integer
Gap of indeterminate length	-	25
Unknown character (any character or symbol not in table)	?	0

Examples

Converting a Simple Sequence

Convert the sequence of letters MATLAB to integers.

```
SeqInt = aa2int('MATLAB')

SeqInt =

    13     1    17    11     1    21
```

Converting a Random Sequence

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

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

SeqChar =

    d w c z t e c a k f u e c v i f c h d s
```

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

```
SeqInt = aa2int(SeqChar)

SeqInt =

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

    Columns 14 through 20
```

20 10 14 5 9 4 16

See Also

Bioinformatics Toolbox™ functions: aminolookup, int2aa, int2nt, nt2int

aa2nt

Purpose

Convert amino acid sequence to nucleotide sequence

Syntax

SeqNT = aa2nt(*SeqAA*)

SeqNT = aa2nt(*SeqAA*, ...'GeneticCode',

GeneticCodeValue, ...)

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

Arguments*SeqAA*

One of the following:

- String of single-letter codes specifying an amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers on page 2-2. Unknown characters are mapped to 0.
- Row vector of integers specifying an amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes on page 2-558.
- MATLAB structure containing a `Sequence` field that contains an amino acid sequence, such as returned by `fastaread`, `getgenpept`, `genpeptread`, `getpdb`, or `pdbread`.

Examples: 'ARN' or [1 2 3]

GeneticCodeValue

Integer or string specifying a genetic code number or code name from the table Genetic Code on page 2-9. Default is 1 or 'Standard'.

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

AlphabetValue

String specifying a nucleotide alphabet. Choices are:

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

Return Values

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

Description

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

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

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

SeqNT = aa2nt(*SeqAA*, ... '*GeneticCode*', *GeneticCodeValue*, ...) specifies a genetic code to use when converting an amino acid sequence to a nucleotide sequence. *GeneticCodeValue* can be an integer or string specifying a code number or code name from the table Genetic Code on page 2-9. Default is 1 or 'Standard'. The amino acid to nucleotide codon mapping for the Standard genetic code is shown in the table Standard Genetic Code on page 2-10.

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

SeqNT = aa2nt(*SeqAA*, ... '*Alphabet*' *AlphabetValue*, ...) specifies a nucleotide alphabet. *AlphabetValue* can be 'DNA', which

uses the symbols A, C, G, and T, or 'RNA', which uses the symbols A, C, G, and U. Default is 'DNA'.

Genetic Code

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

Standard Genetic Code

Amino Acid Name	Amino Acid Code	Nucleotide Codon
Alanine	A	GCT GCC GCA GCG
Arginine	R	CGT CGC CGA CCG AGA AGG
Asparagine	N	ATT AAC
Aspartic acid (Aspartate)	D	GAT GAC
Cysteine	C	TGT TGC
Glutamine	Q	CAA CAG
Glutamic acid (Glutamate)	E	GAA GAG
Glycine	G	GGT GGC GGA GGG
Histidine	H	CAT CAC
Isoleucine	I	ATT ATC ATA
Leucine	L	TTA TTG CTT CTC CTA CTG
Lysine	K	AAA AAG
Methionine	M	ATG
Phenylalanine	F	TTT TTC
Proline	P	CCT CCC CCA CCG
Serine	S	TCT TCC TCA TCG AGT AGC
Threonine	T	ACT ACC ACA ACG
Tryptophan	W	TGG
Tyrosine	Y	TAT, TAC
Valine	V	GTT GTC GTA GTG

Standard Genetic Code (Continued)

Amino Acid Name	Amino Acid Code	Nucleotide Codon
Asparagine or Aspartic acid (Aspartate)	B	Random codon from D and N
Glutamine or Glutamic acid (Glutamate)	Z	Random codon from E and Q
Unknown amino acid (any amino acid)	X	Random codon
Translation stop	*	TAA TAG TGA
Gap of indeterminate length	-	---
Unknown character (any character or symbol not in table)	?	???

Examples

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

```
aa2nt('MATLAP')
```

```
ans =
```

```
ATGGCGACGTTAGCGCCG
```

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

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

```
ans =
```

```
ATGGCAACTCTAGCGCCT
```

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

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

```
ans =
```

```
AUGGCCACAUUGGCACCU
```

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

```
aa2nt('abcd')
```

```
Warning: The sequence contains ambiguous characters.
```

```
ans =
```

```
GCCACATGCGAC
```

See Also

Bioinformatics Toolbox functions: aminolookup, baselookup, geneticcode, nt2aa, revgeneticcode, seqtool

MATLAB function: rand

Purpose Count amino acids in sequence

Syntax

```

AAStruct = aaccount(SeqAA)
AAStruct = aaccount(SeqAA, ... 'Chart', ChartValue, ...)
AAStruct = aaccount(SeqAA, ... 'Others', OthersValue, ...)
AAStruct = aaccount(SeqAA, ... 'Structure', StructureValue,
    ...)

```

Arguments

<i>SeqAA</i>	One of the following:
	<ul style="list-style-type: none"> • String of single-letter codes specifying an amino acid sequence. For valid letter codes, see the table Mapping Amino Acid Letter Codes to Integers on page 2-2. Unknown characters are mapped to 0. • Row vector of integers specifying an amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes on page 2-558. • MATLAB structure containing a <code>Sequence</code> field that contains an amino acid sequence, such as returned by <code>fastaread</code>, <code>getgenpept</code>, <code>genpeptread</code>, <code>getpdb</code>, or <code>pdbread</code>.
	Examples: 'ARN' or [1 2 3]
<i>ChartValue</i>	String specifying a chart type. Choices are 'pie' or 'bar'.

<i>OthersValue</i>	String specifying how to count ambiguous characters (B, Z, X), the stop character (*), and gaps indicated by a hyphen (-). Choices are 'full' (lists the ambiguous characters in separate fields) or 'bundle' (lists the ambiguous characters together in the field <i>Others</i>). Default is 'bundle'.
<i>StructureValue</i>	Suppresses the unknown characters warning when set to 'full' .

Return Values

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

Description

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

- If a sequence contains ambiguous characters (B, Z, or X), the stop character (*), or gaps indicated with a hyphen (-), then these characters are counted in the field *Others* and the following warning message appears:

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

- If a sequence contains characters other than the 20 standard amino acids, ambiguous, stop, and gap characters, then these characters are counted in the field *Others*, and the following warning message appears:

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

- If the property 'Others' is set to 'full', ambiguous characters are listed separately in the fields B, Z, X, Stop, and Gap.

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

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

AAStruct = `aaccount(SeqAA, ...'Others', OthersValue, ...)` specifies how to count ambiguous characters (B, Z, or X), the stop character (*), and gaps indicated by a hyphen (-). Choices are 'full' (lists the ambiguous characters in separate fields) or 'bundle' (lists the ambiguous characters together in the field Others). Default is 'bundle'.

AAStruct = `aaccount(SeqAA, ...'Structure', StructureValue, ...)` suppresses the unknown characters warning when *StructureValue* is set to 'full'.

- `aaccount(SeqAA)` — Displays fields for 20 amino acids, and, if there are ambiguous or unknown characters, adds an Others field with the ambiguous and unknown character counts.
- `aaccount(SeqAA, 'Others', 'full')` — Displays fields for 20 amino acids, 3 ambiguous amino acids, stops, gaps, and, if there are unknown characters, adds an Others field with the unknown counts.
- `aaccount(SeqAA, 'Structure', 'full')` — Displays fields for 20 amino acids and an Others field. If there are ambiguous or unknown characters, adds the counts to the Others field; otherwise displays 0 in the Others field.
- `aaccount(SeqAA, 'Others', 'full', 'Structure', 'full')` — Displays fields for 20 amino acids, 3 ambiguous amino acids, stops,

gaps, and an `Others` field. If there are unknown characters, add the counts to the `Others` field; otherwise displays 0 in the `Others` field.

Examples

- 1 Create an amino acid sequence.

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

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

```
AA = aaccount(Seq)
```

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

```
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
```

- 4** Create a random character string to represent an amino acid sequence.

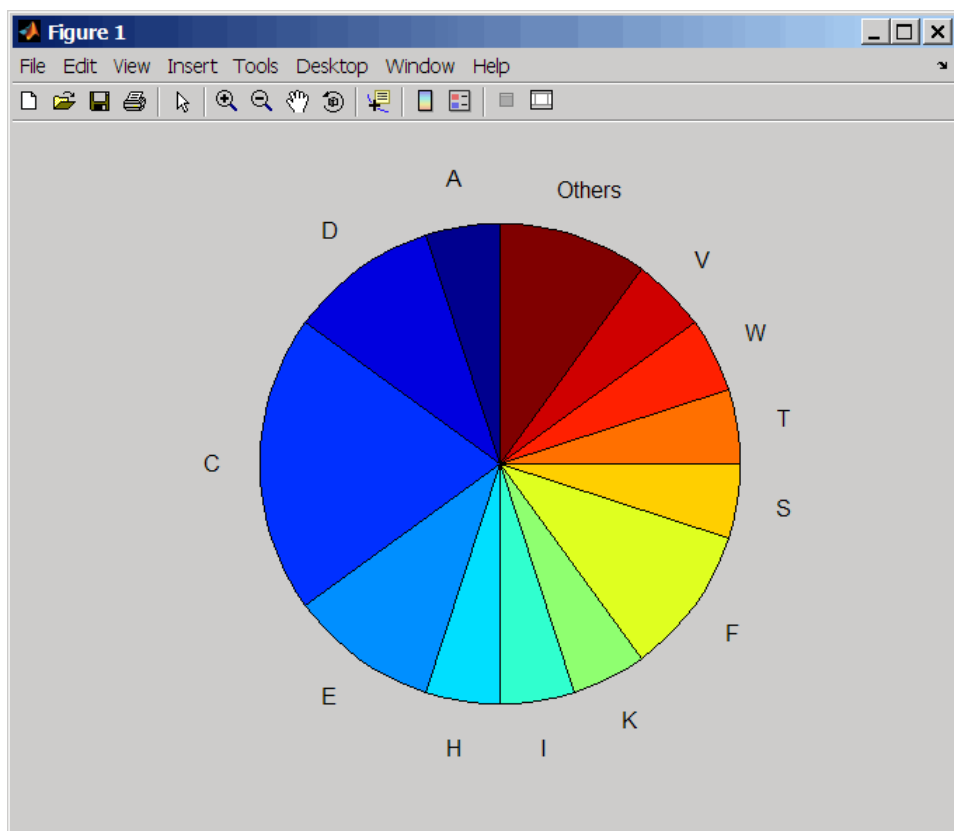
```
Seq = randseq(20, 'alphabet', 'amino')
```

```
Seq =
```

```
dwcztacakfuecvifchds
```

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

```
AA = aaccount(Seq, 'chart', 'pie');
```



See Also

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

Purpose

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

Syntax

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

Arguments

CELFiles

Any of the following:

- String specifying a single CEL file name.
- '*', which reads all CEL files in the current directory.
- ' ', which opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold **Ctrl** or **Shift** while clicking to select multiple CEL files.
- Cell array of CEL file names.

CDFFile

Either of the following:

- String specifying a CDF file name.
- ' ', which opens the Select CDF File dialog box from which you select the CDF file.

<i>SeqFile</i>	<p>Either of the following:</p> <ul style="list-style-type: none">• String specifying a file name of a sequence file (tab-separated or FASTA) that contains the following information for a specific type of Affymetrix GeneChip array:<ul style="list-style-type: none">▪ Probe set IDs▪ Probe <i>x</i>-coordinates▪ Probe <i>y</i>-coordinates▪ Probe sequences in each probe set▪ Affymetrix GeneChip array type (FASTA file only) <p>The sequence file (tab-separated or FASTA) must be on the MATLAB search path or in the Current Directory (unless you use the <i>SeqPath</i> property). In a tab-separated file, each row represents a probe; in a FASTA file, each header represents a probe.</p> <ul style="list-style-type: none">• An <i>N</i>-by-25 matrix of sequence information, such as returned by <i>affyprobeseqread</i>.
<i>ProbeStructure</i>	<p>MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs, returned by the <i>celintensityread</i> function.</p>
<i>CELPathValue</i>	<p>String specifying the path and directory where the files specified in <i>CELFiles</i> are stored.</p>
<i>CDFPathValue</i>	<p>String specifying the path and directory where the file specified in <i>CDFFile</i> is stored.</p>
<i>SeqPathValue</i>	<p>String specifying a directory or path and directory where <i>SeqFile</i> is stored.</p>

<i>ChipIndexValue</i>	Positive integer specifying a chip. This chip's sequence information and mismatch probe intensity data is used to compute probe affinities. Default is 1.
<i>OpticalCorrValue</i>	Controls the use of optical background correction on the input probe intensity values. Choices are <code>true</code> (default) or <code>false</code> .
<i>CorrConstValue</i>	Value that specifies the correlation constant, ρ , for log background intensity for each PM/MM probe pair. Choices are any value ≥ 0 and ≤ 1 . Default is 0.7.
<i>MethodValue</i>	String that specifies the method to estimate the signal. Choices are 'MLE', a faster, ad hoc Maximum Likelihood Estimate method, or 'EB', a slower, more formal, empirical Bayes method. Default is 'MLE'.
<i>TuningParamValue</i>	Value that specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

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

<i>GSBCorrValue</i>	Specifies whether to perform gene-specific binding (GSB) correction using probe affinity data. Choices are <code>true</code> (default) or <code>false</code> . If there is no probe affinity information, this property is ignored.
<i>MedianValue</i>	Specifies the use of the median of the ranked values instead of the mean for normalization. Choices are <code>true</code> or <code>false</code> (default).

OutputValue

Specifies the scale of the returned gene expression values. Choices are:

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

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

ShowplotValue

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

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

VerboseValue

Controls the display of the status of the reading of files and GCRMA processing. Choices are `true` (default) or `false`.

Return Values

Expression DataMatrix object containing the \log_2 gene expression values that have been background adjusted, normalized, and summarized using the GC Robust Multi-array Average (GCRMA) procedure.

Each row in *Expression* corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file.

Description

Note This function does not work on the Solaris™ platform.

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

CELFiles is a string or cell array of CEL file names. *CDFFile* is a string specifying a CDF file name. If you set *CELFiles* to '*', then it reads all CEL files in the current directory. If you set *CELFiles* to ' ', then it opens the Select CEL Files dialog box from which you select the CEL files. *SeqFile* is a file or matrix containing sequence information for probes on a specific type of Affymetrix GeneChip array.

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

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

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

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

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

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

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

Expression = `affygcma(..., 'OpticalCorr', OpticalCorrValue, ...)` controls the use of optical background

correction on the input probe intensity values. Choices are `true` (default) or `false`.

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

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

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

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

Expression = `affygcRMA(..., 'GSBCorr', GSBCorrValue, ...)` specifies whether to perform gene-specific binding (GSB) correction using probe affinity data. Choices are `true` (default) or `false`. If there is no probe affinity information, this property is ignored.

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

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

- 'log'
- 'log2'

- 'log10'
- 'natural'
- @*functionname*

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

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

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

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

Examples

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

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

References

- [1] Naef, F., and Magnasco, M.O. (2003). Solving the Riddle of the Bright Mismatches: Labeling and Effective Binding in Oligonucleotide Arrays. *Physical Review E* 68, 011906.
- [2] Wu, Z., Irizarry, R.A., Gentleman, R., Murillo, F.M., and Spencer, F. (2004). A Model Based Background Adjustment for Oligonucleotide Expression Arrays. *Journal of the American Statistical Association* 99(468), 909–917.
- [3] Wu, Z., and Irizarry, R.A. (2005). Stochastic Models Inspired by Hybridization Theory for Short Oligonucleotide Arrays. *Proceedings of RECOMB 2004. J Comput Biol.* 12(6), 882–93.
- [4] Wu, Z., and Irizarry, R.A. (2005). A Statistical Framework for the Analysis of Microarray Probe-Level Data. *Johns Hopkins University, Biostatistics Working Papers* 73.
- [5] Wu, Z., and Irizarry, R.A. (2003). A Model Based Background Adjustment for Oligonucleotide Expression Arrays. *RSS Workshop on Gene Expression, Wye, England*, <http://biosun01.biostat.jhsph.edu/%7Eririzarr/Talks/gctalk.pdf>.
- [6] Speed, T. (2006). Background models and GCRMA. Lecture 10, *Statistics 246, University of California Berkeley*. <http://www.stat.berkeley.edu/users/terry/Courses/s246.2006/-Week10/Week10L1.pdf>.
- [7] Abd Rabbo, N.A., and Barakat, H.M. (1979). Estimation Problems in Bivariate Lognormal Distribution. *Indian J. Pure Appl. Math* 10(7), 815–825.
- [8] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate

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

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

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

See Also

Bioinformatics Toolbox functions: `affyprobeaffinities`, `affyprobeseqread`, `affyrma`, `celintensityread`, `gcrrma`, `gcrrmabackadj`, `mafdr`, `matteest`, `quantilenorm`, `rmasummary`

affyinvvarsetnorm

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

Syntax

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

Arguments

<i>Data</i>	Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL or DAT file. (Each CEL or DAT file is generated from a separate chip. All chips should be of the same type.)
<i>MedStructure</i>	Structure of each column's intensity median before and after normalization, and the index of the column chosen as the baseline.
<i>BaselineValue</i>	Property to control the selection of the column index <i>N</i> from <i>Data</i> to be used as the baseline column. Default is the column index whose median intensity is the median of all the columns.

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

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

StopPercentileValue Property to stop the iteration process when the number of data points in the invariant set reaches N percent of the total number of data points. Default is 1.

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

RayPercentileValue Property to select the N percentage of the highest ranked invariant set of data points to fit a straight line through, while the remaining data points are fitted to a running median curve. The final running median curve is a piecewise linear curve. Default is 1.5.

affyinvarsetnorm

<i>MethodValue</i>	Property to select the smoothing method used to normalize the data. Enter 'lowess' or 'runmedian'. Default is 'lowess'.
<i>ShowplotValue</i>	Property to control the plotting of two pairs of scatter plots (before and after normalization). The first pair plots baseline data versus data from a specified column (chip) from the matrix <i>Data</i> . The second is a pair of M-A scatter plots, which plots M (ratio between baseline and sample) versus A (the average of the baseline and sample). Enter either 'all' (plot a pair of scatter plots for each column or chip) or specify a subset of columns (chips) by entering the column number(s) or a range of numbers.

For example:

- ..., 'Showplot', 3, ...) plots data from column 3.
- ..., 'Showplot', [3,5,7], ...) plots data from columns 3, 5, and 7.
- ..., 'Showplot', 3:9, ...) plots data from columns 3 to 9.

Description

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

Specifically, `affyinvarsetnorm`:

- Selects a baseline index, typically the column whose median intensity is the median of all the columns.

- For each column, determines the proportional rank difference (*prd*) for each pair of ranks, *RankX* and *RankY*, from the sample column and the baseline reference.

$$prd = \text{abs}(\text{Rank}X - \text{Rank}Y)$$

- For each column, determines the invariant set of data points by selecting data points whose proportional rank differences (*prd*) are below *threshold*, which is a predetermined threshold for a given data point (defined by the *ThresholdsValue* property). It repeats the process until either no more data points are eliminated, or a predetermined percentage of data points is reached.

The invariant set is data points with a $prd < \text{threshold}$.

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

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

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

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

... `affyinvarsetnorm(..., 'Baseline', BaselineValue, ...)` lets you select the column index *N* from *Data* to be the baseline column.

affyinvarsetnorm

Default is the index of the column whose median intensity is the median of all the columns.

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

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

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

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

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

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

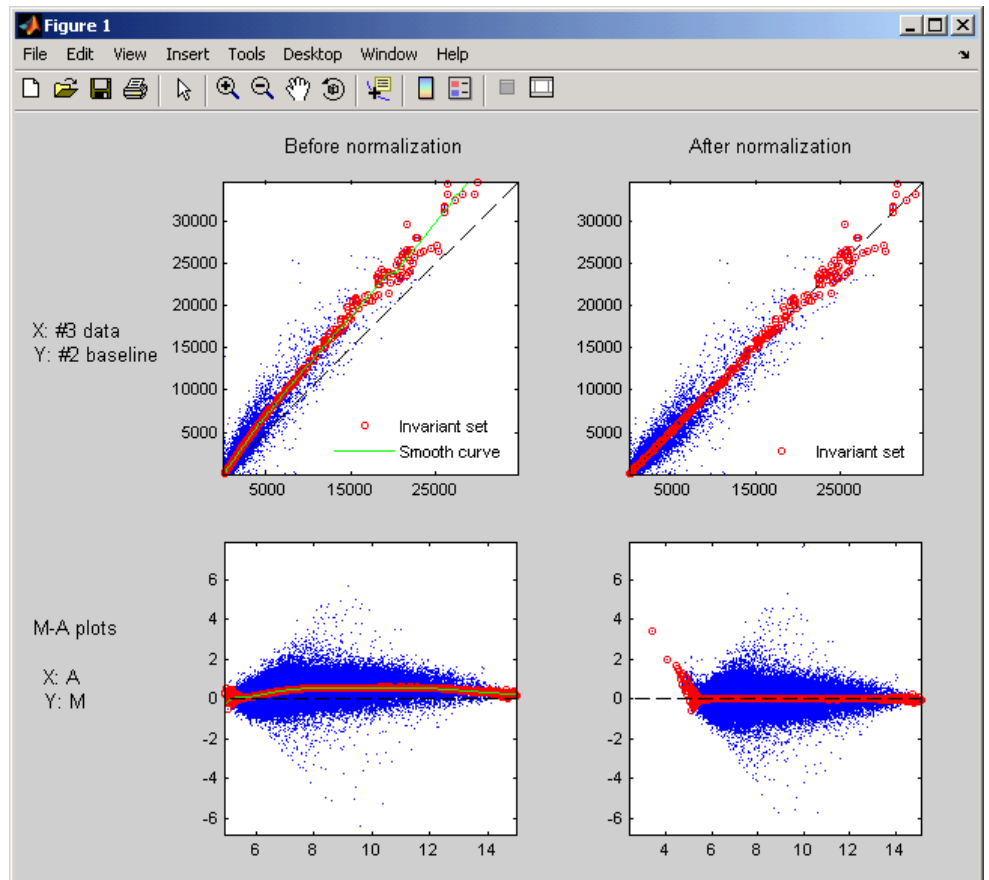
When *MethodValue* is 'runmedian', `affyinvarsetnorm` uses the running median method. Default is 'lowess'.

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

For example:

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

affyinvarsetnorm



Examples

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

```
load prostatecancerrawdata
```

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

```
NormMatrix = affyinvsetnorm(pmMatrix);
```

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

References

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

[2] <http://biosun1.harvard.edu/complab/dchip/normalizing%20arrays.htm#isn>

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

See Also

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

affyprobeaffinities

Purpose Compute Affymetrix probe affinities from their sequences and MM probe intensities

Syntax

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

Arguments

SequenceMatrix

An N -by-25 matrix of sequence information for the perfect match (PM) probes on an Affymetrix GeneChip array, where N is the number of probes on the array. Each row corresponds to a probe, and each column corresponds to one of the 25 sequence positions. Nucleotides in the sequences are represented by one of the following integers:

- 0 — None
- 1 — A
- 2 — C
- 3 — G
- 4 — T

Tip You can use the `affyprobeseqread` function to generate this matrix. If you have this sequence information in letter representation, you can convert it to integer representation using the `nt2int` function.

MMIntensity

Column vector containing mismatch (MM) probe intensities from a CEL file, generated from a single Affymetrix GeneChip array. Each row corresponds to a probe.

Tip You can extract this column vector from the `MMIntensities` matrix returned by the `celintensityread` function.

affyprobeaffinities

ProbeIndicesValue Column vector containing probe indexing information. Probes within a probe set are numbered 0 through $N - 1$, where N is the number of probes in the probe set.

Tip You can use the `affyprobeseqread` function to generate this column vector.

ShowplotValue Controls the display of a plot showing the affinity values of each of the four bases (A, C, G, and T) for each of the 25 sequence positions, for all probes on the Affymetrix GeneChip array. Choices are `true` or `false` (default).

Return Values

AffinPM Column vector of PM probe affinities, computed from their probe sequences and MM probe intensities.

AffinMM Column vector of MM probe affinities, computed from their probe sequences and MM probe intensities.

<i>BaseProf</i>	4-by-4 matrix containing the four parameters for a polynomial of degree 3, for each base, A, C, G, and T. Each row corresponds to a base, and each column corresponds to a parameter. These values are estimated from the probe sequences and intensities, and represent all probes on an Affymetrix GeneChip array.
<i>Stats</i>	Row vector containing four statistics in the following order: <ul style="list-style-type: none"> • R-square statistic • F statistic • p-value • Error variance

Description

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

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

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

affyprobeaffinities

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

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

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

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

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

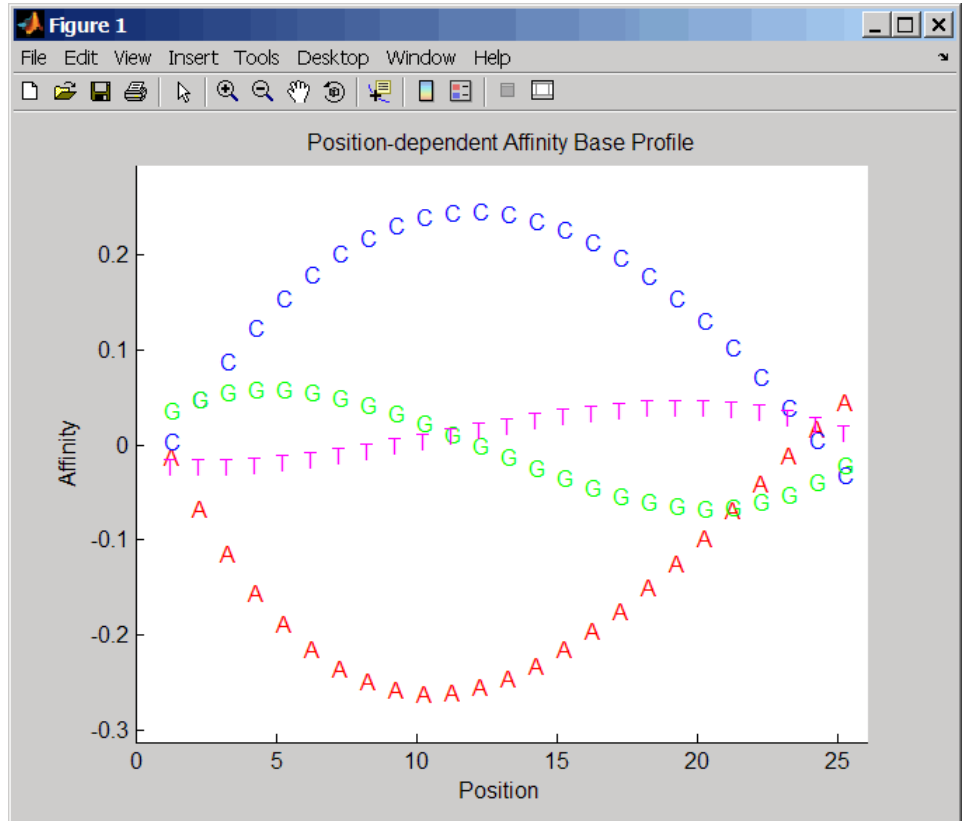
Examples

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

```
load prostatecancerrawdata
```

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

```
[apm, amm] = affyprobeaffinities(seqMatrix, mmMatrix(:,1),...  
                                'ProbeIndices', probeIndices, 'showplot', true);
```



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

References

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

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

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

See Also

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

Purpose Read data file containing probe sequence information for Affymetrix GeneChip array

Syntax

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

affyprobeseqread

Arguments

SeqFile

String specifying a file name of a sequence file (tab-separated or FASTA) that contains the following information for a specific type of Affymetrix GeneChip array:

- Probe set IDs
- Probe *x*-coordinates
- Probe *y*-coordinates
- Probe sequences in each probe set
- Affymetrix GeneChip array type (FASTA file only)

The sequence file (tab-separated or FASTA) must be on the MATLAB search path or in the Current Directory (unless you use the `SeqPath` property). In a tab-separated file, each row represents a probe; in a FASTA file, each header represents a probe.

CDFFile

Either of the following:

- String specifying a file name of an Affymetrix CDF library file, which contains information that specifies which probe set each probe belongs to on a specific type of Affymetrix GeneChip array. The CDF library file must be on the MATLAB search path or in the MATLAB Current Directory (unless you use the `CDFPath` property).
- CDF structure, such as returned by the `affyread` function, which contains information that specifies which probe set each probe belongs to on a specific type of Affymetrix GeneChip array.

Caution Make sure that *SeqFile* and *CDFFile* contain information for the same type of Affymetrix GeneChip array.

SeqPathValue String specifying a directory or path and directory where *SeqFile* is stored.

CDFFPathValue String specifying a directory or path and directory where *CDFFile* is stored.

SeqOnlyValue Controls the return of a structure, *Struct*, with only one field, *SequenceMatrix*. Choices are true or false (default).

Return Values

Struct MATLAB structure containing the following fields:

- ProbeSetIDs
- ProbeIndices
- SequenceMatrix

Description

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

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

affyprobeseqread

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

`Struct = affyprobeseqread(SeqFile, CDFFile, ... 'PropertyName', PropertyValue, ...)` calls `affyprobeseqread` with optional properties that use property name/property value

pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

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

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

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

Examples

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

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

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

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

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

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

See Also

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

affyread

Purpose

Read microarray data from Affymetrix GeneChip file

Syntax

```
AffyStruct = affyread(File)
```

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

Arguments

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

- **EXP** — Data file containing information about experimental conditions and protocols.
- **DAT** — Data file containing raw image data (pixel intensity values).
- **CEL** — Data file containing information about the intensity values of the individual probes.
- **CHP** — Data file containing summary information of the probe sets, including intensity values.
- **CDF** — Library file containing information about which probes belong to which probe set.
- **GIN** — Library file containing information about the probe sets, such as the gene name with which the probe set is associated.

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

LibraryPath String specifying the path and directory of a:

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

Note If you do not specify *LibraryPath* when reading a CHP file, `affyread` looks in the current directory for the CDF file. If it does not find the CDF file, it still reads the CHP file, but the probe set names and types will be omitted from the return value, *AffyStruct*.

Return Values

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

Description

Note This function does not work on the Solaris platform.

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

AffyStruct = `affyread(File, LibraryPath)` specifies the path and directory of a:

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

Note If you do not specify *LibraryPath* when reading a CHP file, `affyread` looks in the current directory for the CDF file. If it does not find the CDF file, it still reads the CHP file, but the probe set names and types will be omitted from the return value, *AffyStruct*.

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

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

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

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

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

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

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

Field	Description
Name	File name.
DataPath	Path and directory of the file.
LibPath	Path and directory of the CDF and GIN library files associated with the file being read.
FullPathName	Path and directory of the file.
ChipType	Name of the Affymetrix GeneChip array (for example, DrosGenome1 or HG-Focus).
Date	Date the file was created.

EXP File

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

DAT File

Field	Description
NumPixelsPerRow	Number of pixels per row in the image created from the GeneChip array (number of columns).
NumRows	Number of rows in the image created from the GeneChip array.
MinData	Minimum intensity value in the image created from the GeneChip array.

DAT File (Continued)

Field	Description
MaxData	Maximum intensity value in the image created from the GeneChip array.
PixelSize	Size of one pixel in the image created from the GeneChip array.
CellMargin	Size of gaps between cells in the image created from the GeneChip array.
ScanSpeed	Speed of the scanner used to create the image.
ScanDate	Date the scan was performed.
ScannerID	Name of the scanning device used.
UpperLeftX UpperLeftY UpperRightX UpperRightY LowerLeftX LowerLeftY LowerRightX LowerRightY	Pixel coordinates of the scanned image.
ServerName	Not used.
Image	A NumRows-by-NumPixelsPerRow image of the scanned GeneChip array.

CEL File

Field	Description
FileVersion	Version of the CEL file format.
Algorithm	Algorithm used in the image processing step that converts from DAT format to CEL format.

CEL File (Continued)

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

CEL File (Continued)

Field	Description
ProbeColumnNames	<p>Cell array containing the eight column names in the Probes field:</p> <ul style="list-style-type: none"> • PosX — <i>x</i>-coordinate of the cell • PosY — <i>y</i>-coordinate of the cell • Intensity — Intensity value of the cell • StdDev — Standard deviation of intensity value • Pixels — Number of pixels in the cell • Outlier — True/false flag indicating if the cell was marked as an outlier • Masked — True/false flag indicating if the cell was masked • ProbeType — Integer indicating the probe type (for example, 1 = expression)
Probes	<p>NumProbes-by-8 array of information about the individual probes, including intensity values. The columns of this array are contained in the ProbeColumnNames field.</p>

CHP File

Field	Description
AssayType	Type of assay that the GeneChip array contained (for example, Expression, Genotyping, or Resequencing).
CellFile	File name of the CEL file from which the CHP file was created.
Algorithm	Algorithm used to convert from CEL format to CHP format.

CHP File (Continued)

Field	Description
AlgVersion	Version of the algorithm used to create the CHP file.
NumAlgParams	Number of parameters in AlgParams.
AlgParams	String containing parameters used in steps needed to create the CHP file (for example, background correction).
NumChipSummary	Number of entries in ChipSummary.
ChipSummary	Summary information for the GeneChip array, including background average, standard deviation, max, and min.
BackgroundZones	Structure containing information about the zones used in the background adjustment step.
Rows	Number of rows of probes.
Cols	Number of columns of probes.
NumProbeSets	Number of probe sets on the GeneChip array.
NumQCProbeSets	Number of QC probe sets on the GeneChip array.

CHP File (Continued)

Field	Description
ProbeSets (Expression GeneChip array)	<p>A NumProbeSets-by-1 structure array containing information for each expression probe set, including the following fields:</p> <ul style="list-style-type: none"> • Name — Name of the probe set. • ProbeSetType — Type of the probe set. • CompDataExists — True/false flag indicating if the probe set has additional computed information. • NumPairs — Number of probe pairs in the probe set. • NumPairsUsed — Number of probe pairs in the probe set used for calculating the probe set signal (not masked). • Signal — Summary intensity value for the probe set. • Detection — Indicator of statistically significant difference between the intensity value of the PM probes and the intensity value of the MM probes in a single probe set (Present, Absent, or Marginal). • DetectionPValue — P value for the Detection indicator. • CommonPairs — When CompDataExists is true, contains the number of common pairs between the experiment and the baseline after outliers and masked probes have been removed. • SignalLogRatio — When CompDataExists is true, contains the change in signal between the experiment and baseline. • SignalLogRatioLow — When CompDataExists is true, contains the lowest ratios of probes between the experiment and the baseline. • SignalLogRatioHigh — When CompDataExists is true, contains the highest ratios of probes between the experiment and the baseline. • Change — When CompDataExists is true, describes how the probe is changed versus a baseline experiment. Choices are Increase, Marginal Increase, No Change, Decrease, or Marginal Decrease. • ChangePValue — When CompDataExists is true, contains the p-value associated with Change.

CHP File (Continued)

Field	Description
ProbeSets (Genotyping GeneChip array)	A NumProbeSets-by-1 structure array containing information for each genotyping probe set, including the following fields: <ul style="list-style-type: none"> • Name — Name of the probe set. • AlleleCall — Allele that is present for the probe set. Possibilities are AA (homozygous for the major allele), AB (heterozygous for the major and minor allele), BB (homozygous for the minor allele), or NoCall (unable to determine allele). • Confidence — A measure of the accuracy of the allele call. • RAS1 — Relative Allele Signal 1 for the SNP site, which is calculated using sense probes. • RAS2— Relative Allele Signal 2 for the SNP site, which is calculated using antisense probes. • PValueAA — p-value for an AA call. • PValueAB — p-value for an AB call. • PValueBB — p-value for a BB call. • PValueNoCall — p-value for a NoCall call.
ProbeSets (Resequencing GeneChip array)	A NumProbeSets-by-1 structure array containing information for each resequencing probe set, including the following fields: <ul style="list-style-type: none"> • CalledBases — A 1-by-NumProbeSets character array containing the bases called by the resequencing algorithm. Possible values are a, c, g, t, and n. • Scores — A 1-by-NumProbeSets array containing the score associated with each base call.

CDF File

Field	Description
Rows	Number of rows of probes.
Cols	Number of columns of probes.
NumProbeSets	Number of probe sets on the GeneChip array.
NumQCProbeSets	Number of QC probe sets on the GeneChip array.

CDF File (Continued)

Field	Description
ProbeSetColumnNames	<p>Cell array containing the six column names in the ProbePairs field in the ProbeSets array:</p> <ul style="list-style-type: none"> • GroupNumber — Number identifying the group to which the probe pair belongs. For expression arrays, this is always 1. For genotyping arrays, this is typically 1 (allele A, sense), 2 (allele B, sense), 3 (allele A, antisense), or 4 (allele B, antisense). • Direction — Number identifying the direction of the probe pair. 1 = sense and 2 = antisense. • PMPosX — <i>x</i>-coordinate of the perfect match probe. • PMPosY — <i>y</i>-coordinate of the perfect match probe. • MMPosX — <i>x</i>-coordinate of the mismatch probe. • MMPosY — <i>y</i>-coordinate of the mismatch probe.
ProbeSets	<p>A NumProbeSets-by-1 structure array containing information for each probe set, including the following fields:</p> <ul style="list-style-type: none"> • Name — Name of the probe set. • ProbeSetType — Type of the probe set. • CompDataExists — True/false flag indicating if the probe set has additional computed information. • NumPairs — Number of probe pairs in the probe set. • NumQCProbes — Number of QC probes in the probe set. • QCType — Type of QC probes. • GroupNames — Name of the group to which the probe set belongs. For expression arrays, this is the name of the probe set. For genotyping arrays, this is the name of the alleles, for example {'A' 'C' 'A' 'C'}. • ProbePairs — NumPairs-by-6 array of information about the probe pairs. The column names of this array are contained in the ProbeSetColumnNames field.

GIN File

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

Examples

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

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

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

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

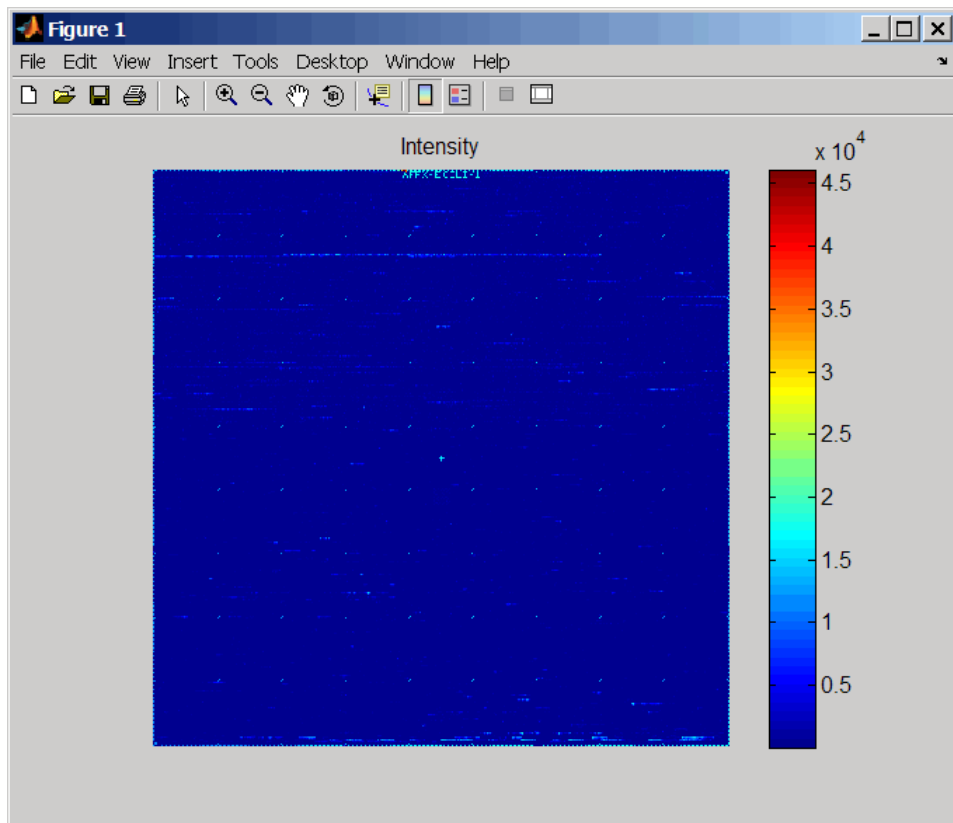
The following example assumes that files `Ecoli-antisense-121502.CEL`, `Ecoli-antisense-121502.dat`, and `Ecoli-antisense-121502.chp` are stored on the MATLAB search path or in the current directory. It also assumes that the associated CDF library file, `Ecoli_ASv2.CDF`, is stored at `D:\Affymetrix\LibFiles\Ecoli`.

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

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

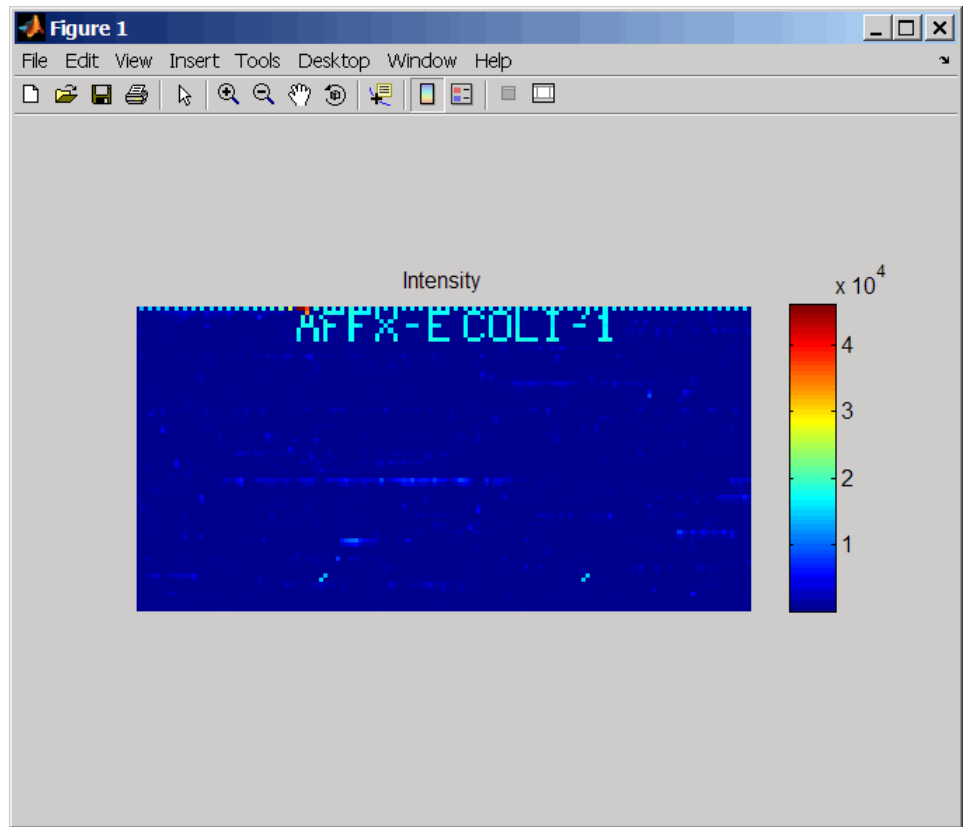
- 2 Display a spatial plot of the probe intensities.

```
mimage(celStruct, 'Intensity')
```



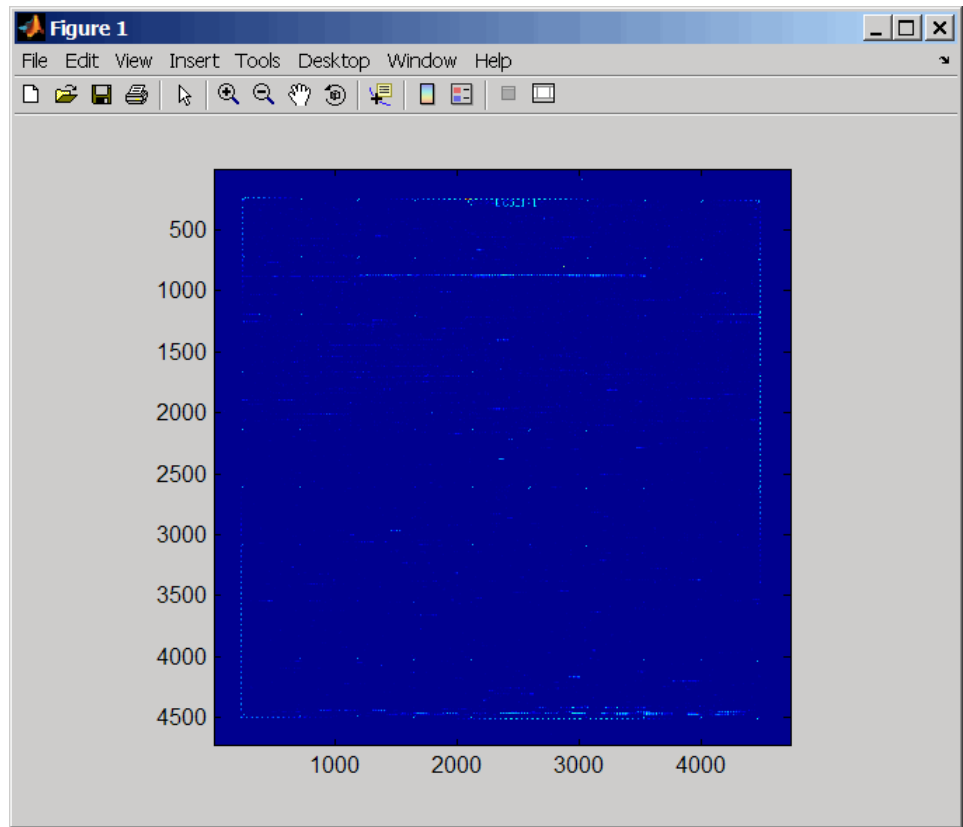
- 3 Zoom in on a specific region of the plot.

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

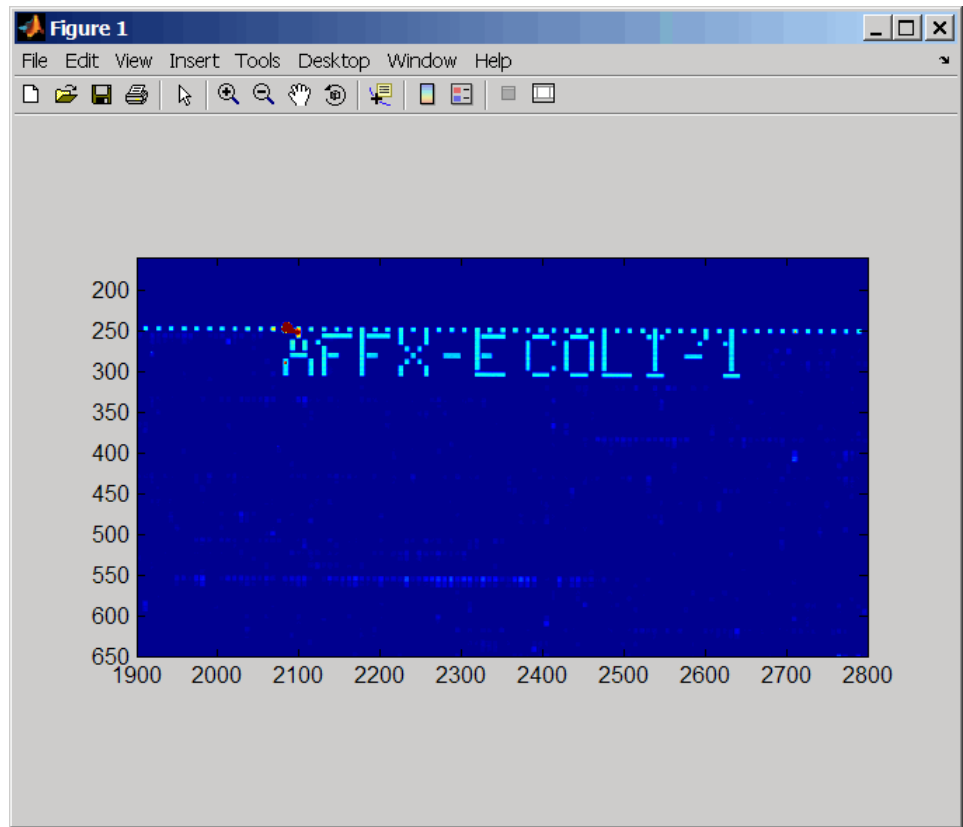
- 4 Read the contents of a DAT file into a MATLAB structure, display the raw image data, and then use the `axis image` function to set the correct aspect ratio.

```
datStruct = affyread('Ecoli-antisense-121502.dat');  
imagesc(datStruct.Image)  
axis image
```



5 Zoom in on a specific region of the plot.

```
axis([1900 2800 160 650])
```



- 6 Read the contents of a CHP file into a MATLAB structure, specifying the location of the associated CDF library file. Then extract information for probe set 3315278.

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

```
geneName =
```

affyread

```
Identifier: '3315278'  
ProbeSetName: 'argG_b3172_at'  
CDFIndex: 5213  
GINIndex: 3074  
Description: [1x82 char]  
Source: 'NCBI EColi Genome'  
SourceURL: [1x74 char]
```

See Also

Bioinformatics Toolbox functions: `affysnannotread`,
`affysnpintensitysplit`, `agferead`, `celintensityread`,
`geoseriesread`, `gprread`, `ilmnbsread`, `probelibraryinfo`,
`probesetlink`, `probesetlookup`, `probesetplot`, `probesetvalues`,
`sptread`

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

Syntax

```

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

```

Arguments

<i>CELFiles</i>	Any of the following: <ul style="list-style-type: none"> • String specifying a single CEL file name. • '*', which reads all CEL files in the current directory. • ' ', which opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold Ctrl or Shift while clicking to select multiple CEL files. • Cell array of CEL file names.
<i>CDFFile</i>	Either of the following: <ul style="list-style-type: none"> • String specifying a CDF file name. • ' ', which opens the Select CDF File dialog box from which you select the CDF file.

<i>ProbeStructure</i>	MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs, returned by the <code>celintensityread</code> function.
<i>CELPathValue</i>	String specifying the path and directory where the files specified in <i>CELFiles</i> are stored.
<i>CDFPathValue</i>	String specifying the path and directory where the file specified in <i>CDFFile</i> is stored.
<i>MethodValue</i>	Specifies the estimation method for the background adjustment model parameters. Choices are 'RMA' (to use estimation method described by Bolstad, 2005) or 'MLE' (to estimate the parameters using maximum likelihood). Default is 'RMA'.
<i>TruncateValue</i>	Specifies the background noise model. Choices are <code>true</code> (use a truncated Gaussian distribution) or <code>false</code> (use a nontruncated Gaussian distribution). Default is <code>true</code> .
<i>MedianValue</i>	Specifies the use of the median of the ranked values instead of the mean for normalization. Choices are <code>true</code> or <code>false</code> (default).

OutputValue Specifies the scale of the returned gene expression values. Choices are:

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

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

ShowplotValue Controls the plotting of a histogram showing the distribution of PM probe intensity values (blue) and the convoluted probability distribution function (red), with estimated parameters mu, sigma and alpha. Enter either 'all' (plot a histogram for each column or chip) or specify a subset of columns (chips) by entering the column number, list of numbers, or range of numbers.

For example:

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

VerboseValue Controls the display of the status of the reading of files and RMA processing. Choices are true (default) or false.

Return Values

Expression DataMatrix object containing the log₂ based gene expression values that have been background adjusted, normalized, and summarized using the Robust Multi-array Average (RMA) procedure.

Each row in *Expression* corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file.

Description

Note This function does not work on the Solaris platform.

Expression = `affyrma(CELFiles, CDFFile)` reads the specified Affymetrix CEL files and the associated CDF library file (created from Affymetrix GeneChip arrays for expression or genotyping assays), processes the probe intensity values using RMA background adjustment, quantile normalization, and summarization procedures, then returns *Expression*, a DataMatrix object containing the log₂ based gene expression values in a matrix, the probe set IDs as row names, and the CEL file names as column names. Note that each row in *Expression* corresponds to a gene (probe set), and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)

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

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

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

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

Expression = affyrma(*CELFiles*, *CDFFile*, ...'CELPPath', *CELPPathValue*, ...) specifies a path and directory where the files specified by *CELFiles* are stored.

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

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

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

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

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

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

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

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

For example:

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

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

Examples

The following example assumes that you have the HG_U95Av2.CDF library file stored at `D:\Affymetrix\LibFiles\HGGenome`, and that

your current directory points to a location containing CEL files associated with this CDF library file. In this example, the `affyRMA` function reads all the CEL files in the current directory and a CDF file in a specified directory. It also performs RMA background adjustment, quantile normalization, and summarization procedures on the PM probe intensity values, and returns a `DataMatrix` object, containing the metadata and processed data.

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

References

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

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

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

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

See Also

`affyGCRMA`, `celIntensityRead`, `GCRMA`, `MAFDR`, `MAtest`, `quantileNorm`, `RMAbackAdj`, `RMAsummary`

affysnpannotread

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

Syntax
`AnnotStruct = affysnpannotread(File, PID)`
`AnnotStruct = affysnpannotread(File, PID, 'LookUpField', LookUpFieldValue)`

Arguments

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

Return Values

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

Description `AnnotStruct = affysnpannotread(File, PID)` reads *File*, an Affymetrix CSV annotation file for a Mapping 10K array set, Mapping 100K array set, or Mapping 500K array set, and returns *AnnotStruct*,

a MATLAB structure containing annotation information for one or more probe sets specified by *PID*, a string or cell array of strings specifying one or more probe set IDs. *AnnotStruct* contains a subset of the fields in *File*. The fields are described in the following table.

Structure Created from an Affymetrix CSV Annotation File

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

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

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

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

Examples

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

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

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

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

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

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

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

```
snpInfo =
```

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

```
AlleleB: {59024x1 cell}
Accession: {59024x1 cell}
FragmentLength: [59024x1 double]
```

See Also

Bioinformatics Toolbox functions: `affysnpintensitysplit`, `affyread`

affysnpintensitysplit

Purpose Split Affymetrix SNP probe intensity information for alleles A and B

Syntax
ProbeStructSplit = affysnpintensitysplit(*ProbeStruct*)
ProbeStructSplit = affysnpintensitysplit(*ProbeStruct*,
'Controls', *ControlsValue*)

Arguments

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

Return Values

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

Description

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

ProbeStructSplit contains the following fields.

Field	Description
CDFName	File name of the Affymetrix CDF library file.
CELNames	Cell array of names of the Affymetrix CEL files.
NumChips	Number of CEL files read into the input structure.

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

affysnpintensitysplit

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

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

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

Examples

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

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

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

```
ps =
```

```

    CDFName: 'Mapping50K_Hind240.CDF'
    CELNames: {18x1 cell}
    NumChips: 18
    NumProbeSets: 57299
    NumProbes: 1145780
    ProbeSetIDs: {57299x1 cell}
    ProbeIndices: [1145780x1 uint8]
    GroupNumbers: [1145780x1 uint8]
    PMIntensities: [1145780x18 single]

```

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

```
ps_split = affysnpintensitysplit(ps)
```

```
ps_split =
```

```

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

```

affysnpintensitiesplit

PMAIntensities: [572750x18 single]
PMBIntensities: [572750x18 single]

See Also

Bioinformatics Toolbox functions: `affysnpannotread`, `affyread`,
`celintensityread`

Purpose Create table of SNP probe quartet results for Affymetrix probe set

Syntax `SNPQStruct = affysnpquartets(CELStruct, CDFStruct, PS)`

Arguments

CELStruct Structure created by the `affyread` function from an Affymetrix CEL file, which contains information about the intensity values of the individual probes.

CDFStruct Structure created by the `affyread` function from an Affymetrix CDF library file associated with the CEL file. The CDF library file contains information about which probes belong to which probe set.

PS Probe set index or the probe set ID/name.

Return Values

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

Description

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

Field	Description
'ProbeSet'	Identifier for the probe set.
'AlleleA'	String specifying the base that is allele A for the probe set.

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

Examples

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

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

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

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

<http://www.affymetrix.com/support/technical/byproduct.affx?product=100k>

The following example assumes that the NA06985_Hind_B5_3005533.CEL file is stored on the MATLAB search path or in the current directory. It also assumes that the associated CDF library file, Mapping50K_Hind240.cdf, is stored at D:\Affymetrix\LibFiles\.

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

```
celStruct = affyread('NA06985_Hind_B5_3005533.CEL');
```

- 2 Read the contents of a CDF file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Mapping50K_Hind240.cdf');
```

- 3 Create a structure containing SNP probe quartet results for the SNP_A-1684395 probe set.

```
SNPQStruct = affysnpquartets(celStruct,cdfStruct,'SNP_A-1684395')
```

```
SNPQStruct =
```

```
ProbeSet: 'SNP_A-1684395'  
AlleleA: 'A'  
AlleleB: 'G'  
Quartet: [1x5 struct]
```

- 4 View the intensity values of the first probe pair in the probe set.

```
SNPQStruct.Quartet(1)
```

```
ans =
```

```
A_Sense_PM: 5013  
B_Sense_PM: 1290  
A_Sense_MM: 1485  
B_Sense_MM: 686  
A_Antisense_PM: 3746  
B_Antisense_PM: 1406  
A_Antisense_MM: 1527  
B_Antisense_MM: 958
```

See Also

Bioinformatics Toolbox functions: `affyread`, `probesetvalues`

agferead

Purpose Read Agilent Feature Extraction Software file

Syntax `AGFEData = agferead(File)`

Arguments

<i>File</i>	Microarray data file generated with the Agilent Feature Extraction Software.
-------------	--

Description `AGFEData = agferead(File)` reads files generated with the Feature Extraction Software from Agilent microarray scanners and creates a structure (*AGFEData*) containing the following fields:

- Header
- Stats
- Columns
- Rows
- Names
- IDs
- Data
- ColumnNames
- TextData
- TextColumnNames

The Feature Extraction Software takes an image from an Agilent microarray scanner and generates raw intensity data for each spot on the plate. For more information about this software, see:

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

Examples

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


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

2 Plot the median foreground.

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

See Also

Bioinformatics Toolbox functions: `affyread`, `celintensityread`, `galread`, `geoseriesread`, `geosoftread`, `gprread`, `ilmnbsread`, `imageneread`, `magetfield`, `sptread`

aminolookup

Purpose Find amino acid codes, integers, abbreviations, names, and codons

Syntax

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

Arguments

<i>SeqAA</i>	String of single-letter codes or three-letter abbreviations representing an amino acid sequence. For valid codes and abbreviations, see the table Amino Acid Lookup on page 2-91.
<i>CodeValue</i>	String specifying a single-letter code representing an amino acid. For valid single-letter codes, see the table Amino Acid Lookup on page 2-91.
<i>IntegerValue</i>	Single integer representing an amino acid. For valid integers, see the table Amino Acid Lookup on page 2-91.
<i>AbbreviationValue</i>	String specifying a three-letter abbreviation representing an amino acid. For valid three-letter abbreviations, see the table Amino Acid Lookup on page 2-91.
<i>NameValue</i>	String specifying an amino acid name. For valid amino acid names, see the table Amino Acid Lookup on page 2-91.

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

Amino Acid Lookup

Code	Integer	Abbreviation	Amino Acid Name	Codons
A	1	Ala	Alanine	GCU GCC GCA GCG
R	2	Arg	Arginine	CGU CGC CGA CGG AGA AGG
N	3	Asn	Asparagine	AAU AAC
D	4	Asp	Aspartic acid (Aspartate)	GAU GAC
C	5	Cys	Cysteine	UGU UGC
Q	6	Gln	Glutamine	CAA CAG
E	7	Glu	Glutamic acid (Glutamate)	GAA GAG
G	8	Gly	Glycine	GGU GGC GGA GGG
H	9	His	Histidine	CAU CAC
I	10	Ile	Isoleucine	AUU AUC AUA
L	11	Leu	Leucine	UUA UUG CUU CUC CUA CUG
K	12	Lys	Lysine	AAA AAG
M	13	Met	Methionine	AUG
F	14	Phe	Phenylalanine	UUU UUC
P	15	Pro	Proline	CCU CCC CCA CCG
S	16	Ser	Serine	UCU UCC UCA UCG AGU AGC

Amino Acid Lookup (Continued)

Code	Integer	Abbreviation	Amino Acid Name	Codons
T	17	Thr	Threonine	ACU ACC ACA ACG
W	18	Trp	Tryptophan	UGG
Y	19	Tyr	Tyrosine	UAU UAC
V	20	Val	Valine	GUU GUC GUA GUG
B	21	Asx	Asparagine or Aspartic acid (Aspartate)	AAU AAC GAU GAC
Z	22	Glx	Glutamine or Glutamic acid (Glutamate)	CAA CAG GAA GAG
X	23	Xaa	Any amino acid	All codons
*	24	END	Termination codon (translation stop)	UAA UAG UGA
-	25	GAP	Gap of unknown length	NA

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

If you enter one of the ambiguous single-letter codes B, Z, or X, this function displays the corresponding abbreviation for the ambiguous amino acid character.

```
aminolookup('abc')
```

```
ans =
```

```
AlaAsxCys
```

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

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

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

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

Examples

- Convert an amino acid sequence in single-letter codes to the corresponding three-letter abbreviations.

```
aminolookup('MWKQAEDIRDIYDF')
```

```
ans =
```

```
MetTrpLysGlnAlaGluAspIleArgAspIleTyrAspPhe
```

- Convert an amino acid sequence in three-letter abbreviations to the corresponding single-letter codes.

```
aminolookup('MetTrpLysGlnAlaGluAspIleArgAspIleTyrAspPhe')
```

```
ans =
```

```
MWKQAEDIRDIYDF
```

aminolookup

- Display the three-letter abbreviation and name for the amino acid corresponding to the single-letter code R.

```
aminolookup('Code', 'R')
```

```
ans =
```

```
Arg Arginine
```

- Display the single-letter code, three-letter abbreviation, and name for the amino acid corresponding to the integer 1.

```
aminolookup('Integer', 1)
```

```
ans =
```

```
A Ala Alanine
```

- Display the single-letter code and name for the amino acid corresponding to the three-letter abbreviation asn.

```
aminolookup('Abbreviation', 'asn')
```

```
ans =
```

```
N Asparagine
```

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

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

```
ans =
```

```
P Pro
```

See Also

Bioinformatics Toolbox functions: `aa2int`, `aa2nt`, `aacount`, `geneticcode`, `int2aa`, `nt2aa`, `revgeneticcode`

Purpose Calculate atomic composition of protein

Syntax `NumberAtoms = atomiccomp(SeqAA)`

Arguments

`SeqAA` Amino acid sequence. Enter a character string or vector of integers from the table Mapping Amino Acid Letter Codes to Integers on page 2-2. You can also enter a structure with the field `Sequence`.

Description

`NumberAtoms = atomiccomp(SeqAA)` counts the type and number of atoms in an amino acid sequence (`SeqAA`) and returns the counts in a 1-by-1 structure (`NumberAtoms`) with fields C, H, N, O, and S.

Examples

- 1 Get an amino acid sequence from the NCBI GenPept database.

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

- 2 Count the atoms in a sequence.

```
rhodopsinAC = atomiccomp(rhodopsin)
```

```
rhodopsinAC =
```

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

- 3 Retrieve the number of carbon atoms in the sequence.

```
rhodopsinAC.C
```

```
ans =
```

```
1814
```

atomiccomp

See Also

Bioinformatics Toolbox functions: `aaccount`, `molweight`, `proteinplot`

Purpose	Count nucleotides in sequence	
Syntax	<pre> NTStruct = basecount(SeqNT) NTStruct = basecount(SeqNT, ...'Chart', ChartValue, ...) NTStruct = basecount(SeqNT, ...'Others', OthersValue, ...) NTStruct = basecount(SeqNT, ...'Structure', StructureValue, ...)</pre>	
Arguments	<i>SeqNT</i>	<p>One of the following:</p> <ul style="list-style-type: none"> • String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers on page 2-794 • Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes on page 2-562 • MATLAB structure containing a <code>Sequence</code> field that contains a nucleotide sequence, such as returned by <code>fastaread</code>, <code>emblread</code>, <code>getembl</code>, <code>genbankread</code>, or <code>getgenbank</code>.
	<i>ChartValue</i>	String specifying a chart type. Choices are 'pie' or 'bar'.
	<i>OthersValue</i>	String specifying how to count ambiguous characters, including gaps indicated by a hyphen (-). Choices are 'full' (lists the ambiguous characters in separate fields) or 'bundle' (lists the ambiguous characters together in the field <code>Others</code>). Default is 'bundle'.
	<i>StructureValue</i>	Suppresses the unknown characters warning when set to 'full'.

basecount

Return Values

NTStruct 1-by-1 MATLAB structure containing the fields A, C, G, and T.

Description

NTStruct = `basecount(SeqNT)` counts the number of each type of base in a `SeqNT`, a nucleotide sequence, and returns the counts in *NTStruct*, a 1-by-1 MATLAB structure containing the fields A, C, G, and T.

- For sequences with the character U, the number of U characters is added to the T field.
- If a sequence contains ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N), or gaps indicated by a hyphen (-), then these characters are counted in the field `Others`, and the following warning message appears:

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

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

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

- If the property `'Others'` is set to `'full'`, ambiguous characters are listed separately in the fields R, Y, K, M, S, W, B, D, H, V, N, and Gap.

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

NTStruct = `basecount(SeqNT, ...'Chart', ChartValue, ...)` creates a chart showing the relative proportions of the nucleotides. *ChartValue* can be `'pie'` or `'bar'`.

NTStruct = basecount(*SeqNT*, ...'Others', *OthersValue*, ...) specifies how to count ambiguous characters (R, Y, K, M, S, W, B, D, H, V, and N), or gaps indicated by a hyphen (-). Choices are 'full' (lists the ambiguous characters in separate fields) or 'bundle' (lists the ambiguous characters together in the field *Others*). Default is 'bundle'.

NTStruct = basecount(*SeqNT*, ...'Structure', *StructureValue*, ...) suppresses the unknown characters warning when set to 'full'.

- basecount(*SeqNT*) — Displays fields for four nucleotides, and, if there are ambiguous and unknown characters, add an *Others* field with the ambiguous and unknown character counts.
- basecount(*SeqNT*, 'Others', 'full') — Displays fields for 4 nucleotides, 11 ambiguous nucleotides, gaps, and, if there are unknown characters, adds an *Others* field with the unknown counts.
- basecount(*SeqNT*, 'Structure', 'full') — Displays fields for four nucleotides and an *Others* field. If there are ambiguous or unknown characters, adds the counts to the *Others* field; otherwise displays 0 in the *Others* field.
- basecount(*SeqNT*, 'Others', 'full', 'Structure', 'full') — Displays fields for 4 nucleotides, 11 ambiguous nucleotides, gaps, and an *Others* field. If there are unknown characters, adds the counts to the *Others* field; otherwise displays 0 in the *Others* field.

Examples

- 1 Count the bases in a DNA sequence and return the results in a structure.

```
Bases = basecount('TAGCTGGCCAAGCGAGCTTG')
```

```
Bases =
```

```
A: 4
C: 5
G: 7
T: 4
```

basecount

- 2** Get the count for adenosine (A) bases.

```
Bases.A  
  
ans =  
  
4
```

- 3** Count the bases in a DNA sequence containing ambiguous characters, listing the ambiguous characters in separate fields.

```
basecount('ABCDGGCCAAGCGAGCTTG','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  
Gap: 0
```

See Also

Bioinformatics Toolbox functions: `aaccount`, `baselookup`, `codoncount`, `cpgisland`, `dimercount`, `nmercount`, `ntdensity`, `seqtool`

Purpose

Find nucleotide codes, integers, names, and complements

Syntax

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

Arguments

SeqNT

Nucleotide sequence(s) represented by one of the following:

- String of single-letter codes from the table Nucleotide Lookup on page 2-102
- Cell array of sequences
- Two-dimensional character array of sequences

Note If the input is multiple sequences, the complement for each sequence is determined independently.

CodeValue

Nucleotide letter code represented by one of the following:

- String specifying a single-letter code representing a nucleotide. For valid single-letter codes, see the table Nucleotide Lookup on page 2-102.
- Cell array of letter codes.
- Two-dimensional character array of letter codes.

baselookup

IntegerValue Single integer representing a nucleotide. For valid integers, see the table Nucleotide Lookup on page 2-102.

NameValue Nucleotide name represented by one of the following:

- String specifying a nucleotide name. For valid nucleotide names, see the table Nucleotide Lookup on page 2-102.
- Cell array of names.
- Two-dimensional character array of names.

Description

baselookup displays a table of nucleotide codes, integers, names, and complements.

Nucleotide Lookup

Code	Integer	Nucleotide Name	Meaning	Complement
A	1	Adenine	A	T
C	2	Cytosine	C	G
G	3	Guanine	G	C
T	4	Thymine	T	A
U	4	Uracil	U	A
R	5	Purine	A or G	Y
Y	6	Pyrimidine	C or T	R
K	7	Keto	G or T	M
M	8	Amino	A or C	K

Nucleotide Lookup (Continued)

Code	Integer	Nucleotide Name	Meaning	Complement
S	9	Strong interaction (3 H bonds)	C or G	S
W	10	Weak interaction (2 H bonds)	A or T	W
B	11	Not A	C or G or T	V
D	12	Not C	A or G or T	H
H	13	Not G	A or C or T	D
V	14	Not T or U	A or C or G	B
N, X	15	Any nucleotide	A or C or G or T or U	N
-	16	Gap of indeterminate length	Gap	-

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

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

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

`baselookup('Name', NameValue)` displays the corresponding letter code, meaning, and nucleotide name or descriptive name.

baselookup

Examples

- Convert a nucleotide sequence to its complementary sequence.

```
baselookup('Complement', 'TAGCTGRCCAAGGCCAAGCGAGCTTN')  
  
ans =  
  
ATCGACYGGTTCGGTTCGCTCGAAN
```

- Display the meaning and nucleotide name or descriptive name for the nucleotide codes G and Y.

```
>> baselookup('Code', 'G')  
  
ans =  
  
G Guanine
```

```
>> baselookup('Code', 'Y')  
  
ans =  
  
T|C pYrimidine
```

- Display the nucleotide letter code, meaning, and nucleotide name or descriptive name for the integers 1 and 7.

```
baselookup('Integer', 1)  
  
ans =  
  
A A - Adenine
```

```
>> baselookup('Integer', 7)  
  
ans =
```


K G|T - Keto

- Display the corresponding nucleotide letter code, meaning, and name for cytosine and purine.

```
baselookup('Name','cytosine')
```

```
ans =
```

C C - Cytosine

```
baselookup('Name','purine')
```

```
ans =
```

R G|A - puRine

See Also

Bioinformatics Toolbox functions: `aa2nt`, `basecount`, `codoncount`, `dimercount`, `geneticcode`, `int2nt`, `nt2aa`, `nt2int`, `revgeneticcode`, `seqtool`

biograph

Purpose

Create biograph object

Syntax

```
BGobj = biograph(CMatrix)
BGobj = biograph(CMatrix, NodeIDs)
BGobj = biograph(CMatrix, NodeIDs, ...'ID', IDValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'Label', LabelValue,
    ...)
BGobj = biograph(CMatrix, NodeIDs, ...'Description',
    DescriptionValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'LayoutType',
    LayoutTypeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeType',
    EdgeTypeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'Scale', ScaleValue,
    ...)
BGobj = biograph(CMatrix, NodeIDs, ...'LayoutScale',
    LayoutScaleValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeTextColor',
    EdgeTextColorValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeFontSize',
    EdgeFontSizeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ShowArrows',
    ShowArrowsValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ArrowSize',
    ArrowSizeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ShowWeights',
    ShowWeightsValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'ShowTextInNodes',
    ShowTextInNodesValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'NodeAutoSize',
    NodeAutoSizeValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'NodeCallback',
    NodeCallbackValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'EdgeCallback',
    EdgeCallbackValue, ...)
BGobj = biograph(CMatrix, NodeIDs, ...'CustomNodeDrawFcn',
    CustomNodeDrawFcnValue, ...)
```

Arguments

CMatrix Full or sparse square matrix that acts as a connection matrix. That is, a value of 1 indicates a connection between nodes while a 0 indicates no connection. The number of rows/columns is equal to the number of nodes.

NodeIDs Node identification strings. Enter any of the following:

- Cell array of strings with the number of strings equal to the number of rows or columns in the connection matrix *CMatrix*. Each string must be unique.
- Character array with the number of rows equal to the number of nodes. Each row in the array must be unique.
- String with the number of characters equal to the number of nodes. Each character must be unique.

Default values are the row or column numbers.

Note You must specify *NodeIDs* if you want to specify property name/value pairs. Set *NodeIDs* to [] to use the default values of the row/column numbers.

IDValue String to identify the biograph object. Default is ''.

LabelValue String to label the biograph object. Default is ''.

biograph

<i>DescriptionValue</i>	String that describes the biograph object. Default is ''.
<i>LayoutTypeValue</i>	String that specifies the algorithm for the layout engine. Choices are: <ul style="list-style-type: none">• 'hierarchical' (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges.• 'radial' — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges.• 'equilibrium' — Calculates layout by minimizing the energy in a dynamic spring system.
<i>EdgeTypeValue</i>	String that specifies how edges display. Choices are: <ul style="list-style-type: none">• 'straight'• 'curved' (default)• 'segmented'

Note Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with *LayoutType* equal to 'equilibrium' or 'radial' cannot produce curved or segmented edges.

<i>ScaleValue</i>	Positive number that post-scales the node coordinates. Default is 1.
<i>LayoutScaleValue</i>	Positive number that scales the size of the nodes before calling the layout engine. Default is 1.
<i>EdgeTextColorValue</i>	Three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.
<i>EdgeFontSizeValue</i>	Positive number that sets the size of the edge font in points. Default is 8.
<i>ShowArrowsValue</i>	Controls the display of arrows for the edges. Choices are 'on' (default) or 'off'.
<i>ArrowSizeValue</i>	Positive number that sets the size of the arrows in points. Default is 8.
<i>ShowWeightsValue</i>	Controls the display of text indicating the weight of the edges. Choices are 'on' (default) or 'off'.
<i>ShowTextInNodesValue</i>	String that specifies the node property used to label nodes when you display a biograph object using the view method. Choices are: <ul style="list-style-type: none">• 'Label' — Uses the Label property of the node object (default).• 'ID' — Uses the ID property of the node object.• 'None'
<i>NodeAutoSizeValue</i>	Controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.

<i>NodeCallbackValue</i>	User callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the <code>view</code> function to display the biograph in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is <code>@(node) inspect(node)</code> , which displays the Property Inspector dialog box.
<i>EdgeCallbackValue</i>	User callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the <code>view</code> function to display the biograph in the Biograph Viewer, you can double-click an edge to activate the first callback, or right-click and select a callback to activate. Default is <code>@(edge) inspect(edge)</code> , which displays the Property Inspector dialog box.
<i>CustomNodeDrawFcnValue</i>	Function handle to a customized function to draw nodes. Default is <code>[]</code> .

Description

`BGobj = biograph(CMatrix)` creates a biograph object, `BGobj`, using a connection matrix, `CMatrix`. All nondiagonal and positive entries in the connection matrix, `CMatrix`, indicate connected nodes, rows represent the source nodes, and columns represent the sink nodes.

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

- Cell array of strings with the number of strings equal to the number of rows or columns in the connection matrix `CMatrix`. Each string must be unique.

- Character array with the number of rows equal to the number of nodes. Each row in the array must be unique.
- String with the number of characters equal to the number of nodes. Each character must be unique.

Default values are the row or column numbers.

Note If you want to specify property name/value pairs, you must specify *NodeIDs*. Set *NodeIDs* to [] to use the default values of the row/column numbers.

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

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'ID', *IDValue*, ...) specifies an ID for the biograph object. Default is ''.

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

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'Description', *DescriptionValue*, ...) specifies a description of the biograph object. Default is ''.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'LayoutType', *LayoutTypeValue*, ...) specifies the algorithm for the layout engine.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'EdgeType', *EdgeTypeValue*, ...) specifies how edges display.

BGobj = biograph(*CMatrix*, *NodeIDs*, ...'Scale', *ScaleValue*, ...) post-scales the node coordinates. Default is 1.

biograph

BGobj = `biograph(CMatrix, NodeIDs, ...'LayoutScale', LayoutScaleValue, ...)` scales the size of the nodes before calling the layout engine. Default is 1.

BGobj = `biograph(CMatrix, NodeIDs, ...'EdgeTextColor', EdgeTextColorValue, ...)` specifies a three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.

BGobj = `biograph(CMatrix, NodeIDs, ...'EdgeFontSize', EdgeFontSizeValue, ...)` sets the size of the edge font in points. Default is 8.

BGobj = `biograph(CMatrix, NodeIDs, ...'ShowArrows', ShowArrowsValue, ...)` controls the display of arrows for the edges. Choices are 'on' (default) or 'off'.

BGobj = `biograph(CMatrix, NodeIDs, ...'ArrowSize', ArrowSizeValue, ...)` sets the size of the arrows in points. Default is 8.

BGobj = `biograph(CMatrix, NodeIDs, ...'ShowWeights', ShowWeightsValue, ...)` controls the display of text indicating the weight of the edges. Choices are 'on' (default) or 'off'.

BGobj = `biograph(CMatrix, NodeIDs, ...'ShowTextInNodes', ShowTextInNodesValue, ...)` specifies the node property used to label nodes when you display a biograph object using the view method.

BGobj = `biograph(CMatrix, NodeIDs, ...'NodeAutoSize', NodeAutoSizeValue, ...)` controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.

BGobj = `biograph(CMatrix, NodeIDs, ...'NodeCallback', NodeCallbackValue, ...)` specifies user callback for all nodes.

BGobj = `biograph(CMatrix, NodeIDs, ...'EdgeCallback', EdgeCallbackValue, ...)` specifies user callback for all edges.

BGobj = `biograph(CMatrix, NodeIDs, ...'CustomNodeDrawFcn', CustomNodeDrawFcnValue, ...)` specifies function handle to customized function to draw nodes. Default is [].

Examples

- 1 Create a biograph object with default node IDs, and then use the `get` function to display the node IDs.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg1 = biograph(cm)
Biograph object with 5 nodes and 9 edges.
get(bg1.nodes, 'ID')

ans =

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

- 2 Create a biograph object, assign the node IDs, and then use the `get` function to display the node IDs.

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

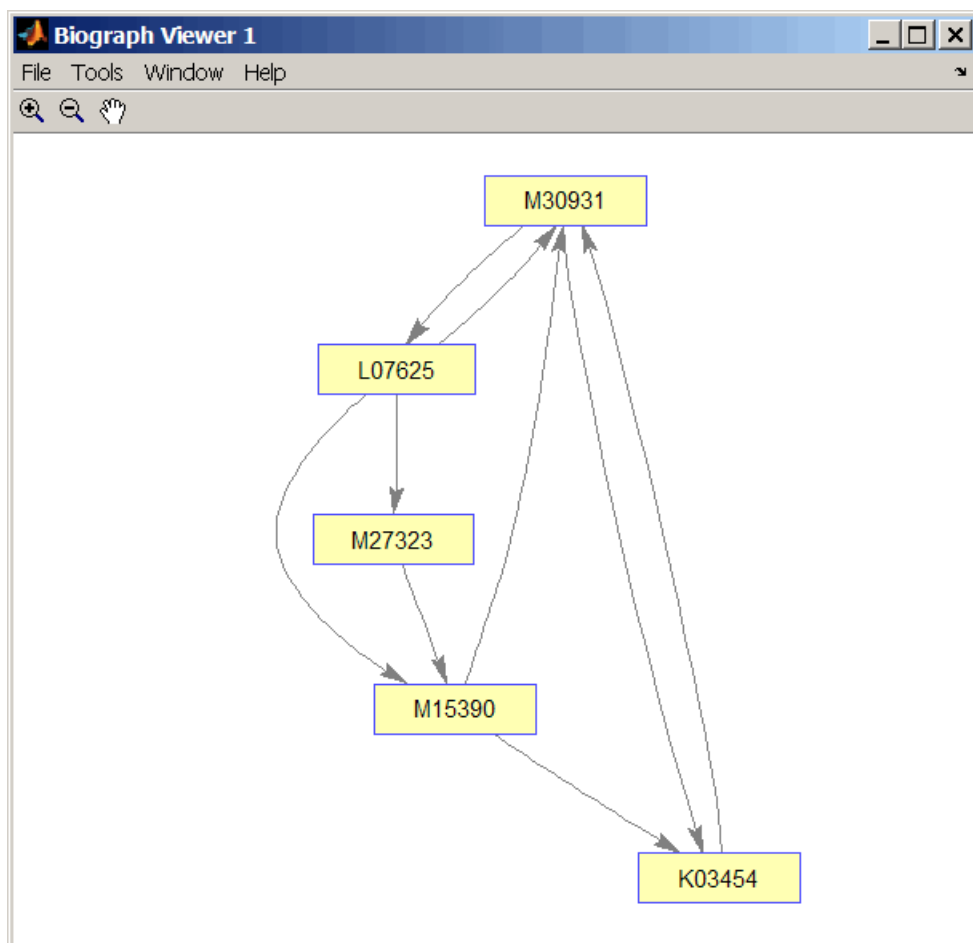
ans =

    'M30931'
    'L07625'
    'K03454'
    'M27323'
    'M15390'
```

- 3 Use the `view` method to display the biograph object.

```
view(bg2)
```

biograph



See Also

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object:
`allshortestpaths`, `conncomp`, `dolayout`, `get`, `getancestors`,
`getdescendants`, `getedgesbynodeid`, `getmatrix`, `getnodesbyid`,
`getrelatives`, `isdag`, `isomorphism`, `isspantree`, `maxflow`,
`minspantree`, `set`, `shortestpath`, `topoorder`, `traverse`, `view`

Purpose Create local BLAST database

Syntax

```
blastformat('Inputdb', InputdbValue)  
blastformat(..., 'FormatPath', FormatPathValue, ...)  
blastformat(..., 'Title', TitleValue, ...)  
blastformat(..., 'Log', LogValue, ...)  
blastformat(..., 'Protein', ProteinValue, ...)  
blastformat(..., 'FormatArgs', FormatArgsValue, ...)
```

Arguments

<i>InputdbValue</i>	String specifying a file name or path and file name of a FASTA file containing a set of sequences to be formatted as a blastable database. If you specify only a file name, that file must be on the MATLAB search path or in the current directory. (This corresponds to the <code>formatdb</code> option <code>-i</code> .)
<i>FormatPathValue</i>	String specifying the full path to the <code>formatdb</code> executable file, including the name and extension of the executable file. Default is the system path.
<i>TitleValue</i>	String specifying the title for the local database. Default is the input FASTA file name. (This corresponds to the <code>formatdb</code> option <code>-t</code> .)
<i>LogValue</i>	String specifying the file name or path and file name for the log file associated with the local database. Default is <code>formatdb.log</code> . (This corresponds to the <code>formatdb</code> option <code>-l</code> .)

- ProteinValue* Specifies whether the sequences formatted as a local BLAST database are protein or not. Choices are true (default) or false. (This corresponds to the `formatdb` option `-p`.)
- FormatArgsValue* NCBI `formatdb` command string, that is, a string containing one or more instances of `-x` and the option associated with it, used to specify input arguments. For an example, see Using `blastformat` with `formatdb` Syntax and Input Arguments on page 2-119.

Description

Note To use the `blastformat` function, you must have a local copy of the NCBI `formatdb` executable file available from your system. You can download the `formatdb` executable file by accessing

<http://blast.ncbi.nlm.nih.gov/download.shtml>

then clicking the **download** link under the **blast** column for your platform. Run the downloaded executable and configure it for your system. For more information, see the readme file on the NCBI ftp site at:

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

For convenience, consider placing the NCBI `formatdb` executable file on your system path.

`blastformat('Inputdb', InputdbValue)` calls a local version of the NCBI `formatdb` executable file with *InputdbValue*, a file name or path and file name of a FASTA file containing a set of sequences. If you specify only a file name, that file must be on the MATLAB search path or in the current directory. (This corresponds to the `formatdb` option `-i`.)

It then formats the sequences as a local, blastable database, by creating multiple files, each with the same name as the *InputdbValue* FASTA

file, but with different extensions. The database files are placed in the same location as the FASTA file.

Note If you rename the database files, make sure they all have the same name.

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

`blastformat(..., 'FormatPath', FormatPathValue, ...)` specifies the full path to the `formatdb` executable file, including the name and extension of the executable file. Default is the system path.

`blastformat(..., 'Title', TitleValue, ...)` specifies the title for the local database. Default is the input FASTA file name. (This corresponds to the `formatdb` option `-t`.)

Note The 'Title' property does not change the file name of the database files. This title is used internally only, and appears in the report structure returned by the `blastlocal` function.

`blastformat(..., 'Log', LogValue, ...)` specifies the file name or path and file name for the log file associated with the local database. Default is `formatdb.log`. The log file captures the progress of the database creation and formatting. (This corresponds to the `formatdb` option `-l`.)

`blastformat(..., 'Protein', ProteinValue, ...)` specifies whether the sequences formatted as a local BLAST database are protein or not. Choices are `true` (default) or `false`. (This corresponds to the `formatdb` option `-p`.)

`blastformat(..., 'FormatArgs', FormatArgsValue, ...)` specifies options using the input arguments for the NCBI `formatdb` function. *FormatArgsValue* is a string containing one or more instances of `-x` and the option associated with it. For example, to specify that the input is a database in ASN.1 format, instead of a FASTA file, you would use the following syntax:

```
blastformat('Inputdb', 'ecoli.asn', 'FormatArgs', '-a T')
```

Tip Use the 'FormatArgs' property to specify `formatdb` options for which there are no corresponding property name/property value pairs.

Note For a complete list of valid input arguments for the NCBI `formatdb` function, make sure that the `formatdb` executable file is located on your system path or current directory, then type the following at your system's command prompt.

```
formatdb -
```

Using `formatdb` Syntax

You can also use the syntax and input arguments accepted by the NCBI `formatdb` function, instead of the property name/property value pairs listed previously. To do so, supply a single string containing multiple options using the `-x option` syntax. For example, you can specify the `ecoli.nt` FASTA file, a title of `myecoli`, and that the sequences are not protein by using

```
blastformat('-i ecoli.nt -t myecoli -p F')
```

Note For a complete list of valid input arguments for the NCBI `formatdb` function, make sure that the `formatdb` executable file is located on your system path or current directory, then type the following at your system's command prompt.

```
formatdb -
```

Examples

Using `blastformat` with Property Name/Value Pairs

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

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

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

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

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

Using `blastformat` with `formatdb` Syntax and Input Arguments

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

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

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

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

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

blastformat

```
'formatargs', '-t myecoli_aa -l ecoli_aa.log');
```

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. (1990). Basic local alignment search tool. *J. Mol. Biol.* *215*, 403–410.

[2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* *25*, 3389–3402.

For more information on the NCBI formatdb function, see:

<http://blast.ncbi.nlm.nih.gov/docs/formatdb.html>

See Also

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

Purpose Perform search on local BLAST database to create BLAST report

Syntax

```
blastlocal('InputQuery', InputQueryValue)
Data = blastlocal('InputQuery', InputQueryValue)
... blastlocal(..., 'Program', ProgramValue, ...)
... blastlocal(..., 'Database', DatabaseValue, ...)
... blastlocal(..., 'BlastPath', BlastPathValue, ...)
... blastlocal(..., 'Expect', ExpectValue, ...)
... blastlocal(..., 'Format', FormatValue, ...)
... blastlocal(..., 'ToFile', ToFileValue, ...)
... blastlocal(..., 'Filter', FilterValue, ...)
... blastlocal(..., 'GapOpen', GapOpenValue, ...)
... blastlocal(..., 'GapExtend', GapExtendValue, ...)
... blastlocal(..., 'BLASTArgs', BLASTArgsValue, ...)
```

blastlocal

Arguments

<i>InputQueryValue</i>	String specifying the file name or path and file name of a FASTA file containing query nucleotide or amino acid sequence(s). (This corresponds to the <code>blastall</code> option <code>-i</code> .)
<i>ProgramValue</i>	String specifying a BLAST program. Choices are: <ul style="list-style-type: none">• 'blastp' (default) — Search protein query versus protein database.• 'blastn' — Search nucleotide query versus nucleotide database.• 'blastx' — Search translated nucleotide query versus protein database.• 'tblastn' — Search protein query versus translated nucleotide database.• 'tblastx' — Search translated nucleotide query versus translated nucleotide database. (The <i>ProgramValue</i> argument corresponds to the <code>blastall</code> option <code>-p</code> .)
<i>DatabaseValue</i>	String specifying a file name or path and file name of a local BLAST database (formatted using the NCBI <code>formatdb</code> function) to search. Default is a local version of the <code>nr</code> database in the MATLAB current directory. (This corresponds to the <code>blastall</code> option <code>-d</code> .)
<i>BlastPathValue</i>	String specifying the full path to the <code>blastall</code> executable file, including the name and extension of the executable file. Default is the system path.

<i>ExpectValue</i>	Value specifying the statistical significance threshold for matches against database sequences. Choices are any real number. Default is 10. (This corresponds to the <code>blastall</code> option <code>-e</code> .)
<i>FormatValue</i>	Integer specifying the alignment format of the BLAST search results. Choices are: <ul style="list-style-type: none">• 0 (default) — Pairwise• 1 — Query-anchored, showing identities• 2 — Query-anchored, no identities• 3 — Flat query-anchored, showing identities• 4 — Flat query-anchored, no identities• 5 — Query-anchored, no identities and blunt ends• 6 — Flat query-anchored, no identities and blunt ends• 8 — Tabular• 9 — Tabular with comment lines (This corresponds to the <code>blastall</code> option <code>-m</code> .)
<i>ToFileValue</i>	String specifying a file name or path and file name in which to save the contents of the BLAST report. (This corresponds to the <code>blastall</code> option <code>-o</code> .)
<i>FilterValue</i>	Controls the application of a filter (DUST filter for the <code>blastn</code> program or SEG filter for other programs) to the query sequence(s). Choices are <code>true</code> (default) or <code>false</code> . (This corresponds to the <code>blastall</code> option <code>-F</code> .)

blastlocal

<i>GapOpenValue</i>	Integer that specifies the penalty for opening a gap in the alignment of sequences. Default is -1. (This corresponds to the <code>blastall</code> option -G.)
<i>GapExtendValue</i>	Integer that specifies the penalty for extending a gap in the alignment of sequences. Default is -1. (This corresponds to the <code>blastall</code> option -E.)
<i>BLASTArgsValue</i>	NCBI <code>blastall</code> command string, that is a string containing one or more instances of <code>-x</code> and the option associated with it, used to specify input arguments. For an example, see step 7 in “Examples” on page 2-131.

Return Values

<i>Data</i>	MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report.
-------------	---

Description

This function assumes that

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

Note To use the `blastlocal` function, you must have a local copy of the NCBI `blastall` executable file (version 2.2.17) available from your system. You can download the `blastall` executable file by accessing

<http://blast.ncbi.nlm.nih.gov/download.shtml>

then clicking the **download** link under the **blast** column for your platform. Run the downloaded executable and configure it for your system. For more information, see the readme file on the NCBI ftp site at:

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

For convenience, consider placing the NCBI `blastall` executable file on your system path.

`blastlocal('InputQuery', InputQueryValue)` submits query sequence(s) specified by *InputQueryValue*, a FASTA file containing nucleotide or amino acid sequence(s), for a BLAST search of a local BLAST database, by calling a local version of the NCBI `blastall` executable file. The BLAST search results are displayed in the MATLAB Command Window. (This corresponds to the `blastall` option `-i`.)

`Data = blastlocal('InputQuery', InputQueryValue)` returns the BLAST search results in *Data*, a MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report.

Data contains a subset of the following fields, based on the specified alignment format.

Field	Description
Algorithm	NCBI algorithm used to do a BLAST search.

Field	Description
Query	Identifier of the query sequence submitted to a BLAST search.
Length	Length of the query sequence.
Database	All databases searched.
Hits.Name	Name of a database sequence (subject sequence) that matched the query sequence.
Hits.Score	Alignment score between the query sequence and the subject sequence.
Hits.Expect	Expectation value for the alignment between the query sequence and the subject sequence.
Hits.Length	Length of a subject sequence.
Hits.HSPs.Score	Pairwise alignment score for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Expect	Expectation value for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Identities	Identities (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.

Field	Description
Hits.HSPs.Positives	<p>Identical or similar residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject amino acid sequence.</p> <hr/> <p>Note This field applies only to translated nucleotide or amino acid query sequences and/or databases.</p> <hr/>
Hits.HSPs.Gaps	<p>Nonaligned residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</p>
Hits.HSPs.Mismatches	<p>Residues that are not similar to each other (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</p>
Hits.HSPs.Frame	<p>Reading frame of the translated nucleotide sequence for a high-scoring sequence pair between the query sequence and a subject sequence.</p> <hr/> <p>Note This field applies only when performing translated searches, that is, when using tblastx, tblastn, and blastx.</p> <hr/>

Field	Description
Hits.HSPs.Strand	Sense (Plus = 5' to 3' and Minus = 3' to 5') of the DNA strands for a high-scoring sequence pair between the query sequence and a subject sequence. <hr/> Note This field applies only when using a nucleotide query sequence and database. <hr/>
Hits.HSPs.Alignment	Three-row matrix showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.QueryIndices	Indices of the query sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.SubjectIndices	Indices of the subject sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.AlignmentLength	Length of the pairwise alignment for a high-scoring sequence pair between the query sequence and a subject sequence.
Alignment	Entire alignment for the query sequence and the subject sequence(s).
Statistics	Summary of statistical details about the performed search, such as lambda values, gap penalties, number of sequences searched, and number of hits.

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

... `blastlocal(..., 'Program', ProgramValue, ...)` specifies the BLAST program. Choices are 'blastp' (default), 'blastn', 'blastx', 'tblastn', and 'tblastx'. (This corresponds to the `blastall` option `-p`.) For help in selecting an appropriate BLAST program, visit:

<http://blast.ncbi.nlm.nih.gov/producttable.shtml>

... `blastlocal(..., 'Database', DatabaseValue, ...)` specifies the local BLAST database (formatted using the NCBI `formatdb` function) to search. Default is a local version of the `nr` database in the MATLAB current directory. (This corresponds to the `blastall` option `-d`.)

... `blastlocal(..., 'BlastPath', BlastPathValue, ...)` specifies the full path to the `blastall` executable file, including the name and extension of the executable file. Default is the system path.

... `blastlocal(..., 'Expect', ExpectValue, ...)` specifies a statistical significance threshold for matches against database sequences. Choices are any real number. Default is 10. (This corresponds to the `blastall` option `-e`.) You can learn more about the statistics of local sequence comparison at:

<http://blast.ncbi.nlm.nih.gov/tutorial/Altschul-1.html#head2>

... `blastlocal(..., 'Format', FormatValue, ...)` specifies the alignment format of the BLAST search results. Choices are:

- 0 (default) — Pairwise
- 1 — Query-anchored, showing identities

- 2 — Query-anchored, no identities
- 3 — Flat query-anchored, showing identities
- 4 — Flat query-anchored, no identities
- 5 — Query-anchored, no identities and blunt ends
- 6 — Flat query-anchored, no identities and blunt ends
- 7 — Not used
- 8 — Tabular
- 9 — Tabular with comment lines

(This corresponds to the `blastall` option `-m`.)

... `blastlocal(..., 'ToFile', ToFileValue, ...)` saves the contents of the BLAST report to the specified file. (This corresponds to the `blastall` option `-o`.)

... `blastlocal(..., 'Filter', FilterValue, ...)` specifies whether a filter (DUST filter for the `blastn` program or SEG filter for other programs) is applied to the query sequence(s). Choices are `true` (default) or `false`. (This corresponds to the `blastall` option `-F`.)

... `blastlocal(..., 'GapOpen', GapOpenValue, ...)` specifies the penalty for opening a gap in the alignment of sequences. Default is `-1`. (This corresponds to the `blastall` option `-G`.)

... `blastlocal(..., 'GapExtend', GapExtendValue, ...)` specifies the penalty for extending a gap in the alignment of sequences. Default is `-1`. (This corresponds to the `blastall` option `-E`.)

... `blastlocal(..., 'BLASTArgs', BLASTArgsValue, ...)` specifies options using the input arguments for the NCBI `blastall` function. *BLASTArgsValue* is a string containing one or more instances or `-x` and the option associated with it. For example, to specify the BLOSUM 45 matrix, you would use the following syntax:

```
blastlocal('InputQuery', ecoliquery.txt, 'BLASTArgs', '-M BLOSUM45')
```

Tip Use the 'BlastArgs' property to specify `blastall` options for which there are no corresponding property name/property value pairs.

Note For a complete list of valid input arguments for the NCBI `blastall` function, make sure that the `blastall` executable file is located on your system path or current directory, then type the following at your system's command prompt.

```
blastall -
```

Using blastall Syntax

You can also use the syntax and input arguments accepted by the NCBI `blastall` function, instead of the property name/property value pairs listed previously. To do so, supply a single string containing multiple options using the `-x option` syntax. For example, you can specify the `ecoliquery.txt` FASTA file as your query sequences, the `blastp` program, and the `ecoli` local database, by using

```
blastlocal('-i ecoliquery.txt -p blastp -d ecoli')
```

Note For a complete list of valid input arguments for the NCBI `blastall` function, make sure that the `blastall` executable file is located on your system path or current directory, then type the following at your system's command prompt.

```
blastall -
```

Examples

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

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

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

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

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

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

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

- 4 Use the `fastawrite` function to create a FASTA file named `query_nt.fa` from this sequence information, using only the accession number as the header.

```
S.Header = S.Accession;  
fastawrite('query_nt.fa', S);
```

- 5 Use MATLAB syntax to submit the query sequence in the `query_nt.fa` FASTA file for a BLAST search of the local amino acid database `ecoli.aa`. Specify the BLAST program `blastx`. Return the BLAST search results in `results`, a MATLAB structure.

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

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

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

- 7 Submit the query sequence in the `query_nt.fa` FASTA file for a BLAST search of the local nucleotide database `ecoli.nt`. Specify the BLAST program `blastn` and a tabular alignment format. Save the contents of the BLAST report to a file named `myecoli_nt.txt`.

```
blastlocal('inputquery', 'query_nt.fa',...  
          'database', 'ecoli.nt', 'tofile', 'myecoli_nt.txt',...  
          'blastargs', '-p blastn -m 8')
```

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. (1990). Basic local alignment search tool. *J. Mol. Biol.* *215*, 403–410.

[2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* *25*, 3389–3402.

For more information on the NCBI `blastall` function, see:

<http://blast.ncbi.nlm.nih.gov/docs/blastall.html>

See Also

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

blastncbi

Purpose

Create remote NCBI BLAST report request ID or link to NCBI BLAST report

Syntax

```
blastncbi(Seq, Program)
RID = blastncbi(Seq, Program)
[RID, RTOE] = blastncbi(Seq, Program)
... blastncbi(Seq, Program, ...'Database',
DatabaseValue, ...)
... blastncbi(Seq, Program, ...'Descriptions',
DescriptionsValue, ...)
... blastncbi(Seq, Program, ...'Alignments',
AlignmentsValue,
...)
... blastncbi(Seq, Program, ...'Filter', FilterValue, ...)
... blastncbi(Seq, Program, ...'Expect', ExpectValue, ...)
... blastncbi(Seq, Program, ...'Word', WordValue, ...)
... blastncbi(Seq, Program, ...'Matrix', MatrixValue, ...)
... blastncbi(Seq, Program, ...'GapOpen',
GapOpenValue, ...)
... blastncbi(Seq, Program, ...'ExtendGap', ExtendGapValue,
...)
... blastncbi(Seq, Program, ...'GapCosts', GapCostsValue,
...)
... blastncbi(Seq, Program, ...'Inclusion', InclusionValue,
...)
... blastncbi(Seq, Program, ...'Pct', PctValue, ...)
... blastncbi(Seq, Program, ...'Entrez', EntrezValue, ...)
```

Arguments*Seq*

Nucleotide or amino acid sequence specified by any of the following:

- GenBank, GenPept, or RefSeq accession number
- GI sequence identifier
- FASTA file
- URL pointing to a sequence file
- String
- Character array
- MATLAB structure containing a `Sequence` field

Program

String specifying a BLAST program. Choices are:

- 'blastn' — Search nucleotide query versus nucleotide database.
- 'blastp' — Search protein query versus protein database.
- 'blastx' — Search translated nucleotide query versus protein database.
- 'megablast' — Quickly search for highly similar nucleotide sequences.
- 'psiblast' — Search protein query using position-specific iterative BLAST.
- 'tblastn' — Search protein query versus translated nucleotide database.
- 'tblastx' — Search translated nucleotide query versus translated nucleotide database.

<i>DatabaseValue</i>	<p>String specifying a database. Compatible databases depend on the type of sequence specified by <i>Seq</i>, and the program specified by <i>Program</i>.</p> <p>For a list of database choices for nucleotide sequences and amino acid sequences, see the lists in the section “Description” on page 2-141.</p>
<i>DescriptionsValue</i>	<p>Value specifying the number of short descriptions to include in the report. Default is 100, unless <i>Program</i> = 'psiblast', then default is 500.</p>
<i>AlignmentsValue</i>	<p>Value specifying the number of sequences for which high-scoring segment pairs (HSPs) are reported. Default is 100, unless <i>Program</i> = 'psiblast', then default is 500.</p>
<i>FilterValue</i>	<p>String specifying a filter. Possible choices are:</p> <ul style="list-style-type: none">• 'L' (default) — Low complexity.• 'R' — Human repeats.• 'm' — Mask for lookup table.• 'lcase' — Turn on the lowercase mask. <p>Choices vary depending on the selected <i>Program</i>. For more information, see the table Choices for Optional Properties by BLAST Program on page 2-146.</p>
<i>ExpectValue</i>	<p>Value specifying the statistical significance threshold for matches against database sequences. Choices are any real number. Default is 10.</p>

WordValue

Value specifying a word length for the query sequence.

Choices for amino acid sequences are:

- 2
- 3 (default)

Choices for nucleotide sequences are:

- 7
- 11 (default)
- 15

Choices when *Program* = 'megablast' are:

- 11
- 12
- 16
- 20
- 24
- 28 (default)
- 32
- 48
- 64

<i>MatrixValue</i>	<p>String specifying the substitution matrix for amino acid sequences only. The matrix assigns the score for a possible alignment of any two amino acid residues. Choices are:</p> <ul style="list-style-type: none">• 'PAM30'• 'PAM70'• 'BLOSUM45'• 'BLOSUM62' (default)• 'BLOSUM80'
<i>GapOpenValue</i>	<p>Integer that specifies the penalty for opening a gap in the alignment of amino acid sequences.</p> <p>Choices and default depend on the substitution matrix specified by the 'Matrix' property. For more information, see the table Choices for the GapCosts Property by Matrix on page 2-147.</p>
<i>ExtendGapValue</i>	<p>Integer that specifies the penalty for extending a gap in the alignment of amino acid sequences.</p> <p>Choices and default depend on the substitution matrix specified by the 'Matrix' property. For more information, see the table Choices for the GapCosts Property by Matrix on page 2-147.</p>
<i>GapCostsValue</i>	<p>Vector containing two integers: the first is the penalty for opening a gap, and the second is the penalty for extending the gap, in the alignment of amino acid sequences.</p> <p>Choices and default depend on the substitution matrix specified by the 'Matrix' property. For more information, see the table Choices for the GapCosts Property by Matrix on page 2-147.</p>

InclusionValue Value specifying the statistical significance threshold for including a sequence in the Position-Specific Scoring Matrix (PSSM) created by PSI-BLAST for the subsequent iteration. Default is 0.005.

Note Specify an *InclusionValue* only when *Program* = 'psiblast'.

PctValue

Value specifying the percent identity and the corresponding match and mismatch score for matching existing sequences in a public database. Choices are:

- None
- 99 (default) — 99, 1, -3
- 98 — 98, 1, -3
- 95 — 95, 1, -3
- 90 — 90, 1, -2
- 85 — 85, 1, -2
- 80 — 80, 2, -3
- 75 — 75, 4, -5
- 60 — 60, 1, -1

Note Specify a *PctValue* only when *Program* = 'megablast'.

EntrezValue

String specifying Entrez query syntax to search a subset of the selected database.

Tip Use this property to limit searches based on molecule types, sequence lengths, organisms, and so on.

Return Values

<i>RID</i>	Request ID for the NCBI BLAST report.
<i>RTOE</i>	Request Time Of Execution, which is an estimate of the time (in minutes) until completion.

Tip Use this time estimate with the 'WaitTime' property when using the `getblast` function.

Description

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

`blastncbi(Seq, Program)` sends a BLAST request to NCBI against a *Seq*, a nucleotide or amino acid sequence, using *Program*, a specified BLAST program, and then returns a command window link to the NCBI BLAST report. For help in selecting an appropriate BLAST program, visit:

<http://blast.ncbi.nlm.nih.gov/producttable.shtml>

`RID = blastncbi(Seq, Program)` returns *RID*, the Request ID for the report.

`[RID, RTOE] = blastncbi(Seq, Program)` returns both *RID*, the Request ID for the NCBI BLAST report, and *RTOE*, the Request Time Of Execution, which is an estimate of the time until completion.

Tip Use *RTOE* with the 'WaitTime' property when using the `getblast` function.

`... blastncbi(..., 'PropertyName', PropertyValue, ...)` calls `blastncbi` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each

PropertyName must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are explained below. Additional information on these optional properties can be found at:

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

... `blastncbi(Seq, Program, ...'Database', DatabaseValue, ...)` specifies a database for the alignment search. Compatible databases depend on the type of sequence specified by *Seq*, and the program specified by *Program*.

Database choices for nucleotide sequences are:

- 'nr' (default)
- 'refseq_rna'
- 'refseq_genomic'
- 'est'
- 'est_human'
- 'est_mouse'
- 'est_others'
- 'gss'
- 'htgs'
- 'pat'
- 'pdb'
- 'month'
- 'alu_repeats'
- 'dbsts'
- 'chromosome'
- 'wgs'

- 'env_nt'

Database choices for amino acid sequences are:

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

For help in selecting an appropriate database, visit:

<http://blast.ncbi.nlm.nih.gov/producttable.shtml>

... `blastncbi(Seq, Program, ...'Descriptions',
DescriptionsValue, ...)` specifies the number of short descriptions
to include in the report, when you do not specify return values.

... `blastncbi(Seq, Program, ...'Alignments',
AlignmentsValue, ...)` specifies the number of sequences for which
high-scoring segment pairs (HSPs) are reported, when you do not
specify return values.

... `blastncbi(Seq, Program, ...'Filter', FilterValue, ...)`
specifies the filter to apply to the query sequence.

... `blastncbi(Seq, Program, ...'Expect', ExpectValue,
...)` specifies a statistical significance threshold for matches against
database sequences. Choices are any real number. Default is 10. You
can learn more about the statistics of local sequence comparison at:

<http://blast.ncbi.nlm.nih.gov/tutorial/Altschul-1.html#head2>

... `blastncbi(Seq, Program, ...'Word', WordValue, ...)` specifies a word size for the query sequence.

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

... `blastncbi(Seq, Program, ...'GapOpen', GapOpenValue, ...)` specifies the penalty for opening a gap in the alignment of amino acid sequences. Choices and default depend on the substitution matrix specified by the 'Matrix' property. For more information, see the table Choices for the GapCosts Property by Matrix on page 2-147.

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

http://blast.ncbi.nlm.nih.gov/html/sub_matrix.html

... `blastncbi(Seq, Program, ...'ExtendGap', ExtendGapValue, ...)` specifies the penalty for extending a gap greater than one space in the alignment of amino acid sequences. Choices and default depend on the substitution matrix specified by the 'Matrix' property. For more information, see the table Choices for the GapCosts Property by Matrix on page 2-147.

... `blastncbi(Seq, Program, ...'GapCosts', GapCostsValue, ...)` specifies the penalty for opening and extending a gap in the alignment of amino acid sequences. *GapCostsValue* is a vector containing two integers: the first is the penalty for opening a gap, and the second is the penalty for extending the gap. Choices and default depend on the substitution matrix specified by the 'Matrix' property. For more information, see the table Choices for the GapCosts Property by Matrix on page 2-147.

... `blastncbi(Seq, Program, ...'Inclusion', InclusionValue, ...)` specifies the statistical significance threshold for including a sequence in the Position-Specific Scoring Matrix (PSSM) created by PSI-BLAST for the subsequent iteration. Default is 0.005.

Note Specify an *InclusionValue* only when *Program* = 'psiblast'.

... `blastncbi(Seq, Program, ...'Pct', PctValue, ...)`
specifies the percent identity and the corresponding match and mismatch score for matching existing sequences in a public database. Default is 99.

Note Specify a *PctValue* only when *Program* = 'megablast'.

... `blastncbi(Seq, Program, ...'Entrez', EntrezValue, ...)`
specifies Entrez query syntax to search a subset of the selected database.

Note For more information about Entrez query syntax, see:

<http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=helpentrez.chapter.EntrezHelp>

Tip Use this property to limit searches based on molecule types, sequence lengths, organisms, and so on. For more information on limiting searches, see:

http://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml#entrez_query

Choices for Optional Properties by BLAST Program

2-14

Then choices for the following properties are...						
When BLAST program is...	Database	Filter	Word	Matrix	GapCosts	Pct
'blastn'	'nr' (default) 'est' 'est_human' 'est_mouse' 'est_others'	'L' (default) 'R' 'm' 'lcase'	7 11 (default) 15	—	—	—
'megablast'	'gss' 'htgs' 'pat' 'pdb' 'month' 'alu_repeats' 'dbsts' 'chromosome' 'wgs'	'L'	11 12 16 20 24 28 (default) 32 48 64			None 99 (default) 98 95 90 85 80 75 60
'tblastn'	'refseq_rna' 'refseq_genomic' 'env_nt'	'L' (default) 'm' 'lcase'	2 3 (default)	'PAM30' 'PAM70' 'BLOSUM45' 'BLOSUM62' (default) 'BLOSUM80'	See the next table.	—
'tblastx'		'L' (default) 'R' 'm' 'lcase'				
'blastp'	'nr' (default) 'swissprot'	'L' (default) 'm'				
'blastx'	'pat'	'lcase'				
'psiblast'	'pdb' 'month' 'refseq_protein' 'env_nr'					

Choices for the GapCosts Property by Matrix

When substitution matrix is...	Then choices for GapCosts are...
'PAM30'	[7 2] [6 2] [5 2] [10 1] [9 1] (default) [8 1]
'PAM70'	[8 2] [7 2] [6 2] [11 1] [10 1] (default) [9 1]
'BLOSUM80'	[13 3] [12 3] [11 3] [10 3] [15 2] (default) [14 2] [13 2] [12 2] [19 1] [18 1] [17 1] [16 1]
'BLOSUM45'	[9 2] [8 2] [7 2] [12 1] [11 1] (default) [10 1]
'BLOSUM62'	[9 2] [8 2] [7 2] [12 1] [11 1] (default) [10 1]

blastncbi

Examples

```
% Get a sequence from the Protein Data Bank and create
% a MATLAB structure.
S = getpdb('1CIV')

% Use the structure as input for a BLAST search with an
% expectation of 1e-10.
blastncbi(S,'blastp','expect',1e-10)

% Click the URL link (Link to NCBI BLAST Request) to go
% directly to the NCBI request.

% You can also perform a typical BLAST protein search directly with
% an accession number and an alternative scoring matrix.
RID = blastncbi('AAA59174','blastp','matrix','PAM70','...
               'expect',1e-10)

% You can pass the RID to GETBLAST to parse the report and
% load it into a MATLAB structure.
Struct = getblast(RID)
```

References

- [1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990). Basic local alignment search tool. *J. Mol. Biol.* *215*, 403–410.
- [2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* *25*, 3389–3402.

See Also

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

Purpose	Read data from NCBI BLAST report file		
Syntax	<code>Data = blastread(BLASTReport)</code>		
Arguments	<table><tr><td><i>BLASTReport</i></td><td>NCBI BLAST-formatted report specified by any of the following:<ul style="list-style-type: none">• File name or path and file name, such as returned by the <code>getblast</code> function with the <code>'ToFile'</code> property.• URL pointing to an NCBI BLAST report.• MATLAB character array that contains the text for an NCBI BLAST report.<p>If you specify only a file name, that file must be on the MATLAB search path or in the current directory.</p></td></tr></table>	<i>BLASTReport</i>	NCBI BLAST-formatted report specified by any of the following: <ul style="list-style-type: none">• File name or path and file name, such as returned by the <code>getblast</code> function with the <code>'ToFile'</code> property.• URL pointing to an NCBI BLAST report.• MATLAB character array that contains the text for an NCBI BLAST report. <p>If you specify only a file name, that file must be on the MATLAB search path or in the current directory.</p>
<i>BLASTReport</i>	NCBI BLAST-formatted report specified by any of the following: <ul style="list-style-type: none">• File name or path and file name, such as returned by the <code>getblast</code> function with the <code>'ToFile'</code> property.• URL pointing to an NCBI BLAST report.• MATLAB character array that contains the text for an NCBI BLAST report. <p>If you specify only a file name, that file must be on the MATLAB search path or in the current directory.</p>		
Return Values	<table><tr><td><i>Data</i></td><td>MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from an NCBI BLAST report.</td></tr></table>	<i>Data</i>	MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from an NCBI BLAST report.
<i>Data</i>	MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from an NCBI BLAST report.		
Description	<p>The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of protein and nucleotide sequences against known sequences in online databases. BLAST reports can be lengthy, and parsing the data from the various formats can be cumbersome.</p> <p><code>Data = blastread(BLASTReport)</code> reads a BLAST report from <i>BLASTReport</i>, an NCBI-formatted report, and returns <i>Data</i>, a MATLAB structure or array of structures (if multiple query sequences) containing</p>		

blastread

fields corresponding to the BLAST keywords. `blastread` parses the basic BLAST reports BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX.

Data contains the following fields.

Field	Description
RID	Request ID for retrieving results for a specific NCBI BLAST search.
Algorithm	NCBI algorithm used to do a BLAST search.
Query	Identifier of the query sequence submitted to a BLAST search.
Database	All databases searched.
Hits.Name	Name of a database sequence (subject sequence) that matched the query sequence.
Hits.Length	Length of a subject sequence.
Hits.HSPs.Score	Pairwise alignment score for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Expect	Expectation value for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Identities	Identities (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.

Field	Description
Hits.HSPs.Positives	<p data-bbox="842 331 1331 487">Identical or similar residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject amino acid sequence.</p> <hr data-bbox="842 543 1331 546"/> <p data-bbox="842 557 1331 649">Note This field applies only to translated nucleotide or amino acid query sequences and/or databases.</p> <hr data-bbox="842 656 1331 659"/>
Hits.HSPs.Gaps	<p data-bbox="842 701 1331 821">Nonaligned residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</p>
Hits.HSPs.Frame	<p data-bbox="842 843 1331 963">Reading frame of the translated nucleotide sequence for a high-scoring sequence pair between the query sequence and a subject sequence.</p> <hr data-bbox="842 1019 1331 1022"/> <p data-bbox="842 1032 1331 1152">Note This field applies only when performing translated searches, that is, when using tblastx, tblastn, and blastx.</p> <hr data-bbox="842 1159 1331 1163"/>

blastread

Field	Description
Hits.HSPs.Strand	Sense (Plus = 5' to 3' and Minus = 3' to 5') of the DNA strands for a high-scoring sequence pair between the query sequence and a subject sequence. <hr/> Note This field applies only when using a nucleotide query sequence and database. <hr/>
Hits.HSPs.Alignment	Three-row matrix showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.QueryIndices	Indices of the query sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.SubjectIndices	Indices of the subject sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
Statistics	Summary of statistical details about the performed search, such as lambda values, gap penalties, number of sequences searched, and number of hits.

Examples

- 1 Create an NCBI BLAST report request using a GenPept accession number.

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



```
RID =
```

```
'1175088155-31624-126008617054.BLASTQ3'
```

- 2 Pass the Request ID for the report to the `getblast` function, and save the report data to a text file.

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

Note You may need to wait for the report to become available on the NCBI Web site before you can run the preceding command.

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

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

```
resultsStruct =
```

```
      RID: '1175093446-29831-201366571074.BLASTQ2'  
  Algorithm: 'BLASTP 2.2.16 [Mar-11-2007]'  
      Query: [1x63 char]  
   Database: [1x96 char]  
        Hits: [1x50 struct]  
 Statistics: [1x1034 char]
```

References

- [1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990). Basic local alignment search tool. *J. Mol. Biol.* *215*, 403–410.
- [2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* *25*, 3389–3402.

For more information about reading and interpreting NCBI BLAST reports, see:

blastread

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

See Also

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

Purpose Read data from local BLAST report

Syntax `Data = blastreadlocal(BLASTReport, Format)`

Arguments *BLASTReport* BLAST report specified by any of the following:

- File name or path and file name of a locally created BLAST report file, such as returned by the `blastlocal` function with the 'ToFile' property.
- MATLAB character array that contains the text for a local BLAST report.

If you specify only a file name, that file must be on the MATLAB search path or in the current directory.

Format Integer specifying the alignment format used to create *BLASTReport*. Choices are:

- 0 — Pairwise
- 1 — Query-anchored, showing identities
- 2 — Query-anchored, no identities
- 3 — Flat query-anchored, showing identities
- 4 — Flat query-anchored, no identities
- 5 — Query-anchored, no identities and blunt ends
- 6 — Flat query-anchored, no identities and blunt ends
- 7 — Not used
- 8 — Tabular
- 9 — Tabular with comment lines

blastreadlocal

Return Values

Data MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report.

Description

The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of protein and nucleotide sequences against known sequences in online and local databases. BLAST reports can be lengthy, and parsing the data from the various formats can be cumbersome.

Data = `blastreadlocal(BLASTReport, Format)` reads *BLASTReport*, a locally created BLAST report file, and returns *Data*, a MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from a local BLAST report. *Format* is an integer specifying the alignment format used to create *BLASTReport*.

Note The function assumes the BLAST report was produced using version 2.2.17 of the `blastall` executable.

Data contains a subset of the following fields, based on the specified alignment format.

Field	Description
Algorithm	NCBI algorithm used to do a BLAST search.
Query	Identifier of the query sequence submitted to a BLAST search.
Length	Length of the query sequence.
Database	All databases searched.

Field	Description
Hits.Name	Name of a database sequence (subject sequence) that matched the query sequence.
Hits.Score	Alignment score between the query sequence and the subject sequence.
Hits.Expect	Expectation value for the alignment between the query sequence and the subject sequence.
Hits.Length	Length of a subject sequence.
Hits.HSPs.Score	Pairwise alignment score for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Expect	Expectation value for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Identities	Identities (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.

blastreadlocal

Field	Description
Hits.HSPs.Positives	<p>Identical or similar residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject amino acid sequence.</p> <hr/> <p>Note This field applies only to translated nucleotide or amino acid query sequences and/or databases.</p> <hr/>
Hits.HSPs.Gaps	<p>Nonaligned residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</p>
Hits.HSPs.Mismatches	<p>Residues that are not similar to each other (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</p>

Field	Description
Hits.HSPs.Frame	<p>Reading frame of the translated nucleotide sequence for a high-scoring sequence pair between the query sequence and a subject sequence.</p> <hr/> <p>Note This field applies only when performing translated searches, that is, when using <code>tblastx</code>, <code>tblastn</code>, and <code>blastx</code>.</p>
Hits.HSPs.Strand	<p>Sense (Plus = 5' to 3' and Minus = 3' to 5') of the DNA strands for a high-scoring sequence pair between the query sequence and a subject sequence.</p> <hr/> <p>Note This field applies only when using a nucleotide query sequence and database.</p>
Hits.HSPs.Alignment	<p>Three-row matrix showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence.</p>
Hits.HSPs.QueryIndices	<p>Indices of the query sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.</p>

blastreadlocal

Field	Description
Hits.HSPs.SubjectIndices	Indices of the subject sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.AlignmentLength	Length of the pairwise alignment for a high-scoring sequence pair between the query sequence and a subject sequence.
Alignment	Entire alignment for the query sequence and the subject sequence(s).
Statistics	Summary of statistical details about the performed search, such as lambda values, gap penalties, number of sequences searched, and number of hits.

Examples

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

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

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

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

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

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

```
S1 = getgenbank('M28570.1');  
S2 = getgenbank('M12565');
```


- 4 Use the `fastawrite` function to create a FASTA file named `query_multi_nt.fa` from these two sequences, using the only accession number as the header.

```
Seqs(1).Header = S1.Accession;
Seqs(1).Sequence = S1.Sequence;
Seqs(2).Header = S2.Accession;
Seqs(2).Sequence = S2.Sequence;
fastawrite('query_multi_nt.fa', Seqs);
```

- 5 Submit the query sequences in the `query_multi_nt.fa` FASTA file for a BLAST search of the local nucleotide database `ecoli.nt`. Specify the BLAST program `blastn` and a tabular alignment format. Save the contents of the BLAST report to a file named `myecoli_nt8.txt`, and then read the local BLAST report, displaying the results in the MATLAB Command Window.

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

- 6 Submit the query sequences in the `query_multi_nt.fa` FASTA file for a BLAST search of the local nucleotide database `ecoli.nt`. Specify the BLAST program `blastn` and a query-anchored format. Save the contents of the BLAST report to a file named `myecoli_nt1.txt`, and then read the local BLAST report, saving the results in `results`, an array of structures.

```
blastlocal('inputquery', 'query_multi_nt.fa',...
           'database', 'ecoli.nt',...
           'tofile', 'myecoli_nt1.txt', 'program', 'blastn',...
           'format', 1);
results = blastreadlocal('myecoli_nt1.txt', 1);
```

References

- [1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. (1990). Basic local alignment search tool. *J. Mol. Biol.* *215*, 403–410.

blastreadlocal

[2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* 25, 3389–3402.

For more information about reading and interpreting BLAST reports, see:

<http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/information3.html>

See Also

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

Purpose

Return BLOSUM scoring matrix

Syntax

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

Arguments

<i>Identity</i>	Scalar specifying a percent identity level. Choices are: <ul style="list-style-type: none"> • Values from 30 to 90 in increments of 5 • 62 • 100
<i>ExtendedValue</i>	Controls the listing of extended amino acid codes. Choices are <code>true</code> (default) or <code>false</code> .
<i>OrderValue</i>	Character string of legal amino acid characters that specifies the order amino acids are listed in the matrix. The length of the character string must be 20 or 24.

blosum

Return Values

<i>Matrix</i>	BLOSUM (Blocks Substitution Matrix) scoring matrix with a specified percent identity.
<i>MatrixInfo</i>	Structure of information about <i>Matrix</i> containing the following fields: <ul style="list-style-type: none">• Name• Scale• Entropy• ExpectedScore• HighestScore• LowestScore• Order

Description

Matrix = `blosum(Identity)` returns a BLOSUM (Blocks Substitution Matrix) scoring matrix with a specified percent identity. The default ordering of the output includes the extended characters B, Z, X, and *.

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

`[Matrix, MatrixInfo]` = `blosum(Identity)` returns *MatrixInfo*, a structure of information about *Matrix*, a BLOSUM matrix. *MatrixInfo* contains the following fields:

- Name
- Scale
- Entropy
- ExpectedScore
- HighestScore
- LowestScore

- Order

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

`...` = `blosum(Identity, ...'Extended', ExtendedValue, ...)` controls the listing of extended amino acid codes. Choices are `true` (default) or `false`. If *ExtendedValue* is `false`, returns the scoring matrix for the standard 20 amino acids. Ordering of the output when *ExtendedValue* is `false` is

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

`...` = `blosum(Identity, ...'Order', OrderValue, ...)` returns a BLOSUM matrix ordered by *OrderValue*, a character string of legal amino acid characters that specifies the order amino acids are listed in the matrix. The length of the character string must be 20 or 24.

Examples

Return a BLOSUM matrix with a percent identity level of 50.

```
B50 = blosum(50)
```

Return a BLOSUM matrix with the amino acids in a specific order.

```
B75 = blosum(75, 'Order', 'CSTPAGNDEQHRKMILVFW')
```

See Also

Bioinformatics Toolbox functions: `dayhoff`, `gonnet`, `nwalign`, `pam`, `swalign`

celintensityread

Purpose Read probe intensities from Affymetrix CEL files

Syntax

```
ProbeStructure = celintensityread(CELFiles, CDFFile)
ProbeStructure = celintensityread(..., 'CELPath',
CELPathValue, ...)
ProbeStructure = celintensityread(..., 'CDFPath',
CDFPathValue, ...)
ProbeStructure = celintensityread(..., 'PMOnly',
PMOnlyValue,
...)
ProbeStructure = celintensityread(..., 'Verbose',
VerboseValue, ...)
```

Arguments

<i>CELFiles</i>	Any of the following: <ul style="list-style-type: none">• String specifying a single CEL file name.• '*', which reads all CEL files in the current directory.• ' ', which opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold Ctrl or Shift while clicking to select multiple CEL files.• Cell array of CEL file names.
<i>CDFFile</i>	Either of the following: <ul style="list-style-type: none">• String specifying a CDF file name.• ' ', which opens the Select CDF File dialog box from which you select the CDF file.
<i>CELPathValue</i>	String specifying the path and directory where the files specified in <i>CELFiles</i> are stored.

<i>CDFPathValue</i>	String specifying the path and directory where the file specified in <i>CDFFile</i> is stored.
<i>PMOnlyValue</i>	Property to include or exclude the mismatch (MM) probe intensity values in the returned structure. Enter <code>true</code> to return only perfect match (PM) probe intensities. Enter <code>false</code> to return both PM and MM probe intensities. Default is <code>true</code> .
<i>VerboseValue</i>	Controls the display of a progress report showing the name of each CEL file as it is read. When <i>VerboseValue</i> is <code>false</code> , no progress report is displayed. Default is <code>true</code> .

Return Values

<i>ProbeStructure</i>	MATLAB structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs.
-----------------------	--

Description

Note This function does not work on the Solaris platform.

ProbeStructure = `celintensityread(CELFiles, CDFFile)` reads the specified Affymetrix CEL files and the associated CDF library file (created from Affymetrix GeneChip arrays for expression or genotyping assays), and then creates *ProbeStructure*, a structure containing information from the CEL files, including probe intensities, probe indices, and probe set IDs. *CELFiles* is a string or cell array of CEL file names. *CDFFile* is a string specifying a CDF file name.

If you set *CELFiles* to `'*'`, then it reads all CEL files in the current directory. If you set *CELFiles* to `' '`, then it opens the Select CEL Files dialog box from which you select the CEL files. From this dialog box, you can press and hold **Ctrl** or **Shift** while clicking to select multiple CEL files.

celintensityread

If you set *CDFFile* to ' ', then it opens the Select CDF File dialog box from which you select the CDF file.

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

ProbeStructure = celintensityread(..., '*CELPPath*', *CELPPathValue*, ...) specifies a path and directory where the files specified by *CELFiles* are stored.

ProbeStructure = celintensityread(..., '*CDFPath*', *CDFPathValue*, ...) specifies a path and directory where the file specified by *CDFFile* is stored.

ProbeStructure = celintensityread(..., '*PMOnly*', *PMOnlyValue*, ...) includes or excludes the mismatch (MM) probe intensity values. When *PMOnlyValue* is true, celintensityread returns only perfect match (PM) probe intensities. When *PMOnlyValue* is false, celintensityread returns both PM and MM probe intensities. Default is true.

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

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

Note Some Affymetrix CEL files are combined with other data files in a DTT or CAB file. You must download and use the Affymetrix Data Transfer Tool to extract these files from the DTT or CAB file. You can download the Affymetrix Data Transfer Tool from:

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

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

ProbeStructure contains the following fields.

Field	Description
CDFName	File name of the Affymetrix CDF library file.
CELNames	Cell array of names of the Affymetrix CEL files.
NumChips	Number of CEL files read into the structure.
NumProbeSets	Number of probe sets in each CEL file.
NumProbes	Number of probes in each CEL file.
ProbeSetIDs	Cell array of the probe set IDs from the Affymetrix CDF library file.
ProbeIndices	Column vector containing probe indexing information. Probes within a probe set are numbered 0 through $N - 1$, where N is the number of probes in the probe set.

celintensityread

Field	Description
GroupNumbers	Column vector containing group numbers for probes within the probe set. For gene expression data, the group number for all probes is 1. For SNP (genotyping) data, the group numbers for probes are: <ul style="list-style-type: none">• 1 — Allele A – (sense)• 2 — Allele B – (sense)• 3 — Allele A + (antisense)• 4 — Allele B + (antisense)
PMIntensities	Matrix containing perfect match (PM) probe intensity values. Each row corresponds to a probe, and each column corresponds to a CEL file. The rows are ordered the same way as in <code>ProbeIndices</code> , and the columns are ordered the same way as in the <code>CELFiles</code> input argument.
MMIntensities (optional)	Matrix containing mismatch (MM) probe intensity values. Each row corresponds to a probe, and each column corresponds to a CEL file. The rows are ordered the same way as in <code>ProbeIndices</code> , and the columns are ordered the same way as in the <code>CELFiles</code> input argument.

`ProbeStructure = celintensityread(..., 'Verbose', VerboseValue, ...)` controls the display of a progress report showing the name of each CEL file as it is read. When `VerboseValue` is `false`, no progress report is displayed. Default is `true`.

Examples

The following example assumes that you have the `HG_U95Av2.CDF` library file stored at `D:\Affymetrix\LibFiles\HGGenome`, and that your current directory points to a location containing CEL files associated

with this CDF library file. In this example, the `celintensityread` function reads all the CEL files in the current directory and a CDF file in a specified directory. The next command line uses the `rmabackadj` function to perform background adjustment on the PM probe intensities in the `PMIntensities` field of `PMProbeStructure`.

```
PMProbeStructure = celintensityread('*', 'HG_U95Av2.CDF',...  
                                     'CDFPath', 'D:\Affymetrix\LibFiles\HGGenome');  
BackAdjustedMatrix = rmabackadj(PMProbeStructure.PMIntensities);
```

The following example lets you select CEL files and a CDF file to read using Open File dialog boxes:

```
PMProbeStructure = celintensityread(' ', ' ');
```

See Also

Bioinformatics Toolbox functions: `affygcrrma`, `affyinvarsetnorm`, `affyprobeseqread`, `affyread`, `affyrma`, `affysnpintensitysplit`, `agferead`, `gcrma`, `gcrmabackadj`, `gprread`, `ilmnbsread`, `probelibraryinfo`, `probesetlink`, `probesetlookup`, `probesetplot`, `probesetvalues`, `rmabackadj`, `rmasummary`, `sptread`

Purpose

Perform circular binary segmentation (CBS) on array-based comparative genomic hybridization (aCGH) data

Syntax

```
SegmentStruct = cghcbs(CGHData)
SegmentStruct = cghcbs(CGHData, ...'Alpha',
AlphaValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Permutations',
PermutationsValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Method', MethodValue,
...)
SegmentStruct = cghcbs(CGHData, ...'Smooth', SmoothValue,
...)
SegmentStruct = cghcbs(CGHData, ...'Prune',
PruneValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Errsum', ErrsumValue,
...)
SegmentStruct = cghcbs(CGHData, ...'WindowSize',
WindowSizeValue, ...)
SegmentStruct = cghcbs(CGHData, ...'SampleIndex',
SampleIndexValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Chromosome',
ChromosomeValue, ...)
SegmentStruct = cghcbs(CGHData, ...'Showplot',
ShowplotValue,
...)
SegmentStruct = cghcbs(CGHData, ...'Verbose', VerboseValue,
...)
```

Arguments

CGHData

Array-based comparative genomic hybridization (aCGH) data in either of the following forms:

- Structure with the following fields:
 - **Sample** — Cell array of strings containing the sample names (optional).
 - **Chromosome** — Vector containing the chromosome numbers on which the clones are located.
 - **GenomicPosition** — Vector containing the genomic positions (in any unit) to which the clones are mapped.
 - **Log2Ratio** — Matrix containing \log_2 ratio of test to reference signal intensity for each clone. Each row corresponds to a clone, and each column corresponds to a sample.
- Matrix in which each row corresponds to a clone. The first column contains the chromosome number, the second column contains the genomic position, and the remaining columns each contain the \log_2 ratio of test to reference signal intensity for a sample.

AlphaValue

Scalar that specifies the significance level for the statistical tests to accept change points. Default is 0.01.

PermutationsValue

Scalar that specifies the number of permutations used for p-value estimation. Default is 10,000.

<i>MethodValue</i>	String that specifies the method to estimate the p-values. Choices are 'Perm' or 'Hybrid' (default). 'Perm' does a full permutation, while 'Hybrid' uses a faster, tail probability-based permutation. When using the 'Hybrid' method, the 'Perm' method is applied automatically when segment data length becomes less than 200.
<i>SmoothValue</i>	Controls the smoothing of outliers before segmenting using the procedure explained by Olshen et al. (2004). Choices are true (default) or false.
<i>PruneValue</i>	Controls the elimination of change points identified due to local trends in the data that are not indicative of real copy number change, using the procedure explained by Olshen et al. (2004). Choices are true or false (default).
<i>ErrsumValue</i>	Scalar that specifies the allowed proportional increase in the error sum of squares when eliminating change points using the 'Prune' property. Commonly used values are 0.05 and 0.1. Default is 0.05.
<i>WindowSizeValue</i>	Scalar that specifies the size of the window (in data points) used to divide the data when using the 'Perm' method on large data sets. Default is 200.
<i>SampleIndexValue</i>	A single sample index or a vector of sample indices that specify the sample(s) to analyze. Default is all sample indices.
<i>ChromosomeValue</i>	A single chromosome number or a vector of chromosome numbers that specify the data to analyze. Default is all chromosome numbers.

ShowplotValue

Controls the display of plots of the segment means over the original data. Choices are either:

- `true` — All chromosomes in all samples are plotted. If there are multiple samples in *CGHData*, then each sample is plotted in a separate Figure window.
- `false` — No plot.
- `W` — The layout displays all chromosomes in the whole genome in one plot in the Figure window.
- `S` — The layout displays each chromosome in a subplot in the Figure window.
- `I` — An integer specifying only one of the chromosomes in *CGHData* to be plotted.

Default is:

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

VerboseValue

Controls the display of a progress report of the analysis. Choices are `true` (default) or `false`.

Return Values

SegmentStruct

Structure containing segmentation information in the following fields:

- **Sample** — Sample name from *CGHData* input argument. If the input argument does not include sample names, then sample names are assigned as *Sample1*, *Sample2*, and so forth.
- **SegmentData** — Structure array containing segment data for the sample in the following fields:
 - **Chromosome** — Chromosome number on which the segment is located.
 - **Start** — Genomic position at the start of the segment (in the same units as used for the *CGHData* input).
 - **End** — Genomic position at the end of the segment (in the same units as used for the *CGHData* input).
 - **Mean** — Mean value of the \log_2 ratio of the test to reference signal intensity for the segment.

Description

SegmentStruct = *cghcbs*(*CGHData*) performs circular binary segmentation (CBS) on array-based comparative genomic hybridization (aCGH) data to determine the copy number alteration segments (neighboring regions of DNA that exhibit a statistical difference in copy number) and change points.

Note The CBS algorithm recursively splits chromosomes into segments based on a maximum t statistic estimated by permutation. This computation can be time consuming. If n = number of data points, then computation time $\sim O(n^2)$.

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

SegmentStruct = cghcbs(*CGHData*, ...'*Alpha*', *AlphaValue*, ...) specifies the significance level for the statistical tests to accept change points. Default is 0.01.

SegmentStruct = cghcbs(*CGHData*, ...'*Permutations*', *PermutationsValue*, ...) specifies the number of permutations used for p-value estimation. Default is 10,000.

SegmentStruct = cghcbs(*CGHData*, ...'*Method*', *MethodValue*, ...) specifies the method to estimate the p-values. Choices are 'Perm' or 'Hybrid' (default). 'Perm' does a full permutation, while 'Hybrid' uses a faster, tail probability-based permutation. When using the 'Hybrid' method, the 'Perm' method is applied automatically when segment data length becomes less than 200.

SegmentStruct = cghcbs(*CGHData*, ...'*Smooth*', *SmoothValue*, ...) controls the smoothing of outliers before segmenting, using the procedure explained by Olshen et al. (2004). Choices are true (default) or false.

SegmentStruct = cghcbs(*CGHData*, ...'*Prune*', *PruneValue*, ...) controls the elimination of change points identified due to local trends in the data that are not indicative of real copy number change, using the procedure explained by Olshen et al. (2004). Choices are true or false (default).

SegmentStruct = `cghcbs(CGHData, ...'Errsum', ErrsumValue, ...)` specifies the allowed proportional increase in the error sum of squares when eliminating change points using the 'Prune' property. Commonly used values are 0.05 and 0.1. Default is 0.05.

SegmentStruct = `cghcbs(CGHData, ...'WindowSize', WindowSizeValue, ...)` specifies the size of the window (in data points) used to divide the data when using the 'Perm' method on large data sets. Default is 200.

SegmentStruct = `cghcbs(CGHData, ...'SampleIndex', SampleIndexValue, ...)` analyzes only the sample(s) specified by *SampleIndexValue*, which can be a single sample index or a vector of sample indices. Default is all sample indices.

SegmentStruct = `cghcbs(CGHData, ...'Chromosome', ChromosomeValue, ...)` analyzes only the data on the chromosomes specified by *ChromosomeValue*, which can be a single chromosome number or a vector of chromosome numbers. Default is all chromosome numbers.

SegmentStruct = `cghcbs(CGHData, ...'Showplot', ShowplotValue, ...)` controls the display of plots of the segment means over the original data. Choices are `true`, `false`, `W`, `S`, or `I`, an integer specifying one of the chromosomes in *CGHData*. When *ShowplotValue* is `true`, all chromosomes in all samples are plotted. If there are multiple samples in *CGHData*, then each sample is plotted in a separate Figure window. When *ShowplotValue* is `W`, the layout displays all chromosomes in one plot in the Figure window. When *ShowplotValue* is `S`, the layout displays each chromosome in a subplot in the Figure window. When *ShowplotValue* is `I`, only the specified chromosome is plotted. Default is either:

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

`SegmentStruct = cghcbs(CGHData, ...'Verbose', VerboseValue, ...)` controls the display of a progress report of the analysis. Choices are true (default) or false.

Examples

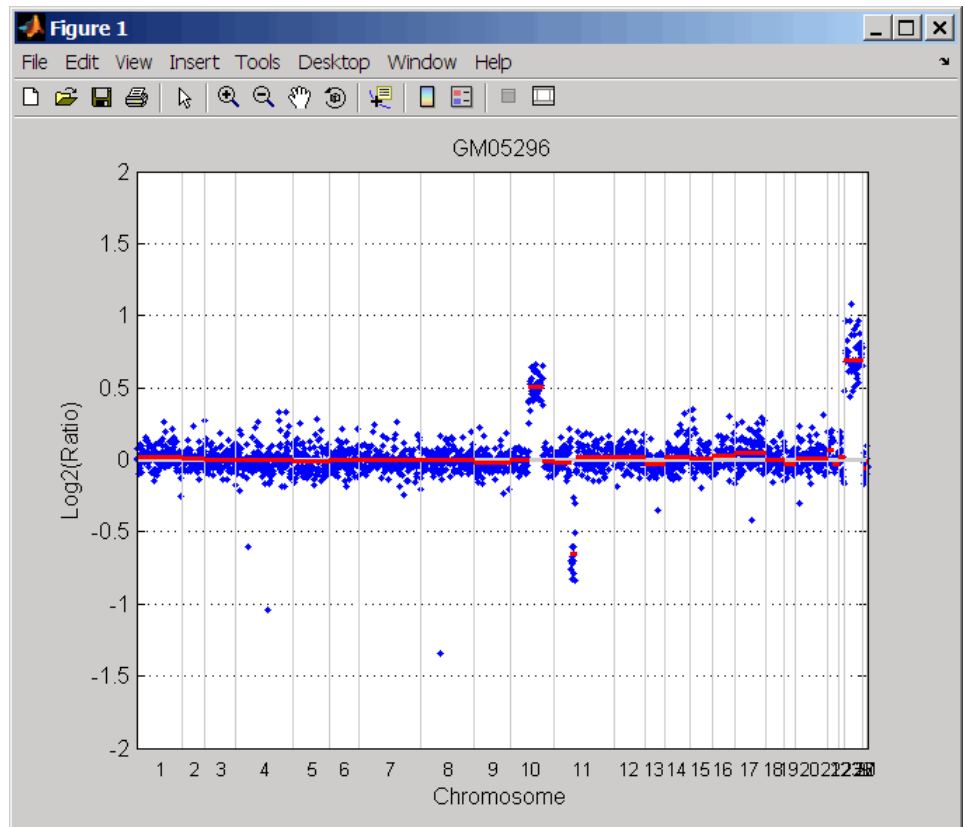
Analyzing Data from the Coriell Cell Line Study

- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains `coriell_data`, a structure of array-based CGH data.

```
load coriell_baccgh
```

- 2 Analyze all chromosomes of sample 3 (GM05296) of the aCGH data and return segmentation data in a structure, `S`. Plot the segment means over the original data for all chromosomes of this sample.

```
S = cghcbs(coriell_data, 'sampleindex', 3, 'showplot', true);
```



Chromosome 10 shows a gain, while chromosome 11 shows a loss.

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

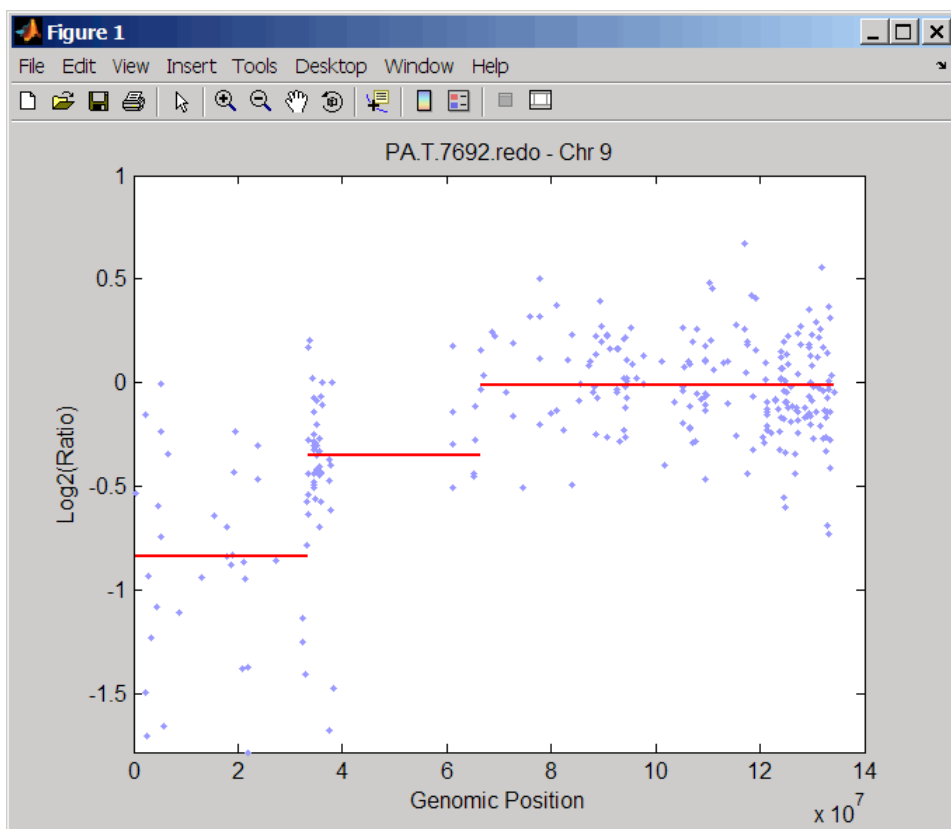
Analyzing Data from a Pancreatic Cancer Study

- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains `pancrea_data`, a structure of array-based CGH data from a pancreatic cancer study.

```
load pancrea_oligocgh
```

- Analyze only chromosome 9 in sample 32 of the CGH data and return the segmentation data in a structure, PS. Plot the segment means over the original data for chromosome 9 in this sample.

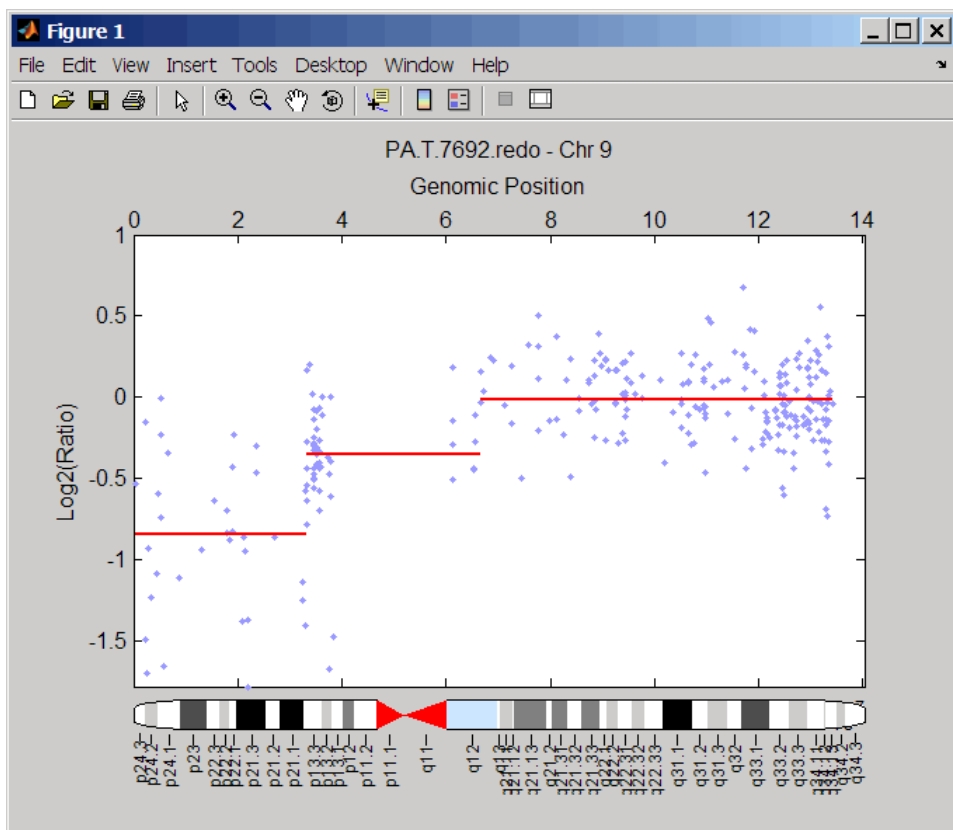
```
PS = cghcbs(pancrea_data, 'sampleindex', 32, 'chromosome', 9, ...  
           'showplot', 9);
```



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

- 3 Use the chromosomeplot function with the 'addtplot' property to add the ideogram of chromosome 9 for *Homo sapiens* to the plot of the segmentation data.

```
chromosomeplot('hs_cytoBand.txt', 9, 'addtplot', gca)
```



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

Displaying Copy Number Alteration Regions Aligned to a Chromosome Ideogram

- 1 Create a structure containing segment gain and loss information for chromosomes 10 and 11 from sample 3 from the Coriell cell line study, making sure the segment data is in bp units. (You can determine copy number variance (CNV) information by exploring `S`, the structure of segments returned by the `cghcbs` function in *Analyzing Data from the Coriell Cell Line Study* on page 2-179.) For the `'CNVType'` field, use 1 to indicate a loss and 2 to indicate a gain.

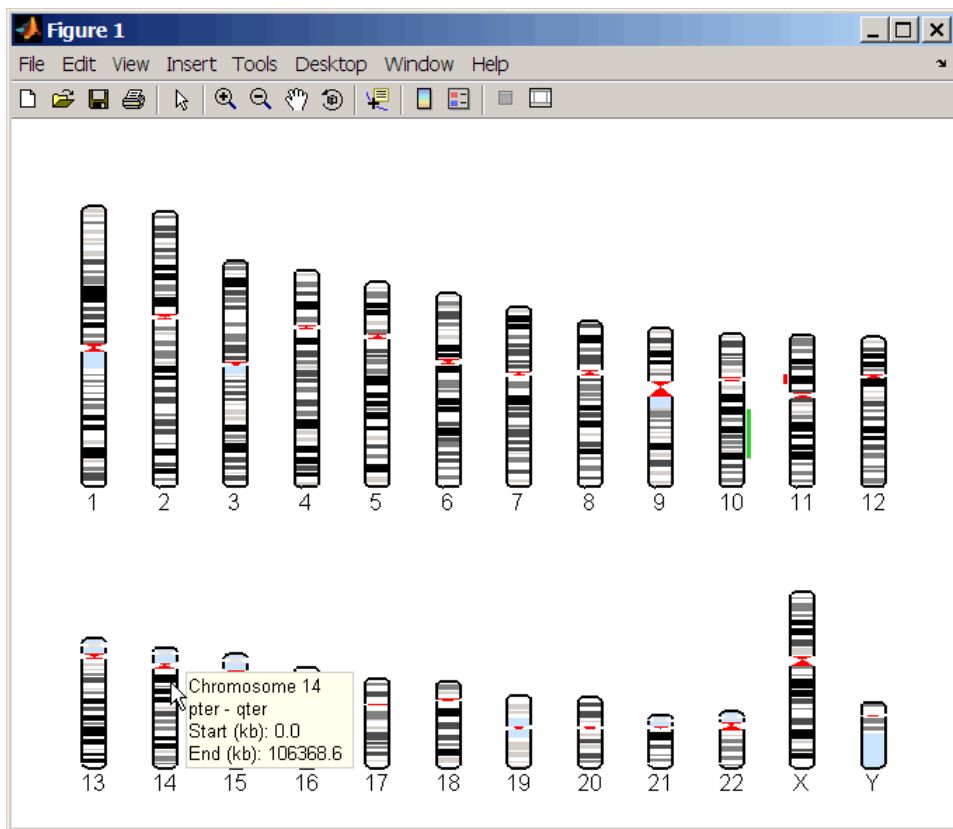
```
cnvStruct = struct('Chromosome', [10 11],...
    'CNVType', [2 1],...
    'Start', [S.SegmentData(10).Start(2),...
    S.SegmentData(11).Start(2)]*1000,...
    'End', [S.SegmentData(10).End(2),...
    S.SegmentData(11).End(2)]*1000)
```

```
cnvStruct =
```

```
Chromosome: [10 11]
CNVType: [2 1]
Start: [66905000 35416000]
End: [110412000 43357000]
```

- 2 Pass the structure to the `chromosomeplot` function using the `'CNV'` property to display the copy number gains (green) and losses (red) aligned to the human chromosome ideogram. Specify kb units for the display of segment information in the data tip.

```
chromosomeplot('hs_cytoBand.txt', 'cnv', cnvStruct, 'unit', 2)
```



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

References

- [1] Olshen, A.B., Venkatraman, E.S., Lucito, R., and Wigler, M. (2004). Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics* 5, 4, 557–572.
- [2] Venkatraman, E.S., and Olshen, A.B. (2007). A Faster Circular Binary Segmentation Algorithm for the Analysis of Array CGH Data. *Bioinformatics* 23(6), 657–663.

[3] Venkatraman, E.S., and Olshen, A.B. (2006).
DNAcopy: A Package for Analyzing DNA Copy Data.
<http://www.bioconductor.org/packages/2.1/bioc/html/DNAcopy.html>

[4] Snijders, A.M., Nowak, N., Segreaves, R., Blackwood, S., Brown, N., Conroy, J., Hamilton, G., Hindle, A.K., Huey, B., Kimura, K., Law, S., Myambo, K., Palmer, J., Ylstra, B., Yue, J.P., Gray, J.W., Jain, A.N., Pinkel, D., and Albertson, D.G. (2001). Assembly of microarrays for genome-wide measurement of DNA copy number. *Nature Genetics* 29, 263–264.

[5] Aguirre, A.J., Brennan, C., Bailey, G., Sinha, R., Feng, B., Leo, C., Zhang, Y., Zhang, J., Gans, J.D., Bardeesy, N., Cauwels, C., Cordon-Cardo, C., Redston, M.S., DePinho, R.A., and Chin, L. (2004). High-resolution characterization of the pancreatic adenocarcinoma genome. *PNAS* 101, 24, 9067–9072.

See Also

Bioinformatics Toolbox functions: `chromosomeplot`, `cytobandread`

cghfreqplot

Purpose

Display frequency of DNA copy number alterations across multiple samples

Syntax

```
FreqStruct = cghfreqplot(CGHData)  
FreqStruct = cghfreqplot(CGHData, ...'Threshold',  
ThresholdValue, ...)  
FreqStruct = cghfreqplot(CGHData, ...'Group', GroupValue,  
...)  
FreqStruct = cghfreqplot(CGHData, ...'Subgrp', SubgrpValue,  
...)  
FreqStruct = cghfreqplot(CGHData, ...'Subplot',  
SubplotValue,  
...)  
FreqStruct = cghfreqplot(CGHData, ...'Cutoff', CutoffValue,  
...)  
FreqStruct = cghfreqplot(CGHData, ...'Chromosome',  
ChromosomeValue, ...)  
FreqStruct = cghfreqplot(CGHData, ...'IncludeX',  
IncludeXValue, ...)  
FreqStruct = cghfreqplot(CGHData, ...'IncludeY',  
IncludeYValue, ...)  
FreqStruct = cghfreqplot(CGHData, ...'Chrominfo',  
ChrominfoValue, ...)  
FreqStruct = cghfreqplot(CGHData, ...'ShowCentr',  
ShowCentrValue, ...)  
FreqStruct = cghfreqplot(CGHData, ...'Color', ColorValue,  
...)  
FreqStruct = cghfreqplot(CGHData, ...'YLim',  
YLimValue, ...)  
FreqStruct = cghfreqplot(CGHData, ...'Titles', TitlesValue,  
...)
```

Arguments

CGHData

Array-based comparative genomic hybridization (aCGH) data in either of the following forms:

- Structure with the following fields:
 - **Sample** — Cell array of strings containing the sample names (optional).
 - **Chromosome** — Vector containing the chromosome numbers on which the clones are located.
 - **GenomicPosition** — Vector containing the genomic positions (in bp, kb, or mb units) to which the clones are mapped.
 - **Log2Ratio** — Matrix containing \log_2 ratio of test to reference signal intensity for each clone. Each row corresponds to a clone, and each column corresponds to a sample.
- Matrix in which each row corresponds to a clone. The first column contains the chromosome number, the second column contains the genomic position, and the remaining columns each contain the \log_2 ratio of test to reference signal intensity for a sample.

ThresholdValue

Positive scalar or vector that specifies the gain/loss threshold. A clone is considered to be a gain if its \log_2 ratio is above *ThresholdValue*, and a loss if its \log_2 ratio is below negative *ThresholdValue*.

The *ThresholdValue* is applied as follows:

- If a positive scalar, it is the gain and loss threshold for all the samples.
- If a two-element vector, the first element is the gain threshold for all samples, and the second element is the loss threshold for all samples.
- If a vector of the same length as the number of samples, each element in the vector is considered as a unique gain and loss threshold for each sample.

Default is 0.25.

<i>GroupValue</i>	<p>Specifies the sample groups to calculate the frequency from. Choices are:</p> <ul style="list-style-type: none">• A vector of sample column indices (for data with only one group). The samples specified in the vector are considered a group.• A cell array of vectors of sample column indices (for data divided into multiple groups). Each element in the cell array is considered a group. <p>Default is a single group of all the samples in <i>CGHData</i>.</p>
<i>SubgrpValue</i>	<p>Controls the analysis of samples by subgroups. Choices are <code>true</code> (default) or <code>false</code>.</p>
<i>SubplotValue</i>	<p>Controls the display of all plots in one Figure window when more than one subgroup is analyzed. Choices are <code>true</code> (default) or <code>false</code> (displays plots in separate windows).</p>
<i>CutoffValue</i>	<p>Scalar or two-element numeric vector that specifies a cutoff, which controls the plotting of only the clones with frequency gains or losses greater than or equal to <i>CutoffValue</i>. If a two-element vector, the first element is the cutoff for gains, and the second element is for losses. Default is 0.</p>
<i>ChromosomeValue</i>	<p>Single chromosome number or a vector of chromosome numbers that specify the chromosomes for which to display frequency plots. Default is all chromosomes in <i>CGHData</i>.</p>
<i>IncludeXValue</i>	<p>Controls the inclusion of the X chromosome in the analysis. Choices are <code>true</code> (default) or <code>false</code>.</p>
<i>IncludeYValue</i>	<p>Controls the inclusion of the Y chromosome in the analysis. Choices are <code>true</code> or <code>false</code> (default).</p>

- ChrominfoValue* Cytogenetic banding information specified by either of the following:
- Structure returned by the cytobandread function
 - String specifying the file name of an NCBI ideogram text file or a UCSC Genome Browser cytoband text file

Default is *Homo sapiens* cytogenetic banding information from the UCSC Genome Browser, NCBI Build 36.1 (<http://genome.ucsc.edu>).

- ShowCentrValue* Controls the display of the centromere positions as vertical dashed lines in the frequency plot. Choices are true (default) or false.

Tip The centromere positions are obtained from *ChrominfoValue*.

<i>ColorValue</i>	<p>Color scheme for the vertical lines in the plot, indicating the frequency of the gains and losses, specified by either of the following:</p> <ul style="list-style-type: none">• Name of or handle to a function that returns a colormap• M-by-3 matrix containing RGB values. If M equals 1, then that single color is used for all gains and losses. If M equals 2 or more, then the first row is used for gains, the second row is used for losses, and remaining rows are ignored. For example, [0 1 0;1 0 0] specifies green for gain and red for loss. <p>The default color scheme is a range of colors from pure green (gain = 1) through yellow (0) to pure red (loss = -1).</p>
<i>YLimValue</i>	<p>Two-element vector specifying the minimum and maximum values on the vertical axis. Default is [1, -1].</p>
<i>TitlesValue</i>	<p>Single string or a cell array of strings that specifies titles for the group(s), which are added to the tops of the plot(s).</p>

Return Values

FreqStruct

Structure containing frequency data in the following fields:

- **Group** — Structure array, with each structure representing a group of samples. Each structure contains the following fields:
 - **Sample** — Cell array containing names of samples within the group.
 - **GainFrequency** — Column vector containing the average gain for each clone for a group of samples.
 - **LossFrequency** — Column vector containing the average loss for each clone for a group of samples.
- **Chromosome** — Column vector containing the chromosome numbers on which the clones are located.
- **GenomicPosition** — Column vector containing the genomic positions of the clones.

Tip You can use this output structure as input to the `cghfreqplot` function.

Description

FreqStruct = `cghfreqplot(CGHData)` displays the frequency of copy number gain or loss across multiple samples for each clone on an array against their genomic position along the chromosomes.

FreqStruct = `cghfreqplot(CGHData, ... 'PropertyName', PropertyValue, ...)` calls `cghfreqplot` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single

quotation marks and is case insensitive. These property name/property value pairs are as follows:

FreqStruct = cghfreqplot(*CGHData*, ... 'Threshold', *ThresholdValue*, ...) specifies the gain/loss threshold. A clone is considered to be a gain if its \log_2 ratio is above *ThresholdValue*, and a loss if its \log_2 ratio is below negative *ThresholdValue*.

The *ThresholdValue* is applied as follows:

- If a positive scalar, it is the gain and loss threshold for all the samples.
- If a two-element vector, the first element is the gain threshold for all samples, and the second element is the loss threshold for all samples.
- If a vector of the same length as the number of samples, each element in the vector is considered as a unique gain and loss threshold for each sample.

Default is 0.25.

FreqStruct = cghfreqplot(*CGHData*, ... 'Group', *GroupValue*, ...) specifies the sample groups to calculate the frequency from.

Choices are:

- A vector of sample column indices (for data with only one group). The samples specified in the vector are considered a group.
- A cell array of vectors of sample column indices (for data divided into multiple groups). Each element in the cell array is considered a group.

Default is a single group of all the samples in *CGHData*.

FreqStruct = cghfreqplot(*CGHData*, ... 'Subgrp', *SubgrpValue*, ...) controls the analysis of samples by subgroups. Choices are true (default) or false.

FreqStruct = cghfreqplot(*CGHData*, ... 'Subplot', *SubplotValue*, ...) controls the display of all plots in one Figure

window when more than one subgroup is analyzed. Choices are `true` (default) or `false` (displays plots in separate windows).

FreqStruct = `cghfreqplot(CGHData, ... 'Cutoff', CutoffValue, ...)` specifies a cutoff value, which controls the plotting of only the clones with frequency gains or losses greater than or equal to *CutoffValue*. *CutoffValue* is a scalar or two-element numeric vector. If a two-element numeric vector, the first element is the cutoff for gains, and the second element is for losses. Default is 0.

FreqStruct = `cghfreqplot(CGHData, ... 'Chromosome', ChromosomeValue, ...)` displays the frequency plots only of chromosome(s) specified by *ChromosomeValue*, which can be a single chromosome number or a vector of chromosome numbers. Default is all chromosomes in *CGHData*.

FreqStruct = `cghfreqplot(CGHData, ... 'IncludeX', IncludeXValue, ...)` controls the inclusion of the X chromosome in the analysis. Choices are `true` (default) or `false`.

FreqStruct = `cghfreqplot(CGHData, ... 'IncludeY', IncludeYValue, ...)` controls the inclusion of the Y chromosome in the analysis. Choices are `true` or `false` (default).

FreqStruct = `cghfreqplot(CGHData, ... 'Chrominfo', ChrominfoValue, ...)` specifies the cytogenetic banding information for the chromosomes. *ChrominfoValue* can be either of the following

- Structure returned by the `cytobandread` function
- String specifying the file name of an NCBI ideogram text file or a UCSC Genome Browser `cytoband` text file

Default is *Homo sapiens* cytogenetic banding information from the UCSC Genome Browser, NCBI Build 36.1 (<http://genome.ucsc.edu>).

Tip You can download files containing cytogenetic G-banding data from the NCBI or UCSC Genome Browser ftp site. For example, you can download the cytogenetic banding data for *Homo sapiens* from:

```
ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/mapview/ideogram.gz
```

or

```
ftp://hgdownload.cse.ucsc.edu/goldenPath/hg18/database/cytoBandIdeo.txt.gz
```

FreqStruct = `cghfreqplot(CGHData, ... 'ShowCentr', ShowCentrValue, ...)` controls the display of the centromere positions as vertical dashed lines in the frequency plot. Choices are `true` (default) or `false`.

Tip The centromere positions are obtained from *ChromInfoValue*.

FreqStruct = `cghfreqplot(CGHData, ... 'Color', ColorValue, ...)` specifies a color scheme for the vertical lines in the plot, indicating the frequency of the gains and losses. Choices are:

- Name of or handle to a function that returns a colormap.
- M-by-3 matrix containing RGB values. If M equals 1, then that single color is used for all gains and losses. If M equals 2 or more, then the first row is used for gains, the second row is used for losses, and remaining rows are ignored. For example, `[0 1 0; 1 0 0]` specifies green for gain and red for loss.

The default color scheme is a range of colors from pure green (gain = 1) through yellow (0) to pure red (loss = -1).

FreqStruct = `cghfreqplot(CGHData, ... 'YLim', YLimValue, ...)` specifies the y vertical limits for the frequency plot. *YLimValue* is

a two-element vector specifying the minimum and maximum values on the vertical axis. Default is [1, -1].

FreqStruct = cghfreqplot(*CGHData*, ...'Titles', *TitlesValue*, ...) specifies titles for the group(s), which are added to the tops of the plot(s). *TitlesValue* can be a single string or a cell array of strings.

Examples

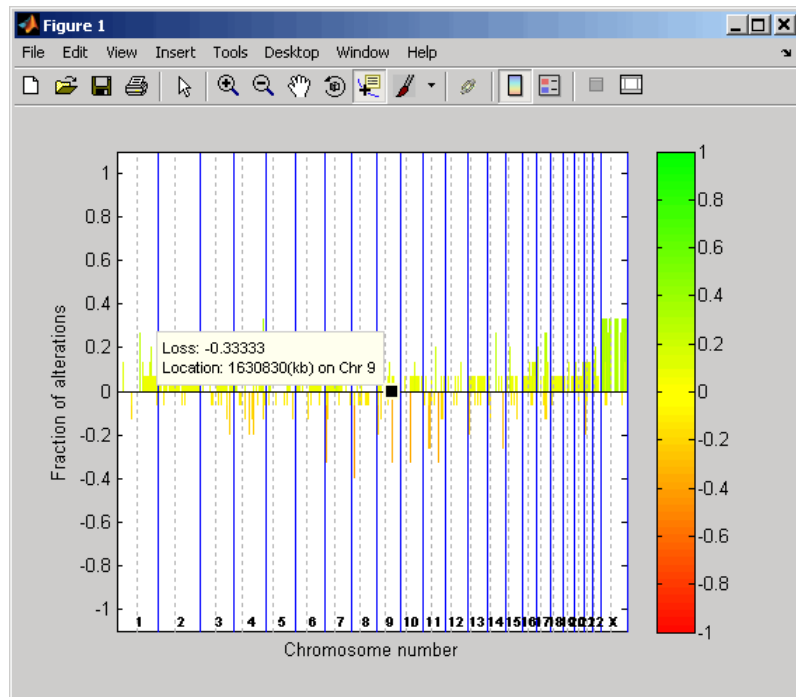
Plotting Data from the Coriell Cell Line Study



- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains `coriell_data`, a structure of array-based CGH data.

```
load coriell_baccgh
```

- 2 Display a frequency plot of the copy number alterations across all samples in the Coriell aCGH data.

```
Struct = cghfreqplot(coriell_data);
```



- 3 View data tips for the data, chromosomes, and centromeres by clicking the Data Cursor  button on the toolbar, then clicking data, a blue chromosome boundary line, or a dotted centromere line in the plot. To delete this data tip, right-click it, then select **Delete Current Datatip**.
- 4 Display a color bar indicating the degree of gain or loss by clicking the Insert Colorbar  button on the toolbar.

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

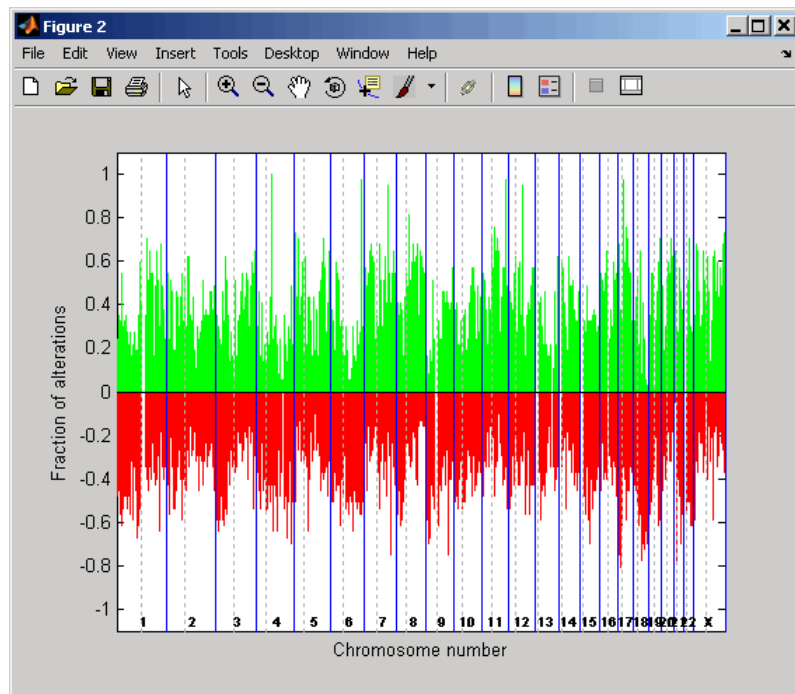
Plotting Pancreatic Cancer Study Data Using a Green and Red Color Scheme

- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains `pancrea_data`, a structure of array-based CGH data from a pancreatic cancer study.

```
load pancrea_oligocgh
```

- 2 Display a frequency plot of the copy number alterations across all samples in the pancreatic cancer data, using a green and red color scheme.

```
cghfreqplot(pancrea_data, 'Color', [0 1 0; 1 0 0])
```



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

Plotting Groups of aCGH Data, Specifying a Frequency Value Cutoff, and Adding a Chromosome Ideogram

- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains `pancrea_data`, a structure of array-based CGH data from a pancreatic cancer study.

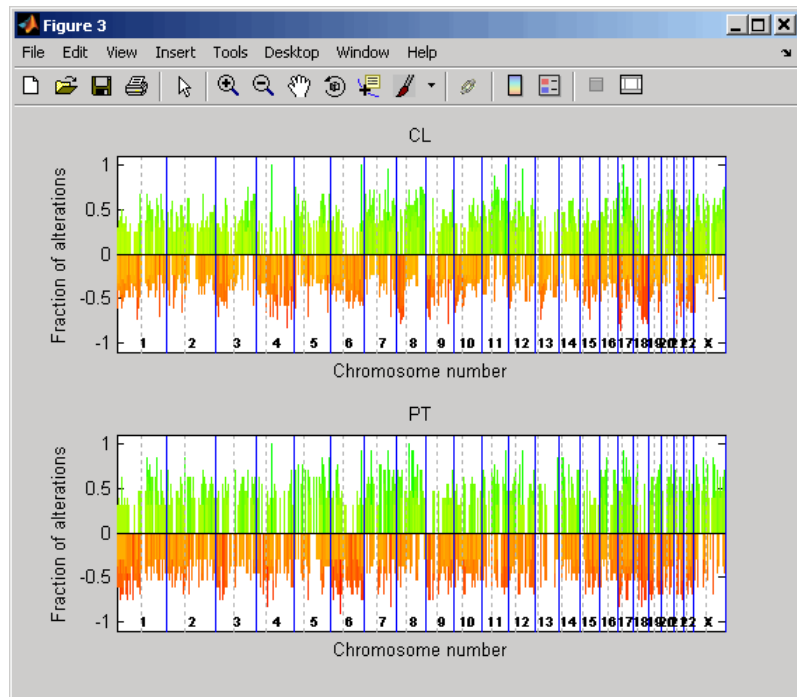
```
load pancrea_oligocgh
```

- 2 Define two groups of data.

```
grp1=strmatch('PA.C', pancrea_data.Sample);  
grp2=strmatch('PA.T', pancrea_data.Sample);
```

- 3 Display a frequency plot of the copy number alterations across all samples in the two groups and limit the plotting to only the clones with frequency gains or losses greater than or equal to 0.25.

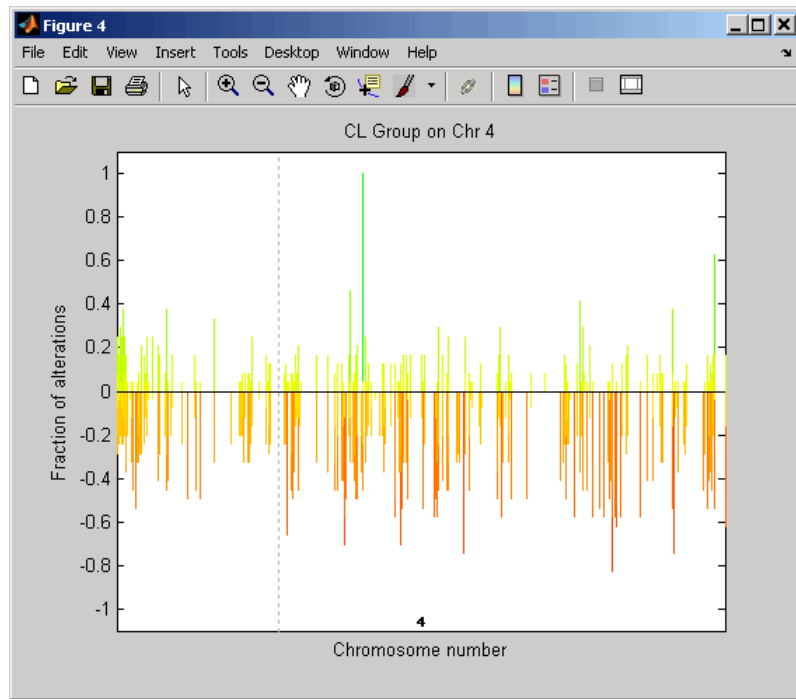
```
SP = cghfreqplot(pancrea_data, 'Group', {grp1, grp2},...  
                'Title', {'CL', 'PT'}, 'Cutoff', 0.25);
```



- 4 Display a frequency plot of the copy number alterations across all samples in the first group and limit the plot to chromosome 4 only.

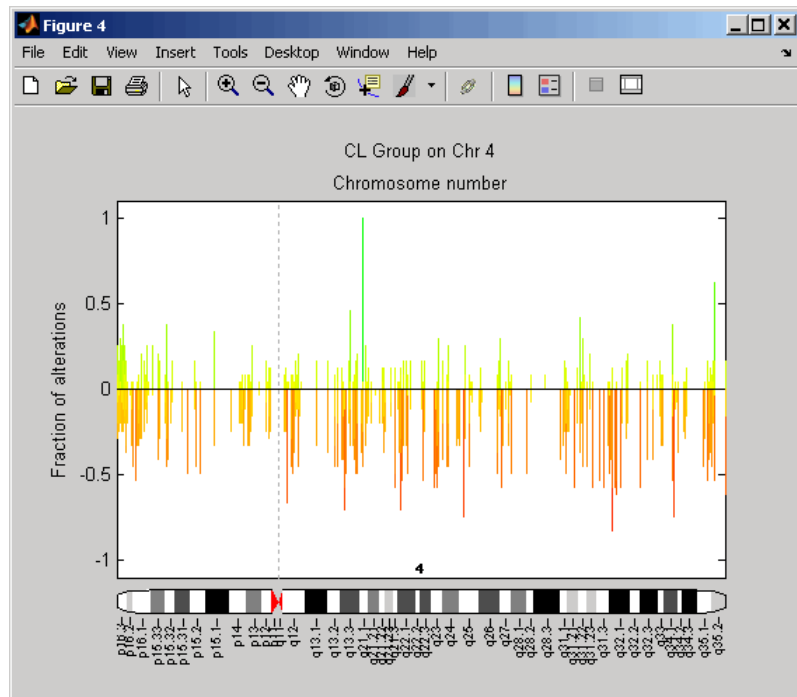
```
SP = cghfreqplot(pancrea_data, 'Group', grp1, ...  
                'Title', 'CL Group on Chr 4', 'Chromosome', 4);
```

cghfreqplot



- 5 Use the `chromosomeplot` function with the `'addtoplot'` property to add the ideogram of chromosome 4 for *Homo sapiens* to this frequency plot. Because the plot of the frequency data from the pancreatic cancer study is in kb units, use the `'Unit'` property to convert the ideogram data to kb units.

```
chromosomeplot('hs_cytoBand.txt', 4, 'addtoplot', gca, 'unit', 2);
```

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

References

- [1] Snijders, A.M., Nowak, N., Segreaves, R., Blackwood, S., Brown, N., Conroy, J., Hamilton, G., Hindle, A.K., Huey, B., Kimura, K., Law, S., Myambo, K., Palmer, J., Ylstra, B., Yue, J.P., Gray, J.W., Jain, A.N., Pinkel, D., and Albertson, D.G. (2001). Assembly of microarrays for genome-wide measurement of DNA copy number. *Nature Genetics* 29, 263–264.
- [2] Aguirre, A.J., Brennan, C., Bailey, G., Sinha, R., Feng, B., Leo, C., Zhang, Y., Zhang, J., Gans, J.D., Bardeesy, N., Cauwels, C., Cordon-Cardo, C., Redston, M.S., DePinho, R.A., and Chin, L. (2004).

cghfreqplot

High-resolution characterization of the pancreatic adenocarcinoma genome. *PNAS* *101*, *24*, 9067–9072.

See Also

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

Purpose

Plot chromosome ideogram with G-banding pattern

Syntax

```
chromosomeplot(CytoData)  
chromosomeplot(CytoData, ChromNum)  
chromosomeplot(CytoData, ChromNum, ..., 'Orientation',  
OrientationValue, ...)  
chromosomeplot(CytoData, ChromNum, ..., 'ShowBandLabel',  
ShowBandLabelValue, ...)  
chromosomeplot(CytoData, ChromNum, ..., 'AddToPlot',  
AddToPlotValue, ...)  
chromosomeplot(..., 'Unit', UnitValue, ...)  
chromosomeplot(..., 'CNV', CNVValue, ...)
```

chromosomeplot

Arguments

CytoData

Either of the following:

- String specifying a file containing cytogenetic G-banding data (in bp units), such as an NCBI ideogram text file or a UCSC Genome Browser cytoband text file.
- Structure containing cytogenetic G-banding data (in bp units) in the following fields:
 - ChromLabels
 - BandStartBPs
 - BandEndBPs
 - BandLabels
 - GieStains

Tip Use the `cytobandread` function to create the structure to use for *CytoData*.

ChromNum

Scalar or string specifying a single chromosome to plot. Valid entries are integers, 'X', and 'Y'.

Note Setting *ChromNum* to 0 will plot ideograms for all chromosomes.

OrientationValue

String or number that specifies the orientation of the ideogram of a single chromosome specified by *ChromNum*. Choices are 'Vertical' or 1 (default) and 'Horizontal' or 2.

<i>ShowBandLabelValue</i>	Controls the display of band labels (such as q25.3) when plotting a single chromosome ideogram, specified by <i>ChromNum</i> . Choices are true (default) or false.
<i>AddToPlotValue</i>	Variable name of a figure axis to which to add the single chromosome ideogram, specified by <i>ChromNum</i> .

Note If you use this property to add the ideogram to a plot of genomic data that is in units other than bp, use the 'Unit' property to convert the ideogram data to the appropriate units.

Tip Before printing a figure containing an added chromosome ideogram, change the background to white by issuing the following command:

```
set(gcf, 'color', 'w')
```

chromosomeplot

<i>UnitValue</i>	Integer that specifies the units (base pairs, kilo base pairs, or mega base pairs) for the starting and ending genomic positions. This unit is used in the data tip displayed when you hover the cursor over chromosomes in the ideogram. This unit can also be used when using the 'AddToPlot' property to add the ideogram to a plot that is in units other than bp. Choices are 1 (bp), 2 (kb), or 3 (mb). Default is 1 (bp).
<i>CNVValue</i>	Controls the display of copy number variance (CNV) data, provided by <i>CNVValue</i> , aligned to the chromosome ideogram. Gains are shown in green to the right or above the ideogram, while losses are shown in red to the left or below the ideogram. <i>CNVValue</i> is a structure array containing the four fields described in the table below.

Description

chromosomeplot(*CytoData*) plots the ideogram of all chromosomes, using information from *CytoData*, a structure containing cytogenetic G-banding data (in bp units), or a string specifying a file containing cytogenetic G-banding data (in bp units), such as an NCBI ideogram text file or a UCSC Genome Browser cytoband text file. The G bands distinguish different areas of the chromosome. For example, for the *Homo sapiens* ideogram, possible G bands are:

- gneg — white
- gpos25 — light gray
- gpos50 — medium gray
- gpos75 — dark gray
- gpos100 — black
- acen — red (centromere)

- stalk — light blue (regions with repeats)
- gvar — indented region

Darker bands are AT-rich, while lighter bands are GC-rich.

`chromosomeplot(CytoData, ChromNum)` plots the ideogram of a single chromosome specified by *ChromNum*.

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

`chromosomeplot(CytoData, ChromNum, ..., 'Orientation', OrientationValue, ...)` specifies the orientation of the ideogram of a single chromosome specified by *ChromNum*. Choices are 'Vertical' or 1 (default) and 'Horizontal' or 2.

Note When plotting the ideogram of all chromosomes, the orientation is always vertical.

`chromosomeplot(CytoData, ChromNum, ..., 'ShowBandLabel', ShowBandLabelValue, ...)` displays band labels (such as q25.3) when plotting a single chromosome ideogram, specified by *ChromNum*. Choices are true (default) or false.

`chromosomeplot(CytoData, ChromNum, ..., 'AddToPlot', AddToPlotValue, ...)` adds the single chromosome ideogram, specified by *ChromNum*, to a figure axis specified by *AddToPlotValue*.

chromosomeplot

Note If you use this property to add the ideogram to a plot of genomic data that is in units other than bp, use the 'Unit' property to convert the ideogram data to the appropriate units.

Tip Before printing a figure containing an added chromosome ideogram, change the background to white by issuing the following command:

```
set(gcf, 'color', 'w')
```

`chromosomeplot(..., 'Unit', UnitValue, ...)` specifies the units (base pairs, kilo base pairs, or mega base pairs) for the starting and ending genomic positions. This unit is used in the data tip displayed when you hover the cursor over chromosomes in the ideogram. This unit can also be used when using the 'AddToPlot' property to add the ideogram to a plot that is in units other than bp. Choices are 1 (bp), 2 (kb), or 3 (mb). Default is 1 (bp).

`chromosomeplot(..., 'CNV', CNVValue, ...)` displays copy number variance (CNV) data, provided by *CNVValue*, aligned to the chromosome ideogram. Gains are shown in green to the right or above the ideogram, while losses are shown in red to the left or below the ideogram. *CNVValue* is a structure array containing the following fields. Each field must contain the same number of elements.

Field	Description
Chromosome	<p>Either of the following:</p> <ul style="list-style-type: none"> Numeric vector containing the chromosome number on which each CNV is located. <hr/> <p>Note For the sex chromosome, X, use N, where N = number of autosomes + 1. For the sex chromosome, Y, use M, where M = number of autosomes + 2. For example, for <i>Homo sapiens</i> use 23 for X and 24 for Y, and for <i>Mus musculus</i> (lab mouse), use 20 for X and 21 for Y.</p> <hr/> <ul style="list-style-type: none"> Character array containing the chromosome number on which each CNV is located. <hr/> <p>Note Using a character array lets you use the characters X and Y (instead of numbers) for sex chromosomes. However, all elements in the array must be the same width, which may require you to add spaces to the strings. For example:</p> <pre>[' 1'; ' 2'; '10'; ' X']</pre> <p>Or you can use the <code>char</code> function with a cell array to create a character array of the chromosome numbers and letters. For example: .</p> <pre>char({'1', '2', '10', 'X'})</pre> <hr/>

chromosomeplot

Field	Description
CNVType	Numeric vector containing the type of each CNV, either 1 (loss) or 2 (gain).
Start	Numeric vector containing the starting genomic position of each CNV. Units must be in base pairs.
End	Numeric vector containing the ending genomic position of each CNV. Units must be in base pairs.

Examples

Plotting Chromosome Ideograms

- 1 Read the cytogenetic banding information for *Homo sapiens* into a structure.

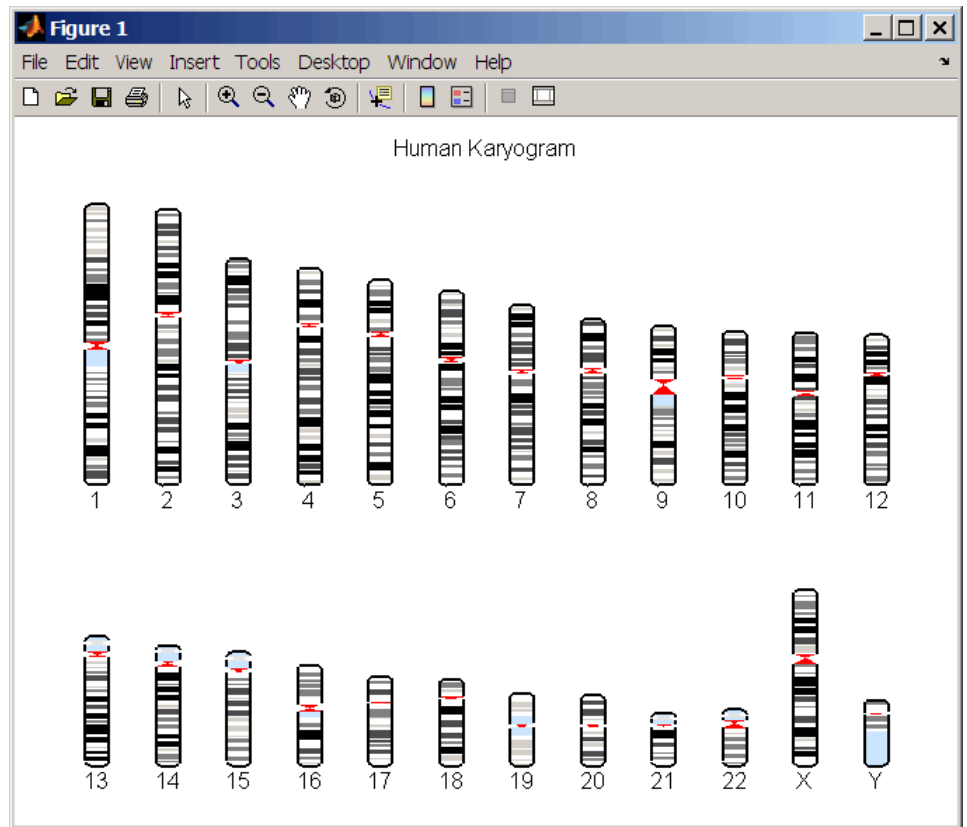
```
hs_cytobands = cytobandread('hs_cytoBand.txt')
```

```
hs_cytobands =
```

```
ChromLabels: {862x1 cell}
BandStartBPs: [862x1 int32]
BandEndBPs: [862x1 int32]
BandLabels: {862x1 cell}
GieStains: {862x1 cell}
```

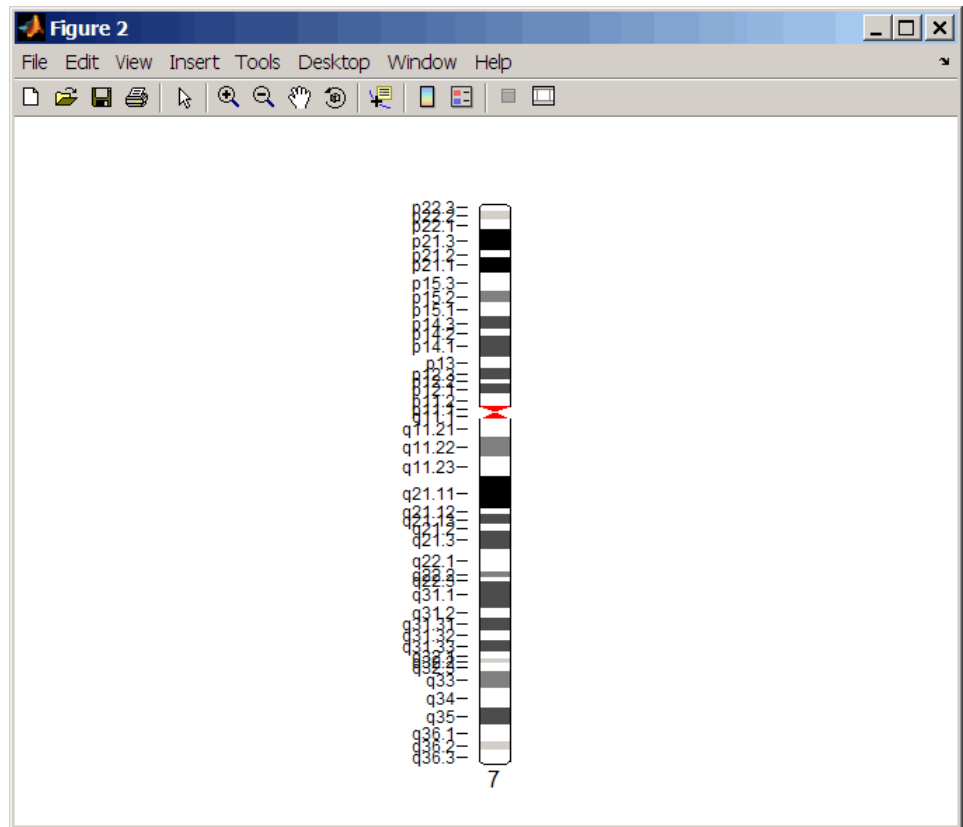
- 2 Plot the entire chromosome ideogram for *Homo sapiens*.

```
chromosomeplot(hs_cytobands);
title('Human Karyogram')
```



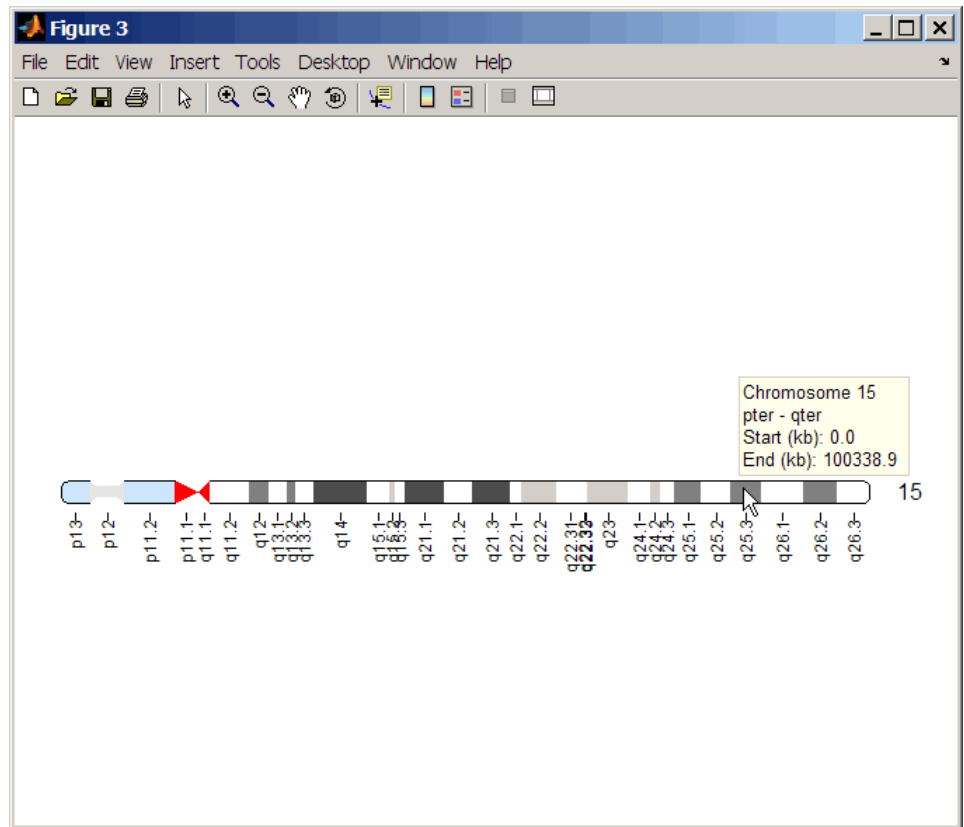
- 3 Display the ideogram of only chromosome 7 for *Homo sapiens* by right-clicking chromosome 7 in the plot, then selecting **Display in New Figure > Vertical**.


chromosomeplot



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

```
chromosomeplot(hs_cytobands, 15, 'Orientation', 2, 'Unit', 2);
```



- 5 View a data tip with information about the chromosome by hovering the cursor over the chromosome. View a data tip with detailed information about a specific band by clicking the Data Cursor  button on the toolbar, then clicking the band in the plot. To delete this data tip, right-click it, then select **Delete Current Datatip**.

Tip You can change the orientation of a single chromosome ideogram by right-clicking, selecting **Display > Vertical** or **Horizontal**. You can show or hide the band labels of a single chromosome ideogram by right-clicking, then selecting **Show G-band Labels** or **Hide G-band Labels**.

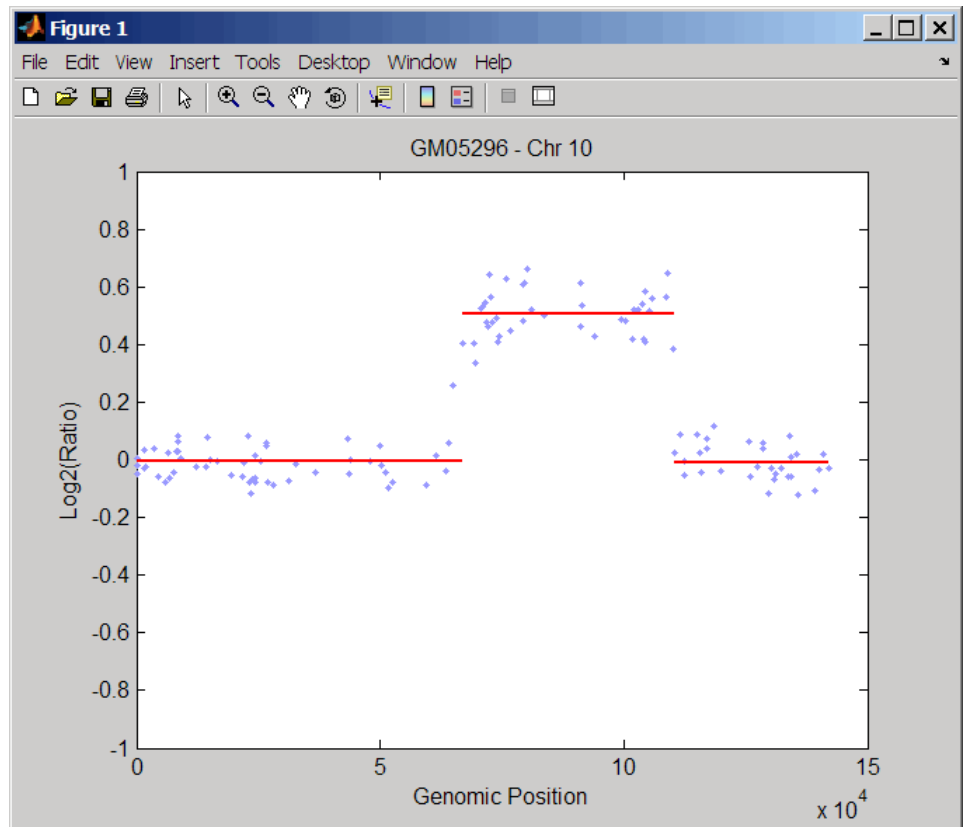
Adding a Chromosome Ideogram to a Plot

- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains `coriell_data`, a structure of CGH data.

```
load coriell_baccgh
```

- 2 Use the `cghcbs` function to analyze chromosome 10 of sample 3 (GM05296) of the CGH data and return copy number variance (CNV) data in a structure, `S`. Plot the segment means over the original data for only chromosome 10 of sample 3.

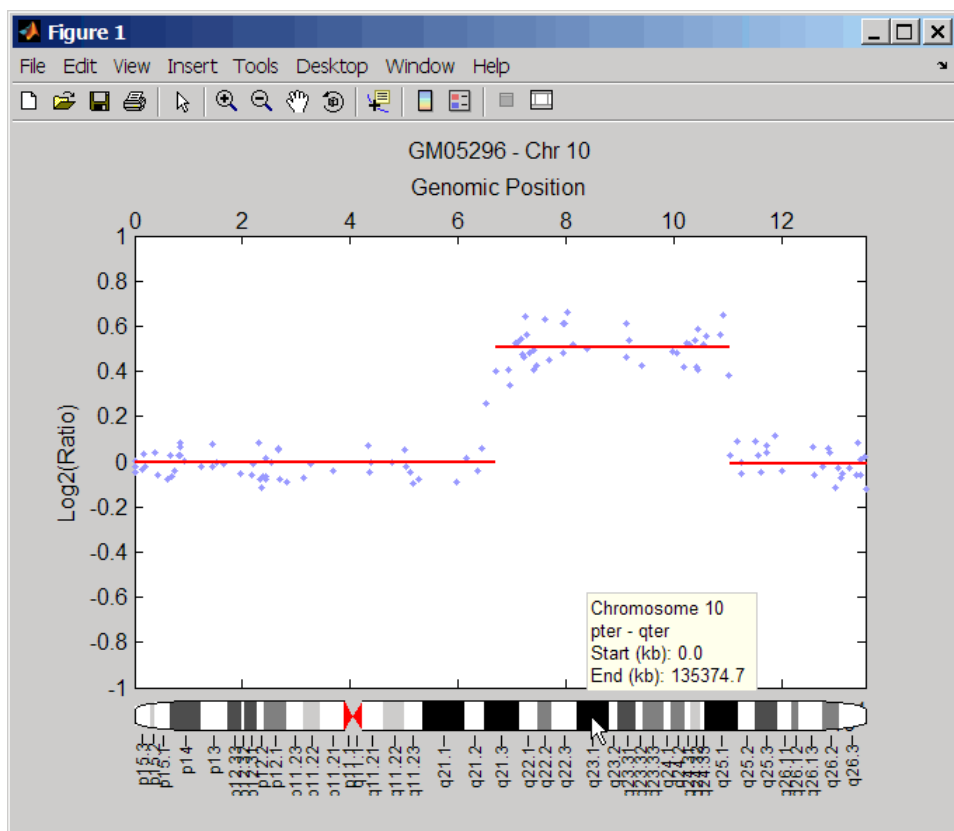
```
S = cghcbs(coriell_data,'sampleindex',3,'chromosome',10,...  
          'showplot',10);
```



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

```
chromosomeplot('hs_cytoBand.txt', 10, 'addtoplot', gca,...  
              'Unit', 2)
```

chromosomeplot



Tip Before printing the above figure containing an added chromosome ideogram, change the background to white by issuing the following command:

```
set(gcf, 'color', 'w')
```


Displaying Copy Number Alteration Regions Aligned to a Chromosome Ideogram

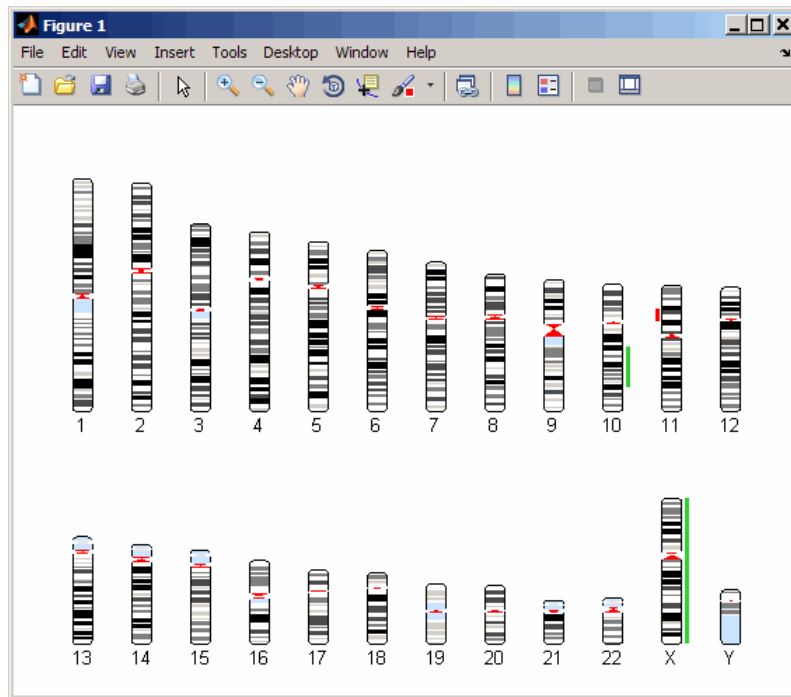
- 1 Create a structure containing segment gain and loss information for chromosomes 10, 11, and X from sample 3 from the Coriell cell line study, making sure the segment data is in bp units. (You can determine copy number variance (CNV) information by exploring `S`, the structure of segments returned by the `cgHcbs` function in step 2 in Adding a Chromosome Ideogram to a Plot on page 2-214.) For the `'CNVType'` field, use 1 to indicate a loss and 2 to indicate a gain.

```
cnvStruct = struct('Chromosome', char({'10', '11', 'X'}),...  
                  'CNVType', [2 1 2],...  
                  'Start', [66905000 25416000 1],...  
                  'End', [110412000 39389000 154913755]);
```

- 2 Pass the structure to the `chromosomeplot` function using the `'CNV'` property to display the copy number gains (green) and losses (red) aligned to the human chromosome ideogram.

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

chromosomeplot



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

References

[1] Snijders, A.M., Nowak, N., Segaves, R., Blackwood, S., Brown, N., Conroy, J., Hamilton, G., Hindle, A.K., Huey, B., Kimura, K., Law, S., Myambo, K., Palmer, J., Ylstra, B., Yue, J.P., Gray, J.W., Jain, A.N., Pinkel, D., and Albertson, D.G. (2001). Assembly of microarrays for genome-wide measurement of DNA copy number. *Nature Genetics* 29, 263–264.

See Also

Bioinformatics Toolbox functions: `cghcbs`, `cytobandread`

Purpose	Evaluate performance of classifier				
Syntax	<pre>classperf CP = classperf(<i>truelabels</i>) CP = classperf(<i>truelabels</i>, <i>classout</i>) CP = classperf(..., 'Positive', <i>PositiveValue</i>, 'Negative', <i>NegativeValue</i>) classperf(<i>CP</i>, <i>classout</i>) classperf(<i>CP</i>, <i>classout</i>, <i>testidx</i>)</pre>				
Arguments	<table><tr><td><i>truelabels</i></td><td>True class labels for each observation, specified by one of the following:<ul style="list-style-type: none">• Numeric vector• Cell array of strings<hr/>Note When used in a cross-validation design experiment, <i>truelabels</i> should have the same size as the total number of observations.<hr/></td></tr><tr><td><i>classout</i></td><td>Classifier output, specified by one of the following:<ul style="list-style-type: none">• Numeric vector• Cell array of strings<hr/>Note <i>classout</i> must contain the same number of elements as <i>truelabels</i>.<hr/></td></tr></table>	<i>truelabels</i>	True class labels for each observation, specified by one of the following: <ul style="list-style-type: none">• Numeric vector• Cell array of strings <hr/> Note When used in a cross-validation design experiment, <i>truelabels</i> should have the same size as the total number of observations. <hr/>	<i>classout</i>	Classifier output, specified by one of the following: <ul style="list-style-type: none">• Numeric vector• Cell array of strings <hr/> Note <i>classout</i> must contain the same number of elements as <i>truelabels</i> . <hr/>
<i>truelabels</i>	True class labels for each observation, specified by one of the following: <ul style="list-style-type: none">• Numeric vector• Cell array of strings <hr/> Note When used in a cross-validation design experiment, <i>truelabels</i> should have the same size as the total number of observations. <hr/>				
<i>classout</i>	Classifier output, specified by one of the following: <ul style="list-style-type: none">• Numeric vector• Cell array of strings <hr/> Note <i>classout</i> must contain the same number of elements as <i>truelabels</i> . <hr/>				

classperf

- PositiveValue* Numeric vector or cell array of strings that specifies the positive labels to identify the target class(es). Default is the first class returned by `grp2idx(trueLabels)`.
- NegativeValue* Numeric vector or cell array of strings that specifies the negative labels to identify the control class(es). Default is all classes other than the first class returned by `grp2idx(trueLabels)`.
- testidx* Vector that indicates the observations that were used in the current validation. Choices are:
- Index vector
 - Logical index vector of the same size as *trueLabels* used to construct the classifier performance object

Return Values

CP Classifier performance object with performance properties listed in the following table.

Description

`classperf` provides an interface to keep track of the performance during the validation of classifiers. `classperf` creates and, optionally, updates a classifier performance object, *CP*, which accumulates the results of the classifier. The performance properties of a classifier performance object are listed in the following table.

`classperf`, without input arguments, displays all the performance properties of a classifier performance object.

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

`CP = classperf(trueLabels, classOut)` creates `CP` using `trueLabels`, then updates `CP` using the classifier output, `classOut`.

Tip This syntax is useful when you want to know the performance of a single validation.

`CP = classperf(..., 'Positive', PositiveValue, 'Negative', NegativeValue)` specifies the positive and negative labels to identify the target and the control classes, respectively. These labels are used to compute clinical diagnostic test performance.

If `trueLabels` is a numeric vector, `PositiveValue` and `NegativeValue` must be numeric vectors whose entries are subsets of `grp2idx(trueLabels)`. If `trueLabels` is a cell array of strings, `PositiveValue` and `NegativeValue` can be cell arrays of strings or numeric vectors whose entries are subsets of `grp2idx(trueLabels)`. `PositiveValue` defaults to the first class returned by `grp2idx(trueLabels)`, while `NegativeValue` defaults to all other classes.

`PositiveValue` and `NegativeValue` must consist of disjoint sets of the labels used in `trueLabels`. For example, if

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

you could set

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

For example, if you have a data set with data from six samples: five different types of cancer (ovarian, lung, prostate, skin, brain) and no cancer, then `ClassLabels = {'Ovarian', 'Lung', 'Prostate', 'Skin', 'Brain', 'Healthy'}`.

You could test a detector for lung cancer by using a `PositiveValue` of 2, and a `NegativeValue = [1 3 4 5 6]`.

Or you can test for any type of cancer by using *PositiveValue* = [1 2 3 4 5] and a *NegativeValue* of 6.

In clinical tests, inconclusive values such as '' or NaN are counted as false negatives for the computation of the specificity, and as false positives for the computation of the sensitivity. That is, inconclusive results may decrease the diagnostic value of the test. Tested observations for which *trueLabels* is not within the union of *PositiveValue* and *NegativeValue* are not considered. However, tested observations that result in a class not covered by the vector *trueLabels* are counted as inconclusive.

`classperf(CP, classout)` updates *CP*, the classifier performance object, with the classifier output *classout*. *classout* must be the same size as *trueLabels*, the vector or cell array used to construct the classifier performance object. When *classout* is a cell array of strings, an empty string, '', represents an inconclusive result of the classifier. For numeric arrays, NaN represents an inconclusive result.

`classperf(CP, classout, testidx)` updates *CP*, the classifier performance object, with the classifier output *classout*. *classout* has a smaller size than *trueLabels*. *testidx* is an index vector or a logical index vector of the same size as *trueLabels*, the vector or cell array used to construct the classifier performance object. *testidx* indicates the observations that were used in the current validation.

Note In the two previous syntaxes, you do not need to create a separate output variable to update the classifier performance object, *CP*.

Properties of a Classifier Performance Object

You can access classifier performance object properties by using the `get` function

```
get(CP, 'ControlClasses')
```

or using dot notation

```
CP.ControlClasses
```

You cannot directly modify the classifier performance object properties by using the `set` function, with the exception of the `Label` and `Description` properties.

Tip To modify properties, use either of the following syntaxes:

```
classperf(CP, classout)
classperf(CP, classout, testidx)
```

Property	Description
Label	String to label the classifier performance object. Default is ''.
Description	String to describe the classifier performance object. Default is ''.
ClassLabels	Numeric vector or cell array of strings specifying a unique set of class labels from <code>unique(trueLabels)</code> .
GroundTruth	Numeric vector or cell array of strings that specifies the true class labels for each observation. The number of elements = <code>NumberOfObservations</code> .
NumberOfObservations	Positive integer specifying the number of observations in the study.

Property	Description
ControlClasses	<p data-bbox="810 319 1267 440">Indices to the <code>ClassLabels</code> vector or cell array, indicating which classes to be considered as the control or negative classes in a diagnostic test.</p> <hr data-bbox="810 499 1285 503"/> <p data-bbox="810 510 1285 736">Tip You set the <code>ControlClasses</code> property with the 'Negative' property name/value pair. If you do not specify the 'Negative' property, <code>ControlClasses</code> defaults to all classes other than the first class returned by <code>grp2idx(trueLabels)</code>.</p> <hr data-bbox="810 739 1285 743"/>
TargetClasses	<p data-bbox="810 784 1282 906">Indices to the <code>ClassLabels</code> vector or cell array, indicating which classes to be considered as the target or positive classes in a diagnostic test.</p> <hr data-bbox="810 965 1285 968"/> <p data-bbox="810 972 1237 1197">Tip You set the <code>TargetClasses</code> property with the 'Positive' property name/value pair. If you do not specify the 'Positive' property, <code>TargetClasses</code> defaults to the first class returned by <code>grp2idx(trueLabels)</code>.</p> <hr data-bbox="810 1201 1285 1204"/>
ValidationCounter	<p data-bbox="810 1249 1285 1305">Positive integer specifying the number of validations performed.</p>

Property	Description
SampleDistribution	<p>Numeric vector indicating how many times each sample was considered in the validation.</p> <p>For example, if you use resubstitution, SampleDistribution is a vector of ones and ValidationCounter = 1. If you have a ten-fold cross-validation, SampleDistribution is also a vector of ones, but ValidationCounter = 10.</p> <hr/> <p>Tip SampleDistribution is more useful when doing Monte Carlo partitions of the test sets, as this will help determine if all the samples are being equally tested.</p> <hr/>
ErrorDistribution	Numeric vector indicating how many times each sample was misclassified.
SampleDistributionByClass	Numeric vector indicating the frequency of the true classes in the validation.
ErrorDistributionByClass	Numeric vector indicating the frequency of errors for each class in the validation.

Property	Description
CountingMatrix	<p>The classification confusion matrix. The order of rows and columns is the same as <code>grp2idx(trueLabels)</code>. Columns represent the true classes, and rows represent the classifier prediction. The last row in <code>CountingMatrix</code> is reserved to count inconclusive results. There are some families of classifiers that can reserve the right to make a hard class assignment; this can be based on metrics, such as the posterior probabilities, or on how close a sample is to the class boundaries.</p>
CorrectRate	<p>Correctly Classified Samples / Classified Samples</p> <hr/> <p>Note Inconclusive results are not counted.</p> <hr/>
ErrorRate	<p>Incorrectly Classified Samples / Classified Samples</p> <hr/> <p>Note Inconclusive results are not counted.</p> <hr/>

Property	Description
LastCorrectRate	<p>The following equation applies only to samples considered the last time the classifier performance object was updated:</p> $\frac{\text{Correctly Classified Samples}}{\text{Classified Samples}}$
LastErrorRate	<p>The following equation applies only to samples considered the last time the classifier performance object was updated:</p> $\frac{\text{Incorrectly Classified Samples}}{\text{Classified Samples}}$
InconclusiveRate	$\frac{\text{Nonclassified Samples}}{\text{Total Number of Samples}}$
ClassifiedRate	$\frac{\text{Classified Samples}}{\text{Total Number of Samples}}$
Sensitivity	<p>Correctly Classified Positive Samples / True Positive Samples</p> <hr/> <p>Note Inconclusive results that are true positives are counted as errors for computing Sensitivity (following a conservative approach). This is the same as being incorrectly classified as negatives.</p> <hr/>

Property	Description
Specificity	<p>Correctly Classified Negative Samples / True Negative Samples</p> <hr/> <p>Note Inconclusive results that are true negatives are counted as errors for computing Specificity (following a conservative approach). This is the same as being incorrectly classified as positives.</p> <hr/>
PositivePredictiveValue	<p>Correctly Classified Positive Samples / Positive Classified Samples</p> <hr/> <p>Note Inconclusive results are classified as negatives when computing PositivePredictiveValue.</p> <hr/>
NegativePredictiveValue	<p>Correctly Classified Negative Samples / Negative Classified Samples</p> <hr/> <p>Note Inconclusive results are classified as positives when computing NegativePredictiveValue.</p> <hr/>
PositiveLikelihood	$Sensitivity / (1 - Specificity)$
NegativeLikelihood	$(1 - Sensitivity) / Specificity$

Property	Description
Prevalence	True Positive Samples / Total Number of Samples
DiagnosticTable	<p>A 2-by-2 numeric array with diagnostic counts. The first row indicates the number of samples that were classified as positive, with the number of true positives in the first column, and the number of false positives in the second column. The second row indicates the number of samples that were classified as negative, with the number of false negatives in the first column, and the number of true negatives in the second column.</p> <p>Correct classifications appear in the diagonal elements, and errors appear in the off-diagonal elements. Inconclusive results are considered errors and counted in the off-diagonal elements.</p> <p>For an illustration of a diagnostic table, see below.</p>

Example Diagnostic Table

In a cancer study of ten patients, suppose we get the following results:

Patient	Classifier Output	Has Cancer
1	Positive	Yes
2	Positive	Yes
3	Positive	Yes
4	Positive	No

classperf

Patient	Classifier Output	Has Cancer
5	Negative	Yes
6	Negative	No
7	Negative	No
8	Negative	No
9	Negative	No
10	Inconclusive	Yes

The diagnostic table would look as follows:

		True State	
		1	0
Classifier Output	1	3	1
	0	1+1	4

Examples

```
% Classify the fisheriris data with a K-Nearest Neighbor classifier
load fisheriris
c = knnclassify(meas,meas,species,4,'euclidean','Consensus');
cp = classperf(species,c)
get(cp)

% 10-fold cross-validation on the fisheriris data using linear
% discriminant analysis and the third column as only feature for
% classification
load fisheriris
indices = crossvalind('Kfold',species,10);
cp = classperf(species); % initializes the CP object
for i = 1:10
    test = (indices == i); train = ~test;
```

```

class = classify(meas(test,3),meas(train,3),species(train));
% updates the CP object with the current classification results
classperf(cp,class,test)
end
cp.CorrectRate % queries for the correct classification rate

```

```
cp =
```

```

biolearning.classperformance

        Label: ''
      Description: ''
    ClassLabels: {3x1 cell}
      trueLabels: [150x1 double]
NumberOfObservations: 150
    ControlClasses: [2x1 double]
      TargetClasses: 1
  ValidationCounter: 1
  SampleDistribution: [150x1 double]
   ErrorDistribution: [150x1 double]
SampleDistributionByClass: [3x1 double]
ErrorDistributionByClass: [3x1 double]
   CountingMatrix: [4x3 double]
      CorrectRate: 1
      ErrorRate: 0
InconclusiveRate: 0.0733
   ClassifiedRate: 0.9267
      Sensitivity: 1
      Specificity: 0.8900
PositivePredictiveValue: 0.8197
NegativePredictiveValue: 1
   PositiveLikelihood: 9.0909
   NegativeLikelihood: 0
      Prevalence: 0.3333
   DiagnosticTable: [2x2 double]

```

classperf

```
ans =  
    0.9467
```

See Also

Bioinformatics Toolbox functions: `crossvalind`, `knnclassify`, `svmclassify`

Statistics Toolbox™ functions: `classify`, `grp2idx`

Purpose

Cleave amino acid sequence with enzyme

Syntax

```
Fragments = cleave(SeqAA, Enzyme)
Fragments = cleave(SeqAA, PeptidePattern, Position)
[Fragments, CuttingSites] = cleave(...)
[Fragments, CuttingSites, Lengths] = cleave(...)
cleave(..., 'PartialDigest', PartialDigestValue)
```

Arguments

SeqAA

One of the following:

- String of single-letter codes specifying an amino acid sequence.
- Row vector of integers specifying an amino acid sequence.
- MATLAB structure containing a `Sequence` field that contains an amino acid sequence, such as returned by `fastaread`, `getgenpept`, `genpeptread`, `getpdb`, or `pdbread`.

Examples: 'ARN' or [1 2 3].

Enzyme

String specifying a name or abbreviation code for an enzyme or compound for which a cleavage rule is specified in the literature.

Tip Use the `cleavelookup` function to display the names of enzymes and compounds in the cleavage rule library.

cleave

PeptidePattern Short amino acid sequence to search for in *SeqAA*, a larger sequence. *PeptidePattern* can be any of the following:

- Character string
- Vector of integers
- Regular expression

Position Integer from 0 to the length of the *PeptidePattern*, that specifies a position in the *PeptidePattern* where the sequence is to be cleaved.

Note Position 0 corresponds to the N terminal end of *PeptidePattern*.

PartialDigestValue Value from 0 to 1 (default) that specifies the probability that a cleavage site will be cleaved.

Return Values

<i>Fragments</i>	Cell array of strings representing the fragments from the cleavage.
<i>CuttingSites</i>	Numeric vector with the indices representing the cleavage sites.

Note A 0 (zero) is added to the list, so `numel(CuttingSites)==numel(Fragments)`. You can use `CuttingSites + 1` to point to the first amino acid of every fragment respective to the original sequence.

<i>Lengths</i>	Numeric vector with the lengths of every fragment.
----------------	--

Description

`Fragments = cleave(SeqAA, Enzyme)` cuts `SeqAA`, an amino acid sequence, into parts at the cleavage sites specific for `Enzyme`, a string specifying a name or abbreviation code for an enzyme or compound for which a cleavage rule is specified in the literature. It returns `Fragments`, a cell array of strings representing the fragments from the cleavage.

Tip Use the `cleavelookup` function to display the names of enzymes and compounds in the cleavage rule library.

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

`[Fragments, CuttingSites] = cleave(...)` returns a numeric vector with the indices representing the cleavage sites.

cleave

Note A 0 (zero) is added to the list, so `numel(CuttingSites)==numel(Fragments)`. 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(..., 'PartialDigest', PartialDigestValue)` simulates a partial digestion where *PartialDigestValue* is the probability of a cleavage site being cut. *PartialDigestValue* is a value from 0 to 1 (default).

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

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

Examples

1 Retrieve a protein sequence from the GenPept database.

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

2 Cleave the sequence using proteinase K.

```
[partsPK, sitesPK, lengthsPK] = cleave(S.Sequence, 'proteinase K');
```

3 Display the indices of the cleavage sites, lengths, and sequences of the first 10 fragments.

```

for i=1:10
    fprintf('%5d%5d  %s\n',sitesPK(i),lengthsPK(i),partsPK{i})
end

```

```

0    3  MGT
3    6  GGRRGA
9    1  A
10   1  A
11   1  A
12   2  PL
14   1  L
15   1  V
16   1  A
17   1  V

```

- 4** Cleave the same sequence using one of trypsin's cleavage rules (cleave after K or R when the next residue is not P).

```

[partsT, sitesT, lengthsT] = cleave(S.Sequence, '[KR](?!P)',1);

```

- 5** Display the indices of the cleavage sites, lengths, and sequences of the first 10 fragments.

```

for i=1:10
    fprintf('%5d%5d  %s\n',sitesT(i),lengthsT(i),partsT{i})
end

```

```

0    6  MGTGGR
6    1  R
7   34  GAAAAPLLVAVAALLLGAAGHLYPGEVCPGMDIR
41   5 >NNLTR
46  21  LHELENCVIEGHLQILLMFK
67   7  TRPEDFR
74   6  DLSFPK
80  12  LIMITDYLLLFR
92   8  VYGLESLK
100 10  DLFPNLTVIR

```

cleave

See Also

Bioinformatics Toolbox functions: `cleavelookup`, `rebasecuts`,
`restrict`, `seqshowwords`

MATLAB function: `regexp`

Purpose Find cleavage rule for enzyme or compound

Syntax

```
cleavelookup
cleavelookup('Code', CodeValue)
cleavelookup('Name', NameValue)
```

Arguments

<i>CodeValue</i>	String specifying a code representing an abbreviation code for an enzyme or compound. For valid codes, see the table Cleave Lookup on page 2-239.
<i>NameValue</i>	String specifying an enzyme or compound name. For valid names, see the table Cleave Lookup on page 2-239.

Description cleavelookup displays a table of abbreviation codes, cleavage positions, cleavage patterns, and full names of enzymes and compounds for which cleavage rules are specified by the cleavage rule library. For more information, see the ExPASy PeptideCutter tool.

Cleave Lookup

Code	Position	Pattern	Full Name
ARG-C	1	R	ARG-C proteinase
ASP-N	2	D	ASP-N endopeptidase
BNPS	1	W	BNPS-Skatole
CASP1	1	(?<=[FWYL]\w[HAT])D(?=[^PEDQKR])	Caspase 1
CASP2	1	(?<=DVA)D(?=[^PEDQKR])	Caspase 2
CASP3	1	(?<=DMQ)D(?=[^PEDQKR])	Caspase 3

Cleave Lookup (Continued)

Code	Position	Pattern	Full Name
CASP4	1	(?<=LEV)D(?=[^PEDQKR])	Caspase 4
CASP5	1	(?<=[LW]EH)D	Caspase 5
CASP6	1	(?<=VE[HI])D(?=[^PEDQKR])	Caspase 6
CASP7	1	(?<=DEV)D(?=[^PEDQKR])	Caspase 7
CASP8	1	(?<=[IL]ET)D(?=[^PEDQKR])	Caspase 8
CASP9	1	(?<=LEH)D	Caspase 9
CASP10	1	(?<=IEA)D	Caspase 10
CH-HI	1	([FY](?=[^P])) (W(?=[^MP]))	Chymotrypsin-high specificity
CH-LO	1	([FLY](?=[^P])) (W(?=[^MP])) (M(?=[^PY])) (H(?=[^DMPW]))	Chymotrypsin-low specificity
CLOST	1	R	Clostripain
CNBR	1	M	CNBR
ENTKIN	1	(?<=[DN][DN][DN])K	Enterokinase
FACTXA	1	(?<=[AFGILTVM][DE]G)R	Factor XA
FORMIC	1	D	Formic acid
GLUEND	1	E	Glutamyl endopeptidase
GRANB	1	(?<=IEP)D	Granzyme B
HYDROX	1	N(?=G)	Hydroxylamine
IODOB	1	W	Iodosobenzoic acid
LYSC	1	K	Lysc

Cleave Lookup (Continued)

Code	Position	Pattern	Full Name
NTCB	1	C	NTCB
PEPS	1	((?<=[^HKR][^P])[^R](?=[FLWY][^P])) ((?<=[^HKR][^P])[FLWY](?=\w[^P]))	Pepsin PH = 1.3
PEPS2	1	((?<=[^HKR][^P])[^R](?=[FL][^P])) ((?<=[^HKR][^P])[FL](?=\w[^P]))	Pepsin PH > 2
PROEND	1	(?<=[HKR])P(?=[^P])	Proline endopeptidase
PROTK	1	[AEFILTVWY]	Proteinase K
STAPHP	1	(?<=[^E])E	Staphylococcal peptidase I
THERMO	1	[^DE](?=[AFILMV])	Thermolysin
THROMB	1	((?<=\w\wG)R(?=G)) ((?<=[AFGILTVM][AFGILTVWA]P)R(?=[^DE][^DE]))	Thrombin
TRYPS	1	((?<=\w)[KR](?=[^P])) ((?<=W)K(?=P)) ((?<=M)R(?=P))	Trypsin

`cleavelookup('Code', CodeValue)` displays the cleavage position, cleavage pattern, and full name of the enzyme or compound specified by *CodeValue*, a string specifying an abbreviation code.

`cleavelookup('Name', NameValue)` displays the cleavage position, cleavage pattern, and abbreviation code of the enzyme or compound specified by *NameValue*, a string specifying an enzyme or compound name.

Examples

Using cleavelookup with an Enzyme Name

Display the cleavage position, cleavage pattern, and abbreviation code of the enzyme Caspase 1.

```
cleavelookup('name', 'CASPASE 1')
```

cleavelookup

```
ans =
```

```
1 (?<=[FWYL]\w[HAT])D(?=[^PEDQKR]) CASP1
```

Using cleavelookup with an Abbreviation Code

Display the cleavage position, cleavage pattern, and full name of the enzyme with a abbreviation code of CASP1.

```
cleavelookup('code', 'CASP1')
```

```
ans =
```

```
1 (?<=[FWYL]\w[HAT])D(?=[^PEDQKR]) CASPASE 1
```

See Also

Bioinformatics Toolbox functions: `cleave`, `rebasecuts`, `restrict`

Purpose Compute hierarchical clustering, display dendrogram and heat map, and create clustergram object

Syntax

```
CGobj = clustergram(Data)
CGobj = clustergram(Data, ...'RowLabels',
RowLabelsValue, ...)
CGobj = clustergram(Data, ...'ColumnLabels',
ColumnLabelsValue, ...)
CGobj = clustergram(Data, ...'Standardize',
StandardizeValue,
...)
CGobj = clustergram(Data, ...'Cluster', ClusterValue, ...)
CGobj = clustergram(Data, ...'RowPdist',
RowPdistValue, ...)
CGobj = clustergram(Data, ...'ColumnPdist',
ColumnPdistValue,
...)
CGobj = clustergram(Data, ...'Linkage', LinkageValue, ...)
CGobj = clustergram(Data, ...'Dendrogram', DendrogramValue,
...)
CGobj = clustergram(Data, ...'OptimalLeafOrder',
OptimalLeafOrderValue, ...)
CGobj = clustergram(Data, ...'ColorMap',
ColorMapValue, ...)
CGobj = clustergram(Data, ...'DisplayRange',
DisplayRangeValue, ...)
CGobj = clustergram(Data, ...'SymmetricRange',
SymmetricRangeValue, ...)
CGobj = clustergram(Data, ...'LogTrans',
LogTransValue, ...)
CGobj = clustergram(Data, ...'Ratio', RatioValue, ...)
CGobj = clustergram(Data, ...'Impute', ImputeValue, ...)
CGobj = clustergram(Data, ...'RowMarker', RowMarkerValue,
...)
CGobj = clustergram(Data, ...'ColumnMarker',
ColumnMarkerValue, ...)
```

clustergram

Arguments

<i>Data</i>	DataMatrix object or numeric matrix of data. If the matrix contains gene expression data, typically each row corresponds to a gene and each column corresponds to sample.
<i>RowLabelsValue</i>	Vector of numbers or cell array of text strings to label the rows in the dendrogram and heat map. Default is a vector of values 1 through M , where M is the number of rows in <i>Data</i> .
<i>ColumnLabelsValue</i>	Vector of numbers or cell array of text strings to label the columns in the dendrogram and heat map. Default is a vector of values 1 through N , where N is the number of columns in <i>Data</i> .
<i>StandardizeValue</i>	Numeric value that specifies the dimension for standardizing the values in <i>Data</i> . The standardized values are transformed so that the mean is 0 and the standard deviation is 1 in the specified dimension. Choices are: <ul style="list-style-type: none">• 1 — Standardize along the columns of data.• 2 (default) — Standardize along the rows of data.• 3 — Do not perform standardization.

ClusterValue

Numeric value that specifies the dimension for clustering the values in *Data*. Choices are:

- 1 — Cluster rows of data only.
- 2 — Cluster columns of data only.
- 3 (default) — Cluster rows of data, then cluster columns of row-clustered data.

RowPdistValue

String that specifies the distance metric to pass to the `pdist` function (Statistics Toolbox software) to use to calculate the pairwise distances between rows. For information on choices, see the `pdist` function. Default is 'euclidean'.

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

ColumnPdistValue

String that specifies the distance metric to pass to the `pdist` function (Statistics Toolbox software) to use to calculate the pairwise distances between columns. For information on choices, see the `pdist` function. Default is 'euclidean'.

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

LinkageValue

String or two-element cell array of strings that specifies the linkage method to pass to the `linkage` function (Statistics Toolbox software) to use to create the hierarchical cluster tree for rows and columns. If a two-element cell array of strings, the first element is used for linkage between rows, and the second element is used for linkage between columns. For information on choices, see the `linkage` function. Default is 'average'.

Tip To specify the linkage method for only one dimension, set the other dimension to ''.

DendrogramValue

Scalar or two-element numeric vector or cell array of strings that specifies the 'colorthreshold' property to pass to the `dendrogram` function (Statistics Toolbox software) to create the dendrogram plot. If a two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns. For more information, see the `dendrogram` function.

Tip To specify the 'colorthreshold' property for only one dimension, set the other dimension to ''.

OptimalLeafOrderValue

Property to enable or disable the optimal leaf ordering calculation, which determines the leaf order that maximizes the similarity between neighboring leaves. Choices are `true` (enable) or `false` (disable). Default depends on the size of *Data*. If the number of rows or columns in *Data* is greater than 1000, default is `false`; otherwise, default is `true`.

Note Disabling the optimal leaf ordering calculation can be useful when working with large data sets because this calculation uses a large amount of memory and can be very time consuming.

<i>ColorMapValue</i>	<p>Either of the following:</p> <ul style="list-style-type: none">• M-by-3 matrix of RGB values• Name of or handle to a function that returns a colormap, such as <code>redgreencmap</code> or <code>redbluecmap</code> <p>Default is <code>redgreencmap</code>, in which red represents values above the mean, black represents the mean, and green represents values below the mean of a row (gene) across all columns (samples).</p>
<i>DisplayRangeValue</i>	<p>Positive scalar that specifies the display range of standardized values. Default is 3, which means there is a color variation for values between -3 and 3, but values >3 will be the same color as 3, and values <-3 will be the same color as -3.</p> <p>For example, if you specify <code>redgreencmap</code> for the 'ColorMap' property, pure red represents values \geq <i>DisplayRangeValue</i>, and pure green represents values \leq $-$<i>DisplayRangeValue</i>.</p>
<i>SymmetricRangeValue</i>	<p>Property to force the color scale of the heat map to be symmetric around zero. Choices are <code>true</code> (default) or <code>false</code>.</p>
<i>LogTransValue</i>	<p>Controls the \log_2 transform of <i>Data</i> from natural scale. Choices are <code>true</code> or <code>false</code> (default).</p>

RatioValue

Either of the following:

- Scalar
- Two-element vector

It specifies the ratio of space that the row and column dendrograms occupy relative to the heat map. If *RatioValue* is a scalar, it is used as the ratio for both dendrograms. If *RatioValue* is a two-element vector, the first element is used for the ratio of the row dendrogram width to the heat map width, and the second element is used for the ratio of the column dendrogram height to the heat map height. The second element is ignored for one-dimensional clustergrams. Default is 1/5.

ImputeValue

Any of the following:

- Name of a function that imputes missing data
- Handle to a function that imputes missing data
- Cell array where the first element is the name of or handle to a function that imputes missing data and the remaining elements are property name/property value pairs used as inputs to the function

Caution If you have missing data points, use the 'Impute' property; otherwise, the clustergram function will error.

RowMarkerValue

Optional structure array for annotating the groups (clusters) of rows determined by the `clustergram` function. Each structure in the array represents a group of rows and contains the following fields:

- **GroupNumber** — The row group number to annotate.
- **Annotation** — String specifying text to annotate the row group.
- **Color** — String or three-element vector of RGB values specifying a color, which is used to label the row group. For more information on specifying colors, see `colspec`. If this field is empty, default is 'blue'.

ColumnMarkerValue

Optional structure array for annotating groups (clusters) of columns determined by the `clustergram` function. Each structure in the array represents a group of columns and contains the following fields:

- **GroupNumber** — The column group number to annotate.
- **Annotation** — String specifying text to annotate the column group.
- **Color** — String or three-element vector of RGB values specifying a color, which is used to label the column group. For more information on specifying colors, see `colspec`. If this field is empty, default is 'blue'.

Description

`CGobj = clustergram(Data)` performs hierarchical clustering analysis on the values in *Data*, a `DataMatrix` object or numeric matrix, creates *CGobj*, an object containing the analysis data, and displays a dendrogram and heat map. It uses hierarchical clustering with euclidean distance metric and average linkage to generate the hierarchical tree. The clustering is performed first along the columns (producing row-clustered data), and then along the rows in the matrix *Data*. If *Data* contains gene expression data, typically the rows correspond to genes and the columns correspond to samples.

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

`CGobj = clustergram(Data, ... 'RowLabels', RowLabelsValue, ...)` uses the contents of *RowLabelsValue*, a vector of numbers or cell array of text strings, as labels for the rows in the dendrogram and heat map. Default is a vector of values 1 through *M*, where *M* is the number of rows in *Data*.

`CGobj = clustergram(Data, ... 'ColumnLabels', ColumnLabelsValue, ...)` uses the contents of *ColumnLabelsValue*, a vector of numbers or cell array of text strings, as labels for the columns in the dendrogram and heat map. Default is a vector of values 1 through *M*, where *M* is the number of columns in *Data*.

`CGobj = clustergram(Data, ... 'Standardize', StandardizeValue, ...)` specifies the dimension for standardizing the values in *Data*. The standardized values are transformed so that the mean is 0 and the standard deviation is 1 in the specified dimension. *StandardizeValue* can be:

- 1 — Standardize along the columns of data.
- 2 (default) — Standardize along the rows of data.
- 3 — Do not perform standardization.

clustergram

`CGobj = clustergram(Data, ...'Cluster', ClusterValue, ...)` specifies the dimension for clustering the values in *Data*. *ClusterValue* can be:

- 1 — Cluster rows of data only.
- 2 — Cluster columns of data only.
- 3 (default) — Cluster rows of data, then cluster columns of row-clustered data.

`CGobj = clustergram(Data, ...'RowPdist', RowPdistValue, ...)` specifies the distance metric to pass to the `pdist` function (Statistics Toolbox software) to use to calculate the pairwise distances between rows. *RowPdistValue* is a string. For information on choices, see the `pdist` function. Default is 'euclidean'.

`CGobj = clustergram(Data, ...'ColumnPdist', ColumnPdistValue, ...)` specifies the distance metric to pass to the `pdist` function (Statistics Toolbox software) to use to calculate the pairwise distances between columns. *ColumnPdistValue* is a string. For information on choices, see the `pdist` function. Default is 'euclidean'.

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

`CGobj = clustergram(Data, ...'Linkage', LinkageValue, ...)` specifies the linkage method to pass to the `linkage` function (Statistics Toolbox software) to use to create the hierarchical cluster tree for rows and columns. *LinkageValue* is a string or two-element cell array of strings. If a two-element cell array of strings, the first element is used for linkage between rows, and the second element is used for linkage

between columns. For information on choices, see the `linkage` function. Default is `'average'`.

Tip To specify the linkage method for only one dimension, set the other dimension to `''`.

`CGobj = clustergram(Data, ...'Dendrogram', DendrogramValue, ...)` specifies the `'colorthreshold'` property to pass to the `dendrogram` function (Statistics Toolbox software) to create the dendrogram plot. *DendrogramValue* is a scalar or two-element numeric vector or cell array of strings that specifies the `'colorthreshold'` property. If a two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns. For more information, see the `dendrogram` function.

Tip To specify the `'colorthreshold'` property for only one dimension, set the other dimension to `''`.

`CGobj = clustergram(Data, ...'OptimalLeafOrder', OptimalLeafOrderValue, ...)` enables or disables the optimal leaf ordering calculation, which determines the leaf order that maximizes the similarity between neighboring leaves. Choices are `true` (enable) or `false` (disable). Default depends on the size of *Data*. If the number of rows or columns in *Data* is greater than 1000, default is `false`; otherwise, default is `true`.

Note Disabling the optimal leaf ordering calculation can be useful when working with large data sets because this calculation uses a large amount of memory and can be very time consuming.

clustergram

`CGobj = clustergram(Data, ...'ColorMap', ColorMapValue, ...)` specifies the colormap to use to create the clustergram. This controls the colors used to display the heat map. *ColorMapValue* is either an M-by-3 matrix of RGB values or the name of or handle to a function that returns a colormap, such as `redgreenmap` or `redbluecmap`. Default is `redgreenmap`.

Note In `redgreenmap`, red represents values above the mean, black represents the mean, and green represents values below the mean of a row (gene) across all columns (samples). In `redbluecmap`, red represents values above the mean, white represents the mean, and blue represents values below the mean of a row (gene) across all columns (samples).

`CGobj = clustergram(Data, ...'DisplayRange', DisplayRangeValue, ...)` specifies the display range of standardized values. *DisplayRangeValue* must be a positive scalar. Default is 3, which means there is a color variation for values between -3 and 3, but values >3 will be the same color as 3, and values < -3 will be the same color as -3.

For example, if you specify `redgreenmap` for the 'ColorMap' property, pure red represents values \geq *DisplayRangeValue*, and pure green represents values \leq $-$ *DisplayRangeValue*.

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

`CGobj = clustergram(Data, ...'LogTrans', LogTransValue, ...)` controls the \log_2 transform of *Data* from natural scale. Choices are `true` or `false` (default).

`CGobj = clustergram(Data, ...'Ratio', RatioValue, ...)` specifies the ratio of space that the row and column dendrograms occupy relative to the heat map. If *RatioValue* is a scalar, it is used as the ratio for both dendrograms. If *RatioValue* is a two-element vector,

the first element is used for the ratio of the row dendrogram width to the heat map width, and the second element is used for the ratio of the column dendrogram height to the heat map height. The second element is ignored for one-dimensional clustergrams. Default is 1/5.

`CGobj = clustergram(Data, ...'Impute', ImputeValue, ...)` specifies a function and optional inputs that impute missing data. *ImputeValue* can be any of the following:

- Name of a function that imputes missing data
- Handle to a function that imputes missing data
- Cell array where the first element is the name of or handle to a function that imputes missing data and the remaining elements are property name/property value pairs used as inputs to the function

Tip If you have missing data points, use the 'Impute' property; otherwise, the `clustergram` function will error.

`CGobj = clustergram(Data, ...'RowMarker', RowMarkerValue, ...)` specifies an optional structure array for annotating the groups of rows determined by the `clustergram` function. Each structure in the array represents a group of rows and contains the following fields:


- `GroupNumber` — The row group number to annotate.
- `Annotation` — String specifying text to annotate the row group.
- `Color` — String or three-element vector of RGB values specifying a color, which is used to label the row group. For more information on specifying colors, see `colspec`. If this field is empty, default is 'blue'.

`CGobj = clustergram(Data, ...'ColumnMarker', ColumnMarkerValue, ...)` specifies an optional structure array for annotating the groups of columns determined by the

clustergram

`clustergram` function. Each structure in the array represents a group of columns and contains the following fields:

- **GroupNumber** — The column group number to annotate.
- **Annotation** — String specifying text to annotate the column group.
- **Color** — String or three-element vector of RGB values specifying a color, which is used to label the column group. For more information on specifying colors, see `colspec`. If this field is empty, default is 'blue'.

Tip If necessary, view row labels (right) and column labels (bottom) by using the Zoom In  button on the toolbar to zoom the clustergram.

Examples

The following example uses data from an experiment (DeRisi et al., 1997) that used DNA microarrays to study temporal gene expression of almost all genes in *Saccharomyces cerevisiae* during the metabolic shift from fermentation to respiration. Expression levels were measured at seven time points during the diauxic shift.

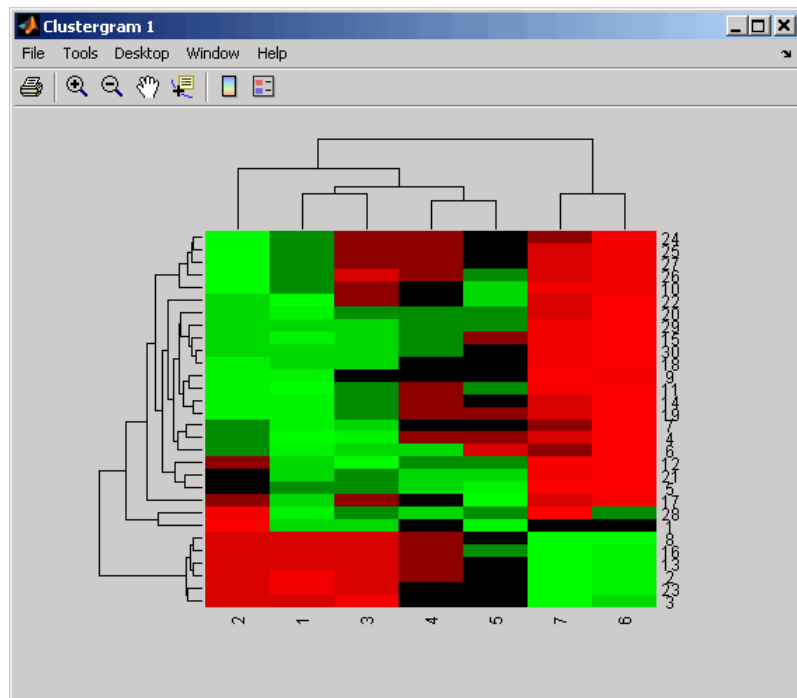
- 1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains filtered yeast data. This MAT-file includes three variables: `yeastvalues`, a matrix of gene expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`.

```
load filteredyeastdata
```

- 2 Create a clustergram object and display the dendrograms and heat map from the gene expression data in the first 30 rows of the `yeastvalues` matrix.

```
cgo = clustergram(yeastvalues(1:30,:))
```

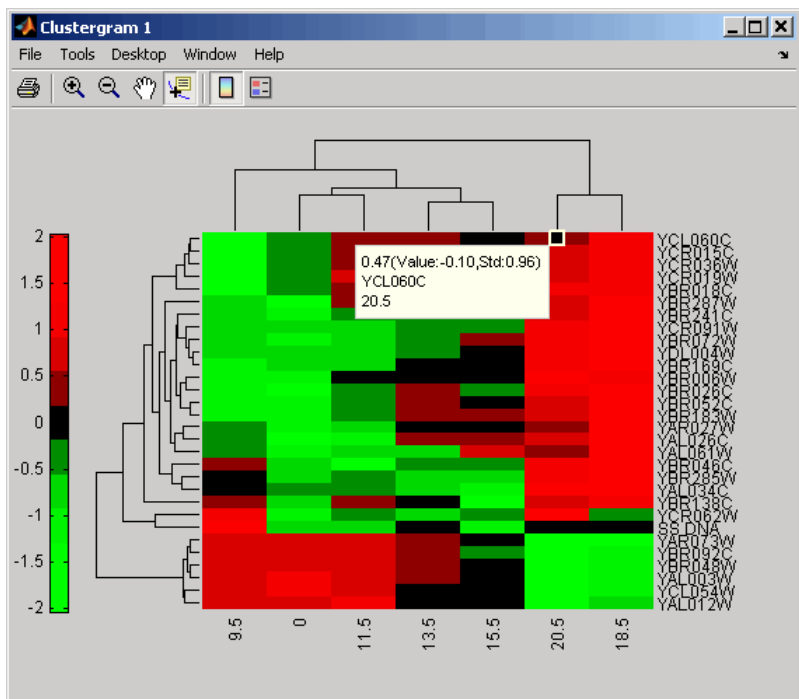

Clustergram object with 30 rows of nodes and 7 columns of nodes.





- 3 Use the `set` method and the `genes` and `times` vectors to add meaningful row and column labels to the clustergram.

```
set(cgo, 'RowLabels', genes(1:30), 'ColumnLabels', times)
```

clustergram



- 4 Add a color bar to the clustergram by clicking the  Insert Colorbar button on the toolbar, then view a data tip containing the intensity value, row label, and column label for a specific area of the heat map by clicking the  Data Cursor button on the toolbar, then clicking an area in the heat map. To delete this data tip, right-click it, then select **Delete Current Datatip**.
- 5 Use the `get` method to display the properties of the clustergram object, `cgo`:

```
get(cgo)
```

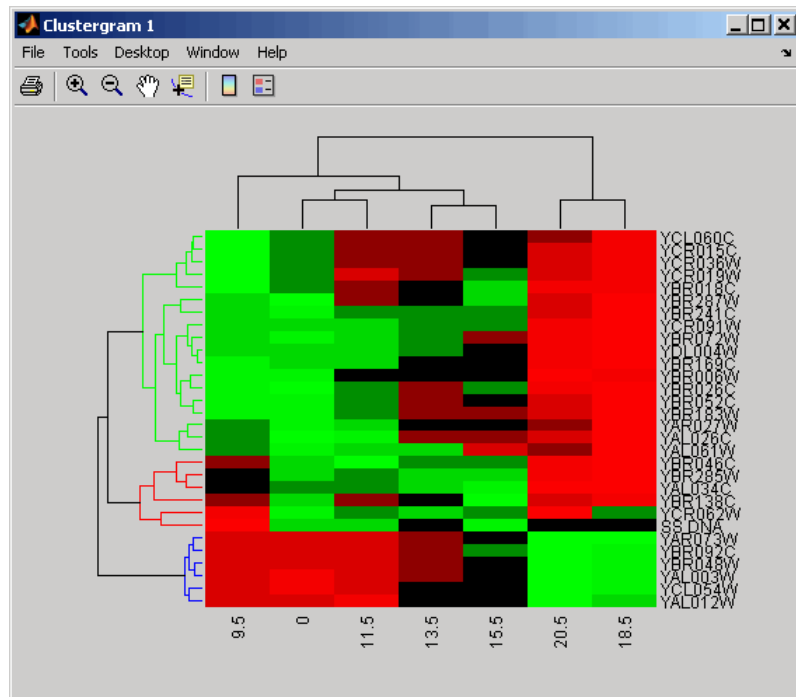
```
RowLabels: {30x1 cell}
```

```
ColumnLabels: {7x1 cell}
Standardize: {'ROW (2)'}
Cluster: {'ALL (3)'}
RowPDist: {'Euclidean'}
ColumnPDist: {'Euclidean'}
Linkage: {'Average'}
Dendrogram: {[0]}
OptimalLeafOrder: 1
LogTrans: 0
Colormap: [11x3 double]
DisplayRange: 3
SymmetricRange: 1
Ratio: [0.2000 0.2000]
Impute: []
RowMarkers: []
ColumnMarkers: []
```

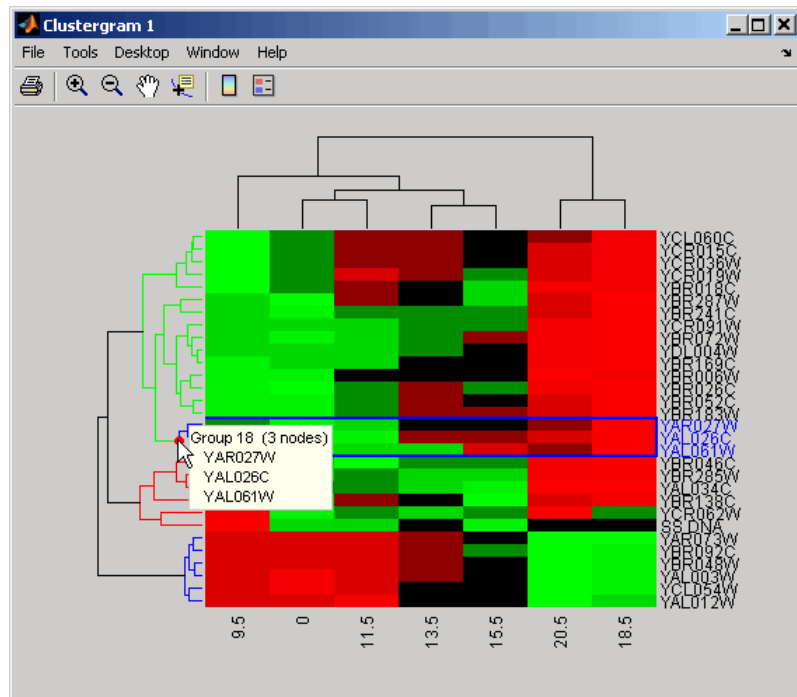
- 6 Change the clustering parameters by changing the linkage method and changing the color of the groups of nodes in the dendrogram whose linkage is less than a threshold of 3.

```
set(cgo,'Linkage','complete','Dendrogram',3)
```

clustergram

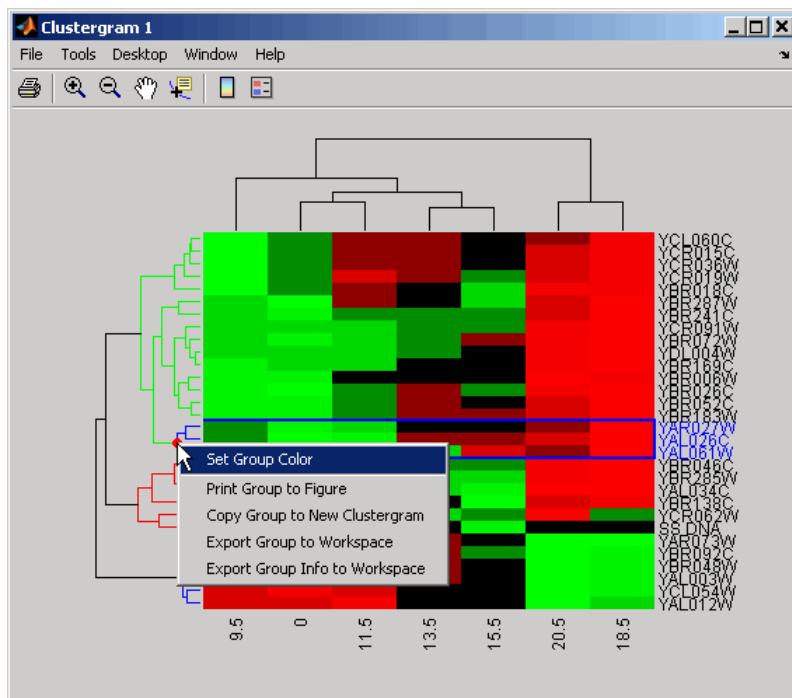


- 7 Place the cursor on a branch node in the dendrogram to highlight (in blue) the group associated with it. Press and hold the mouse button to display a data tip listing the group number and the nodes (genes or samples) in the group.



- 8 Right-click a branch node in the dendrogram to display a menu of options.

clustergram



The following options are available:

- **Set Group Color** — Change the group color.
- **Print Group to Figure** — Print the group to a Figure window.
- **Copy Group to New Clustergram** — Copy the group to a new Clustergram window.
- **Export Group to Workspace** — Create a clustergram object of the group in the MATLAB Workspace.
- **Export Group Info to Workspace** — Create a structure containing information about the group in the MATLAB Workspace. The structure contains these fields: GroupNames, RowNodeNames, ColumnNodeNames, and ExprValues.

- 9 Create a clustergram object in the MATLAB Workspace of Group 18 by right-clicking it, then selecting **Export Group to Workspace**. In the Export to Workspace dialog box, type **Group18**, then click **OK**.
- 10 Use the `get` method to display the properties of the clustergram object, `Group18`.

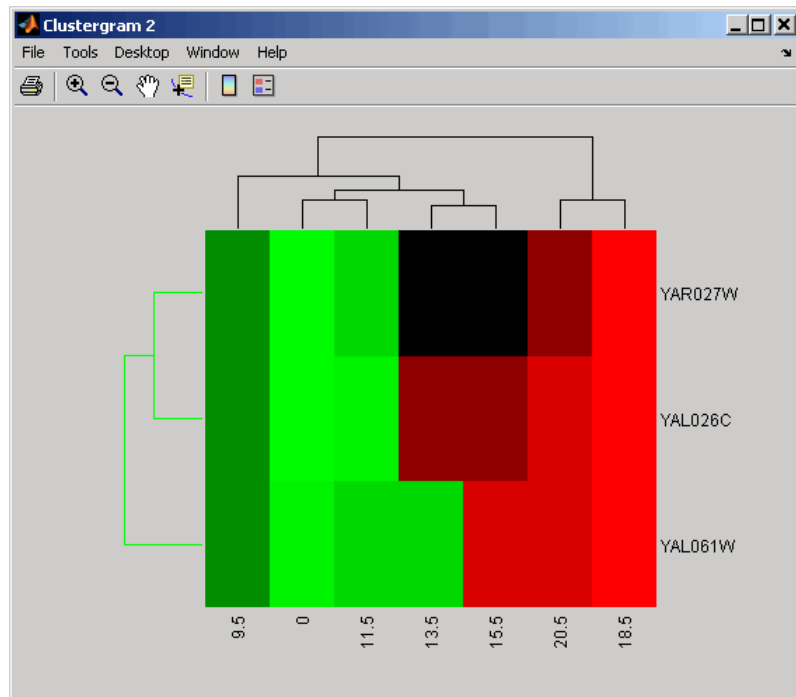
```
get(Group18)

    RowGroupNames: {2x1 cell}
    RowNodeNames: {3x1 cell}
ColumnGroupNames: {6x1 cell}
ColumnNodeNames: {7x1 cell}
    ExprValues: [3x7 double]
    Standardize: {'ROW (2)'}
    Cluster: {'ALL (3)'}
    RowPDist: {'Euclidean'}
    ColumnPDist: {'Euclidean'}
    Linkage: 'complete'
    Dendrogram: 3
OptimalLeafOrder: 1
    LogTrans: 0
    Colormap: [11x3 double]
    DisplayRange: 3
SymmetricRange: 1
    Ratio: [0.2000 0.2000]
    Impute: []
    RowMarkers: []
    ColumnMarkers: []
```

- 11 Use the `view` method to view the clustergram (dendrograms and heat map) of the clustergram object, `Group18`.

```
view(Group18)
```

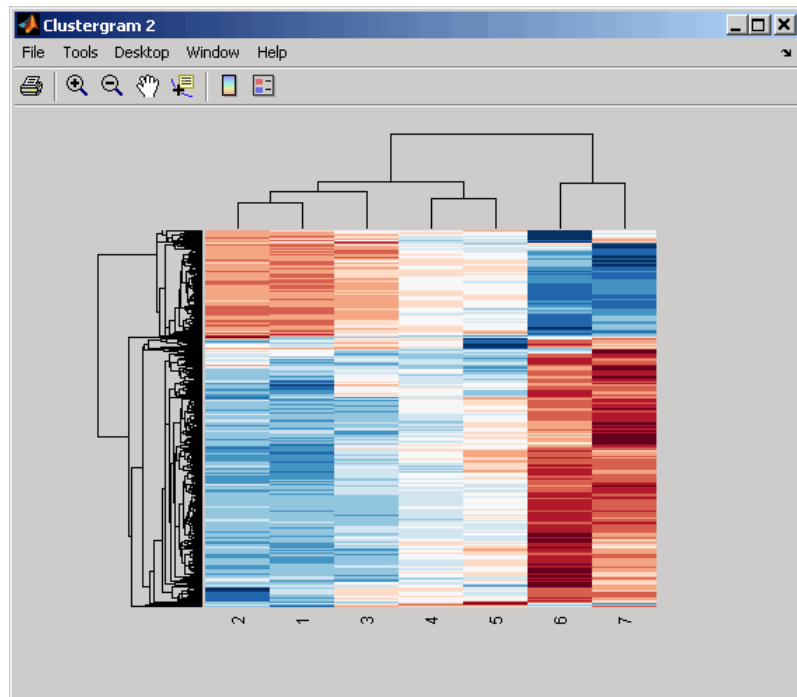
clustergram



Note You cannot use the `set` function with a clustergram object created by exporting a group from another clustergram object.

- 12** View all the gene expression data using a diverging red and blue colormap.

```
cgo_all = clustergram(yeastvalues,'Colormap',redbluecmap)
Clustergram object with 614 rows of nodes and 7 columns of nodes.
```

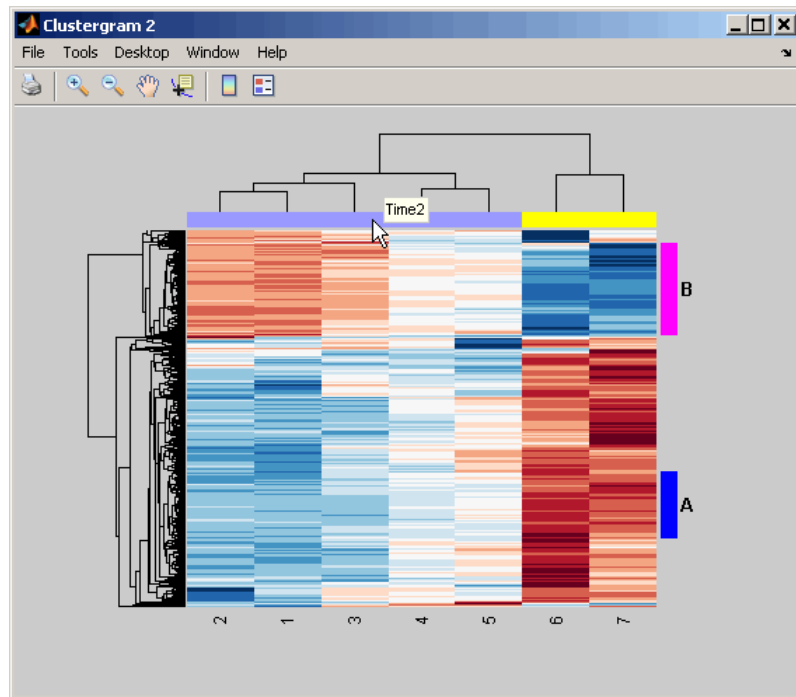
- 13** Create structure arrays to specify marker colors and annotations for two groups of rows (510 and 593) and two groups of columns (4 and 5).

```
rm = struct('GroupNumber', {510,593}, 'Annotation', {'A', 'B'}, ...
           'Color', {'b', 'm'});
cm = struct('GroupNumber', {4,5}, 'Annotation', {'Time1', 'Time2'}, ...
           'Color', {[1 1 0], [0.6 0.6 1]});
```

- 14** Use the 'RowMarker' and 'ColumnMarker' properties to add the color markers and annotations to the clustergram.

```
set(cgo_all, 'RowMarker', rm, 'ColumnMarker', cm)
```

clustergram



Click the color column markers to display the annotations.

References

- [1] Bar-Joseph, Z., Gifford, D.K., and Jaakkola, T.S. (2001). Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics* 17, Suppl 1:S22 – 9. PMID: 11472989.
- [2] Eisen, M.B., Spellman, P.T., Brown, P.O., and Botstein, D. (1998). Cluster analysis and display of genome-wide expression patterns. *Proc Natl Acad Sci USA* 95, 14863–8.
- [3] DeRisi, J.L., Iyer, V.R., and Brown, P.O. (1997). Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science* 278, 680–686s.

[4] Golub, T.R., Slonim, D.K., and Tamayo, P., et al. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science* 286 (15), 531–537.

See Also

Bioinformatics Toolbox functions: `redbluecmap`, `redgreencmap`

Bioinformatics Toolbox object: `clustergram` object

Bioinformatics Toolbox methods of a `clustergram` object: `get`, `plot`, `set`, `view`

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

codonbias

Purpose Calculate codon frequency for each amino acid coded for in nucleotide sequence

Syntax

```
CodonFreq = codonbias(SeqNT)  
CodonFreq = codonbias(SeqNT, ...'GeneticCode',  
GeneticCodeValue, ...)  
CodonFreq = codonbias(SeqNT, ...'Frame', FrameValue, ...)  
CodonFreq = codonbias(SeqNT, ...'Reverse',  
ReverseValue, ...)  
CodonFreq = codonbias(SeqNT, ...'Pie', PieValue, ...)
```

Arguments*SeqNT*

One of the following:

- String of codes specifying a nucleotide sequence
- Row vector of integers specifying a nucleotide sequence
- MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `fastaread`, `emblread`, `getembl`, `genbankread`, or `getgenbank`

Valid characters include A, C, G, T, and U.

`codonbias` does not count ambiguous nucleotides or gaps.*GeneticCodeValue*

Integer or string specifying a genetic code number or code name from the table Genetic Code on page 2-271. Default is 1 or 'Standard'.

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

FrameValue

Integer specifying a reading frame in the nucleotide sequence. Choices are 1 (default), 2, or 3.

<i>ReverseValue</i>	Controls the return of the codon frequency for the reverse complement sequence of the nucleotide sequence specified by <i>SeqNT</i> . Choices are true or false (default).
<i>PieValue</i>	Controls the creation of a figure of 20 pie charts, one for each amino acid. Choices are true or false (default).

Return Values

<i>CodonFreq</i>	MATLAB structure containing a field for each amino acid, each of which contains the associated codon frequencies as percentages.
------------------	--

Description

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

CodonFreq = `codonbias(SeqNT)` calculates the codon frequency in percent for each amino acid coded for in *SeqNT*, a nucleotide sequence, and returns the results in *CodonFreq*, a MATLAB structure containing a field for each amino acid.

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

CodonFreq = `codonbias(SeqNT, ...'GeneticCode', GeneticCodeValue, ...)` specifies a genetic code. Choices for *GeneticCodeValue* are an integer or string specifying a code number or code name from the table Genetic Code on page 2-271. If you use a code name, you can truncate the name to the first two characters of the name. Default is 1 or 'Standard'.

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

CodonFreq = codonbias(*SeqNT*, ...'Frame', *FrameValue*, ...) calculates the codon frequency in the reading frame specified by *FrameValue*, which can be 1 (default), 2, or 3.

CodonFreq = codonbias(*SeqNT*, ...'Reverse', *ReverseValue*, ...) controls the return of the codon frequency for the reverse complement of the nucleotide sequence specified by *SeqNT*. Choices are true or false (default).

CodonFreq = codonbias(*SeqNT*, ...'Pie', *PieValue*, ...) controls the creation of a figure of 20 pie charts, one for each amino acid. Choices are true or false (default).

Genetic Code

Code Number	Code Name
1	Standard
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

Genetic Code (Continued)

Code Number	Code Name
13	Ascidian Mitochondrial
14	Flatworm Mitochondrial
15	Blepharisma Nuclear
16	Chlorophycean Mitochondrial
21	Trematode Mitochondrial
22	Scenedesmus Obliquus Mitochondrial
23	Thraustochytrium Mitochondrial

Examples

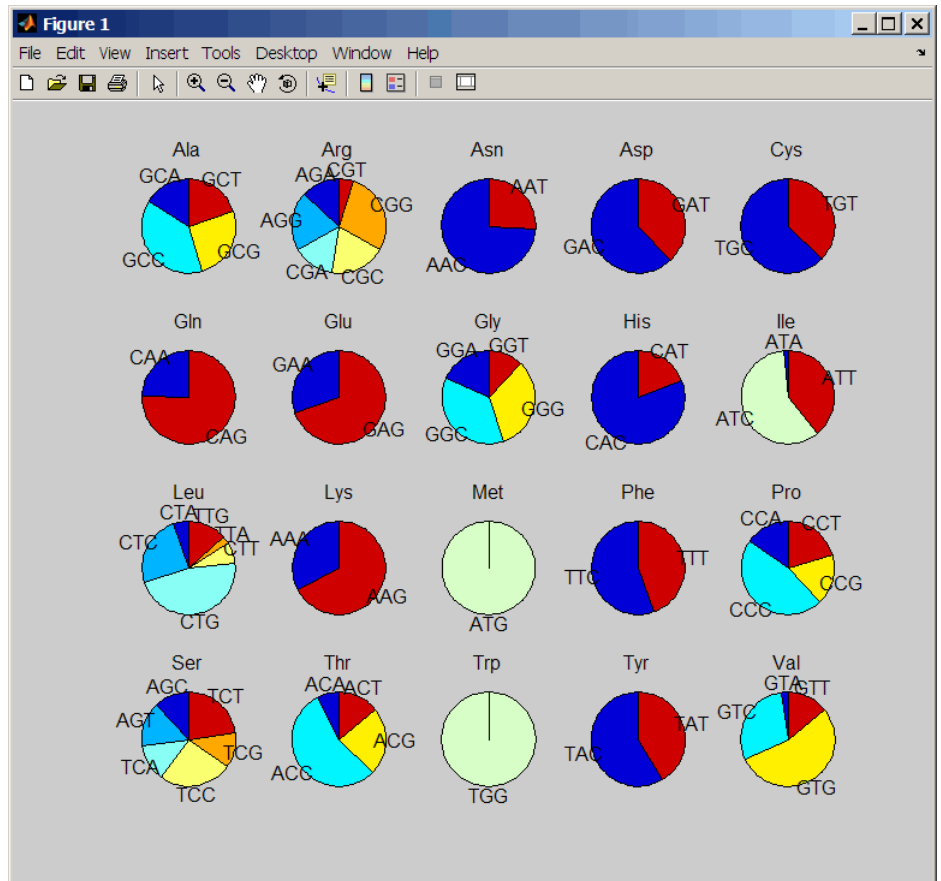
- 1 Import a nucleotide sequence from the GenBank database into the MATLAB software. For example, retrieve the DNA sequence that codes for a human insulin receptor.

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

- 2 Calculate the codon frequency for each amino acid coded for by the DNA sequence, and then plot the results.

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

A figure with 20 pie charts for the 20 amino acids displays.



3 Get the codon frequency for the alanine (A) amino acid.

cb.Ala

ans =

```
Codon: {'GCA' 'GCC' 'GCG' 'GCT'}
Freq: [0.1600 0.3867 0.2533 0.2000]
```

codonbias

See Also

Bioinformatics Toolbox functions: aminolookup, codoncount, geneticcode, nt2aa

Purpose Count codons in nucleotide sequence

Syntax

```

Codons = codoncount(SeqNT)
[Codons, CodonArray] = codoncount(SeqNT)
... = codoncount(SeqNT, ...'Frame', FrameValue, ...)
... = codoncount(SeqNT, ...'Reverse', ReverseValue, ...)
... = codoncount(SeqNT, ...'Figure', FigureValue, ...)
    
```

Arguments

SeqNT One of the following:

- String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers on page 2-794
- Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes on page 2-562
- MATLAB structure containing a *Sequence* field that contains a nucleotide sequence, such as returned by *fastaread*, *emblread*, *getembl*, *genbankread*, or *getgenbank*.

Examples: 'ACGT' or [1 2 3 4]

FrameValue Integer specifying a reading frame in the nucleotide sequence. Choices are 1 (default), 2, or 3.

ReverseValue Controls the return of the codon count for the reverse complement sequence of the nucleotide sequence specified by *SeqNT*. Choices are true or false (default).

FigureValue Controls the display of a heat map of the codon counts. Choices are true or false (default).

Return Values

<i>Codons</i>	MATLAB structure containing fields for the 64 possible codons (AAA, AAC, AAG, ..., TTG, TTT), which contain the codon counts in <i>SeqNT</i> .
<i>CodonArray</i>	A 4-by-4-by-4 array containing the raw count data for each codon. The three dimensions correspond to the three positions in the codon, and the indices to each element are represented by 1 = A, 2 = C, 3 = G, and 4 = T. For example, the element (2,3,4) in the array contains the number of CGT codons.

Description

Codons = `codoncount(SeqNT)` counts the codons in *SeqNT*, a nucleotide sequence, and returns the codon counts in *Codons*, a MATLAB structure containing fields for the 64 possible codons (AAA, AAC, AAG, ..., TTG, TTT).

- For sequences that have codons with the character U, these codons are added to the corresponding codons containing a T.
- If the sequence contains ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N), or gaps indicated by a hyphen (-), then these characters are counted in the field *Others*, and the following warning message appears:

Warning: Ambiguous symbols 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, or Z), then codons containing these characters are counted in the field *Others*, and the following warning message appears:

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

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

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

`...` = `codoncount(SeqNT, ...'Frame', FrameValue, ...)` counts the codons in the reading frame specified by *FrameValue*, which can be 1 (default), 2, or 3.

`...` = `codoncount(SeqNT, ...'Reverse', ReverseValue, ...)` controls the return of the codon count for the reverse complement sequence of *SeqNT*. Choices are `true` or `false` (default).

`...` = `codoncount(SeqNT, ...'Figure', FigureValue, ...)` controls the display of a heat map of the codon counts. Choices are `true` or `false` (default).

Examples

- Count the codons in a nucleotide sequence.

```
codons = codoncount('AACGTTA')
```

```
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
```

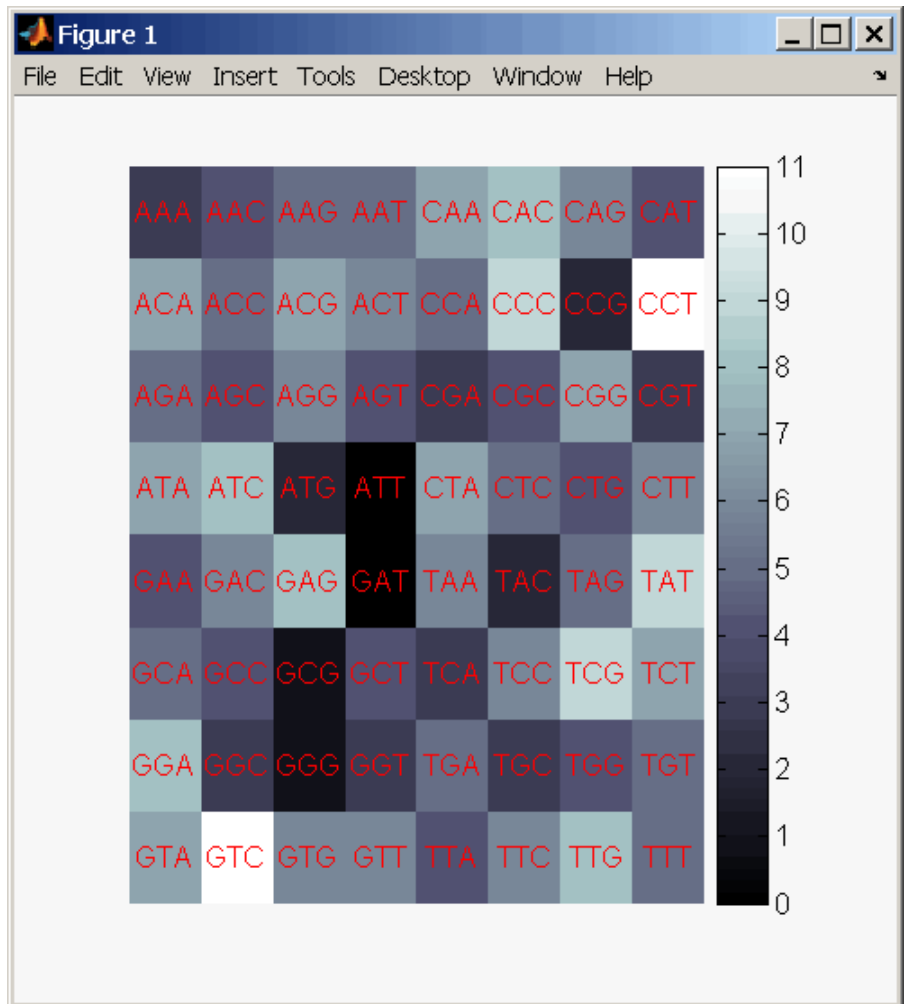
codoncount

- 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 of the codons in a random nucleotide sequence.

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

**See Also**

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

cpgisland

Purpose Locate CpG islands in DNA sequence

Syntax

```
cpgStruct = cpgisland(SeqDNA)  
cpgStruct = cpgisland(SeqDNA, ...'Window',  
WindowValue, ...)  
cpgStruct = cpgisland(SeqDNA, ...'MinIsland',  
MinIslandValue,  
...)  
cpgStruct = cpgisland(SeqDNA, ...'GCmin', GCminValue, ...)  
cpgStruct = cpgisland(SeqDNA, ...'CpGoe', CpGoeValue, ...)  
cpgStruct = cpgisland(SeqDNA, ...'Plot', PlotValue, ...)
```

Arguments

<i>SeqDNA</i>	One of the following: <ul style="list-style-type: none">• String of codes specifying a DNA nucleotide sequence• Row vector of integers specifying a DNA nucleotide sequence• MATLAB structure containing a <code>Sequence</code> field that contains a DNA nucleotide sequence, such as returned by <code>fastaread</code>, <code>emblread</code>, <code>getembl</code>, <code>genbankread</code>, or <code>getgenbank</code> Valid characters include A, C, G, and T. <code>cpgisland</code> does not count ambiguous nucleotides or gaps.
<i>WindowValue</i>	Integer specifying the window size for calculating GC content and CpGobserved/CpGexpected ratios. Default is 100 bases. A smaller window size increases the noise in a plot.
<i>MinIslandValue</i>	Integer specifying the minimum number of consecutive marked bases to report as a CpG island. Default is 200 bases.

<i>GCminValue</i>	Value specifying the minimum GC percent in a window needed to mark a base. Choices are a value between 0 and 1. Default is 0.5.
<i>CpGoeValue</i>	Value specifying the minimum CpGobserved/CpGexpected ratio in each window needed to mark a base. Choices are a value between 0 and 1. Default is 0.6. This ratio is defined as: $\text{CPGobs/CpGexp} = (\text{NumCpGs} * \text{Length}) / (\text{NumGs} * \text{NumCs})$
<i>PlotValue</i>	Controls the plotting of GC content, CpGoe content, CpG islands greater than the minimum island size, and all potential CpG islands for the specified criteria. Choices are true or false (default).

Return Values

<i>cpgStruct</i>	MATLAB structure containing the starting and ending bases of the CpG islands greater than the minimum island size.
------------------	--

Description

cpgStruct = `cpgisland(SeqDNA)` searches *SeqDNA*, a DNA nucleotide sequence, for CpG islands with a GC content greater than 50% and a CpGobserved/CpGexpected ratio greater than 60%. It marks bases meeting this criteria within a moving window of 100 DNA bases and then returns the results in *cpgStruct*, a MATLAB structure containing the starting and ending bases of the CpG islands greater than the minimum island size of 200 bases.

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

`cpgStruct = cpgisland(SeqDNA, ...'Window', WindowValue, ...)` specifies the window size for calculating GC content and CpGobserved/CpGexpected ratios. Default is 100 bases. A smaller window size increases the noise in a plot.

`cpgStruct = cpgisland(SeqDNA, ...'MinIsland', MinIslandValue, ...)` specifies the minimum number of consecutive marked bases to report as a CpG island. Default is 200 bases.

`cpgStruct = cpgisland(SeqDNA, ...'GCmin', GCminValue, ...)` specifies the minimum GC percent in a window needed to mark a base. Choices are a value between 0 and 1. Default is 0.5.

`cpgStruct = cpgisland(SeqDNA, ...'CpGoe', CpGoeValue, ...)` specifies the minimum CpGobserved/CpGexpected ratio in each window needed to mark a base. Choices are a value between 0 and 1. Default is 0.6. This ratio is defined as:

$$\text{CpGobs/CpGexp} = (\text{NumCpGs} * \text{Length}) / (\text{NumGs} * \text{NumCs})$$

`cpgStruct = cpgisland(SeqDNA, ...'Plot', PlotValue, ...)` controls the plotting of GC content, CpGoe content, CpG islands greater than the minimum island size, and all potential CpG islands for the specified criteria. Choices are true or false (default).

Examples

- 1 Import a nucleotide sequence from the GenBank database. For example, retrieve a sequence from *Homo sapiens* chromosome 12.

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

- 2 Calculate the CpG islands in the sequence and plot the results.

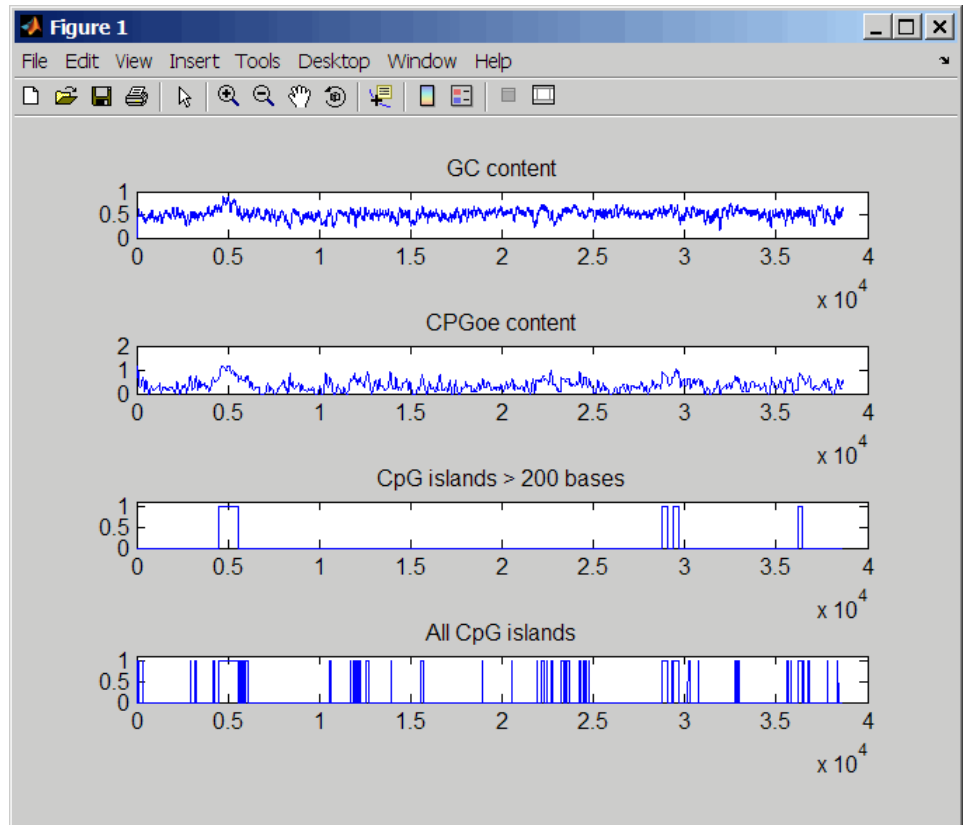
```
cpgisland(S.Sequence, 'PLOT', true)
```

```
ans =
```

```
Starts: [4470 28753 29347 36229]
```

```
Stops: [5555 29064 29676 36450]
```

The CpG islands greater than 200 bases in length are listed and a plot displays.



See Also

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

crossvalind

Purpose Generate cross-validation indices

Syntax

```
Indices = crossvalind('Kfold', N, K)
[Train, Test] = crossvalind('HoldOut', N, P)
[Train, Test] = crossvalind('LeaveMOut', N, M)
[Train, Test] = crossvalind('Resubstitution', N, [P,Q])
[...] = crossvalind(Method, Group, ...)
[...] = crossvalind(Method, Group, ..., 'Classes', C)
[...] = crossvalind(Method, Group, ..., 'Min', MinValue)
```

Description *Indices* = `crossvalind('Kfold', N, K)` returns randomly generated indices for a K-fold cross-validation of N observations. *Indices* contains equal (or approximately equal) proportions of the integers 1 through K that define a partition of the N observations into K disjoint subsets. Repeated calls return different randomly generated partitions. K defaults to 5 when omitted. In K-fold cross-validation, K-1 folds are used for training and the last fold is used for evaluation. This process is repeated K times, leaving one different fold for evaluation each time.

`[Train, Test] = crossvalind('HoldOut', N, P)` returns logical index vectors for cross-validation of N observations by randomly selecting P*N (approximately) observations to hold out for the evaluation set. P must be a scalar between 0 and 1. P defaults to 0.5 when omitted, corresponding to holding 50% out. Using holdout cross-validation within a loop is similar to K-fold cross-validation one time outside the loop, except that non-disjointed subsets are assigned to each evaluation.

`[Train, Test] = crossvalind('LeaveMOut', N, M)`, where M is an integer, returns logical index vectors for cross-validation of N observations by randomly selecting M of the observations to hold out for the evaluation set. M defaults to 1 when omitted. Using `LeaveMOut` cross-validation within a loop does not guarantee disjointed evaluation sets. Use K-fold instead.

`[Train, Test] = crossvalind('Resubstitution', N, [P,Q])` returns logical index vectors of indices for cross-validation of N observations by randomly selecting P*N observations for the evaluation set and Q*N observations for training. Sets are selected in order to

minimize the number of observations that are used in both sets. P and Q are scalars between 0 and 1. $Q=1-P$ corresponds to holding out $(100*P)\%$, while $P=Q=1$ corresponds to full resubstitution. $[P,Q]$ defaults to $[1,1]$ when omitted.

`[...] = crossvalind(Method, Group, ...)` takes the group structure of the data into account. `Group` is a grouping vector that defines the class for each observation. `Group` can be a numeric vector, a string array, or a cell array of strings. The partition of the groups depends on the type of cross-validation: For K -fold, each group is divided into K subsets, approximately equal in size. For all others, approximately equal numbers of observations from each group are selected for the evaluation set. In both cases the training set contains at least one observation from each group.

`[...] = crossvalind(Method, Group, ..., 'Classes', C)` restricts the observations to only those values specified in `C`. `C` can be a numeric vector, a string array, or a cell array of strings, but it is of the same form as `Group`. If one output argument is specified, it contains the value 0 for observations belonging to excluded classes. If two output arguments are specified, both will contain the logical value false for observations belonging to excluded classes.

`[...] = crossvalind(Method, Group, ..., 'Min', MinValue)` sets the minimum number of observations that each group has in the training set. `Min` defaults to 1. Setting a large value for `Min` can help to balance the training groups, but adds partial resubstitution when there are not enough observations. You cannot set `Min` when using K -fold cross-validation.

Examples

Create a 10-fold cross-validation to compute classification error.

```
load fisheriris
indices = crossvalind('Kfold',species,10);
cp = classperf(species);
for i = 1:10
    test = (indices == i); train = ~test;
    class = classify(meas(test,:),meas(train,:),species(train,:));
```

```
        classperf(cp,class,test)
    end
    cp.ErrorRate
```

Approximate a leave-one-out prediction error estimate.

```
load carbig
x = Displacement; y = Acceleration;
N = length(x);
sse = 0;
for i = 1:100
    [train,test] = crossvalind('LeaveMOut',N,1);
    yhat = polyval(polyfit(x(train),y(train),2),x(test));
    sse = sse + sum((yhat - y(test)).^2);
end
CVerr = sse / 100
```

Divide cancer data 60/40 without using the 'Benign' observations.
Assume groups are the true labels of the observations.

```
labels = {'Cancer','Benign','Control'};
groups = labels(ceil(rand(100,1)*3));
[train,test] = crossvalind('holdout',groups,0.6,'classes',...
    {'Control','Cancer'});
sum(test) % Total groups allocated for testing
sum(train) % Total groups allocated for training
```

See Also

Bioinformatics Toolbox functions: `classperf`, `knnclassify`,
`svmclassify`

Statistics Toolbox functions: `classify`, `grp2idx`

Purpose Read cytogenetic banding information

Syntax `CytoStruct = cytobandread(File)`

Arguments

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

Return Values

CytoStruct Structure containing cytogenetic G-banding data in the following fields:

- ChromLabels
- BandStartBPs
- BandEndBPs
- BandLabels
- GieStains

Description

`CytoStruct = cytobandread(File)` reads *File*, which is a string specifying a file containing cytogenetic G-banding data, and returns *CytoStruct*, which is a structure containing the following fields.

Field	Description
ChromLabels	Cell array containing the chromosome label (number or letter) on which each band is located.
BandStartBPs	Column vector containing the number of the base pair at the start of each band.
BandEndBPs	Column vector containing the number of the base pair at the end of each band.

Field	Description
BandLabels	Cell array containing the FISH label of each band, for example, p32.3.
GieStains	Cell array containing the Giemsa staining result for each band. Possible stain results depend on the species. For example, for <i>Homo sapiens</i> , the possibilities are: <ul style="list-style-type: none">• gneg• gpos25• gpos50• gpos75• gpos100• acen• stalk• gvar

Tip You can download files containing cytogenetic G-banding data from the NCBI or UCSC Genome Browser ftp site. For example, you can download the cytogenetic banding data for *Homo sapiens* from:

`ftp://ftp.ncbi.nlm.nih.gov/genomes/H_sapiens/mapview/ideogram.gz`

or

`ftp://hgdownload.cse.ucsc.edu/goldenPath/hg18/database/cytoBandIdeo.txt.gz`

Examples

Read the cytogenetic banding information for *Homo sapiens* into a structure.


```
hs_cytobands = cytobandread('hs_cytoBand.txt')
```

```
hs_cytobands =
```

```
ChromLabels: {862x1 cell}  
BandStartBPs: [862x1 int32]  
BandEndBPs: [862x1 int32]  
BandLabels: {862x1 cell}  
GieStains: {862x1 cell}
```

See Also

Bioinformatics Toolbox function: `chromosomeplot`

DataMatrix

Purpose Create DataMatrix object

Syntax

```
DMobj = DataMatrix(Matrix)
DMobj = DataMatrix(Matrix, RowNames, ColumnNames)
DMobj = DataMatrix('File', FileName)
DMobj = DataMatrix(..., 'RowNames', RowNamesValue, ...)
DMobj = DataMatrix(..., 'ColNames', ColNamesValue, ...)
DMobj = DataMatrix(..., 'Name', NameValue, ...)
DMobj = DataMatrix('File', FileName, ...'Delimiter',
    DelimiterValue, ...)
DMobj = DataMatrix('File', FileName, ...'HLine',
    HLineValue,
    ...)
DMobj = DataMatrix('File', FileName, ...'Rows', RowsValue,
    ...)
DMobj = DataMatrix('File', FileName, ...'Columns',
    ColumnsValue, ...)
```

Arguments

<i>Matrix</i>	Two-dimensional numeric or logical array.
<i>RowNames</i>	Row names for the DataMatrix object, specified by a numeric vector, character array, or cell array of strings, whose elements are equal in number to the number of rows in <i>Matrix</i> . <i>RowNames</i> are typically gene names or probe identifiers from a microarray experiment.

Note The row names do not need to be unique.

ColumnNames

Column names for the DataMatrix object, specified by a numeric vector, character array, or cell array of strings, whose elements are equal in number to the number of columns in *Matrix*. *ColumnNames* are typically sample identifiers from a microarray experiment.

Note The column names do not need to be unique.

FileName

String specifying a file name or a path and file name of a tab-delimited TXT or XLS file that contains table-oriented data and metadata.

Note Typically, the first row of the table contains column names, the first column contains row names, and the numeric data starts at the 2,2 position. The DataMatrix function will detect if the first column does not contain row names, and read data from the first column. However, if the first row does not contain header text (column names), set the HLine property to 0.

RowNamesValue,
ColNamesValue

Row names or column names for the DataMatrix object. Choices are:

- Numeric vector, character array, or a cell array of strings, whose elements are equal in number to the number of rows or number of columns of numeric data in the input matrix.
- A single string, which is used as a prefix for row or column names. Numbers will be appended to the prefix.
- `true` — Unique row or column names will be assigned using the formats `row1`, `row2`, `row3`, etc., or `col1`, `col2`, `col3`, etc.
- `false` — Default. No row or column names are assigned.

Note The row or column names do not need to be unique.

NameValue

String specifying a name for the DataMatrix object. Default is `''`.

DelimiterValue

String specifying a delimiter symbol to use for the input file. Typical choices are:

- `' '`
- `'\t'` (default)
- `','`
- `';'`
- `'|'`

HLineValue

Positive integer that specifies which row of the input file contains the column header text (column names). Default is 1.

When creating the DataMatrix object *DMobj*, the `DataMatrix` function loads data from $(HLineValue + 1)$ to the end of the file.

Tip If the input file does not contain column header text (column names), set *HLineValue* to 0.

RowsValue,
ColumnsValue

A subset of rows or columns in `File`, for the `DataMatrix` function to use to create the DataMatrix object. Choices are:

- Cell array of strings
- Character array
- Numeric or logical vector

Description

A DataMatrix object encapsulates measurement data and feature metadata from a microarray experiment so that it can be indexed by gene names or probe identifiers and by sample identifiers. For examples of creating and using DataMatrix objects, see “Working with DataMatrix Objects” in the Bioinformatics Toolbox User’s Guide.

Note The DataMatrix constructor function is part of the microarray object package. To make it available, type the following in the MATLAB command line:

```
import bioma.data.*
```

Otherwise, use `bioma.data.DataMatrix` instead of `DataMatrix`, in the following syntaxes.

`DMobj = DataMatrix(Matrix)` creates a DataMatrix object, *DMobj*, from *Matrix*, a two-dimensional numeric or logical array. *Matrix* can also be a DataMatrix object.

`DMobj = DataMatrix(Matrix, RowNames, ColumnNames)` creates a DataMatrix object, *DMobj*, from *Matrix*, a two-dimensional numeric or logical array, with row names and column names specified by *RowNames* and *ColumnNames*. *RowNames* and *ColumnNames* can be a numeric vector, character array, or cell array of strings, whose elements are equal in number to the number of rows and number of columns, respectively, in *Matrix*. *RowNames* are typically gene names or probe identifiers, while *ColumnNames* are typically sample identifiers.

Note The row or column names do not need to be unique.

`DMobj = DataMatrix('File', FileName)` creates a DataMatrix object, *DMobj*, from *FileName*, a string specifying a file name or a path and file name of a tab-delimited TXT or XLS file that contains table-oriented data and metadata.

Note Typically, the first row of the table contains column names, the first column contains row names, and the numeric data starts at the 2,2 position. The `DataMatrix` function will detect if the first column does not contain row names, and read data from the first column. However, if the first row does not contain header text (column names), set the `HLine` property to 0.

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

`DMobj = DataMatrix(..., 'RowNames', RowNamesValue, ...)` specifies row names for `DMobj`. *RowNamesValue* can be any of the following:

- Numeric vector, character array, or a cell array of strings, whose elements are equal in number to the number of rows of numeric data in the input matrix.
- A single string, which is used as a prefix for row names. Row numbers will be appended to the prefix.
- `true` — Unique row names will be assigned using the format `row1`, `row2`, `row3`, etc.
- `false` — Default. No row names are assigned.

Note The row names do not need to be unique.

`DMobj = DataMatrix(..., 'ColNames', ColNamesValue, ...)` specifies column names for `DMobj`. *ColNamesValue* can be any of the following:

- Numeric vector, character array, or a cell array of strings, whose elements are equal in number to the number of columns of numeric data in the input matrix.
- A single string, which is used as a prefix for column names. Column numbers will be appended to the prefix.
- `true` — Unique column names will be assigned using the format `col1, col2, col3, etc.`
- `false` — Default. No column names are assigned.

Note The column names do not need to be unique.

`DMobj = DataMatrix(..., 'Name', NameValue, ...)` specifies a name for `DMobj`. Default is `''`.

`DMobj = DataMatrix('File', FileName, ...'Delimiter', DelimiterValue, ...)` specifies a delimiter symbol to use for the input file. Typical choices are:

- `''`
- `'\t'` (default)
- `','`
- `';'`
- `'|'`

`DMobj = DataMatrix('File', FileName, ...'HLine', HLineValue, ...)` specifies which row of the input file contains the column header text (column names). `HLineValue` is a positive integer. Default is 1. When creating the `DataMatrix` object `DMobj`, the `DataMatrix` function loads data from `(HLineValue + 1)` to the end of the file.

Tip If the input file does not contain column header text (column names), set *HLineValue* to 0.

DMobj = DataMatrix('File', *FileName*, ... 'Rows', *RowsValue*, ...) specifies a subset of row names in *File* for the DataMatrix function to use to create *DMobj*. *RowsValue* can be a cell array of strings, a character array, or a numeric or logical vector.

DMobj = DataMatrix('File', *FileName*, ... 'Columns', *ColumnsValue*, ...) specifies a subset of column names in *File* for the DataMatrix function to use to create *DMobj*. *ColumnsValue* can be a cell array of strings, a character array, or a numeric or logical vector.

Examples

For examples of creating and using DataMatrix objects, see “Working with DataMatrix Objects” in the Bioinformatics Toolbox User’s Guide.

See Also

Bioinformatics Toolbox object: DataMatrix object

Bioinformatics Toolbox methods of a DataMatrix object: colnames, disp, darrayfun, dmsxfun, double, eq, ge, get, gt, horzcat, isequal, isequalwithequalnans, ldivide, le, lt, max, mean, median, min, minus, ndims, ne, numel, plot, plus, power, rdivide, rownames, set, single, sortcols, sortrows, std, sum, times, var, vertcat

dayhoff

Purpose Return Dayhoff scoring matrix

Syntax *ScoringMatrix* = dayhoff

Description *ScoringMatrix* = dayhoff returns a PAM250 type scoring matrix. The order of amino acids in the matrix is A R N D C Q E G H I L K M F P S T W Y V B Z X *.

See Also Bioinformatics Toolbox functions: blosum, gonnet, pam

Purpose Count dimers in nucleotide sequence

Syntax

```
Dimers = dimercount(SeqNT)  
[Dimers, Percent] = dimercount(SeqNT)  
... = dimercount(SeqNT, 'Chart', ChartValue)
```

Arguments

<i>SeqNT</i>	One of the following:
	<ul style="list-style-type: none">• String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers on page 2-794.• Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes on page 2-562.• MATLAB structure containing a <i>Sequence</i> field that contains a nucleotide sequence, such as returned by <i>fastaread</i>, <i>emblread</i>, <i>getembl</i>, <i>genbankread</i>, or <i>getgenbank</i>.

Examples: 'ACGT' or [1 2 3 4]

<i>ChartValue</i>	String specifying a chart type. Choices are 'pie' or 'bar'.
-------------------	---

dimercount

Return Values

- Dimers* MATLAB structure containing the fields AA, AC, AG, AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, and TT, which contain the dimer counts in *SeqNT*.
- Percent* A 4-by-4 matrix with the relative proportions of the dimers in *SeqNT*. The rows correspond to A, C, G, and T in the first element of the dimer, and the columns correspond to A, C, G, and T in the second element of the dimer.

Description

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

- For sequences that have dimers with the character U, these dimers are added to the corresponding dimers containing a T.
- If the sequence contains ambiguous nucleotide characters (R, Y, K, M, S, W, B, D, H, V, or N), or gaps indicated by a hyphen (-), then these characters are counted in the field *Others*, and the following warning message appears.

Warning: Ambiguous symbols 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, or Z), then dimers containing these characters are counted in the field *Others*, and the following warning message appears:

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

[*Dimers*, *Percent*] = dimercount(*SeqNT*) returns *Percent*, a 4-by-4 matrix with the relative proportions of the dimers in *SeqNT*. The rows correspond to A, C, G, and T in the first element of the dimer, and the columns correspond to A, C, G, and T in the second element of the dimer.

... = dimercount(*SeqNT*, 'Chart', *ChartValue*) creates a chart showing the relative proportions of the dimers. *ChartValue* can be 'pie' or 'bar'.

Examples

Count the dimers in a nucleotide sequence and display a matrix of the percentage of each dimer.

```
[Dimers, Percent] = dimercount('TAGCTGGCCAAGCGAGCTTG')
```

Dimers =

```
AA: 1
AC: 0
AG: 3
AT: 0
CA: 1
CC: 1
CG: 1
CT: 2
GA: 1
GC: 4
GG: 1
GT: 0
TA: 1
TC: 0
TG: 2
TT: 1
```

Percent =

```
0.0526      0      0.1579      0
0.0526     0.0526    0.0526     0.1053
0.0526     0.2105    0.0526      0
0.0526      0      0.1053     0.0526
```

dimercount

See Also

Bioinformatics Toolbox functions: `aaccount`, `basecount`, `baselookup`, `codoncount`, `nmercount`, `ntdensity`

Purpose Convert DNA sequence to RNA sequence

Syntax `SeqRNA = dna2rna(SeqDNA)`

Arguments

<i>SeqDNA</i>	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 Letter Codes to Integers on page 2-794. You can also enter a structure with the field <i>Sequence</i> .
<i>SeqRNA</i>	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: `regex`, `strrep`

dnds

Purpose

Estimate synonymous and nonsynonymous substitution rates

Syntax

```
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2)
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2,
... 'GeneticCode', GeneticCodeValue, ...)
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ... 'Method',
MethodValue, ...)
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ... 'Window',
WindowValue, ...)
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2,
... 'AdjustStops', AdjustStopsValue, ...)
[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ... 'Verbose',
VerboseValue, ...)
```

Arguments

<i>SeqNT1, SeqNT2</i>	Nucleotide sequences. Enter either a string or a structure with the field <code>Sequence</code> .
<i>GeneticCodeValue</i>	Property to specify a genetic code. Enter a Code Number or a string with a Code Name from the table Genetic Code on page 2-9. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard.

MethodValue

String specifying the method for calculating substitution rates. Choices are:

- NG (default) — Nei-Gojobori method (1986) uses the number of synonymous and nonsynonymous substitutions and the number of potentially synonymous and nonsynonymous sites. Based on the Jukes-Cantor model.
- LWL — Li-Wu-Luo method (1985) uses the number of transitional and transversional substitutions at three different levels of degeneracy of the genetic code. Based on Kimura's two-parameter model.
- PBL — Pamilo-Bianchi-Li method (1993) is similar to the Li-Wu-Luo method, but with bias correction. Use this method when the number of transitions is much larger than the number of transversions.

WindowValue

Integer specifying the sliding window size, in codons, for calculating substitution rates and variances.

<i>AdjustStopsValue</i>	Controls whether stop codons are excluded from calculations. Choices are <code>true</code> (default) or <code>false</code> .
<i>VerboseValue</i>	Property to control the display of the codons considered in the computations and their amino acid translations. Choices are <code>true</code> or <code>false</code> (default).

Tip Specify `true` to use this display to manually verify the codon alignment of the two input sequences. The presence of stop codons (*) in the amino acid translation can indicate that *SeqNT1* and *SeqNT2* are not codon-aligned.

Return Values

<i>Dn</i>	Nonsynonymous substitution rate(s).
<i>Ds</i>	Synonymous substitution rate(s).
<i>Vardn</i>	Variance for the nonsynonymous substitution rate(s).
<i>Vards</i>	Variance for the synonymous substitutions rate(s).

Description

$[Dn, Ds, Vardn, Vards] = \text{dnds}(\text{SeqNT1}, \text{SeqNT2})$ estimates the synonymous and nonsynonymous substitution rates per site between the two homologous nucleotide sequences, *SeqNT1* and *SeqNT2*, by comparing codons using the Nei-Gojobori method.

`dnds` returns:

- *Dn* — Nonsynonymous substitution rate(s).
- *Ds* — Synonymous substitution rate(s).

- *Vardn* — Variance for the nonsynonymous substitution rate(s).
- *Vards* — Variance for the synonymous substitutions rate(s).

This analysis:

- Assumes that the nucleotide sequences, *SeqNT1* and *SeqNT2*, are codon-aligned, that is, do not have frame shifts

Tip If your sequences are not codon-aligned, use the `nt2aa` function to convert them to amino acid sequences, use the `nwalign` function to globally align them, then use the `seqinsertgaps` function to recover the corresponding codon-aligned nucleotide sequences. See *Estimating Synonymous and Nonsynonymous Substitution Rates Between Two Nucleotide Sequences That Are Not Codon-Aligned* on page 2-310.

- Excludes codons that include ambiguous nucleotide characters or gaps
- Considers the number of codons in the shorter of the two nucleotide sequences

Caution

If *SeqNT1* and *SeqNT2* are too short or too divergent, saturation can be reached, and `dnds` returns NaNs and a warning message.

`[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ... 'PropertyName', PropertyValue, ...)` calls `dnds` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ...'GeneticCode', GeneticCodeValue, ...)` calculates synonymous and nonsynonymous substitution rates using the specified genetic code. Enter a Code Number or a string with a Code Name from the table Genetic Code on page 2-9. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard.

`[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ...'Method', MethodValue, ...)` allows you to calculate synonymous and nonsynonymous substitution rates using the following algorithms:

- NG (default) — Nei-Gojobori method (1986) uses the number of synonymous and nonsynonymous substitutions and the number of potentially synonymous and nonsynonymous sites. Based on the Jukes-Cantor model.
- LWL — Li-Wu-Luo method (1985) uses the number of transitional and transversional substitutions at three different levels of degeneracy of the genetic code. Based on Kimura's two-parameter model.
- PBL — Pamilo-Bianchi-Li method (1993) is similar to the Li-Wu-Luo method, but with bias correction. Use this method when the number of transitions is much larger than the number of transversions.

`[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ...'Window', WindowValue, ...)` performs the calculations over a sliding window, specified in codons. Each output is an array containing a rate or variance for each window.

`[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ...'AdjustStops', AdjustStopsValue, ...)` controls whether stop codons are excluded from calculations. Choices are true (default) or false.

Tip When the 'AdjustStops' property is set to true, the following are true:

- Stop codons are excluded from frequency tables.
 - Paths containing stop codons are not counted in the Nei-Gojobori method.
-

`[Dn, Ds, Vardn, Vards] = dnds(SeqNT1, SeqNT2, ... 'Verbose', VerboseValue, ...)` controls the display of the codons considered in the computations and their amino acid translations. Choices are true or false (default).

Tip Specify true to use this display to manually verify the codon alignment of the two input sequences, *SeqNT1* and *SeqNT2*. The presence of stop codons (*) in the amino acid translation can indicate that *SeqNT1* and *SeqNT2* are not codon-aligned.

Examples

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

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

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

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

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

```
dn =
```

```
0.0244
ds =
0.0697
vardn =
2.3509e-005
vars =
2.3438e-004
```

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

- 1 Retrieve two nucleotide sequences from the GenBank database for the neuraminidase (NA) protein of two strains of the influenza A virus (H5N1).

```
hk01 = getgenbank('AF509094');
vt04 = getgenbank('DQ094287');
```

- 2 Extract the coding region from the two nucleotide sequences.

```
hk01_cds = featuresparse(hk01,'feature','CDS','Sequence',true);
vt04_cds = featuresparse(vt04,'feature','CDS','Sequence',true);
```

- 3 Align the amino acids sequences converted from the nucleotide sequences.

```
[sc,a1] = nwalign(nt2aa(hk01_cds),nt2aa(vt04_cds),'extendgap',1);
```

- 4 Use the `seqinsertgaps` function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

```
hk01_aligned = seqinsertgaps(hk01_cds,a1(1,:))
vt04_aligned = seqinsertgaps(vt04_cds,a1(3,:))
```

- 5 Estimate the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences and also display the codons considered in the computations and their amino acid translations.

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

References

- [1] Li, W., Wu, C., and Luo, C. (1985). A new method for estimating synonymous and nonsynonymous rates of nucleotide substitution considering the relative likelihood of nucleotide and codon changes. *Molecular Biology and Evolution* 2(2), 150–174.
- [2] Nei, M., and Gojobori, T. (1986). Simple methods for estimating the numbers of synonymous and nonsynonymous nucleotide substitutions. *Molecular Biology and Evolution* 3(5), 418–426.
- [3] Nei, M., and Jin, L. (1989). Variances of the average numbers of nucleotide substitutions within and between populations. *Molecular Biology and Evolution* 6(3), 290–300.
- [4] Nei, M., and Kumar, S. (2000). Synonymous and nonsynonymous nucleotide substitutions” in *Molecular Evolution and Phylogenetics* (Oxford University Press).
- [5] Pamilo, P., and Bianchi, N. (1993). Evolution of the *Zfx* And *Zfy* genes: rates and interdependence between the genes. *Molecular Biology and Evolution* 10(2), 271–281.

See Also

Bioinformatics Toolbox functions: `dndsm1`, `featuresparse`, `geneticcode`, `nt2aa`, `nwalgn`, `seqinsertgaps`, `seqpdist`

dndsm1

Purpose

Estimate synonymous and nonsynonymous substitution rates using maximum likelihood method

Syntax

```
[Dn, Ds, Like] = dndsm1(SeqNT1, SeqNT2)
[Dn, Ds, Like] = dndsm1(SeqNT1, SeqNT2, ...'GeneticCode',
GeneticCodeValue, ...)
[Dn, Ds, Like] = dndsm1(SeqNT1, SeqNT2, ...'Verbose',
VerboseValue, ...)
```

Arguments

<i>SeqNT1, SeqNT2</i>	Nucleotide sequences. Enter either a string or a structure with the field <i>Sequence</i> .
<i>GeneticCodeValue</i>	Property to specify a genetic code. Enter a Code Number or a string with a Code Name from the table Genetic Code on page 2-9. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard.
<i>VerboseValue</i>	Property to control the display of the codons considered in the computations and their amino acid translations. Choices are <code>true</code> or <code>false</code> (default).

Tip Specify `true` to use this display to manually verify the codon alignment of the two input sequences. The presence of stop codons (*) in the amino acid translation can indicate that *SeqNT1* and *SeqNT2* are not codon-aligned.

Return Values

<i>Dn</i>	Nonsynonymous substitution rate(s).
<i>Ds</i>	Synonymous substitution rate(s).
<i>Like</i>	Likelihood of estimate of substitution rates.

Description

$[Dn, Ds, Like] = \text{dndsml}(\text{SeqNT1}, \text{SeqNT2})$ estimates the synonymous and nonsynonymous substitution rates between the two homologous sequences, *SeqNT1* and *SeqNT2*, using the Goldman-Yang method (1994). This maximum likelihood method estimates an explicit model for codon substitution that accounts for transition/transversion rate bias and base/codon frequency bias. Then it uses the model to correct synonymous and nonsynonymous counts to account for multiple substitutions at the same site. The maximum likelihood method is best suited when the sample size is significant (larger than 100 bases) and when the sequences being compared can have transition/transversion rate biases and base/codon frequency biases.

`dndsml` returns:

- *Dn* — Nonsynonymous substitution rate(s).
- *Ds* — Synonymous substitution rate(s).
- *Like* — Likelihood of this estimate.

This analysis:

- Assumes that the nucleotide sequences, *SeqNT1* and *SeqNT2*, are codon-aligned, that is, do not have frame shifts.

Tip If your sequences are not codon-aligned, use the `nt2aa` function to convert them to amino acid sequences, use the `nwalignment` function to globally align them, then use the `seqinsertgaps` function to recover the corresponding codon-aligned nucleotide sequences. See [Estimating Synonymous and Nonsynonymous Substitution Rates Between Two Nucleotide Sequences That Are Not Codon-Aligned](#) on page 2-315

- Excludes any ambiguous nucleotide characters or codons that include gaps.

- Considers the number of codons in the shorter of the two nucleotide sequences.

Caution

If *SeqNT1* and *SeqNT2* are too short or too divergent, saturation can be reached, and dndsm1 returns NaNs and a warning message.

[*Dn*, *Ds*, *Like*] = dndsm1(*SeqNT1*, *SeqNT2*, ...'*PropertyName*', *PropertyValue*, ...) calls dnds with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

[*Dn*, *Ds*, *Like*] = dndsm1(*SeqNT1*, *SeqNT2*, ...'*GeneticCode*', *GeneticCodeValue*, ...) calculates synonymous and nonsynonymous substitution rates using the specified genetic code. Enter a Code Number or a string with a Code Name from the table Genetic Code on page 2-9. If you use a Code Name, you can truncate it to the first two characters. Default is 1 or Standard.

[*Dn*, *Ds*, *Like*] = dndsm1(*SeqNT1*, *SeqNT2*, ...'*Verbose*', *VerboseValue*, ...) controls the display of the codons considered in the computations and their amino acid translations. Choices are true or false (default).

Tip Specify true to use this display to manually verify the codon alignment of the two input sequences, *SeqNT1* and *SeqNT2*. The presence of stop codons (*) in the amino acid translation can indicate that *SeqNT1* and *SeqNT2* are not codon-aligned.

Examples

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

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

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

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

```
[dn ds like] = dndsml(gag1, gag2)  
  
dn =  
    0.0259  
ds =  
    0.0623  
like =  
   -2.1857e+003
```

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

- 1 Retrieve two nucleotide sequences from the GenBank database for the neuraminidase (NA) protein of two strains of the Influenza A virus (H5N1).

```
hk01 = getgenbank('AF509094');  
vt04 = getgenbank('DQ094287');
```

- 2 Extract the coding region from the two nucleotide sequences.

```
hk01_cds = featuresparse(hk01, 'feature', 'CDS', 'Sequence', true);  
vt04_cds = featuresparse(vt04, 'feature', 'CDS', 'Sequence', true);
```

- 3** Align the amino acids sequences converted from the nucleotide sequences.

```
[sc,al]=nwalignment(nt2aa(hk01_cds),nt2aa(vt04_cds),'extendgap',1);
```

- 4** Use the `seqinsertgaps` function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

```
hk01_aligned = seqinsertgaps(hk01_cds,al(1,:));  
vt04_aligned = seqinsertgaps(vt04_cds,al(3,:));
```

- 5** Estimate the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences and also display the codons considered in the computations and their amino acid translations.

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

```
dn =  
    0.0445
```

```
ds =  
    0.1576
```

References

- [1] Tamura, K., and Mei, M. (1993). Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. *Molecular Biology and Evolution* *10*, 512–526.
- [2] Yang, Z., and Nielsen, R. (2000). Estimating synonymous and nonsynonymous substitution rates under realistic evolutionary models. *Molecular Biology and Evolution* *17*, 32–43.
- [3] Goldman, N., and Yang, Z. (1994). A Codon-based Model of Nucleotide Substitution for Protein-coding DNA Sequences. *Mol. Biol. Evol.* *11(5)*, 725–736.

See Also

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

emblread

Purpose Read data from EMBL file

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

Arguments

<i>File</i>	Either of the following: <ul style="list-style-type: none">• String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is an EMBL-formatted file.• MATLAB character array that contains the text of an EMBL-formatted file
<i>SequenceOnlyValue</i>	Controls the reading of only the sequence without the metadata. Choices are true or false (default).

Return Values

<i>EMBLData</i>	MATLAB structure with fields corresponding to EMBL data.
<i>EMBLSeq</i>	MATLAB character string representing the sequence.

Description *EMBLData* = emblread(*File*) reads data from *File*, an EMBL-formatted file, and creates *EMBLData*, a MATLAB structure with fields corresponding to the EMBL two-character line type code. Each line type code is stored as a separate element in the structure.

EMBLData contains the following fields.

Field
Identification.EntryName

Field
Identification.Version
Identification.Topology
Identification.Molecule
Identification.DataClass
Identification.Division
Identification.SequenceLength
Accession
SequenceVersion
DateCreated
DateUpdated
Description
Keyword
OrganismSpecies
OrganismClassification
Organelle
Reference{#}.Number
Reference{#}.Comment
Reference{#}.Position
Reference{#}.MedLine
Reference{#}.PubMed
Reference{#}.Group
Reference{#}.Authors
Reference{#}.Title
Reference{#}.Location
DatabaseCrossReference

Field
Comments
Assembly
Feature
Basecount.BP
Basecount.A
Basecount.C
Basecount.G
Basecount.T
Basecount.Other
Sequence

Note Topology information was not included in EMBL flat files before release 87 of the database. When reading a file created before release 87, EMBLREAD returns an empty `Identification.Topology` field.

Note The entry name is no longer displayed in the ID line of EMBL flat files in release 87. When reading a file created in release 87, EMBLREAD returns the accession number in the `Identification.EntryName` field.

EMBLSeq = `emblread (File, 'SequenceOnly', SequenceOnlyValue)` controls the reading of only the sequence without the metadata. Choices are `true` or `false` (default).

Examples

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

- 1 Use the `getembl` function and `ToFile` property to retrieve sequence information from the Web and save to an EMBL-formatted file.

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

- 2 Read data from the EMBL-formatted file and create a MATLAB structure.

```
EMBLData = emblread('rat_protein.txt')
```

See Also

Bioinformatics Toolbox functions: `fastaread`, `genbankread`, `genpeptread`, `getembl`, `pdbread`, `seqtool`

evalrasmolscript

Purpose Send RasMol script commands to Molecule Viewer window

Syntax `evalrasmolscript(FigureHandle, Command)`
`evalrasmolscript(FigureHandle, 'File', FileValue)`

Arguments

FigureHandle Figure handle to a molecule viewer returned by the `molviewer` function.

Command Either of the following:

- String specifying one or more RasMol script commands. Use a ; to separate commands.
- Character array or cell array containing strings specifying RasMol script commands.

Note For a complete list of RasMol script commands, see

<http://www.stolaf.edu/academics/chemapps/jmol/docs/>

FileValue String specifying a file name or a path and file name of a text file containing Jmol script commands. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Directory.

Description `evalrasmolscript(FigureHandle, Command)` sends the RasMol script commands specified by *Command* to *FigureHandle*, the figure handle of a Molecule Viewer window created using the `molviewer` function.

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

figure handle of a Molecule Viewer window created using the `molviewer` function.

Examples

- 1 Use the `molviewer` function to create a figure handle to a Molecule Viewer window.

```
FH = molviewer('2DHB')
```

- 2 Use the `evalrasmolscript` function to send script commands to the molecule viewer that change the background to black and spin the molecule.

```
evalrasmolscript(FH, 'background white; spin')
```

See Also

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

exprprofrange

Purpose Calculate range of gene expression profiles

Syntax
`Range = exprprofrange(Data)`
`[Range, LogRange] = exprprofrange(Data)`
`... = exprprofrange(Data, 'ShowHist', ShowHistValue)`

Arguments

Data DataMatrix object or numeric matrix of expression values, where each row corresponds to a gene.

ShowHistValue Controls the display of a histogram with range data. Choices are true or false (default).

Description `Range = exprprofrange(Data)` calculates the range of each expression profile in *Data*, a DataMatrix object or numeric matrix of expression values, where each row corresponds to a gene.

`[Range, LogRange] = exprprofrange(Data)` returns the log range, that is, $\log(\max(\text{prof})) - \log(\min(\text{prof}))$, of each expression profile. If you do not specify output arguments, `exprprofrange` displays a histogram bar plot of the range.

`... = exprprofrange(Data, 'ShowHist', ShowHistValue)` controls the display of a histogram with range data. Choices for *ShowHistValue* are true or false (default).

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

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

See Also Bioinformatics Toolbox functions: `exprprofvar`, `generangefilter`

Purpose Calculate variance of gene expression profiles

Syntax

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

Arguments

Data DataMatrix object or numeric matrix of expression values, where each row corresponds to a gene.

ShowHistValue Property to control the display of a histogram with variance data. Enter either true or false (default).

Description

Variance = `exprprofvar(Data)` calculates the variance of each expression profile in *Data*, a DataMatrix object or numeric matrix of expression values, where each row corresponds to a gene. If you do not specify output arguments, this function displays a histogram bar plot of the range.

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

`exprprofvar(..., 'ShowHist', ShowHistValue)`, 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	<pre>FASTAData = fastaread(File) [Header, Sequence] = fastaread(File) ... = fastaread(File, ...'IgnoreGaps', IgnoreGapsValue, ...) ... = fastaread(File, ...'Blockread', BlockreadValue, ...)</pre>	
Arguments	<i>File</i>	FASTA-formatted file (ASCII text file). Enter a file name, a path and file name, or a URL pointing to a file. <i>File</i> can also be a MATLAB character array that contains the text for a file name.
	<i>IgnoreGapsValue</i>	Property to control removing gap symbols. Enter either true or false (default).
	<i>BlockreadValue</i>	Property to control reading a single entry or block of entries from a file containing multiple sequences. Enter a scalar <i>N</i> , to read the <i>N</i> th entry in the file. Enter a 1-by-2 vector [<i>M1</i> , <i>M2</i>], to read the block of entries starting at entry <i>M1</i> and ending at entry <i>M2</i> . To read all remaining entries in the file starting at entry <i>M1</i> , enter a positive value for <i>M1</i> and enter Inf for <i>M2</i> .
Return Values	<i>FASTAData</i>	MATLAB structure with the fields Header and Sequence.
Description	fastaread reads data from a FASTA-formatted file into a MATLAB structure with the following fields.	

Field
Header
Sequence

A file with a FASTA format begins with a right angle bracket (>) and a single line description. Following this description is the sequence as a series of lines with fewer than 80 characters. Sequences are expected to use the standard IUB/IUPAC amino acid and nucleotide letter codes.

For a list of codes, see `aminolookup` and `baselookup`.

`FASTAData = fastaread(File)` reads a file with a FASTA format and returns the data in a structure. `FASTAData.Header` is the header information, while `FASTAData.Sequence` is the sequence stored as a string of letters.

`[Header, Sequence] = fastaread(File)` reads data from a file into separate variables. If the file contains more than one sequence, then header and sequence are cell arrays of header and sequence information.

`... = fastaread(File, ...'PropertyName', PropertyValue, ...)` calls `fastaread` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each `PropertyName` must be enclosed in single quotation marks and is case insensitive. The property name/value pairs can be in any format supported by the function `set` (for example, name-value string pairs, structures, and name-value cell array pairs). These property name/property value pairs are as follows:

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

`... = fastaread(File, ...'Blockread', BlockreadValue, ...)` lets you read in a single entry or block of entries from a file containing multiple sequences. If `BlockreadValue` is a scalar N, then `fastaread` reads the Nth entry in the file. If `BlockreadValue` is a 1-by-2 vector `[M1, M2]`, then `fastaread` reads the block of entries starting at entry `M1` and

fastaread

ending at entry *M2*. To read all remaining entries in the file starting at entry *M1*, enter a positive value for *M1* and enter *Inf* for *M2*.

Examples

Read the sequence for the human p53 tumor gene.

```
p53nt = fastaread('p53nt.txt')
```

Read the sequence for the human p53 tumor protein.

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

Read the human mitochondrion genome in FASTA format.

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

See Also

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

Purpose

Write to file using FASTA format

Syntax

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

Arguments

<i>File</i>	String specifying either a file name or a path and file name for saving the FASTA-formatted data. If you specify only a file name, the file is saved to the MATLAB Current Directory.
<i>Data</i>	Any of the following: <ul style="list-style-type: none">• String containing a sequence• MATLAB structure containing the fields <code>Header</code> and <code>Sequence</code>• MATLAB structure containing sequence information from the GenBank or GenPept database, such as returned by <code>genbankread</code>, <code>getgenbank</code>, <code>genpeptread</code>, or <code>getgenpept</code>.
<i>Header</i>	String or name of variable containing information about the sequence. This text will be included in the header of the FASTA-formatted file, <i>File</i> .
<i>Sequence</i>	String or name of variable containing an amino acid or nucleotide sequence using the standard IUB/IUPAC letter or integer codes. For a list of valid characters, see Amino Acid Lookup on page 2-91 or Nucleotide Lookup on page 2-102.

Description

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

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

Tip To append FASTA-formatted data to an existing file, simply specify that file name, and the data will be added to the end of the file.

If you are using `fastawrite` in a script, you can disable the append warning message by entering the following command lines before the `fastawrite` command:

```
warnState = warning %Save the current warning state
warning('off', 'Bioinfo:fastawrite:AppendToFile');
```

Then entering the following command line after the `fastawrite` command:

```
warning(warnState) %Reset warning state to previous settings
```

Examples

Writing a Coding Region to a FASTA-Formatted File

- 1 Retrieve the sequence for the human p53 gene from the GenBank database.

```
seq = getgenbank('NM_000546');
```

- 2 Find the CDS line in the FEATURES information.

```
cdsline = strmatch('CDS', seq.Features)
```

```
cdsline =
```

```
23
```

- 3 Read the coordinates of the coding region in the CDS line.

```
[start, stop] = strread(seq.Features(cdsline, :), '%*s%d..%d')
```

```
start =
```

```
252
```

```
stop =
```

```
1433
```

- 4 Extract the coding region.

```
codingSeq = seq.Sequence(start:stop);
```

- 5 Write the coding region to a FASTA-formatted file, specifying Coding region for p53 for the Header in the file, and p53coding.txt for the file name.

```
fastawrite('p53coding.txt','Coding region for p53',codingSeq);
```

Saving Multiple Sequences to a FASTA-Formatted File

- 1 Write two nucleotide sequences to a MATLAB structure containing the fields Header and Sequence.

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

- 2 Write the sequences to a FASTA-formatted file, specifying my_sequences.txt for the file name.

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

- 3 Display the FASTA-formatted file, my_sequences.txt.

```
type('my_sequences.txt')
```

```
>First sequence  
ACACAGGAAA
```

```
>Second sequence
```

```
ACGTCAGGTC
```

Appending Sequences to a FASTA-Formatted File

1 If you haven't already done so, create the FASTA-formatted file, `my_sequences.txt`, described in Saving Multiple Sequences to a FASTA-Formatted File on page 2-331.

2 Append a third sequence to the file.

```
fastawrite('my_sequences.txt', 'Third sequence', 'TACTGACTTC')
```

3 Display the FASTA-formatted file, `my_sequences.txt`.

```
type('my_sequences.txt')
```

```
>First sequence  
ACACAGGAAA
```

```
>Second sequence  
ACGTCAGGTC
```

```
>Third sequence  
TACTGACTTC
```

See Also

Bioinformatics Toolbox functions: `fastaread`, `genbankread`, `genpeptread`, `getgenbank`, `getgenpept`, `multialignwrite`, `seqtool`

Purpose

Draw linear or circular map of features from GenBank structure

Syntax

```
featuresmap(GBStructure)
featuresmap(GBStructure, FeatList)
featuresmap(GBStructure, FeatList, Levels)
featuresmap(GBStructure, Levels)
[Handles, OutFeatList] = featuresmap(...)
featuresmap(..., 'FontSize', FontSizeValue, ...)
featuresmap(..., 'ColorMap', ColorMapValue, ...)
featuresmap(..., 'Qualifiers', QualifiersValue, ...)
featuresmap(..., 'ShowPositions', ShowPositionsValue, ...)
```

Arguments

GBStructure

GenBank structure, typically created using the `getgenbank` or the `genbankread` function.

FeatList

Cell array of features (from the list of all features in the GenBank structure) to include in or exclude from the map.

- If *FeatList* is a cell array of features, these features are mapped. Any features in *FeatList* not found in the GenBank structure are ignored.
- If *FeatList* includes '-' as the first string in the cell array, then the remaining strings (features) are not mapped.

By default, *FeatList* is the a list of all features in the GenBank structure.

featuresmap

<i>Levels</i>	Vector of N integers, where N is the number of features. Each integer represents the level in the map for the corresponding feature. For example, if <i>Levels</i> = [1, 1, 2, 3, 3], the first two features would appear on level 1, the third feature on level 2, and the fourth and fifth features on level 3. By default, <i>Levels</i> = [1:N].
<i>FontSizeValue</i>	Scalar that sets the font size (points) for the annotations of the features. Default is 9.
<i>ColorMapValue</i>	Three-column matrix, to specify a list of colors to use for each feature. This matrix replaces the default matrix, which specifies the following colors and order: blue, green, red, cyan, magenta, yellow, brown, light green, orange, purple, gold, and silver. In the matrix, each row corresponds to a color, and each column specifies red, green, and blue intensity respectively. Valid values for the RGB intensities are 0.0 to 1.0.

QualifiersValue

Cell array of strings to specify an ordered list of qualifiers to search for in the structure and use as annotations. For each feature, the first matching qualifier found from the list is used for its annotation. If a feature does not include any of the qualifiers, no annotation displays for that feature. By default, *QualifiersValue* = {'gene', 'product', 'locus_tag', 'note', 'db_xref', 'protein_id'}. Provide your own *QualifiersValue* to limit or expand the list of qualifiers or change the search order.

Tip Set *QualifiersValue* = {} to create a map with no annotations.

Tip To determine all qualifiers available for a given feature, do either of the following:

- Create the map, and then click a feature or its annotation to list all qualifiers for that feature.
 - Use the `featuresparse` command to parse all the features into a new structure, and then use the `fieldnames` command to list the qualifiers for a specific feature. See [Determining Qualifiers for a Specific Feature](#) on page 2-341.
-

ShowPositionsValue

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

Description

`featuresmap(GBStructure)` creates a linear or circular map of all features from a GenBank structure, typically created using the `getgenbank` or the `genbankread` function.

`featuresmap(GBStructure, FeatList)` creates a linear or circular map of a subset of features from a GenBank structure. *FeatList* lets you specify features (from the list of all features in the GenBank structure) to include in or exclude from the map.

- If *FeatList* is a cell array of features, these features are mapped. Any features in *FeatList* not found in the GenBank structure are ignored.
- If *FeatList* includes ' - ' as the first string in the cell array, then the remaining strings (features) are not mapped.

By default, *FeatList* is a list of all features in the GenBank structure.

`featuresmap(GBStructure, FeatList, Levels)` or `featuresmap(GBStructure, Levels)` indicates which level on the map each feature is drawn. Level 1 is the left-most (linear map) or inner-most (circular map) level, and level N is the right-most (linear map) or outer-most (circular map) level, where N is the number of features.

Levels is a vector of N integers, where N is the number of features. Each integer represents the level in the map for the corresponding feature. For example, if *Levels* = [1, 1, 2, 3, 3], the first two features would appear on level 1, the third feature on level 2, and the fourth and fifth features on level 3. By default, *Levels* = [1:N].

`[Handles, OutFeatList] = featuresmap(...)` returns a list of handles for each feature in *OutFeatList*. It also returns *OutFeatList*, which is a cell array of the mapped features.

Tip Use *Handles* and *OutFeatList* with the `legend` command to create a legend of features.

`featuresmap(..., 'PropertyName', PropertyValue, ...)` defines optional properties that use property name/value pairs in any order. These property name/value pairs are as follows:

`featuresmap(..., 'FontSize', FontSizeValue, ...)` sets the font size (points) for the annotations of the features. Default *FontSizeValue* is 9.

`featuresmap(..., 'ColorMap', ColorMapValue, ...)` specifies a list of colors to use for each feature. This matrix replaces the default matrix, which specifies the following colors and order: blue, green, red, cyan, magenta, yellow, brown, light green, orange, purple, gold, and silver. *ColorMapValue* is a three-column matrix, where each row corresponds to a color, and each column specifies red, green, and blue intensity respectively. Valid values for the RGB intensities are 0.0 to 1.0.

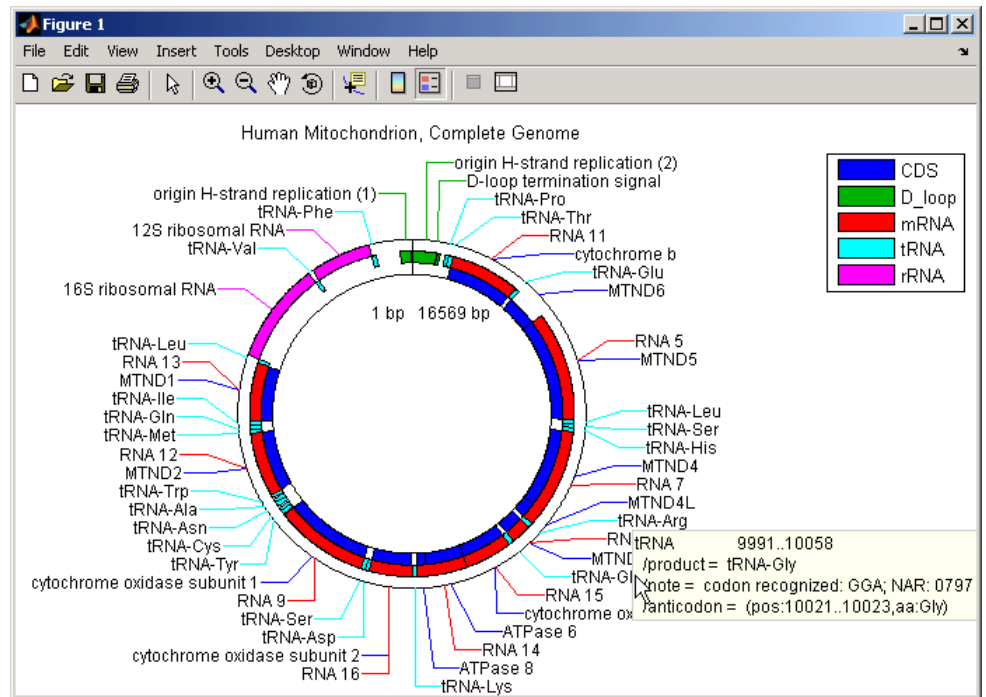
`featuresmap(..., 'Qualifiers', QualifiersValue, ...)` lets you specify an ordered list of qualifiers to search for and use as annotations. For each feature, the first matching qualifier found from the list is used for its annotation. If a feature does not include any of the qualifiers, no annotation displays for that feature. *QualifiersValue* is a cell array of strings. By default, *QualifiersValue* = {'gene', 'product', 'locus_tag', 'note', 'db_xref', 'protein_id'}. Provide your own *QualifiersValue* to limit or expand the list of qualifiers or change the search order.

Tip Set *QualifiersValue* = {} to create a map with no annotations.

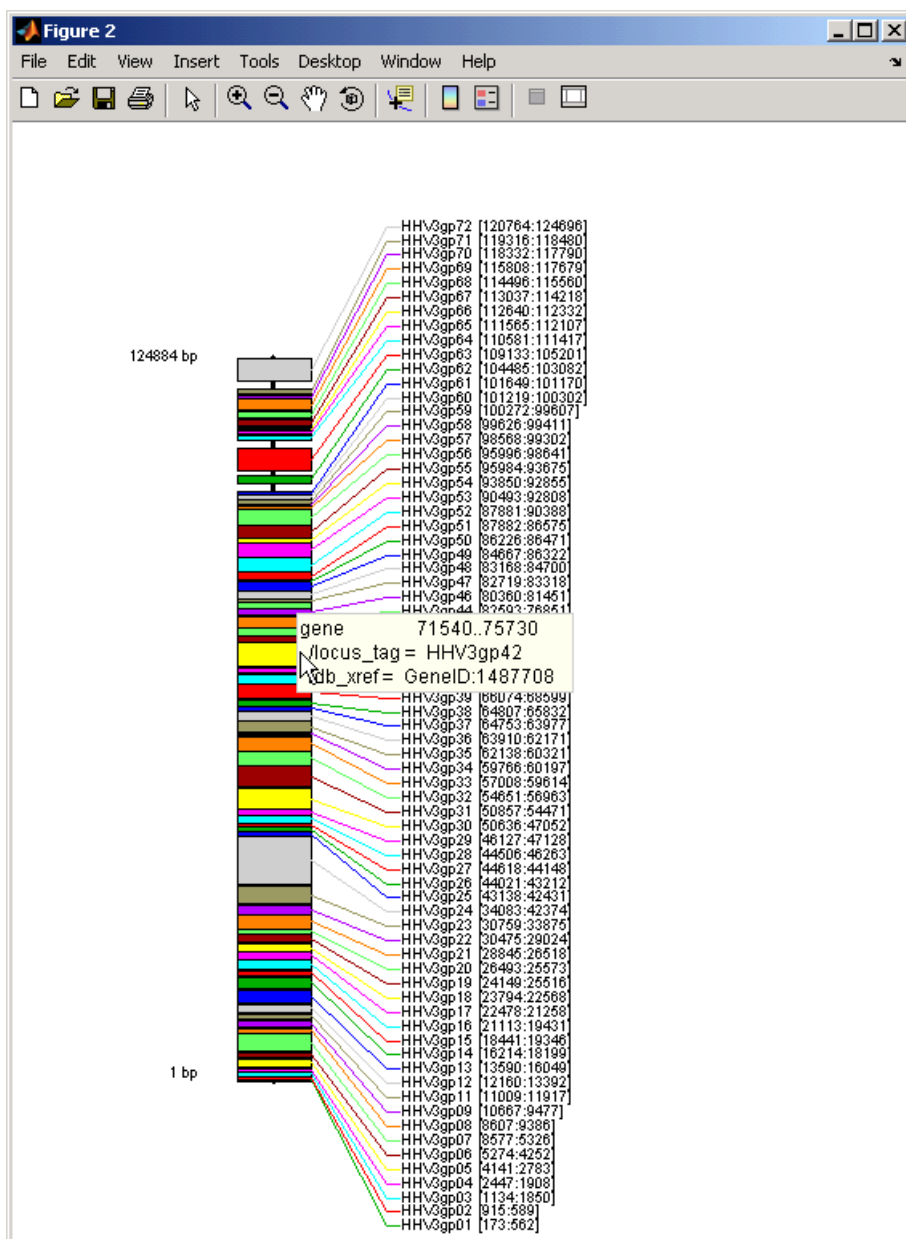
Tip To determine all qualifiers available for a given feature, do either of the following:

- Create the map, and then click a feature or its annotation to list all qualifiers for that feature.
 - Use the `featuresparse` command to parse all the features into a new structure, and then use the `fieldnames` command to list the qualifiers for a specific feature. See [Determining Qualifiers for a Specific Feature](#) on page 2-341.
-

`featuresmap(..., 'ShowPositions', ShowPositionsValue, ...)`
lets you add the sequence position to the annotation label. If *ShowPositionsValue* is `true`, sequence positions are added to the annotation labels. Default is `false`.

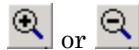


featuresmap



After creating a map:

- Click a feature or annotation to display a list of all qualifiers for that feature.
- Zoom the plot by clicking the following buttons:



Examples

Creating a Circular Map with a Legend

The following example creates a circular map of five different features mapped on three levels. It also uses outputs from the `featuresmap` function as inputs to the `legend` function to add a legend to the map.

```
GBStructure = getgenbank('J01415');
[Handles, OutFeatList] = featuresmap(GBStructure, ...
    {'CDS', 'D_loop', 'mRNA', 'tRNA', 'rRNA'}, [1 2 2 2 3])
legend(Handles, OutFeatList, 'interpreter', 'none', ...
    'location', 'bestoutside')
title('Human Mitochondrion, Complete Genome')
```

Creating a Linear Map with Sequence Position Labels and Changed Font Size

The following example creates a linear map showing only the gene feature. It changes the font of the labels to seven points and includes the sequence position in the labels.

```
herpes = getgenbank('NC_001348');
featuresmap(herpes, {'gene'}, 'fontsize', 7, 'showpositions', true)
title('Genes in Human herpesvirus 3 (strain Dumas)')
```

Determining Qualifiers for a Specific Feature

The following example uses the `getgenbank` function to create a GenBank structure, `GBStructure`. It then uses the `featuresparse` function to parse the features in the GenBank structure into a new

featuresmap

structure, features. It then uses the `fieldnames` function to return all qualifiers for one of the features, `D_loop`.

```
GenBankStructure = getgenbank('J01415');
features = featuresparse (GenBankStructure)
features =

    source: [1x1 struct]
    D_loop: [1x2 struct]
    rep_origin: [1x3 struct]
    repeat_unit: [1x4 struct]
    misc_signal: [1x1 struct]
    misc_RNA: [1x1 struct]
    variation: [1x17 struct]
    tRNA: [1x22 struct]
    rRNA: [1x2 struct]
    mRNA: [1x10 struct]
    CDS: [1x13 struct]
    conflict: [1x1 struct]

fieldnames(features.D_loop)

ans =

    'Location'
    'Indices'
    'note'
    'citation'
```

See Also

`featuresparse`, `genbankread`, `getgenbank`, `seqtool`

Purpose

Parse features from GenBank, GenPept, or EMBL data

Syntax

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

Arguments

<i>Features</i>	Any of the following: <ul style="list-style-type: none"> • String containing GenBank, GenPept, or EMBL features • MATLAB character array including text describing GenBank, GenPept, or EMBL features • MATLAB structure with fields corresponding to GenBank, GenPept, or EMBL data, such as those returned by <code>genbankread</code>, <code>genpeptread</code>, <code>emblread</code>, <code>getgenbank</code>, <code>getgenpept</code>, or <code>getembl</code>
<i>FeatureValue</i>	Name of a feature contained in <i>Features</i> . When specified, <code>featuresparse</code> returns only the substructure that corresponds to this feature. If there are multiple features with the same <i>FeatureValue</i> , then <i>FeatStruct</i> is an array of structures.
<i>SequenceValue</i>	Property to control the extraction, when possible, of the sequences respective to each feature, joining and complementing pieces of the source sequence and storing them in the <code>Sequence</code> field of the returned structure, <i>FeatStruct</i> . When extracting the sequence from an incomplete CDS feature, <code>featuresparse</code> uses the <code>codon_start</code> qualifier to adjust the frame of the sequence. Choices are <code>true</code> or <code>false</code> (default).

featuresparse

Return Values

FeatStruct Output structure containing a field for every database feature. Each field name in *FeatStruct* matches the corresponding feature name in the GenBank, GenPept, or EMBL database, with the exceptions listed in the table below. Fields in *FeatStruct* contain substructures with feature qualifiers as fields. In the GenBank, GenPept, and EMBL databases, for each feature, the only mandatory qualifier is its location, which `featuresparse` translates to the field `Location`. When possible, `featuresparse` also translates this location to numeric indices, creating an `Indices` field.

Note If you use the `Indices` field to extract sequence information, you may need to complement the sequences.

Description

FeatStruct = `featuresparse(Features)` parses the features from *Features*, which contains GenBank, GenPept, or EMBL features. *Features* can be a:

- String containing GenBank, GenPept, or EMBL features
- MATLAB character array including text describing GenBank, GenPept, or EMBL features
- MATLAB structure with fields corresponding to GenBank, GenPept, or EMBL data, such as those returned by `genbankread`, `genpeptread`, `emblread`, `getgenbank`, `getgenpept`, or `getembl`

FeatStruct is the output structure containing a field for every database feature. Each field name in *FeatStruct* matches the corresponding

feature name in the GenBank, GenPept, or EMBL database, with the following exceptions.

Feature Name in GenBank, GenPept, or EMBL Database	Field Name in MATLAB Structure
-10_signal	minus_10_signal
-35_signal	minus_35_signal
3'UTR	three_prime_UTR
3'clip	three_prime_clip
5'UTR	five_prime_UTR
5'clip	five_prime_clip
D-loop	D_loop

Fields in *FeatStruct* contain substructures with feature qualifiers as fields. In the GenBank, GenPept, and EMBL databases, for each feature, the only mandatory qualifier is its location, which *featuresparse* translates to the field *Location*. When possible, *featuresparse* also translates this location to numeric indices, creating an *Indices* field.

Note If you use the *Indices* field to extract sequence information, you may need to complement the sequences.

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

FeatStruct = *featuresparse*(*Features*, ...'*Feature*', *FeatureValue*, ...) returns only the substructure that corresponds to *FeatureValue*,

the name of a feature contained in *Features*. If there are multiple features with the same *FeatureValue*, then *FeatStruct* is an array of structures.

FeatStruct = `featuresparse(Features, ... 'Sequence', SequenceValue, ...)` controls the extraction, when possible, of the sequences respective to each feature, joining and complementing pieces of the source sequence and storing them in the field *Sequence*. When extracting the sequence from an incomplete CDS feature, `featuresparse` uses the `codon_start` qualifier to adjust the frame of the sequence. Choices are `true` or `false` (default).

Examples

Obtaining All Features from a GenBank File

The following example obtains all the features stored in the GenBank file `nm175642.txt`:

```
gbkStruct = genbankread('nm175642.txt');
features = featuresparse(gbkStruct)

features =

    source: [1x1 struct]
    gene: [1x1 struct]
    CDS: [1x1 struct]
```

Obtaining a Subset of Features from a GenBank Record

The following example obtains only the coding sequences (CDS) feature of the *Caenorhabditis elegans* cosmid record (accession number Z92777) from the GenBank database:

```
worm = getgenbank('Z92777');
CDS = featuresparse(worm, 'feature', 'cds')

CDS =

    1x12 struct array with fields:
        Location
```

```
Indices
locus_tag
standard_name
note
codon_start
product
protein_id
db_xref
translation
```

Extracting Sequences for Each Feature

- 1 Retrieve two nucleotide sequences from the GenBank database for the neuraminidase (NA) protein of two strains of the Influenza A virus (H5N1).

```
hk01 = getgenbank('AF509094');
vt04 = getgenbank('DQ094287');
```

- 2 Extract the sequence of the coding region for the neuraminidase (NA) protein from the two nucleotide sequences. The sequences of the coding regions are stored in the Sequence fields of the returned structures, hk01_cds and vt04_cds.

```
hk01_cds = featureparse(hk01, 'feature', 'CDS', 'Sequence', true);
vt04_cds = featureparse(vt04, 'feature', 'CDS', 'Sequence', true);
```

- 3 Once you have extracted the nucleotide sequences, you can use the nt2aa and nwalignment functions to align the amino acid sequences converted from the nucleotide sequences.

```
[sc, al] = nwalignment(nt2aa(hk01_cds), nt2aa(vt04_cds), 'extendgap', 1);
```

- 4 Then you can use the seqinsertgaps function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

```
hk01_aligned = seqinsertgaps(hk01_cds, al(1,:))
```

```
vt04_aligned = seqinsertgaps(vt04_cds,al(3,:))
```

- 5 Once you have code aligned the two sequences, you can use them as input to other functions such as `dnds`, which calculates the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences. By setting `Verbose` to `true`, you can also display the codons considered in the computations and their amino acid translations.

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

See Also

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

Purpose Read microarray data from GenePix array list file

Syntax `GALData = galread('File')`

Arguments *File* GenePix array list formatted file (GAL). Enter a file name, or enter a path and file name.

Description `galread` reads data from a GenePix formatted file into a MATLAB structure.

`GALData = galread('File')` reads in a GenePix array list formatted file (*File*) and creates a structure (`GALData`) containing the following fields.

Field
Header
BlockData
IDs
Names

The field `BlockData` is an N-by-3 array. The columns of this array are the block data, the column data, and the row data respectively. For more information on the GAL format, see

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

For a list of supported file format versions, see

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

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

Purpose

Perform GC Robust Multi-array Average (GCRMA) background adjustment, quantile normalization, and median-polish summarization on Affymetrix microarray probe-level data

Syntax

```
ExpressionMatrix = gcrma(PMatrix, MMMatrix, ProbeIndices,  
    AffinPM, AffinMM)  
ExpressionMatrix = gcrma(PMatrix, MMMatrix, ProbeIndices,  
    SequenceMatrix)  
ExpressionMatrix = gcrma(..., 'ChipIndex',  
    ChipIndexValue, ...)  
ExpressionMatrix = gcrma(..., 'OpticalCorr',  
    OpticalCorrValue, ...)  
ExpressionMatrix = gcrma(..., 'CorrConst', CorrConstValue,  
    ...)  
ExpressionMatrix = gcrma(..., 'Method', MethodValue, ...)  
ExpressionMatrix = gcrma(..., 'TuningParam',  
    TuningParamValue, ...)  
ExpressionMatrix = gcrma(..., 'GSBCorr', GSBCorrValue, ...)  
ExpressionMatrix = gcrma(..., 'Normalize', NormalizeValue,  
    ...)  
ExpressionMatrix = gcrma(..., 'Verbose', VerboseValue, ...)
```

Arguments

PMMatrix

Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)

Tip You can use the `PMIntensities` matrix returned by the `celintensityread` function.

MMMatrix

Matrix of intensity values where each row corresponds to a mismatch (MM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)

Tip You can use the `MMIntensities` matrix returned by the `celintensityread` function.

ProbeIndices

Column vector containing probe indices. Probes within a probe set are numbered 0 through $N - 1$, where N is the number of probes in the probe set.

Tip You can use the `affyprobeseqread` function to generate this column vector.

AffinPM

Column vector of PM probe affinities.

Tip You can use the `affyprobeaffinities` function to generate this column vector.

AffinMM

Column vector of MM probe affinities.

Tip You can use the `affyprobeaffinities` function to generate this column vector.

SequenceMatrix An N -by-25 matrix of sequence information for the perfect match (PM) probes on the Affymetrix GeneChip array, where N is the number of probes on the array. Each row corresponds to a probe, and each column corresponds to one of the 25 sequence positions. Nucleotides in the sequences are represented by one of the following integers:

- 0 — None
- 1 — A
- 2 — C
- 3 — G
- 4 — T

Tip You can use the `affyprobeseqread` function to generate this matrix. If you have this sequence information in letter representation, you can convert it to integer representation using the `nt2int` function.

ChipIndexValue Positive integer specifying a column index in *MMMatrix*, which specifies a chip. This chip intensity data is used to compute probe affinities. Default is 1.

OpticalCorrValue Controls the use of optical background correction on the PM and MM intensity values in *PMMatrix* and *MMMatrix*. Choices are `true` (default) or `false`.

<i>CorrConstValue</i>	Value that specifies the correlation constant, rho, for background intensity for each PM/MM probe pair. Choices are any value ≥ 0 and ≤ 1 . Default is 0.7.
<i>MethodValue</i>	String that specifies the method to estimate the signal. Choices are MLE, a faster, ad hoc Maximum Likelihood Estimate method, or EB, a slower, more formal, empirical Bayes method. Default is MLE.
<i>TuningParamValue</i>	Value that specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

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

<i>GSBCorrValue</i>	Specifies whether to perform gene-specific binding (GSB) correction using probe affinity data. Choices are <code>true</code> (default) or <code>false</code> . If there is no probe affinity information, this property is ignored.
<i>NormalizeValue</i>	Controls whether quantile normalization is performed on background adjusted data. Choices are <code>true</code> (default) or <code>false</code> .
<i>VerboseValue</i>	Controls the display of a progress report showing the number of each chip as it is completed. Choices are <code>true</code> (default) or <code>false</code> .

Return Values

ExpressionMatrix Matrix of \log_2 expression values where each row corresponds to a gene (probe set) and each column corresponds to an Affymetrix CEL file, which represents a single chip.

Description

ExpressionMatrix = `gcrma(PMMatrix, MMMatrix, ProbeIndices, AffinPM, AffinMM)` performs GCRMA background adjustment, quantile normalization, and median-polish summarization on Affymetrix microarray probe-level data using probe affinity data. *ExpressionMatrix* is a matrix of \log_2 expression values where each row corresponds to a gene (probe set) and each column corresponds to an Affymetrix CEL file, which represents a single chip.

Note There is no column in *ExpressionMatrix* that contains probe set or gene information.

ExpressionMatrix = `gcrma(PMMatrix, MMMatrix, ProbeIndices, SequenceMatrix)` performs GCRMA background adjustment, quantile normalization, and Robust Multi-array Average (RMA) summarization on Affymetrix microarray probe-level data using probe sequence data to compute probe affinity data. *ExpressionMatrix* is a matrix of \log_2 expression values where each row corresponds to a gene (probe set) and each column corresponds to an Affymetrix CEL file, which represents a single chip.

Note If *AffinPM* and *AffinMM* affinity data and *SequenceMatrix* sequence data are not available, you can still use the `gcrma` function by entering an empty matrix for these inputs in the syntax.

ExpressionMatrix = `gcrma(...'PropertyName', PropertyValue, ...)` calls `gcrma` with optional properties that use property

name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

ExpressionMatrix = `gcrma(..., 'ChipIndex', ChipIndexValue, ...)` computes probe affinities from MM probe intensity data from the chip with the specified column index in *MMMatrix*. Default *ChipIndexValue* is 1. If *AffinPM* and *AffinMM* affinity data are provided, this property is ignored.

ExpressionMatrix = `gcrma(..., 'OpticalCorr', OpticalCorrValue, ...)` controls the use of optical background correction on the PM and MM intensity values in *PMMatrix* and *MMMatrix*. Choices are true (default) or false.

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

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

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

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

ExpressionMatrix = `gcrma(..., 'GSBCorr', GSBCorrValue, ...)` specifies whether to perform gene specific binding (GSB) correction using

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

`ExpressionMatrix = gcrma(..., 'Normalize', NormalizeValue, ...)` controls whether quantile normalization is performed on background adjusted data. Choices are `true` (default) or `false`.

`ExpressionMatrix = gcrma(..., 'Verbose', VerboseValue, ...)` controls the display of a progress report showing the number of each chip as it is completed. Choices are `true` (default) or `false`.

Examples

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

```
load prostatecancerrawdata
```

- 2 Compute the Affymetrix PM and MM probe affinities from their sequences and MM probe intensities.

```
[apm, amm] = affyprobeaffinities(seqMatrix, mmMatrix(:,1),...  
                                'ProbeIndices', probeIndices);
```

- 3 Perform GCRMA background adjustment, quantile normalization, and Robust Multi-array Average (RMA) summarization on the Affymetrix microarray probe-level data and create a matrix of expression values.

```
expdata = gcrma(pmMatrix, mmMatrix, probeIndices, seqMatrix);
```

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

References

- [1] Wu, Z., Irizarry, R.A., Gentleman, R., Murillo, F.M., and Spencer, F. (2004). A Model Based Background Adjustment for Oligonucleotide Expression Arrays. *Journal of the American Statistical Association* 99(468), 909–917.
- [2] Wu, Z., and Irizarry, R.A. (2005). Stochastic Models Inspired by Hybridization Theory for Short Oligonucleotide Arrays. *Proceedings of RECOMB 2004. J Comput Biol.* 12(6), 882–93.
- [3] Wu, Z., and Irizarry, R.A. (2005). A Statistical Framework for the Analysis of Microarray Probe-Level Data. *Johns Hopkins University, Biostatistics Working Papers* 73.
- [4] Speed, T. (2006). Background models and GCRMA. Lecture 10, *Statistics 246, University of California Berkeley*.
<http://www.stat.berkeley.edu/users/terry/Courses/s246.2006/Week10/Week10L1.pdf>.
- [5] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. *Clinical Cancer Research* 11, 6823–6834.

See Also

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

Purpose

Perform GC Robust Multi-array Average (GCRMA) background adjustment on Affymetrix microarray probe-level data using sequence information

Syntax

```
PMatrix_Adj = gcrmabackadj(PMatrix, MMMatrix, AffinPM,  
                          AffinMM)  
[PMatrix_Adj, nsbStruct] = gcrmabackadj(PMatrix,  
                                         MMMatrix,  
                                         AffinPM, AffinMM)  
... = gcrmabackadj( ...'OpticalCorr',  
                  OpticalCorrValue, ...)  
... = gcrmabackadj( ...'CorrConst', CorrConstValue, ...)  
... = gcrmabackadj( ...'Method', MethodValue, ...)  
... = gcrmabackadj( ...'TuningParam',  
                  TuningParamValue, ...)  
... = gcrmabackadj( ...'AddVariance',  
                  AddVarianceValue, ...)  
... = gcrmabackadj( ...'Showplot', ShowplotValue, ...)  
... = gcrmabackadj( ...'Verbose', VerboseValue, ...)
```

Arguments

<i>PMMatrix</i>	Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)
	<hr/> Tip You can use the <code>PMIntensities</code> matrix returned by the <code>celintensityread</code> function. <hr/>
<i>MMMMatrix</i>	Matrix of intensity values where each row corresponds to a mismatch (MM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)
	<hr/> Tip You can use the <code>MMIntensities</code> matrix returned by the <code>celintensityread</code> function. <hr/>
<i>AffinPM</i>	Column vector of PM probe affinities, such as returned by the <code>affyprobeaffinities</code> function. Each row corresponds to a probe.
<i>AffinMM</i>	Column vector of MM probe affinities, such as returned by the <code>affyprobeaffinities</code> function. Each row corresponds to a probe.
<i>OpticalCorrValue</i>	Controls the use of optical background correction on the PM and MM probe intensity values in <i>PMMatrix</i> and <i>MMMMatrix</i> . Choices are <code>true</code> (default) or <code>false</code> .

<i>CorrConstValue</i>	Value that specifies the correlation constant, rho, for log background intensity for each PM/MM probe pair. Choices are any value ≥ 0 and ≤ 1 . Default is 0.7.
<i>MethodValue</i>	String that specifies the method to estimate the signal. Choices are MLE, a faster, ad hoc Maximum Likelihood Estimate method, or EB, a slower, more formal, empirical Bayes method. Default is MLE.
<i>TuningParamValue</i>	Value that specifies the tuning parameter used by the estimate method. This tuning parameter sets the lower bound of signal values with positive probability. Choices are a positive value. Default is 5 (MLE) or 0.5 (EB).

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

<i>AddVarianceValue</i>	Controls whether the signal variance is added to the weight function for smoothing low signal edge. Choices are true or false (default).
-------------------------	--

ShowplotValue Controls the display of a plot showing the \log_2 of probe intensity values from a specified column (chip) in *MMMatrix*, versus probe affinities in *AffinMM*. Choices are *true*, *false*, or *I*, an integer specifying a column in *MMMatrix*. If set to *true*, the first column in *MMMatrix* is plotted. Default is:

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

VerboseValue Controls the display of a progress report showing the number of each chip as it is completed. Choices are *true* (default) or *false*.

Return Values

PMatrix_Adj Matrix of background adjusted PM (perfect match) intensity values.

nsbStruct Structure containing nonspecific binding background parameters, estimated from the intensities and affinities of probes on an Affymetrix GeneChip array. *nsbStruct* includes the following fields:

- *sigma*
- *mu_pm*
- *mu_mm*

Description

PMatrix_Adj = `gcrmabackadj(PMatrix, MMMatrix, AffinPM, AffinMM)` performs GCRMA background adjustment (including optical background correction and nonspecific binding correction) on Affymetrix microarray probe-level data, using probe sequence information and returns *PMatrix_Adj*, a matrix of background adjusted PM (perfect match) intensity values.

Note If *AffinPM* and *AffinMM* data are not available, you can still use the `gcrmabackadj` function by entering empty column vectors for both of these inputs in the syntax.

`[PMMatrix_Adj, nsbStruct] = gcrmabackadj(PMMatrix, MMMatrix, AffinPM, AffinMM)` returns *nsbStruct*, a structure containing nonspecific binding background parameters, estimated from the intensities and affinities of probes on an Affymetrix GeneChip array. *nsbStruct* includes the following fields:

- `sigma`
- `mu_pm`
- `mu_mm`

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

`... = gcrmabackadj(...'OpticalCorr', OpticalCorrValue, ...)` controls the use of optical background correction on the PM and MM probe intensity values in *PMMatrix* and *MMMatrix*. Choices are `true` (default) or `false`.

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

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

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

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

... = gcrmabackadj(...'AddVariance', *AddVarianceValue*, ...) controls whether the signal variance is added to the weight function for smoothing low signal edge. Choices are true or false (default).

... = gcrmabackadj(...'Showplot', *ShowplotValue*, ...) controls the display of a plot showing the \log_2 of probe intensity values from a specified column (chip) in *MMMatrix*, versus probe affinities in *AffinMM*. Choices are true, false, or *I*, an integer specifying a column in *MMMatrix*. If set to true, the first column in *MMMatrix* is plotted. Default is:

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

... = gcrmabackadj(...'Verbose', *VerboseValue*, ...) controls the display of a progress report showing the number of each chip as it is completed. Choices are true (default) or false.

Examples

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

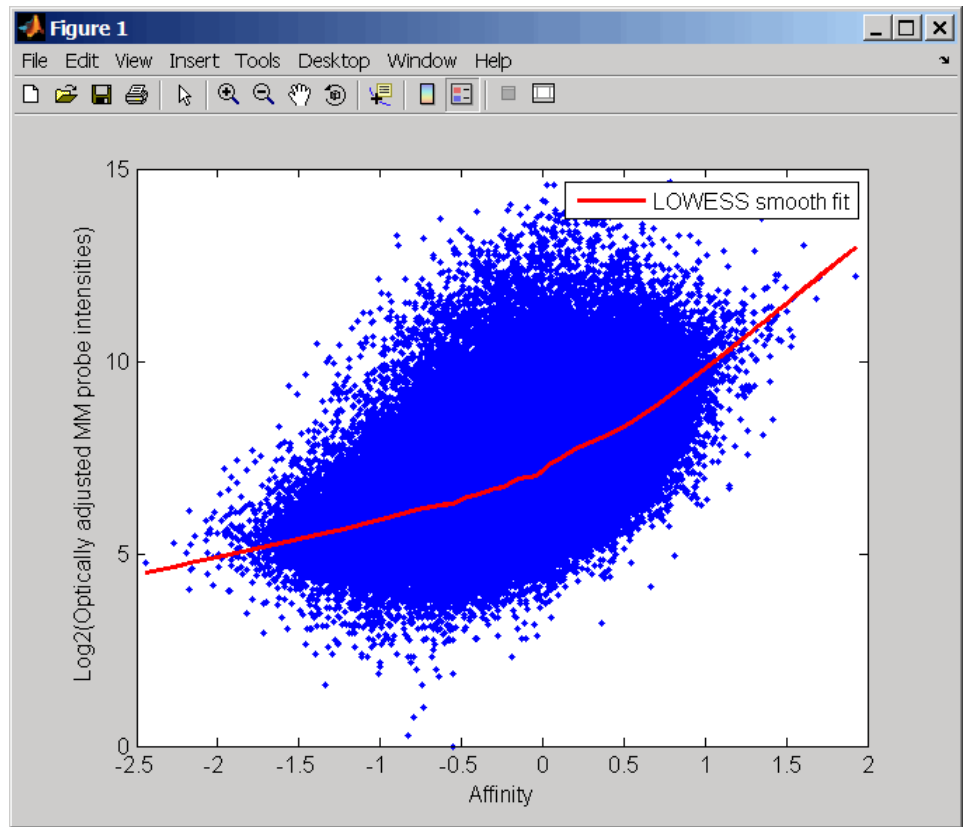
```
load prostatecancerrawdata
```

- 2** Compute the Affymetrix PM and MM probe affinities from their sequences and MM probe intensities.

```
[apm, amm] = affyprobeaffinities(seqMatrix, mmMatrix(:,1),...  
                                'ProbeIndices', probeIndices);
```

- 3** Perform GCRMA background adjustment on the Affymetrix microarray probe-level data, creating a matrix of background adjusted PM intensity values. Also, display a plot showing the \log_2 of probe intensity values from column 3 (chip 3) in `mmMatrix`, versus probe affinities in `amm`.

```
pms_adj = gcrmabackadj(pmMatrix, mmMatrix, apm, amm, 'showplot', 3);
```



- 4 Perform GCRMA background adjustment again, using the slower, more formal, empirical Bayes method.

```
pms_adj2 = gcrmabackadj(pmMatrix, mmMatrix, apm, amm, 'method', 'EB');
```

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

References

- [1] Wu, Z., Irizarry, R.A., Gentleman, R., Murillo, F.M., and Spencer, F. (2004). A Model Based Background Adjustment for Oligonucleotide

Expression Arrays. *Journal of the American Statistical Association* 99(468), 909–917.

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

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

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

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

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

See Also

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

genbankread

Purpose Read data from GenBank file

Syntax `GenBankData = genbankread(File)`

Arguments

File Either of the following:

- String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a GenBank-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Directory.
- MATLAB character array that contains the text of a GenBank-formatted file.

GenBankData MATLAB structure with fields corresponding to GenBank keywords.

Description

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

Examples

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

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

```
s =
```

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



```

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

```

2 Display the source organism for this sequence.

```

s.SourceOrganism

ans =

Homo sapiens
Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
Mammalia; Eutheria; Euarchontoglires; Primates; Haplorrhini;
Catarrhini; Hominidae; Homo.

```

See Also

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

geneentropyfilter

Purpose Remove genes with low entropy expression values

Syntax

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

Arguments

<i>Data</i>	DataMatrix object or numeric matrix where each row corresponds to the experimental results for one gene. Each column is the results for all genes from one experiment.
<i>Names</i>	Cell array with the name of a gene for each row of experimental data. <i>Names</i> has same number of rows as <i>Data</i> with each row containing the name or ID of the gene in the data set.
<i>PercentileValue</i>	Property to specify a percentile below which gene data is removed. Enter a value from 0 to 100.

Description *Mask* = geneentropyfilter(*Data*) identifies gene expression profiles in *Data* with entropy values less than the 10th percentile.

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

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

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

Note If *Data* is a *DataMatrix* object with specified row names, you do not need to provide the second input *Names* to return the third output *FNames*.

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

Examples

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

References

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

See Also

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

genelowvalfilter

Purpose Remove gene profiles with low absolute values

Syntax

```
Mask = genelowvalfilter(Data)
[Mask, FData] = genelowvalfilter(Data)
[Mask, FData, FNames] = genelowvalfilter(Data, Names)
genelowvalfilter(..., 'Percentile', PercentileValue, ...)
genelowvalfilter(..., 'AbsValue', AbsValueValue, ...)
genelowvalfilter(..., 'AnyVal', AnyValValue, ...)
```

Arguments

<i>Data</i>	DataMatrix object or numeric matrix where each row corresponds to the experimental results for one gene. Each column is the results for all genes from one experiment.
<i>Names</i>	Cell array with the same number of rows as <i>Data</i> . Each row contains the name or ID of the gene in the data set.
<i>PercentileValue</i>	Property to specify a percentile below which gene expression profiles are removed. Enter a value from 0 to 100.
<i>AbsValueValue</i>	Property to specify an absolute value below which gene expression profiles are removed.
<i>AnyValValue</i>	Property to select the minimum or maximum absolute value for comparison with <i>AbsValueValue</i> . If <i>AnyValValue</i> is true, selects the minimum absolute value. If <i>AnyValValue</i> is false, selects the maximum absolute value. The default value is false.

Description Gene expression profile experiments have data where the absolute values are very low. The quality of this type of data is often bad due to large quantization errors or simply poor spot hybridization.

Mask = genelowvalfilter(*Data*) identifies gene expression profiles in *Data* with all absolute values less than the 10th percentile.

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

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

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

Note If *Data* is a `DataMatrix` object with specified row names, you do not need to provide the second input *Names* to return the third output *FNames*.

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

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

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

`genelowvalfilter(..., 'AnyVal', AnyValValue, ...)`, when *AnyValValue* is true, calculates the minimum absolute value for

genelowvalfilter

each gene expression profile and removes the profiles with minimum absolute values less than *AnyValValue*.

Examples

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

References

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

See Also

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

Purpose	Create geneont object
Syntax	<pre><i>GeneontObj</i> = geneont <i>GeneontObj</i> = geneont('File', <i>FileValue</i>) <i>GeneontObj</i> = geneont('Live', <i>LiveValue</i>) <i>GeneontObj</i> = geneont('Live', <i>LiveValue</i>, 'ToFile', <i>ToFileValue</i>)</pre>
Arguments	<p><i>FileValue</i> String specifying the file name of an OBO-formatted file that is on the MATLAB search path.</p> <p><i>LiveValue</i> Controls the creation of the most up-to-date geneont object. Enter <code>true</code> to create <i>GeneontObj</i>, a geneont object, from the most recent version of the Gene Ontology database. Default is <code>false</code>.</p> <p><i>ToFileValue</i> String specifying a file name or path and file name to which to save the contents of the current version of the Gene Ontology database.</p>
Return Values	<p><i>GeneontObj</i> MATLAB object containing gene ontology information.</p>
Description	<p><i>GeneontObj</i> = <code>geneont</code> creates <i>GeneontObj</i>, a geneont object, from the <code>gene_ontology.obo</code> file in the MATLAB current directory.</p> <p><i>GeneontObj</i> = <code>geneont('File', <i>FileValue</i>)</code> creates <i>GeneontObj</i>, a geneont object, from <i>FileValue</i>, a string specifying the file name of an OBO-formatted file that is on the MATLAB search path.</p> <p><i>GeneontObj</i> = <code>geneont('Live', <i>LiveValue</i>)</code> controls the creation of <i>GeneontObj</i>, a geneont object, from the current version of the Gene Ontology database, which is the file at:</p> <p style="text-align: center;">http://www.geneontology.org/ontology/gene_ontology.obo</p>

Choices are true or false (default).

Note The full Gene Ontology database may take several minutes to download when you run this function using the 'Live' property.

GeneontObj = `geneont('Live', LiveValue, 'ToFile', ToFileValue)`, when *LiveValue* is true, creates *GeneontObj*, a geneont object, from the most recent version of the Gene Ontology database, which is the file at:

`http://www.geneontology.org/ontology/gene_ontology.obo`

and saves the contents of this file to *ToFileValue*, a string specifying a file name or a path and file name.

Examples

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

```
GO = geneont('LIVE', true)
```

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

```
Gene Ontology object with 24316 Terms.
```

- 2 Display information about the geneont object.

```
get(GO)
```

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


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

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

22x1 struct array with fields:

```
id  
name  
ontology  
definition  
comment  
synonym  
is_a  
part_of  
obsolete
```

See Also

Bioinformatics Toolbox functions: `goannotread`, `num2goid`

Bioinformatics Toolbox object: `geneont` object

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

generangefilter

Purpose Remove gene profiles with small profile ranges

Syntax

```
Mask = generangefilter(Data)
[Mask, FData] = generangefilter(Data)
[Mask, FData, FNames] = generangefilter(Data, Names)
generangefilter(..., 'Percentile', PercentileValue, ...)
generangefilter(..., 'AbsValue', AbsValueValue, ...)
generangefilter(..., 'LogPercentile', LogPercentileValue,
    ...)
generangefilter(..., 'LogValue', LogValueValue, ...)
```

Arguments

<i>Data</i>	DataMatrix object or numeric matrix where each row corresponds to the experimental results for one gene. Each column is the results for all genes from one experiment.
<i>Names</i>	Cell array with the name of a gene for each row of experimental data. <i>Names</i> has same number of rows as <i>Data</i> with each row containing the name or ID of the gene in the data set.
<i>PercentileValue</i>	Property to specify a percentile below which gene expression profiles are removed. Enter a value from 0 to 100.
<i>AbsValueValue</i>	Property to specify an absolute value below which gene expression profiles are removed.
<i>LogPercentileValue</i>	Property to specify the logarithm of a percentile.
<i>LogValueValue</i>	Property to specify the logarithm of an absolute value.

Description *Mask* = generangefilter(*Data*) calculates the range for each gene expression profile in *Data*, a DataMatrix object or matrix of the

experimental data, and then identifies the expression profiles with ranges less than the 10th percentile.

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

`[Mask, FData] = generangefilter(Data)` returns *FData*, a filtered data matrix. You can also create *FData* using `FData = Data(Mask,:)`.

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

Note If *Data* is a `DataMatrix` object with specified row names, you do not need to provide the second input *Names* to return the third output *FNames*.

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

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

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

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

generangefilter

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

Examples

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

References

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

See Also

Bioinformatics Toolbox functions: `exprprofrange`, `exprprofvar`, `geneentropyfilter`, `genelowvalfilter`, `genevarfilter`

Purpose Return nucleotide codon to amino acid mapping for genetic code

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

Arguments *GeneticCode* Integer or string specifying a genetic code number or code name from the table Genetic Code on page 2-382. Default is 1 or 'Standard'.

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

Return Values *Map* Structure containing the mapping of nucleotide codons to amino acids for the standard genetic code. The *Map* structure contains a field for each nucleotide codon.

Description *Map* = geneticcode returns a structure containing the mapping of nucleotide codons to amino acids for the standard genetic code. The *Map* structure contains a field for each nucleotide codon.

Map = geneticcode(*GeneticCode*) returns a structure containing the mapping of nucleotide codons to amino acids for the specified genetic code. *GeneticCode* is either:

- An integer or string specifying a code number or code name from the table Genetic Code on page 2-382
- The `transl_table` (code) number from the NCBI Web page describing genetic codes:

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

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

Genetic Code

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

Examples

Return the mapping of nucleotide codons to amino acids for the Flatworm Mitochondrial genetic code.

```
wormmap = geneticcode('Flatworm Mitochondrial');
```

References

[1] NCBI Web page describing genetic codes:

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

See Also

Bioinformatics Toolbox functions: `aa2nt`, `aminolookup`, `baselookup`, `codonbias`, `dnds`, `dndsm1`, `nt2aa`, `revgeneticcode`, `seqshoworfs`, `seqtool`

genevarfilter

Purpose Filter genes with small profile variance

Syntax

```
Mask = genevarfilter(Data)
[Mask, FData] = genevarfilter(Data)
[Mask, FData, FNames] = genevarfilter(Data, Names)
genevarfilter(..., 'Percentile', PercentileValue, ...)
genevarfilter(..., 'AbsValue', AbsValueValue, ...)
```

Arguments

<i>Data</i>	DataMatrix object or numeric matrix where each row corresponds to a gene. If a matrix, the first column is the names of the genes, and each additional column is the results from an experiment.
<i>Names</i>	Cell array with the name of a gene for each row of experimental data. <i>Names</i> has same number of rows as <i>Data</i> with each row containing the name or ID of the gene in the data set.
<i>PercentileValue</i>	Specifies a percentile below which gene expression profiles are removed. Enter a value from 0 to 100.
<i>AbsValueValue</i>	Property to specify an absolute value below which gene expression profiles are removed.

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

Mask = genevarfilter(*Data*) calculates the variance for each gene expression profile in *Data* and then identifies the expression profiles with a variance less than the 10th percentile.

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

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

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

Note If *Data* is a `DataMatrix` object with specified row names, you do not need to provide the second input *Names* to return the third output *FNames*.

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

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

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

Examples

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

References

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

See Also

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

genpeptread

Purpose Read data from GenPept file

Syntax GenPeptData = genpeptread('File')

Arguments *File* GenPept formatted file (ASCII text file). Enter a file name, a path and file name, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text of a GenPept file.

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

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

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

Field
LocusName
LocusSequenceLength
LocusMoleculeType
LocusGenBankDivision
LocusModificationDate
Definition

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

Examples

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

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

genpeptread

See Also

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

Purpose Read Gene Expression Omnibus (GEO) Series (GSE) format data

Syntax `GEOData = geoseriesread(File)`

Arguments

File Either of the following:

- String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a Gene Expression Omnibus (GEO) Series (GSE) format file. If you specify only a file name, that file must be on the MATLAB search path or in the current directory.
- MATLAB character array that contains the text of a GEO Series (GSE) format file.

Tip You can use the `getgeodata` function with the 'ToFile' property to retrieve GEO Series (GSE) format data from the GEO database and create a GEO Series (GSE) format file.

Return Values

GEOData MATLAB structure containing the following fields:

- **Header** — Header text from the GEO Series (GSE) format file, typically containing a description of the data or experiment information.
- **Data** — `DataMatrix` object containing the data from a GEO Series (GSE) format file. The columns and rows of the `DataMatrix` object correspond to the sample IDs and Ref IDs, respectively, from the GEO Series (GSE) format file.

geoseriesread

Description

`GEOData = geoseriesread(File)` reads a Gene Expression Omnibus (GEO) Series (GSE) format file, and then creates a MATLAB structure, `GEOData`, with the following fields.

Fields	Description
Header	Header text from the GEO Series (GSE) format file, typically containing a description of the data or experiment information.
Data	<code>DataMatrix</code> object containing the data from a GEO Series (GSE) format file. The columns and rows of the <code>DataMatrix</code> object correspond to the sample IDs and Ref IDs, respectively, from the GEO Series (GSE) format file.

Examples

- 1 Retrieve Series (GSE) data from the GEO Web site and save it to a file.

```
geodata = getgeodata('GSE11287','ToFile','GSE11287.txt');
```

- 2 In a subsequent MATLAB session, you can access the Series (GSE) data from your local file, instead of retrieving it from the GEO Web site.

```
geodata = geoseriesread('GSE11287.txt')
```

```
geodata =
```

```
Header: [1x1 struct]
```

```
Data: [45101x6 bioma.data.DataMatrix]
```

- 3 Access the sample IDs using the `colnames` property of a `DataMatrix` object.

```
sampleIDs = geodata.Data.colnames
```

```
sampleIDs =
```

'GSM284935' 'GSM284936' 'GSM284937' 'GSM284938' 'GSM284939' 'GSM284940'

See Also

Bioinformatics Toolbox functions: `affyread`, `agferead`, `galread`, `geosoftread`, `getgeodata`, `gprread`, `ilmnbsread`, `sptread`

Bioinformatics Toolbox object: `DataMatrix` object

geosoftread

Purpose Read Gene Expression Omnibus (GEO) SOFT format data

Syntax `GEOSOFTData = geosoftread(File)`

Arguments *File* Either of the following:

- String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a Gene Expression Omnibus (GEO) SOFT format Sample file (GSM), Data Set file (GDS), or Platform (GPL) file. If you specify only a file name, that file must be on the MATLAB search path or in the current directory.
- MATLAB character array that contains the text of a GEO SOFT format file.

Tip You can use the `getgeodata` function with the 'ToFile' property to retrieve GEO SOFT format data from the GEO database and create a GEO SOFT format file.

Return Values *GEOSOFTData* MATLAB structure containing information from a GEO SOFT format file.

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

Fields	Description
Scope	Type of file read (SAMPLE, DATASET, or PLATFORM)
Accession	Accession number for record in GEO database.
Header	Microarray experiment information.
ColumnDescriptions	Cell array containing descriptions of columns in the data.
ColumnNames	Cell array containing names of columns in the data.
Data	Array containing microarray data.
Identifier (GDS files only)	Cell array containing probe IDs.
IDRef (GDS files only)	Cell array containing indices to probes.

Note Currently, the `geosoftread` function supports Sample (GSM), Data Set (GDS), and Platform (GPL) records.

Examples

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

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

Use `geosoftread` to read a local copy of the GSM file, instead of accessing it from the GEO Web site.

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

```
geodata =
```

```

      Scope: 'SAMPLE'
Accession: 'GSM3258'
```

geosoftread

```
Header: [1x1 struct]
ColumnDescriptions: {6x1 cell}
ColumnNames: {6x1 cell}
Data: {5355x6 cell}
```

Read the GDS file for photosynthesis in proteobacteria.

```
gdsdata = geosoftread('GDS329.soft')
```

```
gdsdata =
```

```
Scope: 'DATASET'
Accession: 'GDS329'
Header: [1x1 struct]
ColumnDescriptions: {6x1 cell}
ColumnNames: {6x1 cell}
IDRef: {5355x1 cell}
Identifier: {5355x1 cell}
Data: [5355x6 double]
```

See Also

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

Purpose

Retrieve BLAST report from NCBI Web site

Syntax

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

Arguments

<i>RID</i>	Request ID for the NCBI BLAST report, such as returned by the <code>blastncbi</code> function.
<i>DescriptionsValue</i>	Integer that specifies the number of descriptions in a report. Choices are any value ≥ 1 and ≤ 500 . Default is 100.
<i>AlignmentsValue</i>	Integer that specifies the number of alignments to include in the report. Choices are any value ≥ 1 and ≤ 500 . Default is 50.
	<hr/> Note This value must be \leq the value you specified for the 'Alignments' property when creating <i>RID</i> using the <code>blastncbi</code> function. <hr/>
<i>ToFileValue</i>	String specifying a file name for saving report data.

<i>FileFormatValue</i>	String specifying the format of the file. Choices are 'text' (default) or 'html'.
<i>WaitTimeValue</i>	Positive value that specifies a time (in minutes) for the MATLAB software to wait for a report from the NCBI Web site to be available. If the report is still not available after the wait time, <code>getblast</code> returns an error message. Default behavior is to not wait for a report.

Tip Use the *RTOE* returned by the `blastncbi` function as the *WaitTimeValue*.

Return Values

<i>Data</i>	MATLAB structure or array of structures (if multiple query sequences) containing fields corresponding to BLAST keywords and data from an NCBI BLAST report.
-------------	---

Description

The Basic Local Alignment Search Tool (BLAST) offers a fast and powerful comparative analysis of protein and nucleotide sequences against known sequences in online databases. `getblast` parses NCBI BLAST reports, including `blastn`, `blastp`, `psiblast`, `blastx`, `tblastn`, `tblastx`, and `megablast` reports.

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

`Data = getblast(RID, ... 'PropertyName', PropertyValue, ...)` calls `getblast` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each

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

Data = getblast(RID, ...'Descriptions', DescriptionsValue, ...) specifies the number of descriptions in a report. Choices are any integer ≥ 1 and ≤ 500 . Default is 100.

Data = getblast(RID, ...'Alignments', AlignmentsValue, ...) specifies the number of alignments to include in the report. Choices are any integer ≥ 1 and ≤ 500 . Default is 50.

Note This value must be \leq the value you specified for the 'Alignments' property when creating *RID* using the `blastncbi` function.

Data = getblast(RID, ...'ToFile', ToFileValue, ...) saves the NCBI BLAST report data to a specified file. The default format for the file is 'text', but you can specify 'html' with the 'FileFormat' property.

Data = getblast(RID, ...'FileFormat', FileFormatValue, ...) specifies the format for the report. Choices are 'text' (default) or 'html'.

Data = getblast(RID, ...'WaitTime', WaitTimeValue, ...) pauses the MATLAB software and waits a specified time (in minutes) for a report from the NCBI Web site to be available. If the report is still unavailable after the wait time, `getblast` returns an error message. Choices are any positive value. Default behavior is to not wait for a report.

Tip Use the *RTOE* returned by the `blastncbi` function as the *WaitTimeValue*.

For more information about reading and interpreting BLAST reports, see:

<http://www.ncbi.nlm.nih.gov/Education/BLASTinfo/tut1.html>

Data contains the following fields.

Field	Description
RID	Request ID for retrieving results for a specific NCBI BLAST search.
Algorithm	NCBI algorithm used to do a BLAST search.
Query	Identifier of the query sequence submitted to a BLAST search.
Database	All databases searched.
Hits.Name	Name of a database sequence (subject sequence) that matched the query sequence.
Hits.Length	Length of a subject sequence.
Hits.HSPs.Score	Pairwise alignment score for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Expect	Expectation value for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.Identities	Identities (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.

Field	Description
Hits.HSPs.Positives	<p data-bbox="842 331 1332 487">Identical or similar residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject amino acid sequence.</p> <hr data-bbox="842 543 1332 546"/> <p data-bbox="842 557 1332 647">Note This field applies only to translated nucleotide or amino acid query sequences and/or databases.</p> <hr data-bbox="842 656 1332 659"/>
Hits.HSPs.Gaps	<p data-bbox="842 701 1332 821">Nonaligned residues (match, possible, and percent) for a high-scoring sequence pair between the query sequence and a subject sequence.</p>
Hits.HSPs.Frame	<p data-bbox="842 843 1332 963">Reading frame of the translated nucleotide sequence for a high-scoring sequence pair between the query sequence and a subject sequence.</p> <hr data-bbox="842 1019 1332 1022"/> <p data-bbox="842 1032 1332 1152">Note This field applies only when performing translated searches, that is, when using tblastx, tblastn, and blastx.</p> <hr data-bbox="842 1161 1332 1164"/>

Field	Description
Hits.HSPs.Strand	<p>Sense (Plus = 5' to 3' and Minus = 3' to 5') of the DNA strands for a high-scoring sequence pair between the query sequence and a subject sequence.</p> <hr/> <p>Note This field applies only when using a nucleotide query sequence and database.</p> <hr/>
Hits.HSPs.Alignment	Three-row matrix showing the alignment for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.QueryIndices	Indices of the query sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
Hits.HSPs.SubjectIndices	Indices of the subject sequence residue positions for a high-scoring sequence pair between the query sequence and a subject sequence.
Statistics	Summary of statistical details about the performed search, such as lambda values, gap penalties, number of sequences searched, and number of hits.

Examples

- 1 Create an NCBI BLAST report request using a GenPept accession number.

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



```
RID =
```

```
'1175088155-31624-126008617054.BLASTQ3'
```

- 2 Pass the Request ID for the report to the `getblast` function to parse the report, and return the report data in a MATLAB structure, and save the report data to a text file.

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

```
reportStruct =
```

```
      RID: '1175093633-2786-174709873694.BLASTQ3'  
  Algorithm: 'BLASTP 2.2.16 [Mar-11-2007]'  
      Query: [1x63 char]  
   Database: [1x96 char]  
       Hits: [1x50 struct]  
 Statistics: [1x1034 char]
```

Note You may need to wait for the report to become available on the NCBI Web site before you can run the preceding command.

References

- [1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W. and Lipman, D.J. (1990). Basic local alignment search tool. *J. Mol. Biol.* *215*, 403–410.
- [2] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* *25*, 3389–3402.

For more information about reading and interpreting NCBI BLAST reports, see:

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

getblast

See Also

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

Purpose	Retrieve sequence information from EMBL database						
Syntax	<pre>EMBLData = getembl(AccessionNumber) EMBLData = getembl(..., 'ToFile', ToFileValue, ...) EMBLSeq = getembl(..., 'SequenceOnly', SequenceOnlyValue, ...)</pre>						
Arguments	<table><tr><td><i>AccessionNumber</i></td><td>Unique identifier for a sequence record. Enter a unique combination of letters and numbers.</td></tr><tr><td><i>ToFileValue</i></td><td>String specifying a file name or a path and file name to which to save the data. If you specify only a file name, the file is stored in the current directory.</td></tr><tr><td><i>SequenceOnlyValue</i></td><td>Controls the retrieving of only the sequence without the metadata. Choices are true or false (default).</td></tr></table>	<i>AccessionNumber</i>	Unique identifier for a sequence record. Enter a unique combination of letters and numbers.	<i>ToFileValue</i>	String specifying a file name or a path and file name to which to save the data. If you specify only a file name, the file is stored in the current directory.	<i>SequenceOnlyValue</i>	Controls the retrieving of only the sequence without the metadata. Choices are true or false (default).
<i>AccessionNumber</i>	Unique identifier for a sequence record. Enter a unique combination of letters and numbers.						
<i>ToFileValue</i>	String specifying a file name or a path and file name to which to save the data. If you specify only a file name, the file is stored in the current directory.						
<i>SequenceOnlyValue</i>	Controls the retrieving of only the sequence without the metadata. Choices are true or false (default).						
Return Values	<table><tr><td><i>EMBLData</i></td><td>MATLAB structure with fields corresponding to EMBL data.</td></tr><tr><td><i>EMBLSeq</i></td><td>MATLAB character string representing the sequence.</td></tr></table>	<i>EMBLData</i>	MATLAB structure with fields corresponding to EMBL data.	<i>EMBLSeq</i>	MATLAB character string representing the sequence.		
<i>EMBLData</i>	MATLAB structure with fields corresponding to EMBL data.						
<i>EMBLSeq</i>	MATLAB character string representing the sequence.						
Description	<p>getembl retrieves information from the European Molecular Biology Laboratory (EMBL) database for nucleotide sequences. This database is maintained by the European Bioinformatics Institute (EBI). For more details about the EMBL database, see</p> <p>http://www.ebi.ac.uk/embl/Documentation/index.html</p> <p><i>EMBLData</i> = getembl(<i>AccessionNumber</i>) searches for the accession number in the EMBL database (http://www.ebi.ac.uk/embl) and returns <i>EMBLData</i>, a MATLAB structure with fields corresponding to</p>						

the EMBL two-character line type code. Each line type code is stored as a separate element in the structure.

EMBLData contains the following fields.

Field
Identification
Accession
SequenceVersion
DateCreated
DateUpdated
Description
Keyword
OrganismSpecies
OrganismClassification
Organelle
Reference
DatabaseCrossReference
Comments
Assembly
Feature
BaseCount
Sequence

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

EMBLData = `getembl(..., 'ToFile', ToFileValue, ...)` saves the information to an EMBL-formatted file. *ToFileValue* is a string specifying a file name or a path and file name to which to save the data. If you specify only a file name, the file is stored in the current directory.

Tip Read an EMBL-formatted file back into the MATLAB software using the `emblread` function.

EMBLSeq = `getembl(..., 'SequenceOnly', SequenceOnlyValue, ...)` controls the retrieving of only the sequence without the metadata. Choices are `true` or `false` (default).

Examples

Retrieve data for the rat liver apolipoprotein A-I.

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

Retrieve data for the rat liver apolipoprotein A-I and save it to the file `rat_protein`. If you specify a file name without a path, the file is stored in the current directory.

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

Retrieve only the sequence for the rat liver apolipoprotein A-I.

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

See Also

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

getgenbank

Purpose Retrieve sequence information from GenBank database

Syntax

```
Data = getgenbank(AccessionNumber)
getgenbank(AccessionNumber)
getgenbank(..., 'PartialSeq', PartialSeqValue, ...)
getgenbank(..., 'ToFile', ToFileValue, ...)
getgenbank(..., 'FileFormat', FileFormatValue, ...)
getgenbank(..., 'SequenceOnly', SequenceOnlyValue, ...)
```

Arguments

<i>AccessionNumber</i>	String specifying a unique alphanumeric identifier for a sequence record.
<i>PartialSeqValue</i>	Two-element array of integers containing the start and end positions of the subsequence [<i>StartBP</i> , <i>EndBP</i>] that specifies a subsequence to retrieve. <i>StartBP</i> is an integer between 1 and <i>EndBP</i> ; <i>EndBP</i> is an integer between <i>StartBP</i> and the length of the sequence.
<i>ToFileValue</i>	String specifying either a file name or a path and file name for saving the GenBank data. If you specify only a file name, the file is saved to the MATLAB Current Directory.
<i>FileFormatValue</i>	String specifying the format for the file specified with the 'ToFile' property. Choices are 'GenBank' (default) or 'FASTA'.
<i>SequenceOnlyValue</i>	Controls the return of only the sequence as a character array. Choices are true or false (default).

Description getgenbank retrieves nucleotide information from the GenBank database. This database is maintained by the National Center for Biotechnology Information (NCBI). For more details about the GenBank database, see

<http://www.ncbi.nlm.nih.gov/Genbank/>

`Data = getgenbank(AccessionNumber)` searches for the accession number in the GenBank database and returns a MATLAB structure containing information for the sequence.

Tip If an error occurs while retrieving the GenBank-formatted information, try rerunning the query. Errors can occur due to Internet connectivity issues that are unrelated to the GenBank record.

`getgenbank(AccessionNumber)` displays the information in the MATLAB Command Window without returning data to a variable. The displayed information includes hyperlinks to the URLs used to search for and retrieve the data.

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

`getgenbank(..., 'PartialSeq', PartialSeqValue, ...)` returns the specified subsequence in the `Sequence` field of the MATLAB structure. *PartialSeqValue* is a two-element array of integers containing the start and end positions of the subsequence [*StartBP*, *EndBP*]. *StartBP* is an integer between 1 and *EndBP*; *EndBP* is an integer between *StartBP* and the length of the sequence.

`getgenbank(..., 'ToFile', ToFileValue, ...)` saves the data returned from the GenBank database to a file. *ToFileValue* is a string specifying either a file name or a path and file name for saving the GenBank data. If you specify only a file name, the file is saved to the MATLAB Current Directory.

Tip You can read a GenBank-formatted file back into MATLAB using the `genbankread` function.

Tip To append GenBank data to an existing file, specify that file name, and the data will be added to the end of the file.

If you are using `getgenbank` in a script, you can disable the append warning message by entering the following command lines before the `getgenbank` command:

```
warnState = warning %Save the current warning state
warning('off','Bioinfo:getncbidata:AppendToFile');
```

Then enter the following command line after the `getgenbank` command:

```
warning(warnState) %Reset warning state to previous settings
```

`getgenbank(..., 'FileFormat', FileFormatValue, ...)` returns the sequence in the specified format. Choices are 'GenBank' (default) or 'FASTA'.

`getgenbank(..., 'SequenceOnly', SequenceOnlyValue, ...)` controls the return of only the sequence as a character array. Choices are true or false (default).

Note When the 'SequenceOnly' and 'ToFile' properties are used together, the output is a FASTA-formatted file.

Examples

Retrieving an RNA Sequence

To retrieve the sequence from chromosome 19 that codes for the human insulin receptor and store it in a structure, `S`, in the MATLAB Command Window, type:

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

```
S =
```

```

        LocusName: 'HUMINSR'
    LocusSequenceLength: '4723'
    LocusNumberOfStrands: ''
        LocusTopology: 'linear'
    LocusMoleculeType: 'mRNA'
    LocusGenBankDivision: 'PRI'
    LocusModificationDate: '06-JAN-1995'
        Definition: 'Human insulin receptor mRNA, complete cds.'
        Accession: 'M10051'
        Version: 'M10051.1'
            GI: '186439'
        Project: []
        Keywords: 'insulin receptor; tyrosine kinase.'
        Segment: []
        Source: 'Homo sapiens (human)'
    SourceOrganism: [4x65 char]
    Reference: {[1x1 struct]}
        Comment: [14x67 char]
        Features: [51x74 char]
            CDS: [1x1 struct]
        Sequence: [1x4723 char]
        SearchURL: [1x105 char]
        RetrieveURL: [1x95 char]

```

Retrieving a Partial RNA Sequence

By looking at the `Features` field of the structure returned in `Retrieving an RNA Sequence` on page 2-409, you can determine that the coding

getgenbank

sequence is positions 139 through 4287. To retrieve only the coding sequence from chromosome 19 that codes for the human insulin receptor and store it in a structure, CDS, in the MATLAB Command Window, type:

```
CDS = getgenbank('M10051','PARTIALSEQ',[139,4287]);
```

See Also

Bioinformatics Toolbox functions: `genbankread`, `getembl`, `getgenpept`, `getpdb`, `seqtool`

Purpose

Retrieve sequence information from GenPept database

Syntax

```
Data = getgenpept(AccessionNumber)
getgenpept(AccessionNumber)
getgenpept(..., 'PartialSeq', PartialSeqValue, ...)
getgenpept(..., 'ToFile', ToFileValue, ...)
getgenpept(..., 'FileFormat', FileFormatValue, ...)
getgenpept(..., 'SequenceOnly', SequenceOnlyValue, ...)
```

Arguments

<i>AccessionNumber</i>	String specifying a unique alphanumeric identifier for a sequence record.
<i>PartialSeqValue</i>	Two-element array of integers containing the start and end positions of the subsequence [<i>StartAA</i> , <i>EndAA</i>] that specifies a subsequence to retrieve. <i>StartAA</i> is an integer between 1 and <i>EndAA</i> ; <i>EndAA</i> is an integer between <i>StartAA</i> and the length of the sequence.
<i>ToFileValue</i>	String specifying either a file name or a path and file name for saving the GenPept data. If you specify only a file name, the file is saved to the MATLAB Current Directory.
<i>FileFormatValue</i>	String specifying the format for the file specified with the 'ToFile' property. Choices are 'Genpept' (default) or 'FASTA'.
<i>SequenceOnlyValue</i>	Controls the return of only the sequence as a character array. Choices are true or false (default).

Description

getgenpept retrieves a protein (amino acid) sequence information from the GenPept database, which is a translation of the nucleotide sequences in the GenBank database and is maintained by the National Center for Biotechnology Information (NCBI).

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

Data = `getgenpept(AccessionNumber)` searches for the accession number in the GenPept database and returns a MATLAB structure containing information for the sequence.

Tip If an error occurs while retrieving the GenPept-formatted information, try rerunning the query. Errors can occur due to Internet connectivity issues that are unrelated to the GenPept record.

`getgenpept(AccessionNumber)` displays the information in the MATLAB Command Window without returning data to a variable. The displayed information includes hyperlinks to the URLs used to search for and retrieve the data.

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

`getgenpept(..., 'PartialSeq', PartialSeqValue, ...)` returns the specified subsequence in the `Sequence` field of the MATLAB structure. *PartialSeqValue* is a two-element array of integers containing the start and end positions of the subsequence [*StartAA*, *EndAA*]. *StartAA* is an integer between 1 and *EndAA*; *EndAA* is an integer between *StartAA* and the length of the sequence.

`getgenpept(..., 'ToFile', ToFileValue, ...)` saves the data returned from the GenPept database to a file. *ToFileValue* is a string specifying either a file name or a path and file name for saving the

GenPept data. If you specify only a file name, the file is saved to the MATLAB Current Directory.

Tip You can read a GenPept-formatted file back into MATLAB using the `genpeptread` function.

Tip To append GenPept data to an existing file, specify that file name, and the data will be added to the end of the file.

If you are using `getgenpept` in a script, you can disable the append warning message by entering the following command lines before the `getgenpept` command:

```
warnState = warning %Save the current warning state
warning('off', 'Bioinfo:getncbidata:AppendToFile');
```

Then enter the following command line after the `getgenpept` command:

```
warning(warnState) %Reset warning state to previous settings
```

`getgenpept(..., 'FileFormat', FileFormatValue, ...)` returns the sequence in the specified format. Choices are 'GenPept' (default) or 'FASTA'.

`getgenpept(..., 'SequenceOnly', SequenceOnlyValue, ...)` controls the return of only the sequence as a character array. Choices are true or false (default).

Note When the 'SequenceOnly' and 'ToFile' properties are used together, the output is a FASTA-formatted file.

Examples

Retrieving a Peptide Sequence

To retrieve the sequence for the human insulin receptor and store it in a structure, `Seq`, in the MATLAB Command Window, type:

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

```
Seq =
```

```
          LocusName: 'AAA59174'  
    LocusSequenceLength: '1382'  
    LocusNumberofStrands: ''  
          LocusTopology: 'linear'  
    LocusMoleculeType: ''  
    LocusGenBankDivision: 'PRI'  
    LocusModificationDate: '06-JAN-1995'  
          Definition: 'insulin receptor precursor.'  
          Accession: 'AAA59174'  
          Version: 'AAA59174.1'  
             GI: '307070'  
          Project: []  
          DBSource: 'locus HUMINSR accession M10051.1'  
          Keywords: ''  
          Source: 'Homo sapiens (human)'  
    SourceOrganism: [4x65 char]  
          Reference: {[1x1 struct]}  
          Comment: [14x67 char]  
          Features: [40x64 char]  
          Sequence: [1x1382 char]  
          SearchURL: [1x104 char]  
          RetrieveURL: [1x92 char]
```

Retrieving a Partial Peptide Sequence

By looking at the `Features` field of the structure returned in `Retrieving a Peptide Sequence` on page 2-414, you can determine that the furin-like repeats domain is positions 234 through 281. To retrieve only the furin-like repeats domain from the sequence for the human insulin

receptor and store it in a structure, `Fur`, in the MATLAB Command Window, type:

```
Fur = getgenpept('AAA59174','PARTIALSEQ',[234,281]);
```

See Also

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

getgeodata

Purpose Retrieve Gene Expression Omnibus (GEO) format data

Syntax
`GEOData = getgeodata(AccessionNumber)`
`getgeodata(AccessionNumber, 'ToFile', ToFileValue)`

Arguments

<i>AccessionNumber</i>	String specifying a unique identifier for a GEO Sample (GSM), Data Set (GDS), Platform (GPL), or Series (GSE) record in the GEO database.
<i>ToFileValue</i>	String specifying a file name or path and file name for saving the data. If you specify only a file name, that file will be saved in the MATLAB Current Directory.

Return Values

<i>GEOData</i>	MATLAB structure containing information for a GEO record retrieved from the GEO database.
----------------	---

Description `GEOData = getgeodata(AccessionNumber)` searches the Gene Expression Omnibus database for the specified accession number of a Sample (GSM), Data Set (GDS), Platform (GPL), or Series (GSE) record and returns a MATLAB structure containing the following fields:

Field	Description
Scope	Type of data retrieved (SAMPLE, DATASET, PLATFORM, or SERIES)
Accession	Accession number for record in GEO database.
Header	Microarray experiment information.
ColumnDescriptions	Cell array containing descriptions of columns in the data.
ColumnNames	Cell array containing names of columns in the data.

Field	Description
Data	Array containing microarray data.
Identifier (GDS files only)	Cell array containing probe IDs.
IDRef (GDS files only)	Cell array containing indices to probes.

Note Currently, the `getgeodata` function supports Sample (GSM), Data Set (GDS), Platform (GPL), and Series (GSE) records.

`getgeodata(AccessionNumber, 'ToFile', ToFileValue)` saves the data returned from the database to a file.

Note You can read a GEO SOFT-formatted file back into the MATLAB software using the `geosoftread` function. You can read a GEO SERIES-formatted file back into the MATLAB software using the `geoseriesread` function.

For more information, see

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

Examples

```
geoStruct = getgeodata('GSM1768')
```

See Also

Bioinformatics Toolbox functions: `geoseriesread`, `geosoftread`, `getgenbank`, `getgenpept`

gethmmalignment

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

Syntax

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

Arguments

<i>PFAMName</i>	String specifying a protein family name (unique identifier) of an HMM profile record in the PFAM database. For example, 7tm_2.
<i>PFAMAccessNumber</i>	String specifying a protein family accession number of an HMM profile record in the PFAM database. For example, PF00002.
<i>PFAMNumber</i>	Integer specifying a protein family number of an HMM profile record in the PFAM database. For example, 2 is the protein family number for the protein family PF0002.
<i>ToFileValue</i>	String specifying a file name or a path and file name for saving the data. If you specify only a file name, that file will be saved in the MATLAB Current Directory.

<i>TypeValue</i>	String that specifies the set of alignments returned. Choices are: <ul style="list-style-type: none"> • full — Default. Returns all alignments that fit the HMM profile. • seed — Returns only the alignments used to generate the HMM profile.
<i>MirrorValue</i>	String that specifies a Web database. Choices are: <ul style="list-style-type: none"> • Sanger (default) • Janelia
<i>IgnoreGapsValue</i>	Controls the removal of the symbols - and . from the sequence. Choices are true or false (default).
Return Values	<i>AlignStruct</i> MATLAB structure array containing the multiple sequence alignment associated with an HMM profile.

Description

AlignStruct = `gethmmalignment(PFAMName)` searches the PFAM database for the HMM profile record represented by *PFAMName*, a protein family name, retrieves the multiple sequence alignment associated with the HMM profile, and returns *AlignStruct*, a MATLAB structure array, with each structure containing the following fields:

Field	Description
Header	Protein name
Sequence	Protein sequence

AlignStruct = `gethmmalignment(PFAMAccessNumber)` searches the PFAM database for the HMM profile record represented by *PFAMAccessNumber*, a protein family accession number, retrieves the

gethmmalignment

multiple sequence alignment associated with the HMM profile, and returns *AlignStruct*, a MATLAB structure array.

AlignStruct = `gethmmalignment(PFAMNumber)` determines a protein family accession number from *PFAMNumber*, an integer, searches the PFAM database for the associated HMM profile record, retrieves the multiple sequence alignment associated with the HMM profile, and returns *AlignStruct*, a MATLAB structure array.

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

AlignStruct = `gethmmalignment(..., 'ToFile', ToFileValue, ...)` saves the data returned from the PFAM database to a file specified by *ToFileValue*.

Note You can read a FASTA-formatted file containing PFAM data back into the MATLAB software using the `fastaread` function.

AlignStruct = `gethmmalignment(..., 'Type', TypeValue, ...)` specifies the set of alignments returned. Choices are:

- `full` — Default. Returns all sequences that fit the HMM profile.
- `seed` — Returns only the sequences used to generate the HMM profile.

AlignStruct = `gethmmalignment(..., 'Mirror', MirrorValue, ...)` specifies a Web database. Choices are:

- `Sanger` (default)
- `Janelia`

You can reach other mirror sites by passing the complete URL to the `fastaread` function.

Note These mirror sites are maintained separately and may have slight variations.

For more information about the PFAM database, see:

```
http://pfam.sanger.ac.uk
http://pfam.janelia.org/
```

`AlignStruct = gethmmalignment(..., 'IgnoreGaps', IgnoreGaps, ...)` controls the removal of the symbols - and . from the sequence. Choices are true or false (default).

Examples

To retrieve a multiple alignment of the sequences used to train the HMM profile for global alignment to the 7-transmembrane receptor protein in the secretin family, enter either of the following:

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

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

pfamalign =

32x1 struct array with fields:
    Header
    Sequence
```

See Also

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

gethmmprof

Purpose Retrieve hidden Markov model (HMM) profile from PFAM database

Syntax

```
HMMStruct = gethmmprof(PFAMName)
HMMStruct = gethmmprof(PFAMNumber)
HMMStruct = gethmmprof(..., 'ToFile', ToFileValue, ...)
HMMStruct = gethmmprof(..., 'Mode', ModeValue, ...)
HMMStruct = gethmmprof(..., 'Mirror', MirrorValue, ...)
```

Arguments

<i>PFAMName</i>	String specifying a protein family name (unique identifier) of an HMM profile record in the PFAM database. For example, 7tm_2.
<i>PFAMNumber</i>	Integer specifying a protein family number of an HMM profile record in the PFAM database. For example, 2 is the protein family number for the protein family PF00002.
<i>ToFileValue</i>	String specifying a file name or a path and file name for saving the data. If you specify only a file name, that file will be saved in the MATLAB Current Directory.
<i>ModeValue</i>	String that specifies the returned alignment mode. Choices are: <ul style="list-style-type: none">• 1s — Default. Global alignment mode.• fs — Local alignment mode.
<i>MirrorValue</i>	String that specifies a Web database. Choices are: <ul style="list-style-type: none">• Sanger (default)• Janelia

Return Values

<i>HMMStruct</i>	MATLAB structure containing information for an HMM profile retrieved from the PFAM database.
------------------	--

Description

HMMStruct = gethmmprof(*PFAMName*) searches the PFAM database for the record represented by *PFAMName*, a protein family name, retrieves the HMM profile information, and stores it in *HMMStruct*, a MATLAB structure containing the following fields corresponding to parameters of an HMM profile.

Field	Description
Name	The protein family name (unique identifier) of the HMM profile record in the PFAM database.
PfamAccessionNumber	The protein family accession number of the HMM profile record in the PFAM database.
ModelDescription	Description of the HMM profile.
ModelLength	The length of the profile (number of MATCH states).
Alphabet	The alphabet used in the model, 'AA' or 'NT'. Note AlphaLength is 20 for 'AA' and 4 for 'NT'.
MatchEmission	Symbol emission probabilities in the MATCH states. The format is a matrix of size ModelLength-by-AlphabetLength, where each row corresponds to the emission distribution for a specific MATCH state.
InsertEmission	Symbol emission probabilities in the INSERT state. The format is a matrix of size ModelLength-by-AlphabetLength, where each row corresponds to the emission distribution for a specific INSERT state.

Field	Description
NullEmission	<p>Symbol emission probabilities in the MATCH and INSERT states for the NULL model.</p> <p>The format is a 1-by-AlphaLength row vector.</p> <hr/> <p>Note NULL probabilities are also known as the background probabilities.</p> <hr/>
BeginX	<p>BEGIN state transition probabilities.</p> <p>Format is a 1-by-(ModelLength + 1) row vector:</p> <p>[B->D1 B->M1 B->M2 B->M3 ... B->Mend]</p>
MatchX	<p>MATCH state transition probabilities.</p> <p>Format is a 4-by-(ModelLength - 1) matrix:</p> <pre>[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]</pre>
InsertX	<p>INSERT state transition probabilities.</p> <p>Format is a 2-by-(ModelLength - 1) matrix:</p> <pre>[I1->M2 I2->M3 ... I[end-1]->Mend; I1->I1 I2->I2 ... I[end-1]->I[end-1]]</pre>
DeleteX	<p>DELETE state transition probabilities.</p> <p>Format is a 2-by-(ModelLength - 1) matrix:</p> <pre>[D1->M2 D2->M3 ... D[end-1]->Mend ; D1->D2 D2->D3 ... D[end-1]->Dend]</pre>

Field	Description
FlankingInsertX	Flanking insert states (N and C) used for LOCAL profile alignment. Format is a 2-by-2 matrix: [N->B C->T ; N->N C->C]
LoopX	Loop states transition probabilities used for multiple hits alignment. Format is a 2-by-2 matrix: [E->C J->B ; E->J J->J]
NullX	Null transition probabilities used to provide scores with log-odds values also for state transitions. Format is a 2-by-1 column vector: [G->F ; G->G]

HMMStruct = gethmmprof(*PFAMNumber*) determines a protein family accession number from *PFAMNumber*, an integer, searches the PFAM database for the associated record, retrieves the HMM profile information, and stores it in *HMMStruct*, a MATLAB structure.

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

HMMStruct = `gethmmprof(..., 'ToFile', ToFileValue, ...)`
saves the data returned from the PFAM database in a file specified
by *ToFileValue*.

Note You can read an HMM-formatted file back into the MATLAB
software using the `pfamhmmread` function.

HMMStruct = `gethmmprof(..., 'Mode', ModeValue, ...)` specifies
the returned alignment mode. Choices are:

- `ls` (default) — Global alignment mode.
- `fs` — Local alignment mode.

HMMStruct = `gethmmprof(..., 'Mirror', MirrorValue, ...)`
specifies a Web database. Choices are:

- Sanger (default)
- Janelia

You can reach other mirror sites by passing the complete URL to the
`pfamhmmread` function.

Note These mirror sites are maintained separately and may have
slight variations.

For more information about the PFAM database, see:

<http://pfam.sanger.ac.uk>
<http://pfam.janelia.org/>

For more information on HMM profile models, see “HMM Profile Model” on page 2-536.

Examples

To retrieve a hidden Markov model (HMM) profile for the global alignment of the 7-transmembrane receptor protein in the secretin family, enter either of the following:

```
hmm = gethmmprof(2)
```

```
hmm = gethmmprof('7tm_2')
```

```
hmm =
```

```
                Name: '7tm_2'  
PfamAccessionNumber: 'PF00002.14'  
ModelDescription: [1x42 char]  
    ModelLength: 296  
        Alphabet: 'AA'  
    MatchEmission: [296x20 double]  
    InsertEmission: [296x20 double]  
    NullEmission: [1x20 double]  
        BeginX: [297x1 double]  
        MatchX: [295x4 double]  
        InsertX: [295x2 double]  
        DeleteX: [295x2 double]  
    FlankingInsertX: [2x2 double]  
        LoopX: [2x2 double]  
        NullX: [2x1 double]
```

See Also

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

gethmmtree

Purpose Retrieve phylogenetic tree data from PFAM database

Syntax

```
Tree = gethmmtree(PFAMName)
Tree = gethmmtree(PFAMAccessionNumber)
Tree = gethmmtree(PFAMNumber)
Tree = gethmmtree(AccessionNumber, ...'ToFile',
ToFileValue, ...)
Tree = gethmmtree(AccessionNumber, ...'Type',
TypeValue, ...)
```

Arguments

<i>PFAMName</i>	String specifying a protein family name (unique identifier) of an HMM profile record in the PFAM database. For example, 7tm_2.
<i>PFAMAccessionNumber</i>	String specifying a protein family accession number of an HMM profile record in the PFAM database. For example, PF00002.
<i>PFAMNumber</i>	Integer specifying a protein family number of an HMM profile record in the PFAM database. For example, 2 is the protein family number for the protein family PF00002.
<i>ToFileValue</i>	Property to specify the location and file name for saving data. Enter either a file name or a path and file name supported by your system (ASCII text file).
<i>TypeValue</i>	String that specifies which alignments to include in the tree. Choices are: <ul style="list-style-type: none">• 'seed' — Returns a tree with only the alignments used to generate the HMM model.• 'full' (default) — Returns a tree with all of the alignments that match the model.

Return Values

Tree An object containing a phylogenetic tree representative of the protein family.

Description

Tree = gethmmtree(*PFAMName*) searches the PFAM database for the record represented by *PFAMName*, a protein family name, retrieves information, and returns *Tree*, an object containing a phylogenetic tree representative of the protein family.

Tree = gethmmtree(*PFAMAccessionNumber*) searches the PFAM database for the record represented by *PFAMAccessionNumber*, a protein family accession number, retrieves information, and returns *Tree*, an object containing a phylogenetic tree representative of the protein family.

Tree = gethmmtree(*PFAMNumber*) determines a protein family accession number from *PFAMNumber*, an integer, searches the PFAM database for the associated record, retrieves information, and returns *Tree*, an object containing a phylogenetic tree representative of the protein family.

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

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

Tree = gethmmtree(*AccessionNumber*, ...'*Type*', *TypeValue*, ...) specifies which alignments to include in the tree. Choices for *TypeValue* are:

- '*seed*' — Returns a tree with only the alignments used to generate the HMM model.

gethmmtree

- 'full' (default) — Returns a tree with all of the alignments that match the model.

Examples

Enter either of the following to retrieve phylogenetic tree data 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')
```

Phylogenetic tree object with 32 leaves (31 branches)

See Also

Bioinformatics Toolbox functions: `gethmmalignment`, `phytreeread`

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

Syntax

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

Arguments

<i>PDBid</i>	String specifying a unique identifier for a protein structure record in the PDB database.
--------------	---

Note Each structure in the PDB database is represented by a four-character alphanumeric identifier. For example, 4hhb is the identifier for hemoglobin.

<i>ToFileValue</i>	String specifying a file name or a path and file name for saving the PDB-formatted data. If you specify only a file name, that file will be saved in the MATLAB Current Directory.
--------------------	--

Tip After you save the protein structure record to a local PDB-formatted file, you can use the `pdbread` function to read the file into the MATLAB software offline or use the `molviewer` function to display and manipulate a 3-D image of the structure.

<i>SequenceOnlyValue</i>	Controls the return of the protein sequence only. Choices are <code>true</code> or <code>false</code> (default). If there is one sequence, it is returned as a character array. If there are multiple sequences, they are returned as a cell array.
--------------------------	---

Return Values

PDBStruct MATLAB structure containing a field for each PDB record.

Description

The Protein Data Bank (PDB) database is an archive of experimentally determined 3-D biological macromolecular structure data. For more information about the PDB format, see:

<http://www.wwpdb.org/documentation/format23/v2.3.html>

`getpdb` retrieves protein structure data from the Protein Data Bank (PDB) database, which contains 3-D biological macromolecular structure data.

PDBStruct = `getpdb(PDBid)` searches the PDB database for the protein structure record specified by the identifier *PDBid* and returns the MATLAB structure *PDBStruct*, which contains a field for each PDB record. The following table summarizes the possible PDB records and the corresponding fields in the MATLAB structure *PDBStruct*:

PDB Database Record	Field in the MATLAB Structure
HEADER	Header
OBSLTE	Obsolete
TITLE	Title
CAVEAT	Caveat
COMPND	Compound
SOURCE	Source
KEYWDS	Keywords
EXPDTA	ExperimentData
AUTHOR	Authors
REVDAT	RevisionDate
SPRSDE	Superseded

PDB Database Record	Field in the MATLAB Structure
JRNL	Journal
REMARK 1	Remark1
REMARK <i>N</i>	Remark <i>n</i>
Note <i>N</i> equals 2 through 999.	Note <i>n</i> equals 2 through 999.
DBREF	DBReferences
SEQADV	SequenceConflicts
SEQRES	Sequence
FTNOTE	Footnote
MODRES	ModifiedResidues
HET	Heterogen
HETNAM	HeterogenName
HETSYN	HeterogenSynonym
FORMUL	Formula
HELIX	Helix
SHEET	Sheet
TURN	Turn
SSBOND	SSBond
LINK	Link
HYDBND	HydrogenBond
SLTBRG	SaltBridge
CISPEP	CISPeptides
SITE	Site

PDB Database Record	Field in the MATLAB Structure
CRYST1	Cryst1
ORIGXn	OriginX
SCALEn	Scale
MTRIXn	Matrix
TVECT	TranslationVector
MODEL	Model
ATOM	Atom
SIGATM	AtomSD
ANISOU	AnisotropicTemp
SIGUIJ	AnisotropicTempSD
TER	Terminal
HETATM	HeterogenAtom
CONNECT	Connectivity

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

PDBStruct = `getpdb(PDBid, ...'ToFile', ToFileValue, ...)` saves the data returned from the database to a PDB-formatted file, *ToFileValue*.

Tip After you save the protein structure record to a local PDB-formatted file, you can use the `pdbread` function to read the file into the MATLAB software offline or use the `molviewer` function to display and manipulate a 3-D image of the structure.

`PDBStruct = getpdb(PDBid, ...'SequenceOnly', SequenceOnlyValue, ...)` controls the return of the protein sequence only. Choices are `true` or `false` (default). If there is one sequence, it is returned as a character array. If there are multiple sequences, they are returned as a cell array.

The Sequence Field

The Sequence field is also a structure containing sequence information in the following subfields:

- NumOfResidues
- ChainID
- ResidueNames — Contains the three-letter codes for the sequence residues.
- Sequence — Contains the single-letter codes for the sequence residues.

Note If the sequence has modified residues, then the ResidueNames subfield might not correspond to the standard three-letter amino acid codes. In this case, the Sequence subfield will contain the modified residue code in the position corresponding to the modified residue. The modified residue code is provided in the ModifiedResidues field.

The Model Field

The Model field is also a structure or an array of structures containing coordinate information. If the MATLAB structure contains one model, the Model field is a structure containing coordinate information for that model. If the MATLAB structure contains multiple models, the Model field is an array of structures containing coordinate information for each model. The Model field contains the following subfields:

- Atom
- AtomSD

- AnisotropicTemp
- AnisotropicTempSD
- Terminal
- HeterogenAtom

The Atom Field

The Atom field is also an array of structures containing the following subfields:

- AtomSerNo
- AtomName
- altLoc
- resName
- chainID
- resSeq
- iCode
- X
- Y
- Z
- occupancy
- tempFactor
- segID
- element
- charge
- AtomNameStruct — Contains three subfields: chemSymbol, remoteInd, and branch.

Examples

Retrieve the structure information for the electron transport (heme) protein that has a PDB identifier of 5CYT, read the information into a MATLAB structure `pdbstruct`, and save the information to a PDB-formatted file `electron_transport.pdb` in the MATLAB Current Directory.

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

See Also

Bioinformatics Toolbox functions: `getembl`, `getgenbank`, `getgenpept`, `molviewer`, `pdbdistplot`, `pdbread`, `pdbsuperpose`, `pdbtransform`, `pdbwrite`

goannotread

Purpose Read annotations from Gene Ontology annotated file

Syntax

```
Annotation = goannotread(File)  
Annotation = goannotread(File, ...'Fields', FieldsValue,  
...)  
Annotation = goannotread(File, ...'Aspect', AspectValue,  
...)
```

Arguments

<i>File</i>	String specifying a file name of a Gene Ontology (GO) annotated file.
<i>FieldsValue</i>	String or cell array of strings specifying one or more fields to read from the Gene Ontology annotated file. Default is to read all fields. Valid fields are listed below.
<i>AspectValue</i>	Character array specifying one or more characters. Valid aspects are: <ul style="list-style-type: none">• P — Biological process• F — Molecular function• C — Cellular component Default is 'CFP', which specifies to read all aspects.

Return Values

<i>Annotation</i>	MATLAB array of structures containing annotations from a Gene Ontology annotated file.
-------------------	--

Description *Annotation* = goannotread(*File*) converts the contents of *File*, a Gene Ontology annotated file, into *Annotation*, an array of structures. Files should have the structure specified in:

<http://www.geneontology.org/GO.annotation.shtml#file>

A list with some annotated files can be found at:

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

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

Annotation = `goannotread(File, ...'Fields', FieldsValue, ...)` specifies the fields to read from the Gene Ontology annotated file. *FieldsValue* is a string or cell array of strings specifying one or more fields. Default is to read all fields. Valid fields are:

- Database
- DB_Object_ID
- DB_Object_Symbol
- Qualifier
- GOid
- DBReference
- Evidence
- WithFrom
- Aspect
- DB_Object_Name
- Synonym
- DB_Object_Type
- Taxon
- Date
- Assigned_by

For more information on these fields, see:

<http://www.geneontology.org/GO.format.annotation.shtml>

`Annotation = goannotread(File, ...'Aspect', AspectValue, ...)` specifies the aspects to read from the Gene Ontology annotated file. `AspectValue` is a character array specifying one or more characters. Valid aspects are:

- P — Biological process
- F — Molecular function
- C — Cellular component

Default is 'CFP', which specifies to read all aspects.

Examples

Reading All Annotations from a Gene Ontology Annotated File

- 1 Open a Web browser to

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

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

- 3 Uncompress the file using the `gunzip` function.

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

- 4 Read the file into the MATLAB software.

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

- 5 Create a structure with GO annotations and display a list of the genes.

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


Reading a Subset of Annotations from a Gene Ontology Annotated File

- 1 Open a Web browser to

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

- 2 Download `gene_association.goa_human.gz`, the file containing GO annotations for the gene products of *Homo sapiens*, to your MATLAB Current Directory.

- 3 Uncompress the file using the `gunzip` function.

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

- 4 Read the file into the MATLAB software, but limit the annotations to genes related to molecular function (F), and to the fields for the gene symbol and the associated ID, that is, `DB_Object_Symbol` and `G0id`.

```
HumanStruct = goannotread('gene_association.goa_human', ...  
                        'Aspect','F','Fields',{ 'DB_Object_Symbol','G0id'});
```

- 5 Create a list of the *Homo sapiens* genes and a list of the associated GO terms.

```
Humangenes = {HumanStruct.DB_Object_Symbol};  
HumanGO = [HumanStruct.G0id];
```

See Also

Bioinformatics Toolbox functions: `geneont` (object constructor), `num2goid`

Bioinformatics Toolbox object: `geneont` object

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

Purpose Return Gonnet scoring matrix

Syntax gonnet

Description gonnet returns the Gonnet matrix.

The Gonnet matrix is the recommended mutation matrix for initially aligning protein sequences. Matrix elements are ten times the logarithmic of the probability that the residues are aligned divided by the probability that the residues are aligned by chance, and then matrix elements are normalized to 250 PAM units.

Expected score = -0.6152, Entropy = 1.6845 bits Lowest score = -8, Highest score = 14.2

Order:

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

References [1] Gaston, H., Gonnet, M., Cohen, A., and Benner, S. (1992). Exhaustive matching of the entire protein sequence database. *Science*. 256, 1443–1445.

See Also Bioinformatics Toolbox functions: `blosum`, `dayhoff`, `pam`

Purpose Read microarray data from GenePix Results (GPR) file

Syntax

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

Arguments

<i>File</i>	GenePix Results (GPR) formatted file. Enter a file name or a path and file name.
<i>CleanColNamesValue</i>	Controls the creation of column names that can be used as variable names.

Description

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

Field
Header
Data
Blocks
Columns
Rows
Names
IDs
ColumnNames
Indices
Shape

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

gprread(..., 'CleanColNames', CleanColNamesValue) controls the creation of column names that can be used as variable names. A

GPR file may contain column names with spaces and some characters that the MATLAB software cannot use in MATLAB variable names. If *CleanColNamesValue* is true, `gprread` returns names in the field `ColumnNames` that are valid MATLAB variable names and names that you can use in functions. By default, *CleanColNamesValue* is false and the field `ColumnNames` may contain characters that are invalid for MATLAB variable names.

The field `Indices` of the structure contains indices that can be used for plotting heat maps of the data.

For more details on the GPR format, see

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

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

For a list of supported file format versions, see

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

Examples

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

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

See Also

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

Purpose Find all shortest paths in graph

Syntax

```
[dist] = graphallshortestpaths(G)
[dist] = graphallshortestpaths(G, ...'Directed',
DirectedValue, ...)
[dist] = graphallshortestpaths(G, ...'Weights', WeightsValue,
...)
```

Arguments

<i>G</i>	N-by-N sparse matrix that represents a graph. Nonzero entries in matrix <i>G</i> represent the weights of the edges.
<i>DirectedValue</i>	Property that indicates whether the graph is directed or undirected. Enter <code>false</code> for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is <code>true</code> .
<i>WeightsValue</i>	Column vector that specifies custom weights for the edges in matrix <i>G</i> . It must have one entry for every nonzero value (edge) in matrix <i>G</i> . The order of the custom weights in the vector must match the order of the nonzero values in matrix <i>G</i> when it is traversed column-wise. This property lets you use zero-valued weights. By default, <code>graphallshortestpaths</code> gets weight information from the nonzero entries in matrix <i>G</i> .

Description

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

`[dist] = graphallshortestpaths(G)` finds the shortest paths between every pair of nodes in the graph represented by matrix *G*, using Johnson’s algorithm. Input *G* is an N-by-N sparse matrix that

graphallshortestpaths

represents a graph. Nonzero entries in matrix G represent the weights of the edges.

Output $dist$ is an N -by- N matrix where $dist(S, T)$ is the distance of the shortest path from source node S to target node T . Elements in the diagonal of this matrix are always 0, indicating the source node and target node are the same. A 0 not in the diagonal indicates that the distance between the source node and target node is 0. An Inf indicates there is no path between the source node and the target node.

Johnson's algorithm has a time complexity of $O(N \cdot \log(N) + N \cdot E)$, where N and E are the number of nodes and edges respectively.

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

`[dist] = graphallshortestpaths(G, ...'Directed', DirectedValue, ...)` indicates whether the graph is directed or undirected. Set *DirectedValue* to `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`.

`[dist] = graphallshortestpaths(G, ...'Weights', WeightsValue, ...)` lets you specify custom weights for the edges. *WeightsValue* is a column vector having one entry for every nonzero value (edge) in matrix G . The order of the custom weights in the vector must match the order of the nonzero values in matrix G when it is traversed column-wise. This property lets you use zero-valued weights. By default, `graphallshortestpaths` gets weight information from the nonzero entries in matrix G .

Examples

Finding All Shortest Paths in a Directed Graph

1 Create and view a directed graph with 6 nodes and 11 edges.

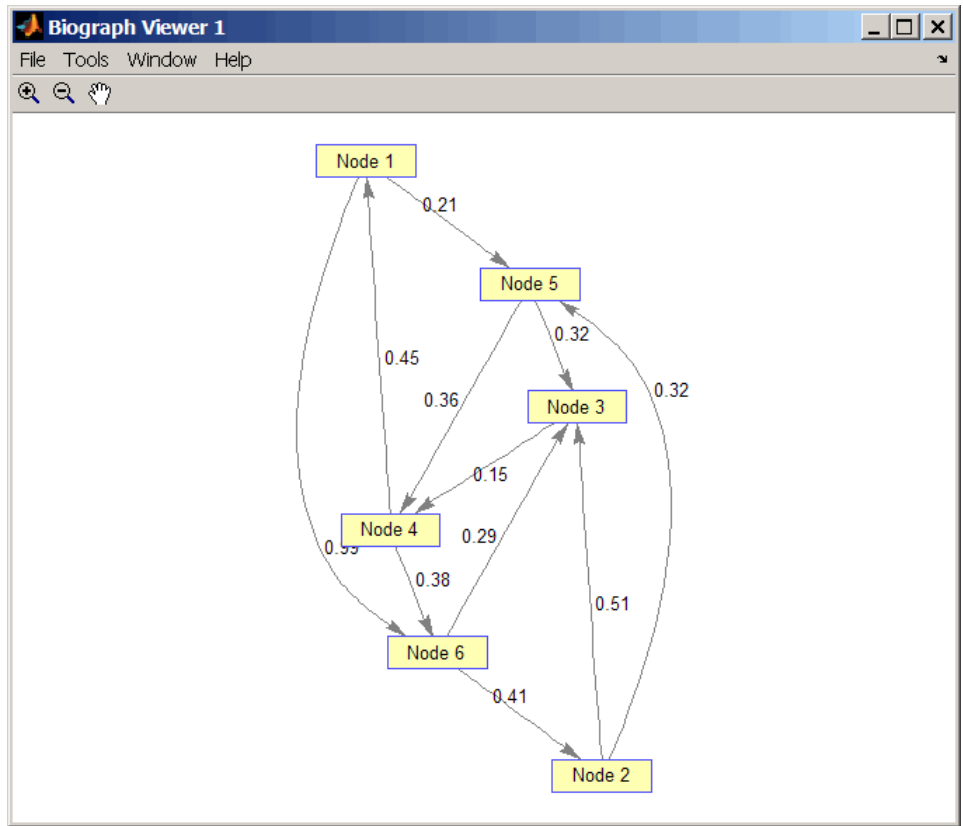
```
W = [.41 .99 .51 .32 .15 .45 .38 .32 .36 .29 .21];  
DG = sparse([6 1 2 2 3 4 4 5 5 6 1],[2 6 3 5 4 1 6 3 4 3 5],W)
```

DG =

(4,1)	0.4500
(6,2)	0.4100
(2,3)	0.5100
(5,3)	0.3200
(6,3)	0.2900
(3,4)	0.1500
(5,4)	0.3600
(1,5)	0.2100
(2,5)	0.3200
(1,6)	0.9900
(4,6)	0.3800

`view(biograph(DG,[],'ShowWeights','on'))`

graphallshortestpaths



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

```
graphallshortestpaths(DG)
```

```
ans =
```

0	1.3600	0.5300	0.5700	0.2100	0.9500
1.1100	0	0.5100	0.6600	0.3200	1.0400
0.6000	0.9400	0	0.1500	0.8100	0.5300

0.4500	0.7900	0.6700	0	0.6600	0.3800
0.8100	1.1500	0.3200	0.3600	0	0.7400
0.8900	0.4100	0.2900	0.4400	0.7300	0

The resulting matrix shows the shortest path from node 1 (first row) to node 6 (sixth column) is 0.95. You can see this in the graph by tracing the path from node 1 to node 5 to node 4 to node 6 ($0.21 + 0.36 + 0.38 = 0.95$).

Finding All Shortest Paths in an Undirected Graph

- 1 Create and view an undirected graph with 6 nodes and 11 edges.

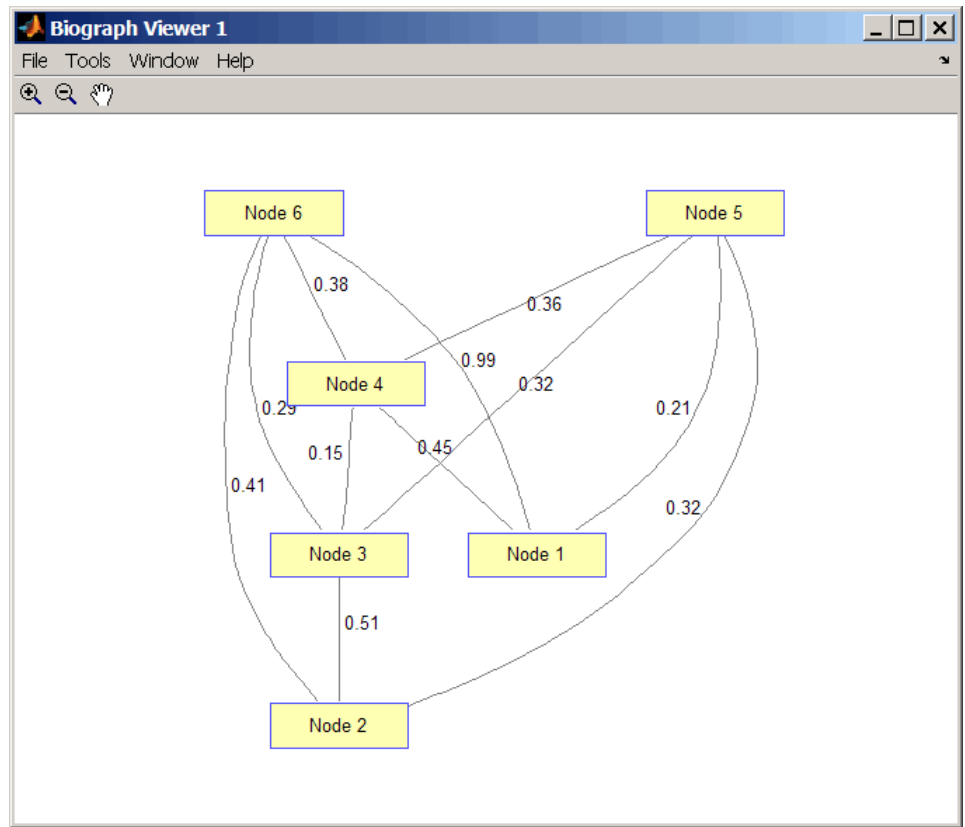
```
UG = tril(DG + DG')
```

```
UG =
```

```
(4,1)    0.4500
(5,1)    0.2100
(6,1)    0.9900
(3,2)    0.5100
(5,2)    0.3200
(6,2)    0.4100
(4,3)    0.1500
(5,3)    0.3200
(6,3)    0.2900
(5,4)    0.3600
(6,4)    0.3800
```

```
view(biograph(UG,[],'ShowArrows','off','ShowWeights','on'))
```

graphallshortestpaths



- 2 Find all the shortest paths between every pair of nodes in the undirected graph.

```
graphallshortestpaths(UG, 'directed', false)
```

```
ans =
```

0	0.5300	0.5300	0.4500	0.2100	0.8300
0.5300	0	0.5100	0.6600	0.3200	0.7000
0.5300	0.5100	0	0.1500	0.3200	0.5300

0.4500	0.6600	0.1500	0	0.3600	0.3800
0.2100	0.3200	0.3200	0.3600	0	0.7400
0.8300	0.7000	0.5300	0.3800	0.7400	0

The resulting matrix is symmetrical because it represents an undirected graph. It shows the shortest path from node 1 (first row) to node 6 (sixth column) is 0.83. You can see this in the graph by tracing the path from node 1 to node 4 to node 6 ($0.45 + 0.38 = 0.83$). Because `UG` is an undirected graph, we can use the edge between node 1 and node 4, which we could not do in the directed graph `DG`.

References

[1] Johnson, D.B. (1977). Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM* *24(1)*, 1-13.

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

See Also

Bioinformatics Toolbox functions: `graphconncomp`, `graphisdag`, `graphisomorphism`, `graphisspanntree`, `graphmaxflow`, `graphminspanntree`, `graphpred2path`, `graphshortestpath`, `graphtopoorder`, `graphtraverse`

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

graphconncomp

Purpose Find strongly or weakly connected components in graph

Syntax
[S, C] = graphconncomp(G)
[S, C] = graphconncomp(G, ...'Directed', *DirectedValue*, ...)
[S, C] = graphconncomp(G, ...'Weak', *WeakValue*, ...)

Arguments

<i>G</i>	N-by-N sparse matrix that represents a graph. Nonzero entries in matrix <i>G</i> indicate the presence of an edge.
<i>DirectedValue</i>	Property that indicates whether the graph is directed or undirected. Enter <code>false</code> for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is <code>true</code> . A DFS-based algorithm computes the connected components. Time complexity is $O(N+E)$, where <i>N</i> and <i>E</i> are number of nodes and edges respectively.
<i>WeakValue</i>	Property that indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set <i>WeakValue</i> to <code>true</code> to find weakly connected components. Default is <code>false</code> , which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is $O(N+E)$, where <i>N</i> and <i>E</i> are number of nodes and edges respectively.

Description

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

`[S, C] = graphconncomp(G)` finds the strongly connected components of the graph represented by matrix *G* using Tarjan’s algorithm. A strongly connected component is a maximal group of nodes that are mutually reachable without violating the edge directions. Input *G* is an N-by-N sparse matrix that represents a graph. Nonzero entries in matrix *G* indicate the presence of an edge.

The number of components found is returned in *S*, and *C* is a vector indicating to which component each node belongs.

Tarjan’s algorithm has a time complexity of $O(N+E)$, where *N* and *E* are the number of nodes and edges respectively.

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

`[S, C] = graphconncomp(G, ...'Directed', DirectedValue, ...)` indicates whether the graph is directed or undirected. Set *directedValue* to `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`. A DFS-based algorithm computes the connected components. Time complexity is $O(N+E)$, where *N* and *E* are number of nodes and edges respectively.

`[S, C] = graphconncomp(G, ...'Weak', WeakValue, ...)` indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set *WeakValue* to `true` to find weakly connected components. Default is `false`, which finds strongly connected components. The state of this

graphconncomp

parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is $O(N+E)$, where N and E are number of nodes and edges respectively.

Note By definition, a single node can be a strongly connected component.

Note A directed acyclic graph (DAG) cannot have any strongly connected components larger than one.

Examples

1 Create and view a directed graph with 10 nodes and 17 edges.

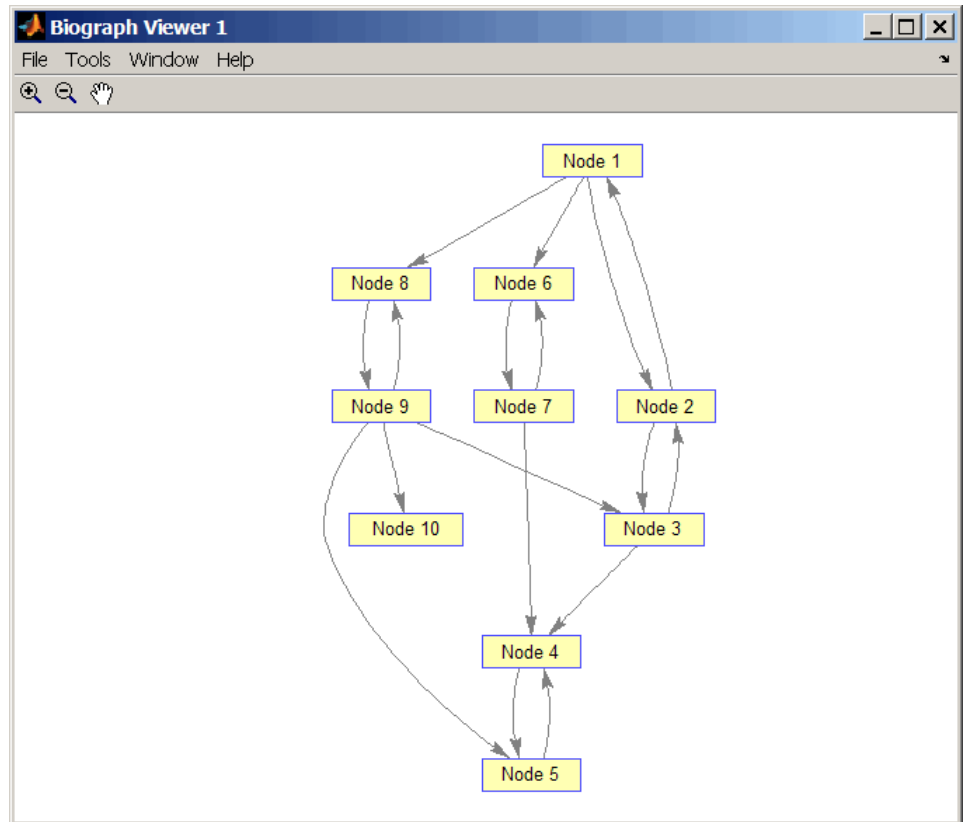
```
DG = sparse([1 1 1 2 2 3 3 4 5 6 7 7 8 9 9 9 9], ...  
           [2 6 8 3 1 4 2 5 4 7 6 4 9 8 10 5 3], true, 10, 10)
```

DG =

(2,1)	1
(1,2)	1
(3,2)	1
(2,3)	1
(9,3)	1
(3,4)	1
(5,4)	1
(7,4)	1
(4,5)	1
(9,5)	1
(1,6)	1
(7,6)	1
(6,7)	1
(1,8)	1
(9,8)	1

```
(8,9)      1
(9,10)     1
```

```
h = view(biograph(DG));
```



- 2** Find the number of strongly connected components in the directed graph and determine to which component each of the 10 nodes belongs.

```
[S,C] = graphconncomp(DG)
```

```
S =
```

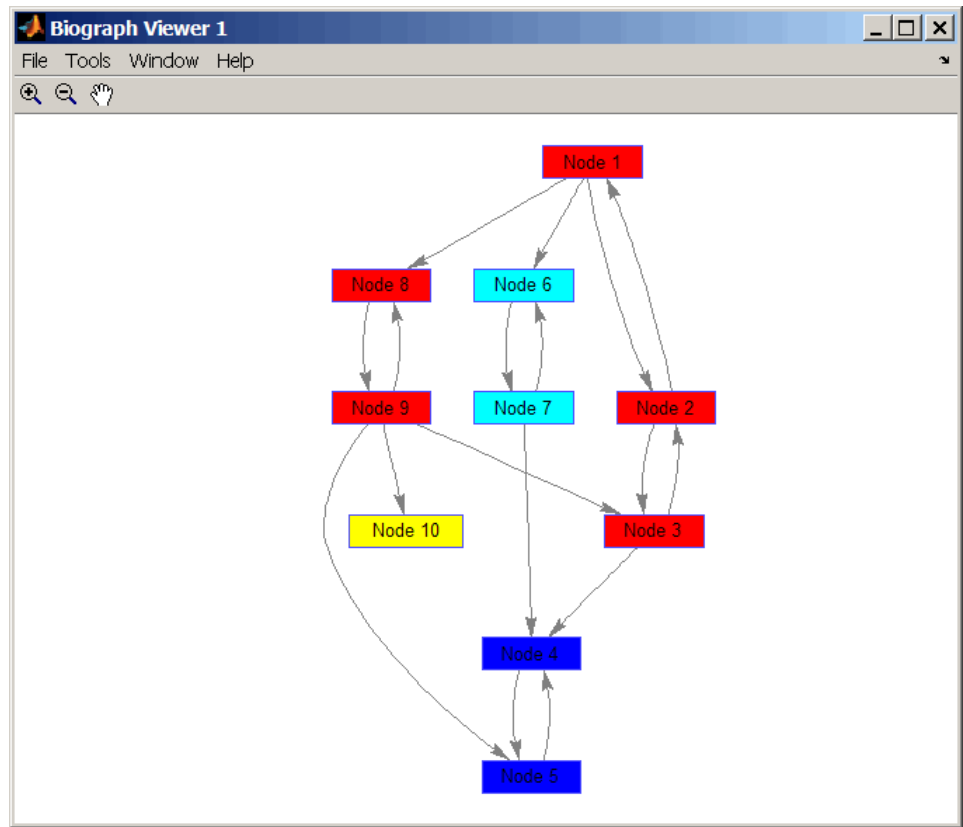
```
4
```

```
C =
```

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

3 Color the nodes for each component with a different color.

```
colors = jet(S);  
for i = 1:numel(h.nodes)  
    h.Nodes(i).Color = colors(C(i),:);  
end
```

References

- [1] Tarjan, R.E., (1972). Depth first search and linear graph algorithms. *SIAM Journal on Computing* 1(2), 146–160.
- [2] Sedgewick, R., (2002). *Algorithms in C++, Part 5 Graph Algorithms* (Addison-Wesley).
- [3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). *The Boost Graph Library User Guide and Reference Manual*, (Upper Saddle River, NJ:Pearson Education).

graphconncomp

See Also

Bioinformatics Toolbox functions: `graphallshortestpaths`, `graphisdag`, `graphisomorphism`, `graphisspanntree`, `graphmaxflow`, `graphminspanntree`, `graphpred2path`, `graphshortestpath`, `graphtopoorder`, `graphtraverse`

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

Purpose Test for cycles in directed graph

Syntax graphisdag(*G*)

Arguments

G N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix *G* indicate the presence of an edge.

Description

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

graphisdag(*G*) returns logical 1 (**true**) if the directed graph represented by matrix *G* is a directed acyclic graph (DAG) and logical 0 (**false**) otherwise. *G* is an N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix *G* indicate the presence of an edge.

Examples

Testing for Cycles in Directed Graphs

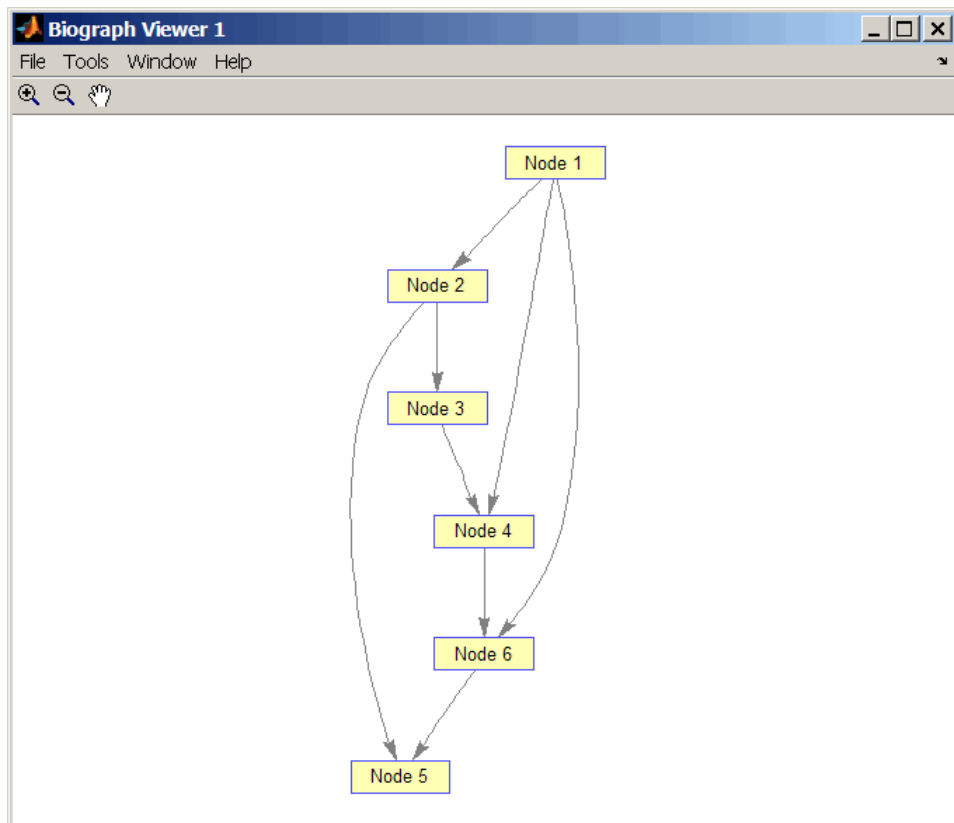
- 1 Create and view a directed acyclic graph (DAG) with six nodes and eight edges.

```
DG = sparse([1 1 1 2 2 3 4 6],[2 4 6 3 5 4 6 5],true,6,6)
```

```
DG =
```

```
(1,2)      1
(2,3)      1
(1,4)      1
(3,4)      1
(2,5)      1
(6,5)      1
(1,6)      1
(4,6)      1
```

```
view(biograph(DG))
```



2 Test for cycles in the DAG.

```
graphisdag(DG)
```

```
ans =
```

```
1
```

- 3** Add an edge to the DAG to make it cyclic, and then view the directed graph.

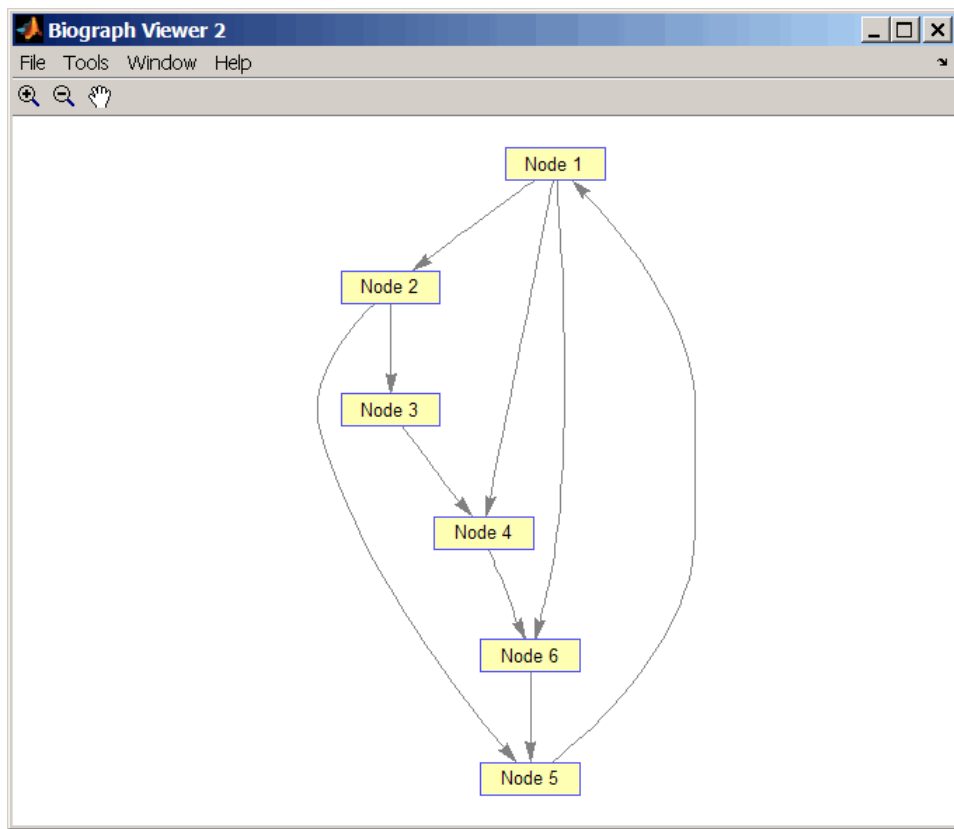
```
DG(5,1) = true
```

```
DG =
```

```
(5,1)      1
(1,2)      1
(2,3)      1
(1,4)      1
(3,4)      1
(2,5)      1
(6,5)      1
(1,6)      1
(4,6)      1
```

```
>> view(bigraph(DG))
```

graphisdag



4 Test for cycles in the new graph.

```
graphisdag(DG)
```

```
ans =
```

```
0
```

Testing for Cycles in a Very Large Graph (Greater Than 20,000 Nodes and 30,000 Edges)

- 1 Download the Gene Ontology database to a geneont object.

```
GO = geneont('live',true);
```

- 2 Convert the geneont object to a matrix.

```
CM = getmatrix(GO);
```

- 3 Test for cycles in the graph.

```
graphisdag(CM)
```

Creating a Random DAG

- 1 Create and view a random directed acyclic graph (DAG) with 15 nodes and 20 edges.

```
g = sparse([],[],true,15,15);  
while nnz(g) < 20  
    edge = randsample(15*15,1); % get a random edge  
    g(edge) = true;  
    g(edge) = graphisdag(g);  
end  
view(biograph(g))
```

- 2 Test for cycles in the graph.

```
graphisdag(g)
```

References

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

See Also

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

graphisdag

graphminspantree, graphpred2path, graphshortestpath,
graphtopoorder, graphtraverse

Bioinformatics Toolbox method of biograph object: isdag

Purpose

Find isomorphism between two graphs

Syntax

```
[Isomorphic, Map] = graphisomorphism(G1, G2)
[Isomorphic, Map] = graphisomorphism(G1, G2, 'Directed',
    DirectedValue)
```

Arguments

- | | |
|----------------------|--|
| <i>G1</i> | N-by-N sparse matrix that represents a directed or undirected graph. Nonzero entries in matrix <i>G1</i> indicate the presence of an edge. |
| <i>G2</i> | N-by-N sparse matrix that represents a directed or undirected graph. <i>G2</i> must be the same (directed or undirected) as <i>G1</i> . |
| <i>DirectedValue</i> | Property that indicates whether the graphs are directed or undirected. Enter <code>false</code> when both <i>G1</i> and <i>G2</i> are undirected graphs. In this case, the upper triangles of the sparse matrices <i>G1</i> and <i>G2</i> are ignored. Default is <code>true</code> , meaning that both graphs are directed. |

Description

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

`[Isomorphic, Map] = graphisomorphism(G1, G2)` returns logical 1 (true) in *Isomorphic* if *G1* and *G2* are isomorphic graphs, and logical 0 (false) otherwise. A graph isomorphism is a 1-to-1 mapping of the nodes in the graph *G1* and the nodes in the graph *G2* such that adjacencies are preserved. *G1* and *G2* are both N-by-N sparse matrices that represent directed or undirected graphs. Return value *Isomorphic* is Boolean. When *Isomorphic* is true, *Map* is a row vector containing the node indices that map from *G2* to *G1*. When *Isomorphic* is false, the worst-case time complexity is $O(N!)$, where N is the number of nodes.

graphisomorphism

`[Isomorphic, Map] = graphisomorphism(G1, G2, 'Directed', DirectedValue)` indicates whether the graphs are directed or undirected. Set `DirectedValue` to `false` when both `G1` and `G2` are undirected graphs. In this case, the upper triangles of the sparse matrices `G1` and `G2` are ignored. Default is `true`, meaning that both graphs are directed.

Examples

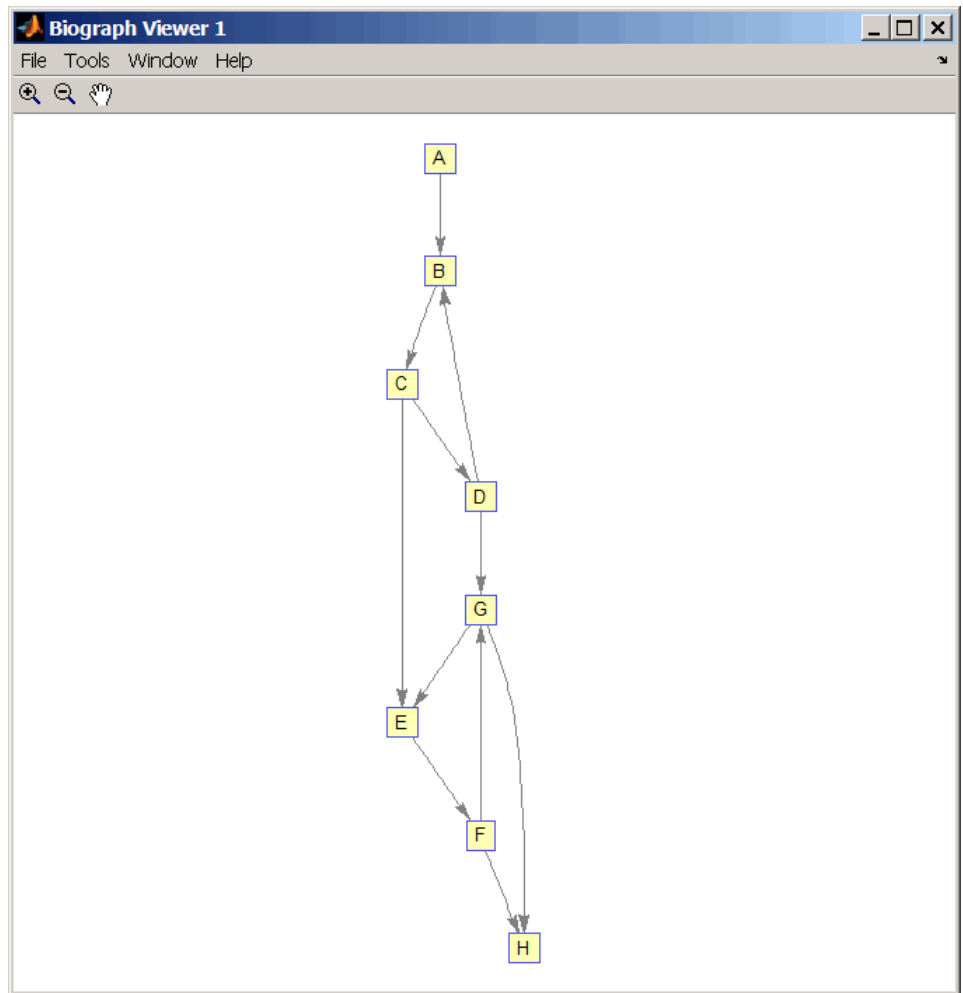
- 1 Create and view a directed graph with 8 nodes and 11 edges.

```
m('ABCDEFGH') = [1 2 3 4 5 6 7 8];  
g1 = sparse(m('ABDCDCGEFFG'),m('BCBDGEEFHGH'),true,8,8)
```

```
g1 =
```

```
(1,2)      1  
(4,2)      1  
(2,3)      1  
(3,4)      1  
(3,5)      1  
(7,5)      1  
(5,6)      1  
(4,7)      1  
(6,7)      1  
(6,8)      1  
(7,8)      1
```

```
view(biograph(g1,'ABCDEFGH'))
```



- 2 Set a random permutation vector and then create and view a new permuted graph.

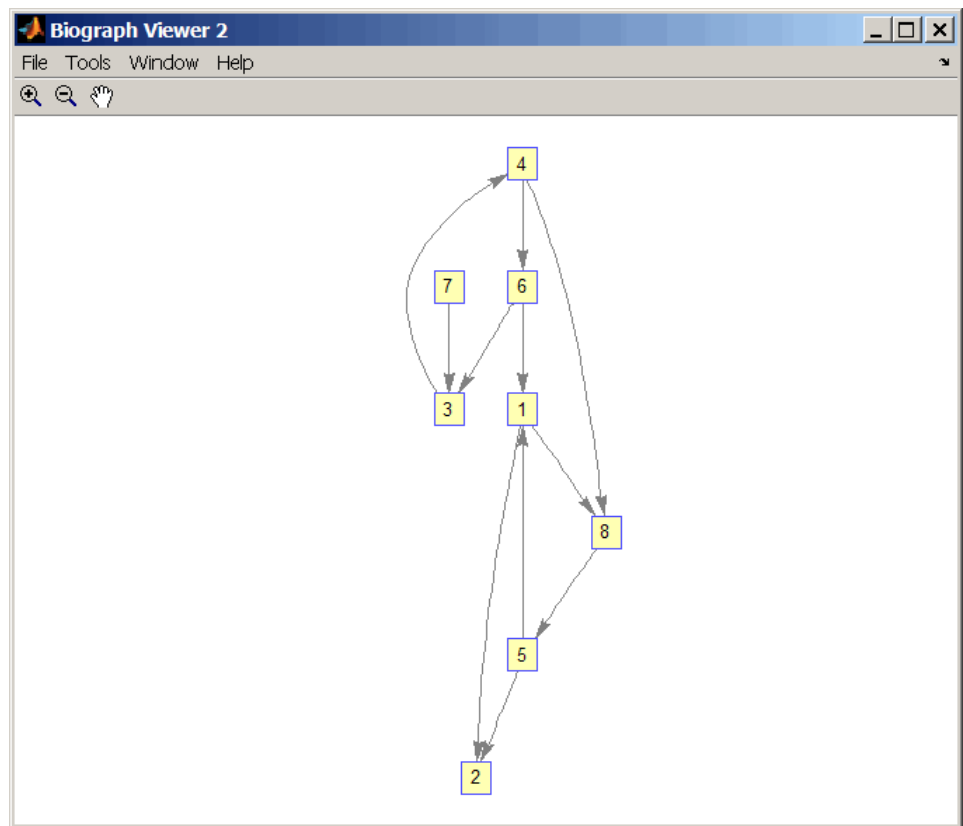
```
p = randperm(8)
```

graphisomorphism

p =

7 8 2 3 6 4 1 5

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



3 Check if the two graphs are isomorphic.

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

F =

1

Map =

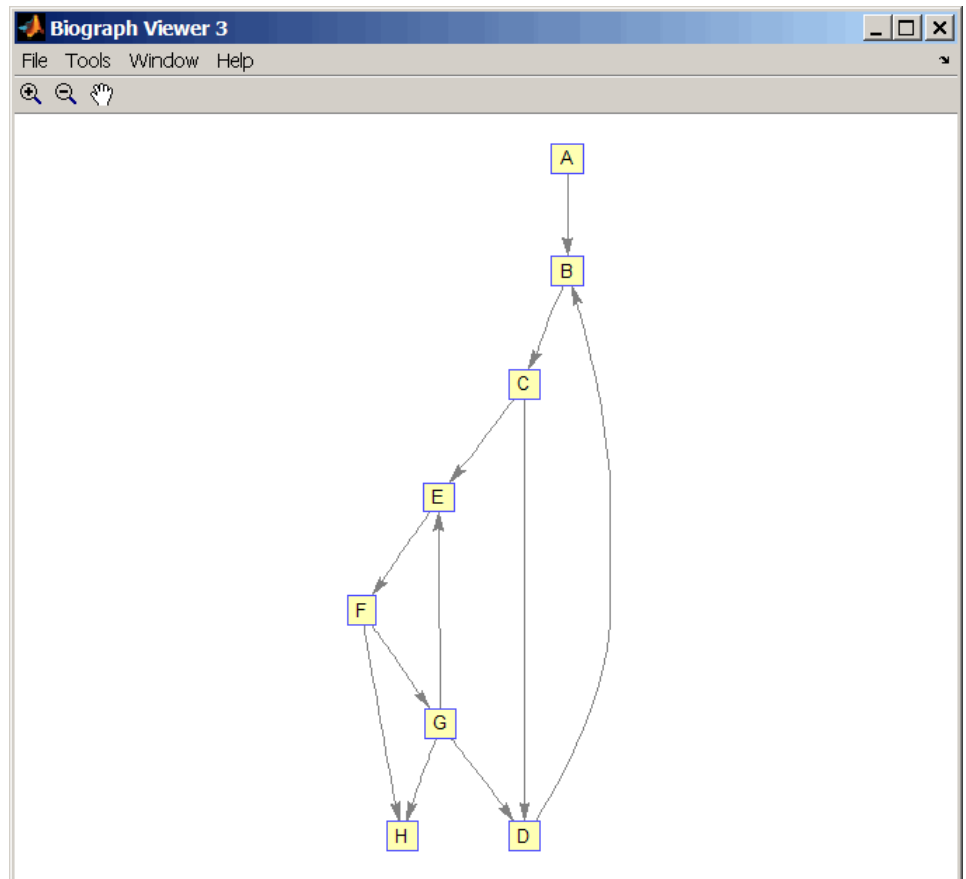
7 8 2 3 6 4 1 5

Note that the Map row vector containing the node indices that map from g2 to g1 is the same as the permutation vector you created in step 2.

- 4 Reverse the direction of the D-G edge in the first graph, and then check for isomorphism again.

```
g1(m('DG'),m('GD')) = g1(m('GD'),m('DG'));  
view(biograph(g1,'ABCDEFGH'))
```

graphisomorphism



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

F =

0

M =

```
[]
```

- 5 Convert the graphs to undirected graphs, and then check for isomorphism.

```
[F,M] = graphisomorphism(g2+g2',g1+g1','directed',false)
```

```
F =
```

```
1
```

```
M =
```

```
7 8 2 3 6 4 1 5
```

References

- [1] Fortin, S. (1996). The Graph Isomorphism Problem. Technical Report, 96-20, Dept. of Computer Science, University of Alberta, Edmonton, Alberta, Canada.
- [2] McKay, B.D. (1981). Practical Graph Isomorphism. *Congressus Numerantium* 30, 45-87.
- [3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

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

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

graphisspantree

Purpose Determine if tree is spanning tree

Syntax `TF = graphisspantree(G)`

Arguments

G N-by-N sparse matrix whose lower triangle represents an undirected graph. Nonzero entries in matrix *G* indicate the presence of an edge.

Description

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

`TF = graphisspantree(G)` returns logical 1 (`true`) if *G* is a spanning tree, and logical 0 (`false`) otherwise. A spanning tree must touch all the nodes and must be acyclic. *G* is an N-by-N sparse matrix whose lower triangle represents an undirected graph. Nonzero entries in matrix *G* indicate the presence of an edge.

Examples

1 Create a phytree object from a phylogenetic tree file.

```
tr = phytread('pf00002.tree')  
Phylogenetic tree object with 33 leaves (32 branches)
```

2 Create a connection matrix from the phytree object.

```
[CM,labels,dist] = getmatrix(tr);
```

3 Determine if the connection matrix is a spanning tree.

```
graphisspantree(CM)
```

```
ans =
```

```
1
```


- 4 Add an edge between the root and the first leaf in the connection matrix.

```
CM(end,1) = 1;
```

- 5 Determine if the modified connection matrix is a spanning tree.

```
graphisspantree(CM)
```

```
ans =
```

```
0
```

References

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

See Also

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

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

graphmaxflow

Purpose Calculate maximum flow in directed graph

Syntax

```
[MaxFlow, FlowMatrix, Cut] = graphmaxflow(G, SNode, TNode)
[...] = graphmaxflow(G, SNode, TNode, ...'Capacity',
CapacityValue, ...)
[...] = graphmaxflow(G, SNode, TNode, ...'Method', MethodValue,
...)
```

Arguments

<i>G</i>	N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix <i>G</i> represent the capacities of the edges.
<i>SNode</i>	Node in <i>G</i> .
<i>TNode</i>	Node in <i>G</i> .
<i>CapacityValue</i>	Column vector that specifies custom capacities for the edges in matrix <i>G</i> . It must have one entry for every nonzero value (edge) in matrix <i>G</i> . The order of the custom capacities in the vector must match the order of the nonzero values in matrix <i>G</i> when it is traversed column-wise. By default, <code>graphmaxflow</code> gets capacity information from the nonzero entries in matrix <i>G</i> .
<i>MethodValue</i>	String that specifies the algorithm used to find the minimal spanning tree (MST). Choices are: <ul style="list-style-type: none">• 'Edmonds' — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the <i>labeling algorithm</i>. Time complexity is $O(N \cdot E^2)$, where <i>N</i> and <i>E</i> are the number of nodes and edges respectively.• 'Goldberg' — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as <i>preflow-push</i>. Time complexity is $O(N^2 \cdot \sqrt{E})$, where <i>N</i> and <i>E</i> are the number of nodes and edges respectively.

Description

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

`[MaxFlow, FlowMatrix, Cut] = graphmaxflow(G, SNode, TNode)` calculates the maximum flow of directed graph *G* from node *SNode* to node *TNode*. Input *G* is an N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix *G* represent the capacities of the edges. Output *MaxFlow* is the maximum flow, and *FlowMatrix* is a sparse matrix with all the flow values for every edge. *FlowMatrix(X,Y)* is the flow from node *X* to node *Y*. Output *Cut* is a logical row vector indicating the nodes connected to *SNode* after calculating the minimum cut between *SNode* and *TNode*. If several solutions to the minimum cut problem exist, then *Cut* is a matrix.

Tip The algorithm that determines *Cut*, all minimum cuts, has a time complexity of $O(2^N)$, where *N* is the number of nodes. If this information is not needed, use the `graphmaxflow` function without the third output.

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

`[...] = graphmaxflow(G, SNode, TNode, ...'Capacity', CapacityValue, ...)` lets you specify custom capacities for the edges. *CapacityValue* is a column vector having one entry for every nonzero value (edge) in matrix *G*. The order of the custom capacities in the vector must match the order of the nonzero values in matrix *G* when it is traversed column-wise. By default, `graphmaxflow` gets capacity information from the nonzero entries in matrix *G*.

graphmaxflow

[...] = graphmaxflow(G, SNode, TNode, ...'Method', MethodValue, ...) lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- 'Edmonds' — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the *labeling algorithm*. Time complexity is $O(N \cdot E^2)$, where N and E are the number of nodes and edges respectively.
- 'Goldberg' — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as *preflow-push*. Time complexity is $O(N^2 \cdot \sqrt{E})$, where N and E are the number of nodes and edges respectively.

Examples

- 1 Create a directed graph with six nodes and eight edges.

```
cm = sparse([1 1 2 2 3 3 4 5],[2 3 4 5 4 5 6 6],...  
           [2 3 3 1 1 1 2 3],6,6)
```

```
cm =
```

(1,2)	2
(1,3)	3
(2,4)	3
(3,4)	1
(2,5)	1
(3,5)	1
(4,6)	2
(5,6)	3

- 2 Calculate the maximum flow in the graph from node 1 to node 6.

```
[M,F,K] = graphmaxflow(cm,1,6)
```

```
M =
```

```
4
```

F =

(1,2)	2
(1,3)	2
(2,4)	1
(3,4)	1
(2,5)	1
(3,5)	1
(4,6)	2
(5,6)	2

K =

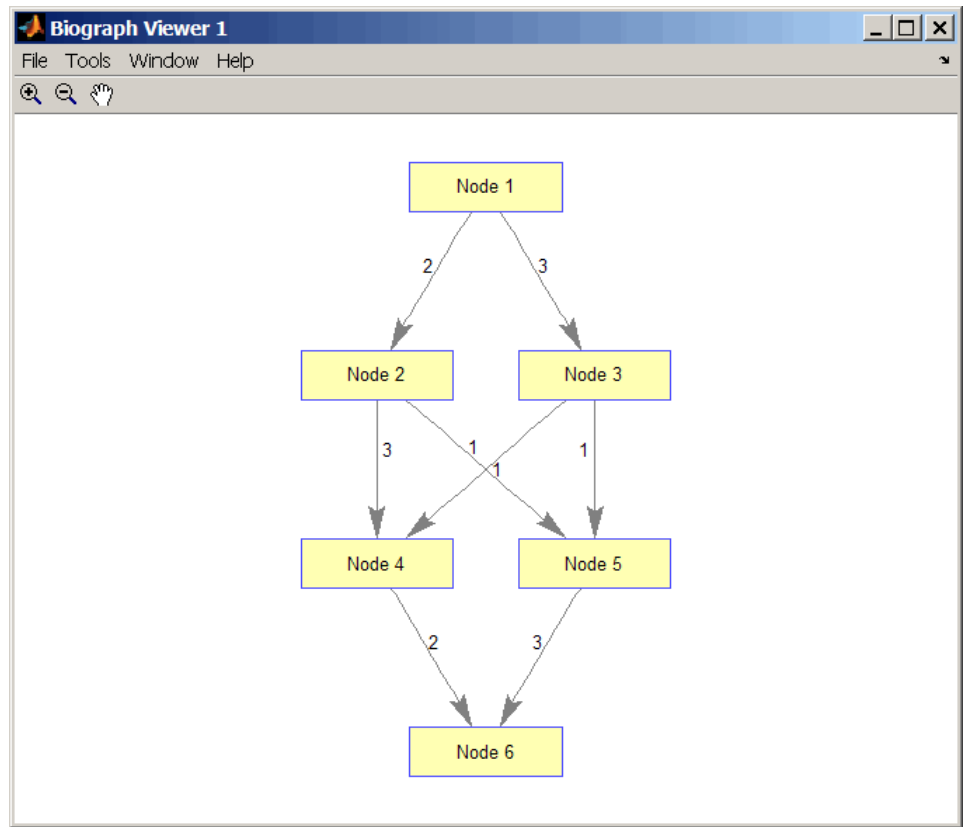
1	1	1	1	0	0
1	0	1	0	0	0

Notice that K is a two-row matrix because there are two possible solutions to the minimum cut problem.

3 View the graph with the original capacities.

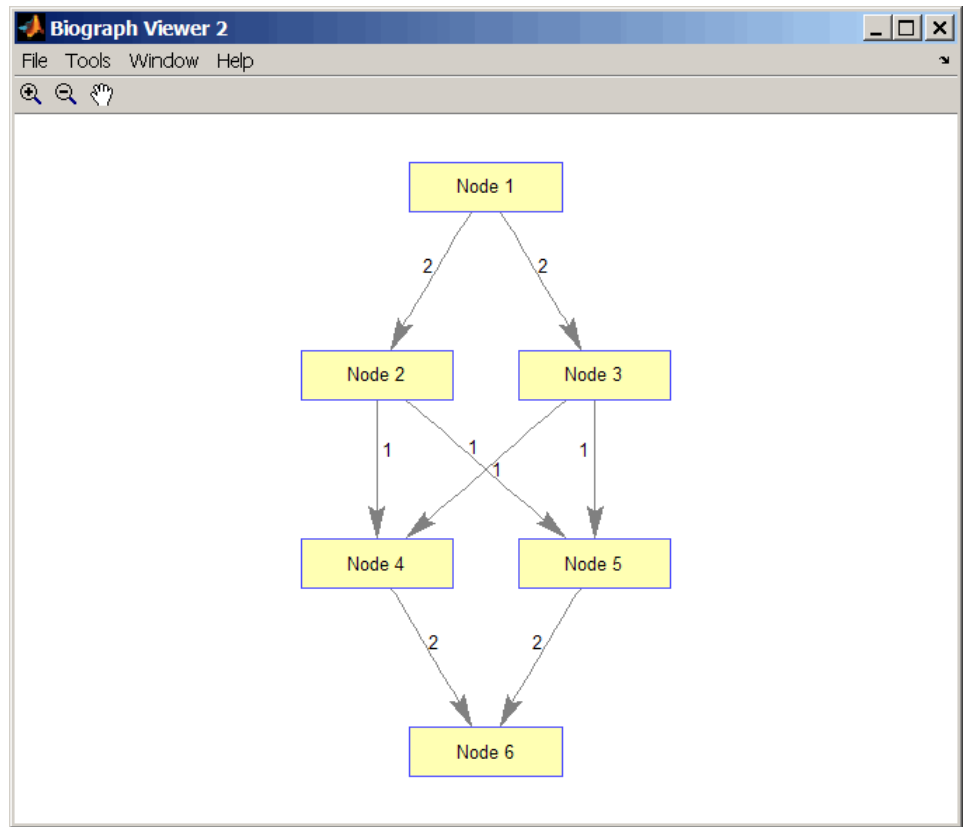
```
h = view(biograph(cm,[], 'ShowWeights', 'on'))
```

graphmaxflow



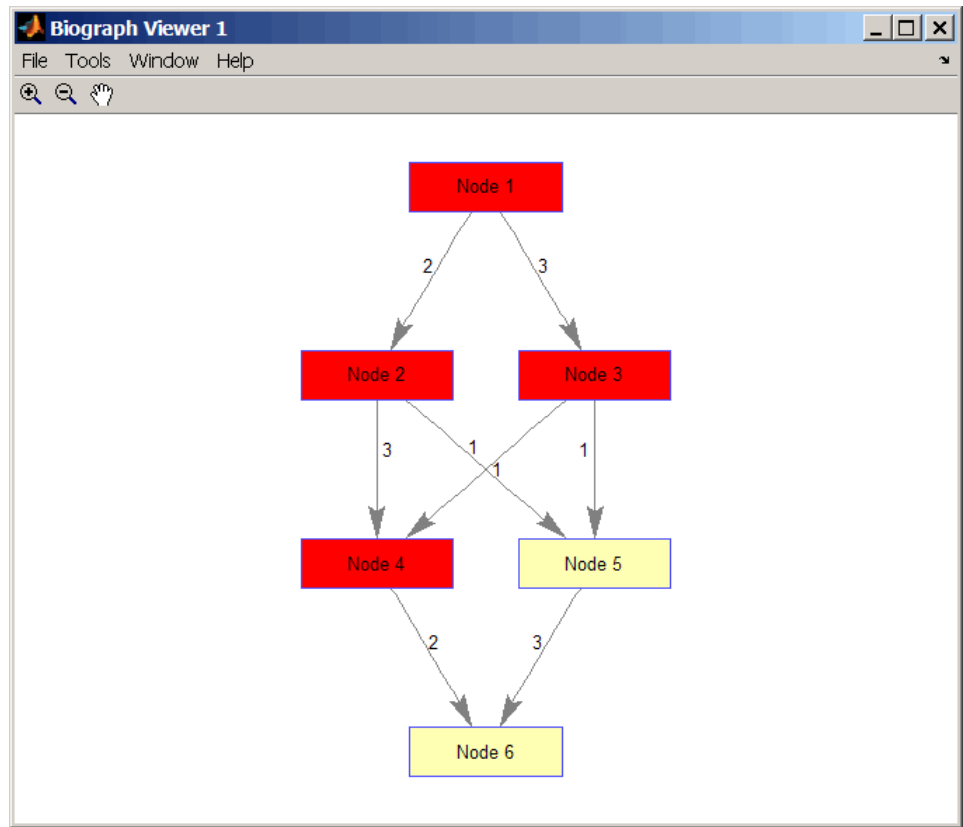
4 View the graph with the calculated maximum flows.

```
view(biograph(F,[], 'ShowWeights', 'on'))
```



5 Show one solution to the minimum cut problem in the original graph.

```
set(h.Nodes(K(1,:)), 'Color', [1 0 0])
```



Notice that in the three edges that connect the source nodes (red) to the destination nodes (yellow), the original capacities and the calculated maximum flows are the same.

References

- [1] Edmonds, J. and Karp, R.M. (1972). Theoretical improvements in the algorithmic efficiency for network flow problems. *Journal of the ACM* 19, 248-264.
- [2] Goldberg, A.V. (1985). A New Max-Flow Algorithm. MIT Technical Report MIT/LCS/TM-291, Laboratory for Computer Science, MIT.

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

See Also

Bioinformatics Toolbox functions: `graphallshortestpaths`, `graphconncomp`, `graphisdag`, `graphisomorphism`, `graphisspanntree`, `graphminspanntree`, `graphpred2path`, `graphshortestpath`, `graphtopoorder`, `graphtraverse`

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

graphminspantree

Purpose Find minimal spanning tree in graph

Syntax

```
[Tree, pred] = graphminspantree(G)
[Tree, pred] = graphminspantree(G, R)
[Tree, pred] = graphminspantree(..., 'Method', MethodValue, ...)
[Tree, pred] = graphminspantree(..., 'Weights', WeightsValue, ...)
```

Arguments

- G* N-by-N sparse matrix that represents an undirected graph. Nonzero entries in matrix *G* represent the weights of the edges.
- R* Scalar between 1 and the number of nodes.

Description

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

`[Tree, pred] = graphminspantree(G)` finds an acyclic subset of edges that connects all the nodes in the undirected graph *G* and for which the total weight is minimized. Weights of the edges are all nonzero entries in the lower triangle of the N-by-N sparse matrix *G*. Output *Tree* is a spanning tree represented by a sparse matrix. Output *pred* is a vector containing the predecessor nodes of the minimal spanning tree (MST), with the root node indicated by 0. The root node defaults to the first node in the largest connected component. This computation requires an extra call to the `graphconncomp` function.

`[Tree, pred] = graphminspantree(G, R)` sets the root of the minimal spanning tree to node *R*.

`[Tree, pred] = graphminspantree(..., 'PropertyName', PropertyValue, ...)` calls `graphminspantree` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes

and is case insensitive. These property name/property value pairs are as follows:

```
[Tree, pred] = graphminspantree(..., 'Method', MethodValue, ...)
```

lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- 'Kruskal' — Grows the minimal spanning tree (MST) one edge at a time by finding an edge that connects two trees in a spreading forest of growing MSTs. Time complexity is $O(E+X*\log(N))$, where X is the number of edges no longer than the longest edge in the MST, and N and E are the number of nodes and edges respectively.
- 'Prim' — Default algorithm. Grows the minimal spanning tree (MST) one edge at a time by adding a minimal edge that connects a node in the growing MST with any other node. Time complexity is $O(E*\log(N))$, where N and E are the number of nodes and edges respectively.

Note When the graph is unconnected, Prim's algorithm returns only the tree that contains R , while Kruskal's algorithm returns an MST for every component.

```
[Tree, pred] = graphminspantree(..., 'Weights',  
WeightsValue, ...) lets you specify custom weights for the  
edges. WeightsValue is a column vector having one entry for every  
nonzero value (edge) in matrix  $G$ . The order of the custom weights in the  
vector must match the order of the nonzero values in matrix  $G$  when it  
is traversed column-wise. By default, graphminspantree gets weight  
information from the nonzero entries in matrix  $G$ .
```

Examples

- 1 Create and view an undirected graph with 6 nodes and 11 edges.

```
W = [.41 .29 .51 .32 .50 .45 .38 .32 .36 .29 .21];  
DG = sparse([1 1 2 2 3 4 4 5 5 6 6],[2 6 3 5 4 1 6 3 4 2 5],W);
```

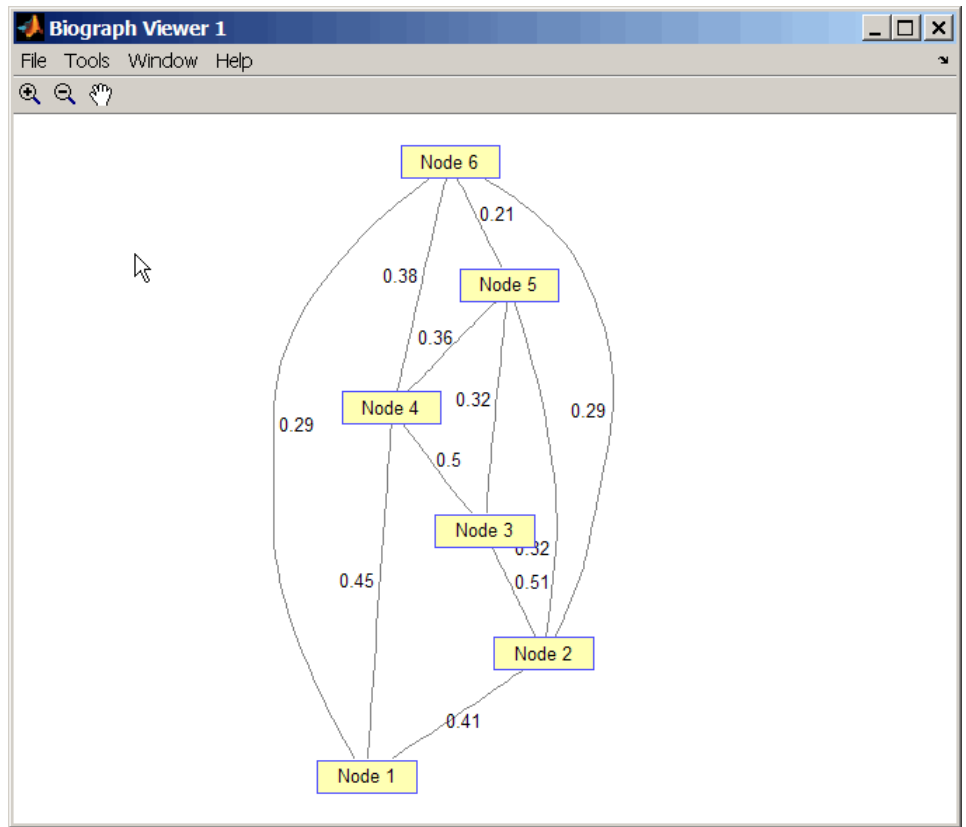
graphminspantree

```
UG = tril(DG + DG')
```

```
UG =
```

(2,1)	0.4100
(4,1)	0.4500
(6,1)	0.2900
(3,2)	0.5100
(5,2)	0.3200
(6,2)	0.2900
(4,3)	0.5000
(5,3)	0.3200
(5,4)	0.3600
(6,4)	0.3800
(6,5)	0.2100

```
view(biograph(UG,[],'ShowArrows','off','ShowWeights','on'))
```



2 Find and view the minimal spanning tree of the undirected graph.

```
[ST,pred] = graphminspantree(UG)
```

ST =

(6,1)	0.2900
(6,2)	0.2900
(5,3)	0.3200
(5,4)	0.3600

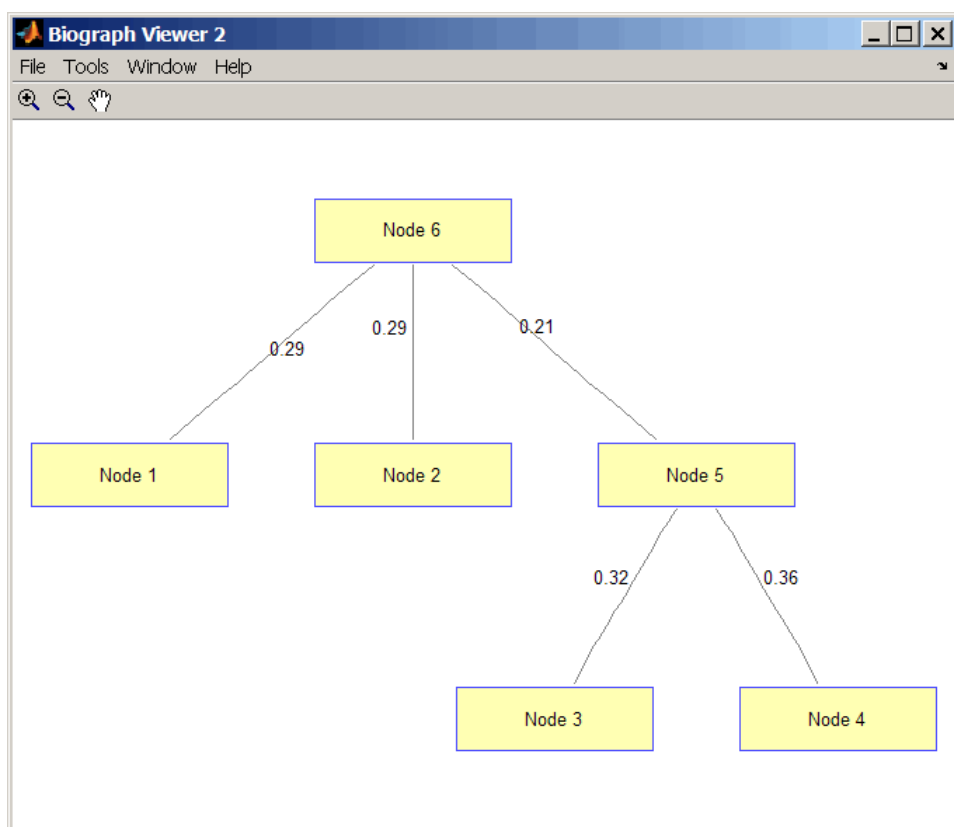
graphminspantree

```
(6,5)      0.2100
```

```
pred =
```

```
0      6      5      5      6      1
```

```
view(biograph(ST,[], 'ShowArrows', 'off', 'ShowWeights', 'on'))
```



References

- [1] Kruskal, J.B. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society 7, 48-50.
- [2] Prim, R. (1957). Shortest Connection Networks and Some Generalizations. Bell System Technical Journal 36, 1389-1401.
- [3] Siek, J.G. Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

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

Bioinformatics Toolbox method of biograph object: minspantree

graphpred2path

Purpose Convert predecessor indices to paths

Syntax $path = \text{graphpred2path}(pred, D)$

Arguments

$pred$ Row vector or matrix of predecessor node indices. The value of the root (or source) node in $pred$ must be 0.

D Destination node in $pred$.

Description

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

$path = \text{graphpred2path}(pred, D)$ traces back a path by following the predecessor list in $pred$ starting at destination node D .

The value of the root (or source) node in $pred$ must be 0. If a NaN is found when following the predecessor nodes, graphpred2path returns an empty path.

If $pred$ is a ...	And D is a ...	Then $path$ is a ...
row vector of predecessor node indices	scalar	row vector listing the nodes from the root (or source) to D .
	row vector	row cell array with every column containing the path to the destination for every element in D .

If <i>pred</i> is a ...	And <i>D</i> is a ...	Then <i>path</i> is a ...
matrix	scalar	column cell array with every row containing the path for every row in <i>pred</i> .
	row vector	matrix cell array with every row containing the paths for the respective row in <i>pred</i> , and every column containing the paths to the respective destination in <i>D</i> .

Note If *D* is omitted, the paths to all the destinations are calculated for every predecessor listed in *pred*.

Examples

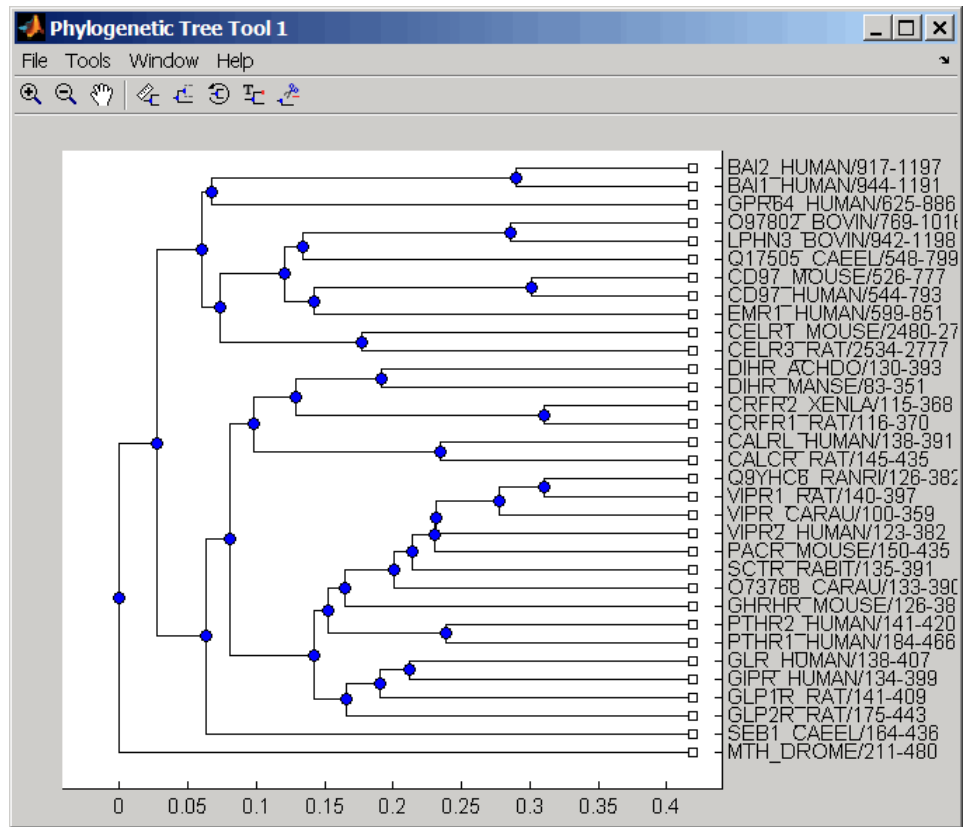
- 1 Create a phytree object from the phylogenetic tree file for the GLR_HUMAN protein.

```
tr = phytread('pf00002.tree')
Phylogenetic tree object with 33 leaves (32 branches)
```

- 2 View the phytree object.

```
view(tr)
```

graphpred2path



- 3** From the phytree object, create a connection matrix to represent the phylogenetic tree.

```
[CM,labels,dist] = getmatrix(tr);
```

- 4** Find the nodes from the root to one leaf in the phylogenetic tree created from the phylogenetic tree file for the GLR_HUMAN protein.

```
root_loc = size(CM,1)
```

```
root_loc =
```

65

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

```
glr_loc =
```

28

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

```
PATH =
```

65 64 53 52 46 45 44 43 28

References

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

See Also

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

graphshortestpath

Purpose Solve shortest path problem in graph

Syntax

```
[dist, path, pred] = graphshortestpath(G, S)  
[dist, path, pred] = graphshortestpath(G, S, T)  
[...] = graphshortestpath(..., 'Directed', DirectedValue, ...)  
[...] = graphshortestpath(..., 'Method', MethodValue, ...)  
[...] = graphshortestpath(..., 'Weights', WeightsValue, ...)
```

Arguments

<i>G</i>	N-by-N sparse matrix that represents a graph. Nonzero entries in matrix <i>G</i> represent the weights of the edges.
<i>S</i>	Node in <i>G</i> .
<i>T</i>	Node in <i>G</i> .
<i>DirectedValue</i>	Property that indicates whether the graph is directed or undirected. Enter <code>false</code> for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is <code>true</code> .

<i>MethodValue</i>	<p>String that specifies the algorithm used to find the shortest path. Choices are:</p> <ul style="list-style-type: none">• 'Bellman-Ford' — Assumes weights of the edges to be nonzero entries in sparse matrix G. Time complexity is $O(N * E)$, where N and E are the number of nodes and edges respectively.• 'BFS' — Breadth-first search. Assumes all weights to be equal, and nonzero entries in sparse matrix G to represent edges. Time complexity is $O(N + E)$, where N and E are the number of nodes and edges respectively.• 'Acyclic' — Assumes G to be a directed acyclic graph and that weights of the edges are nonzero entries in sparse matrix G. Time complexity is $O(N + E)$, where N and E are the number of nodes and edges respectively.• 'Dijkstra' — Default algorithm. Assumes weights of the edges to be positive values in sparse matrix G. Time complexity is $O(\log(N) * E)$, where N and E are the number of nodes and edges respectively.
<i>WeightsValue</i>	<p>Column vector that specifies custom weights for the edges in matrix G. It must have one entry for every nonzero value (edge) in matrix G. The order of the custom weights in the vector must match the order of the nonzero values in matrix G when it is traversed column-wise. This property lets you use zero-valued weights. By default, <code>graphshortestpaths</code> gets weight information from the nonzero entries in matrix G.</p>

Description

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

`[dist, path, pred] = graphshortestpath(G, S)` determines the single-source shortest paths from node *S* to all other nodes in the graph represented by matrix *G*. Input *G* is an N-by-N sparse matrix that represents a graph. Nonzero entries in matrix *G* represent the weights of the edges. *dist* are the N distances from the source to every node (using Inf for nonreachable nodes and 0 for the source node). *path* contains the winning paths to every node. *pred* contains the predecessor nodes of the winning paths.

`[dist, path, pred] = graphshortestpath(G, S, T)` determines the single source-single destination shortest path from node *S* to node *T*.

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

`[...] = graphshortestpath(..., 'Directed', DirectedValue, ...)` indicates whether the graph is directed or undirected. Set *DirectedValue* to `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`.

`[...] = graphshortestpath(..., 'Method', MethodValue, ...)` lets you specify the algorithm used to find the shortest path. Choices are:

- 'Bellman-Ford' — Assumes weights of the edges to be nonzero entries in sparse matrix *G*. Time complexity is $O(N \cdot E)$, where *N* and *E* are the number of nodes and edges respectively.
- 'BFS' — Breadth-first search. Assumes all weights to be equal, and nonzero entries in sparse matrix *G* to represent edges. Time

complexity is $O(N+E)$, where N and E are the number of nodes and edges respectively.

- 'Acyclic' — Assumes G to be a directed acyclic graph and that weights of the edges are nonzero entries in sparse matrix G . Time complexity is $O(N+E)$, where N and E are the number of nodes and edges respectively.
- 'Dijkstra' — Default algorithm. Assumes weights of the edges to be positive values in sparse matrix G . Time complexity is $O(\log(N)*E)$, where N and E are the number of nodes and edges respectively.

[...] = graphshortestpath(..., 'Weights', *WeightsValue*, ...)
lets you specify custom weights for the edges. *WeightsValue* is a column vector having one entry for every nonzero value (edge) in matrix G . The order of the custom weights in the vector must match the order of the nonzero values in matrix G when it is traversed column-wise. This property lets you use zero-valued weights. By default, graphshortestpath gets weight information from the nonzero entries in matrix G .

Examples

Finding the Shortest Path in a Directed Graph

- 1 Create and view a directed graph with 6 nodes and 11 edges.

```
W = [.41 .99 .51 .32 .15 .45 .38 .32 .36 .29 .21];
DG = sparse([6 1 2 2 3 4 4 5 5 6 1],[2 6 3 5 4 1 6 3 4 3 5],W)
```

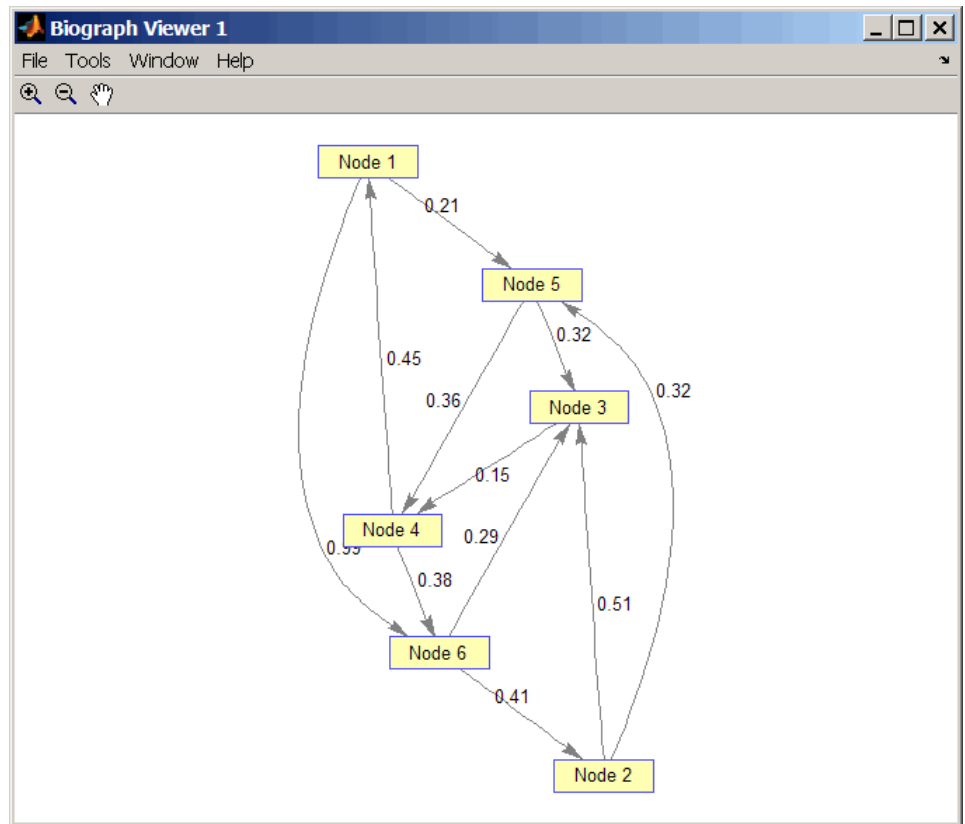
```
DG =

(4,1)    0.4500
(6,2)    0.4100
(2,3)    0.5100
(5,3)    0.3200
(6,3)    0.2900
(3,4)    0.1500
(5,4)    0.3600
(1,5)    0.2100
```

graphshortestpath

```
(2,5)      0.3200  
(1,6)      0.9900  
(4,6)      0.3800
```

```
h = view(biograph(DG,[],'ShowWeights','on'))  
Biograph object with 6 nodes and 11 edges.
```



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

```
[dist,path,pred] = graphshortestpath(DG,1,6)
```



```
dist =
```

```
    0.9500
```

```
path =
```

```
    1     5     4     6
```

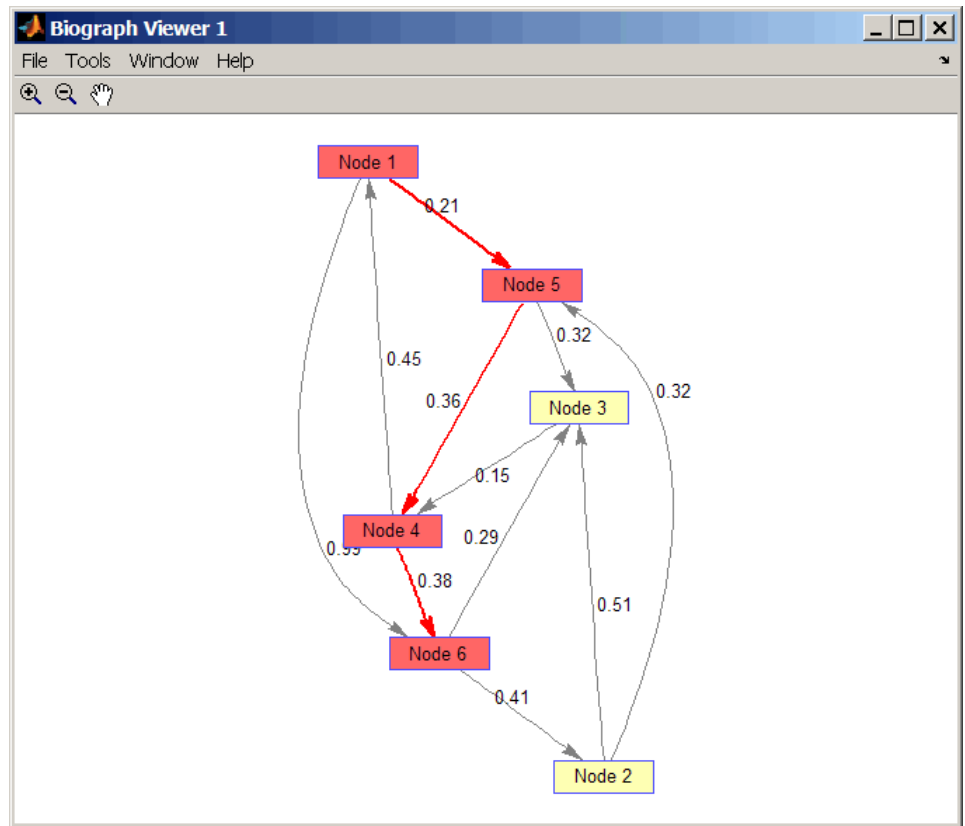
```
pred =
```

```
    0     6     5     5     1     4
```

- 3** Mark the nodes and edges of the shortest path by coloring them red and increasing the line width.

```
set(h.Nodes(path), 'Color', [1 0.4 0.4])  
edges = getedgesbynodeid(h, get(h.Nodes(path), 'ID'));  
set(edges, 'LineColor', [1 0 0])  
set(edges, 'LineWidth', 1.5)
```

graphshortestpath



Finding the Shortest Path in an Undirected Graph

- 1 Create and view an undirected graph with 6 nodes and 11 edges.

```
UG = tril(DG + DG')
```

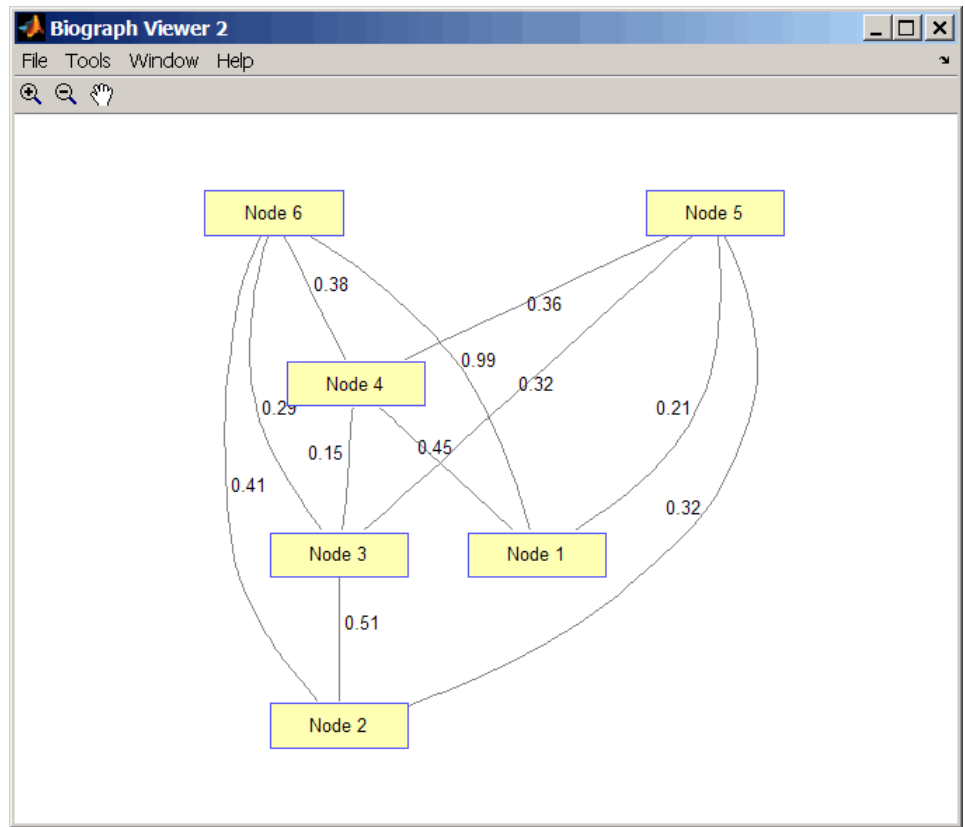
```
UG =
```

```
(4,1)      0.4500  
(5,1)      0.2100
```

(6,1)	0.9900
(3,2)	0.5100
(5,2)	0.3200
(6,2)	0.4100
(4,3)	0.1500
(5,3)	0.3200
(6,3)	0.2900
(5,4)	0.3600
(6,4)	0.3800

```
h = view(biograph(UG,[],'ShowArrows','off','ShowWeights','on'))  
Biograph object with 6 nodes and 11 edges.
```

graphshortestpath



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

```
[dist,path,pred] = graphshortestpath(UG,1,6,'directed',false)
```

```
dist =
```

```
0.8200
```

```
path =
```

```
1    5    3    6
```

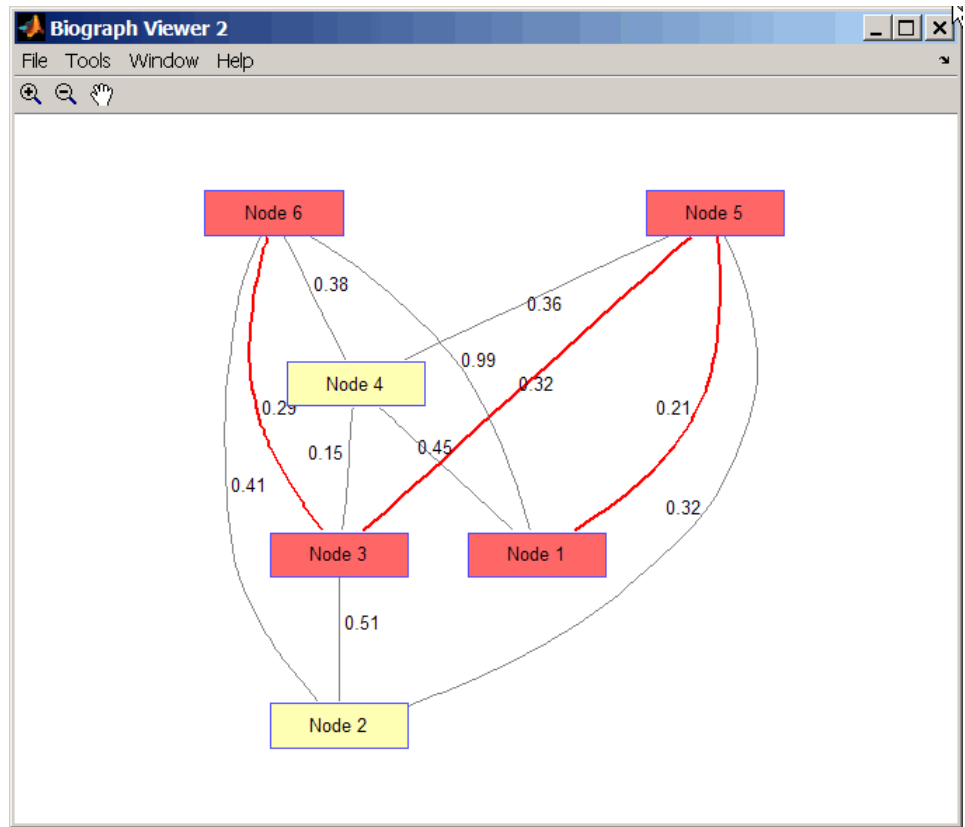
```
pred =
```

```
0    5    5    1    1    3
```

- 3** Mark the nodes and edges of the shortest path by coloring them red and increasing the line width.

```
set(h.Nodes(path), 'Color', [1 0.4 0.4])
fowEdges = getedgesbynodeid(h, get(h.Nodes(path), 'ID'));
revEdges = getedgesbynodeid(h, get(h.Nodes(fliplr(path)), 'ID'));
edges = [fowEdges; revEdges];
set(edges, 'LineColor', [1 0 0])
set(edges, 'LineWidth', 1.5)
```

graphshortestpath



References

- [1] Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269-271.
- [2] Bellman, R. (1958). On a Routing Problem. *Quarterly of Applied Mathematics 16(1)*, 87-90.
- [3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). *The Boost Graph Library User Guide and Reference Manual*, (Upper Saddle River, NJ:Pearson Education).

See Also

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

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

graphtopoorder

Purpose Perform topological sort of directed acyclic graph

Syntax `order = graphtopoorder(G)`

Arguments

`G` N-by-N sparse matrix that represents a directed acyclic graph. Nonzero entries in matrix `G` indicate the presence of an edge.

Description

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

`order = graphtopoorder(G)` returns an index vector with the order of the nodes sorted topologically. In topological order, an edge can exist between a source node `u` and a destination node `v`, if and only if `u` appears before `v` in the vector `order`. `G` is an N-by-N sparse matrix that represents a directed acyclic graph (DAG). Nonzero entries in matrix `G` indicate the presence of an edge.

Examples

- 1 Create and view a directed acyclic graph (DAG) with six nodes and eight edges.

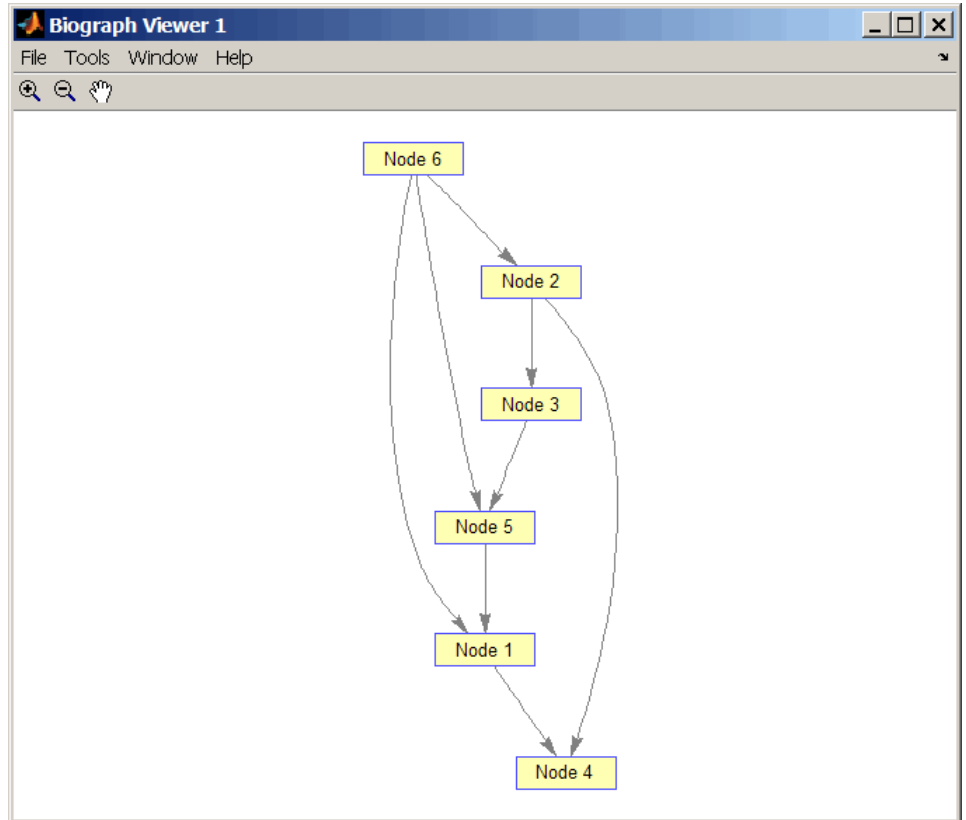
```
DG = sparse([6 6 6 2 2 3 5 1],[2 5 1 3 4 5 1 4],true,6,6)
```

```
DG =
```

```
(5,1)    1
(6,1)    1
(6,2)    1
(2,3)    1
(1,4)    1
(2,4)    1
(3,5)    1
(6,5)    1
```



```
view(biograph(DG))
```



2 Find the topological order of the DAG.

```
order = graphtopoorder(DG)
```

```
order =
```

```
6 2 3 5 1 4
```

3 Permute the nodes so that they appear ordered in the graph display.

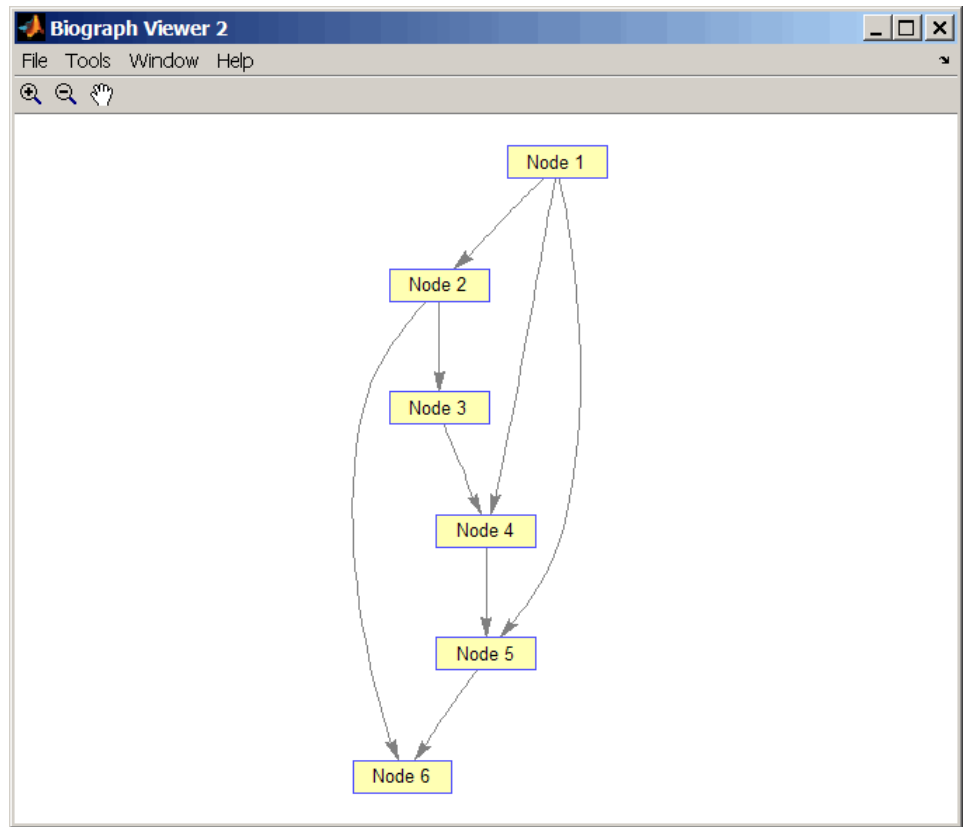
graph_topoorder

```
DG = DG(order,order)
```

```
DG =
```

```
(1,2)      1  
(2,3)      1  
(1,4)      1  
(3,4)      1  
(1,5)      1  
(4,5)      1  
(2,6)      1  
(5,6)      1
```

```
view(bigraph(DG))
```



References

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

See Also

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

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

graphtraverse

Purpose Traverse graph by following adjacent nodes

Syntax

```
[disc, pred, closed] = graphtraverse(G, S)
[...] = graphtraverse(G, S, ...'Depth', DepthValue, ...)
[...] = graphtraverse(G, S, ...'Directed', DirectedValue, ...)
[...] = graphtraverse(G, S, ...'Method', MethodValue, ...)
```

Arguments

<i>G</i>	N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix <i>G</i> indicate the presence of an edge.
<i>S</i>	Integer that indicates the source node in graph <i>G</i> .
<i>DepthValue</i>	Integer that indicates a node in graph <i>G</i> that specifies the depth of the search. Default is Inf (infinity).
<i>DirectedValue</i>	Property that indicates whether graph <i>G</i> is directed or undirected. Enter <code>false</code> for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is <code>true</code> .
<i>MethodValue</i>	String that specifies the algorithm used to traverse the graph. Choices are: <ul style="list-style-type: none">• 'BFS' — Breadth-first search. Time complexity is $O(N+E)$, where <i>N</i> and <i>E</i> are number of nodes and edges respectively.• 'DFS' — Default algorithm. Depth-first search. Time complexity is $O(N+E)$, where <i>N</i> and <i>E</i> are number of nodes and edges respectively.

Description

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

`[disc, pred, closed] = graphtraverse(G, S)` traverses graph *G* starting from the node indicated by integer *S*. *G* is an N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix *G* indicate the presence of an edge. *disc* is a vector of node indices in the order in which they are discovered. *pred* is a vector of predecessor node indices (listed in the order of the node indices) of the resulting spanning tree. *closed* is a vector of node indices in the order in which they are closed.

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

`[...] = graphtraverse(G, S, ...'Depth', DepthValue, ...)` specifies the depth of the search. *DepthValue* is an integer indicating a node in graph *G*. Default is Inf (infinity).

`[...] = graphtraverse(G, S, ...'Directed', DirectedValue, ...)` indicates whether the graph is directed or undirected. Set *DirectedValue* to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

`[...] = graphtraverse(G, S, ...'Method', MethodValue, ...)` lets you specify the algorithm used to traverse the graph. Choices are:

- 'BFS' — Breadth-first search. Time complexity is $O(N+E)$, where *N* and *E* are number of nodes and edges respectively.
- 'DFS' — Default algorithm. Depth-first search. Time complexity is $O(N+E)$, where *N* and *E* are number of nodes and edges respectively.

Examples

- 1 Create a directed graph with 10 nodes and 12 edges.

```
DG = sparse([1 2 3 4 5 5 6 7 8 8 9],...
            [2 4 1 5 3 6 7 9 8 1 10 2],true,10,10)
```

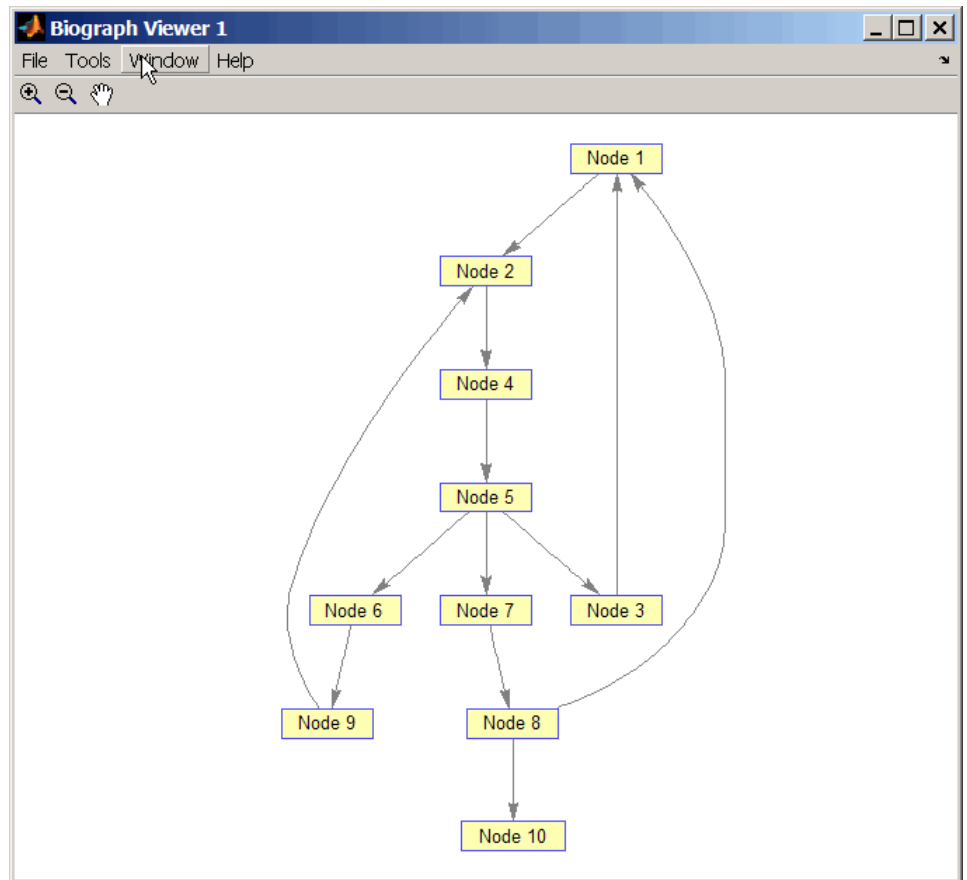
graphtraverse

DG =

(3,1)	1
(8,1)	1
(1,2)	1
(9,2)	1
(5,3)	1
(2,4)	1
(4,5)	1
(5,6)	1
(5,7)	1
(7,8)	1
(6,9)	1
(8,10)	1

h = view(biograph(DG))

Biograph object with 10 nodes and 12 edges.



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

```
order = graphtraverse(DG,4)
```

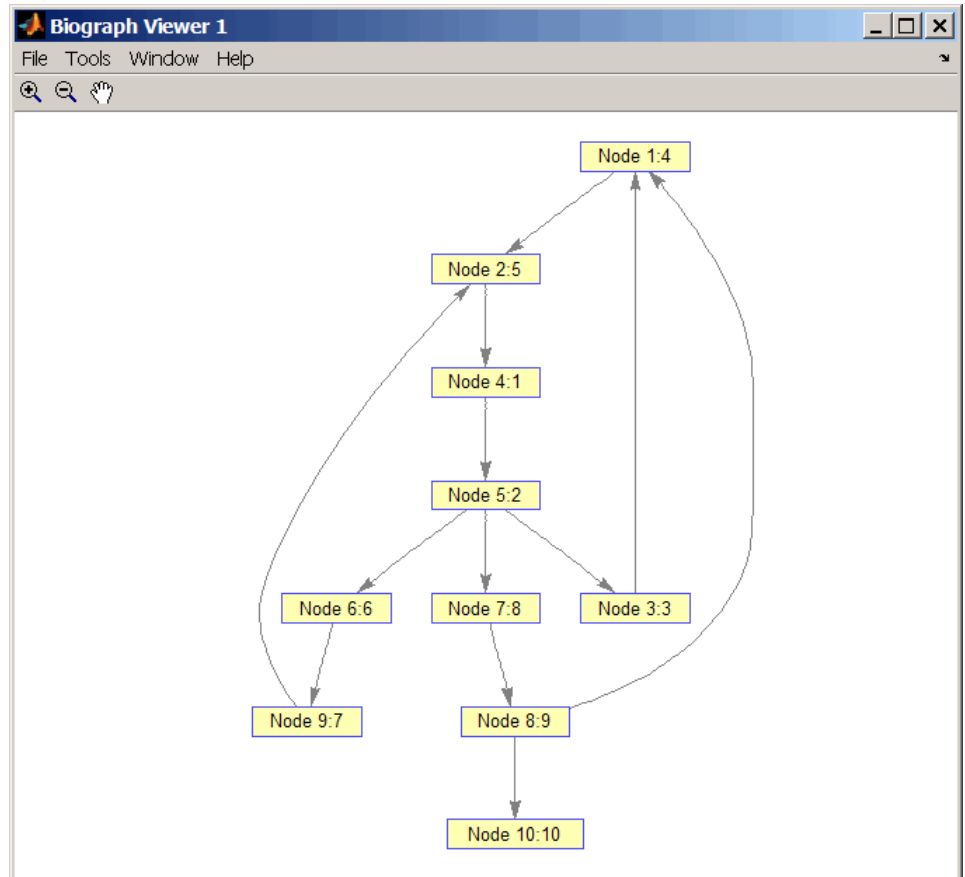
```
order =
```

```
4 5 3 1 2 6 9 7 8 10
```

graphtraverse

3 Label the nodes with the DFS discovery order.

```
for i = 1:10
    h.Nodes(order(i)).Label = ...
    sprintf('%s:%d',h.Nodes(order(i)).ID,i);
end
h.ShowTextInNodes = 'label'
dolayout(h)
```



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

```
order = graphtraverse(DG,4,'Method','BFS')
```

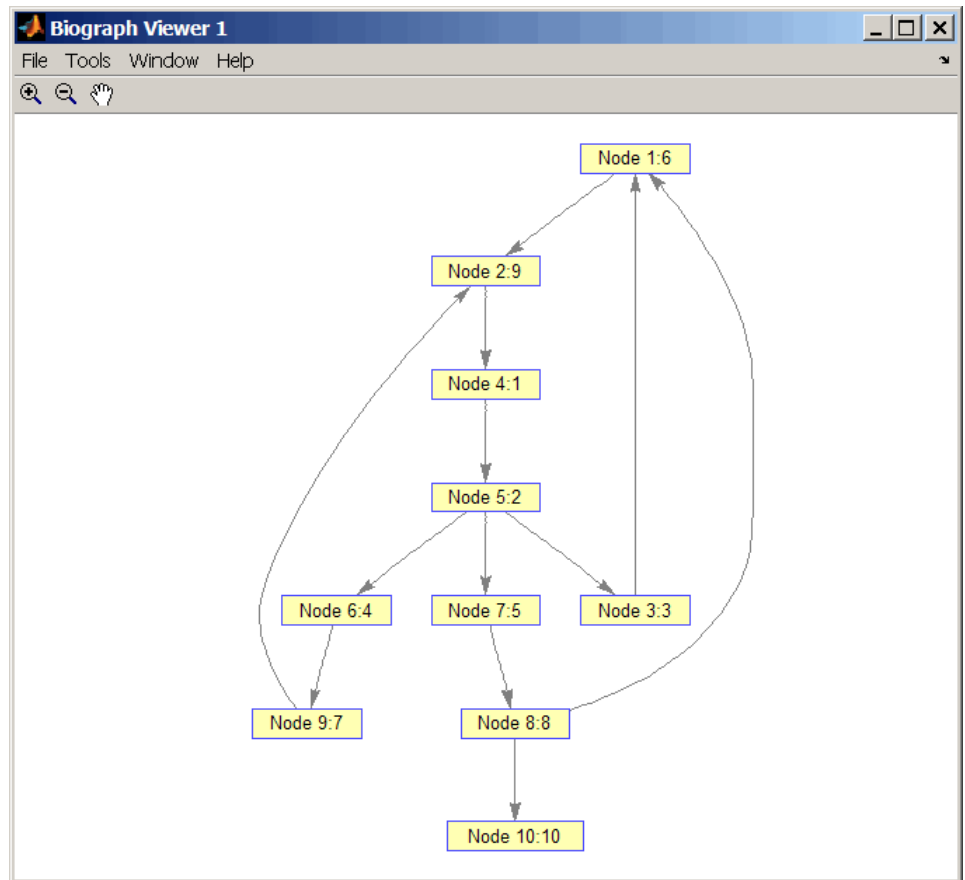
```
order =
```

```
    4    5    3    6    7    1    9    8    2   10
```

- 5** Label the nodes with the BFS discovery order.

```
for i = 1:10
    h.Nodes(order(i)).Label = ...
    sprintf('%s:%d',h.Nodes(order(i)).ID,i);
end
h.ShowTextInNodes = 'label'
dolayout(h)
```

graphtraverse



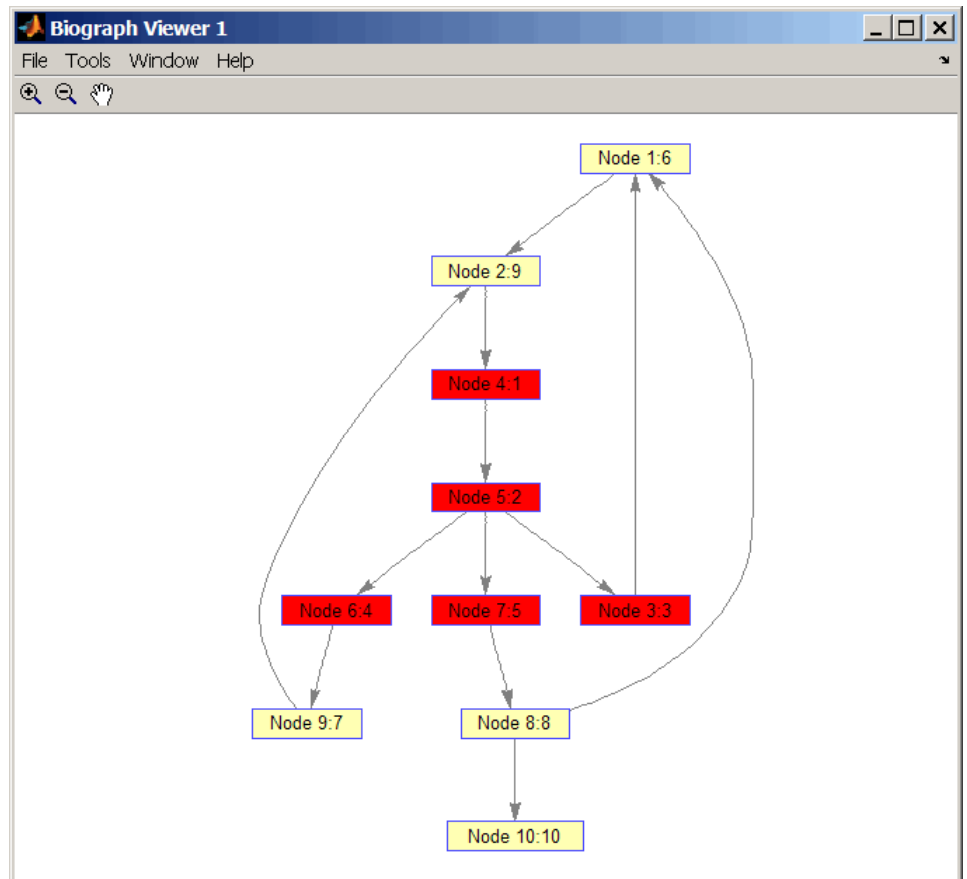
- 6 Find and color nodes that are close to (within two edges of) node 4.

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

```
node_idx =
```

```
4 5 3 6 7
```

```
set(h.nodes(node_idx),'Color',[1 0 0])
```



References

- [1] Sedgewick, R., (2002). Algorithms in C++, Part 5 Graph Algorithms (Addison-Wesley).
- [2] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

graphtraverse

See Also

Bioinformatics Toolbox functions: `graphallshortestpaths`,
`graphconncomp`, `graphisdag`, `graphisomorphism`, `graphisspanntree`,
`graphmaxflow`, `graphminspanntree`, `graphpred2path`,
`graphshortestpath`, `graphtopoorder`

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

Purpose

Align query sequence to profile using hidden Markov model alignment

Syntax

```
Score = hmmprofalign(Model, Seq)  
[Score, Alignment] = hmmprofalign(Model, Seq)  
[Score, Alignment, Pointer] = hmmprofalign(Model, Seq)  
hmmprofalign(..., 'ShowScore', ShowScoreValue, ...)  
hmmprofalign(..., 'Flanks', FlanksValue, ...)  
hmmprofalign(..., 'ScoreFlanks', ScoreFlanksValue, ...)  
hmmprofalign(..., 'ScoreNullTransitions',  
    ScoreNullTransitionsValue, ...)
```

Arguments

<i>Model</i>	Hidden Markov model created with the function <code>hmmprofstruct</code> .
<i>Seq</i>	Amino acid or nucleotide sequence. You can also enter a structure with the field <code>Sequence</code> .
<i>ShowScoreValue</i>	Controls the display of the scoring space and the winning path. Choices are <code>true</code> or <code>false</code> (default).
<i>FlanksValue</i>	Controls the inclusion of the symbols generated by the FLANKING INSERT states in the output sequence. Choices are <code>true</code> or <code>false</code> (default).
<i>ScoreFlanksValue</i>	Controls the inclusion of the transition probabilities for the flanking states in the raw score. Choices are <code>true</code> or <code>false</code> (default).
<i>ScoreNullTransitionsValue</i>	Controls the adjustment of the raw score using the null model for transitions (<code>Model.NullX</code>). Choices are <code>true</code> or <code>false</code> (default).

Description

`Score = hmmprofalign(Model, Seq)` returns the score for the optimal alignment of the query amino acid or nucleotide sequence (*Seq*) to the profile hidden Markov model (*Model*). Scores are computed using log-odd ratios for emission probabilities and log probabilities for state transitions.

`[Score, Alignment] = hmmprofalign(Model, Seq)` also returns a string showing the optimal profile alignment.

Uppercase letters and dashes correspond to MATCH and DELETE states respectively (the combined count is equal to the number of states in the model). Lowercase letters are emitted by the INSERT states. For more information about the HMM profile, see `hmmprofstruct`.

`[Score, Alignment, Pointer] = hmmprofalign(Model, Seq)` also returns a vector of the same length as the profile model with indices pointing to the respective symbols of the query sequence. Null pointers (NaN) mean that such states did not emit a symbol in the aligned sequence because they represent model jumps from the BEGIN state of a MATCH state, model jumps from the from a MATCH state to the END state, or because the alignment passed through DELETE states.

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

`hmmprofalign(..., 'ShowScore', ShowScoreValue, ...)`, when *ShowScoreValue* is true, displays the scoring space and the winning path.

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

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

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

Note Multiple target alignment is not supported in this implementation. All the `Model.LoopX` probabilities are ignored.

Examples

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

See Also

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

hmmprofestimate

Purpose Estimate profile hidden Markov model (HMM) parameters using pseudocounts

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

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

Arguments

Model	Hidden Markov model created with the function <code>hmmprofstruct</code> .
MultipleAlignment	Array of sequences. Sequences can also be a structured array with the aligned sequences in a field <code>Aligned</code> or <code>Sequences</code> , and the optional names in a field <code>Header</code> or <code>Name</code> .
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 <code>Model.NullEmission</code> .
BMx	Property to set the background transition probabilities from any MATCH state (<code>[M->M M->I M->D]</code>). Default values are taken from <code>hmmprofstruct</code> .
BDx	Property to set the background transition probabilities from any DELETE state (<code>[D->M D->D]</code>). Default values are taken from <code>hmmprofstruct</code> .

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 ($[M \rightarrow M \ M \rightarrow I \ M \rightarrow D]$). Default values are taken from `hmmprofstruct`.

hmmprofestimate

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

See Also

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

Purpose Generate random sequence drawn from profile hidden Markov model (HMM)

Syntax

```
Sequence = hmmprofgenerate(Model)
[Sequence, Profptr] = hmmprofgenerate(Model)
... = hmmprofgenerate(Model, ...'Align', AlignValue, ...)
... = hmmprofgenerate(Model, ...'Flanks', FlanksValue, ...)
... = hmmprofgenerate(Model, ...'Signature', SignatureValue,
    ...)
```

Arguments

<i>Model</i>	Hidden Markov model created with the <code>hmmprofstruct</code> function.
<i>AlignValue</i>	Controls the use of uppercase letters for matches and lowercase letters for inserted letters. Choices are <code>true</code> or <code>false</code> (default).
<i>FlanksValue</i>	Controls the inclusion of the symbols generated by the FLANKING INSERT states in the output sequence. Choices are <code>true</code> or <code>false</code> (default).
<i>SignatureValue</i>	Controls the return of the most likely path and symbols. Choices are <code>true</code> or <code>false</code> (default).

Description `Sequence = hmmprofgenerate(Model)` returns the string *Sequence* showing a sequence of amino acids or nucleotides drawn from the profile *Model*. The length, alphabet, and probabilities of the *Model* are stored in a structure. For more information about this structure, see `hmmprofstruct`.

`[Sequence, Profptr] = hmmprofgenerate(Model)` returns a vector of the same length as the profile model pointing to the respective states in the output sequence. Null pointers (0) mean that such states do not exist in the output sequence, either because they are never touched (i.e., jumps from the BEGIN state to MATCH states or from MATCH states to the

hmmprofgenerate

END state), or because DELETE states are not in the output sequence (not aligned output; see below).

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

`... = hmmprofgenerate(Model, ...'Align', AlignValue, ...)` if *Align* is true, the output sequence is aligned to the model as follows: uppercase letters and dashes correspond to MATCH and DELETE states respectively (the combined count is equal to the number of states in the model). Lowercase letters are emitted by the INSERT or FLANKING INSERT states. If *AlignValue* is false, the output is a sequence of uppercase symbols. The default value is true.

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

`... = hmmprofgenerate(Model, ...'Signature', SignatureValue, ...)` if *SignatureValue* is true, returns the most likely path and symbols. The default value is false.

Examples

```
load('hmm_model_examples','model_7tm_2') % load a model example
rand_sequence = hmmprofgenerate(model_7tm_2)
```

See Also

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

Purpose

Concatenate prealigned strings of several sequences to profile hidden Markov model (HMM)

Syntax

```
hmmprofmerge(Sequences)  
hmmprofmerge(Sequences, Names)  
hmmprofmerge(Sequences, Names, Scores)
```

Arguments

Sequences Array of sequences. *Sequences* can also be a structured array with the aligned sequences in a field `Aligned` or `Sequences`, and the optional names in a field `Header` or `Name`.

Names Names for the sequences. Enter a vector of names.

Scores Pairwise alignment scores from the function `hmmprofalign`. Enter a vector of values with the same length as the number of sequences in *Sequences*.

Description

`hmmprofmerge(Sequences)` displays a set of prealigned sequences to an HMM model profile. The output is aligned corresponding to the HMM states.

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

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

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

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

hmmprofmerge

Examples

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

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

See Also

Bioinformatics Toolbox functions: `hmmprofalign`, `hmmprofstruct`

Purpose

Create or edit hidden Markov model (HMM) profile structure

Syntax

```
Model = hmmprofstruct(Length)  
Model = hmmprofstruct(Length, Field1, Field1Value, Field2,  
    Field2Value, ...)  
NewModel = hmmprofstruct(Model, Field1,  
    Field1Value, Field2,  
    Field2Value, ...)
```

Arguments

<i>Length</i>	Number of match states in the model.
<i>Model</i>	MATLAB structure containing fields for the parameters of an HMM profile created with the <code>hmmprofstruct</code> function.
<i>Field</i>	String containing a field name in the structure <i>Model</i> . See the table below for field names.
<i>FieldValue</i>	Value associated with <i>Field</i> . See the table below for descriptions.

Return Values

<i>Model</i>	MATLAB structure containing fields for the parameters of an HMM profile.
--------------	--

Description

Model = `hmmprofstruct(Length)` returns *Model*, a MATLAB structure containing fields for the parameters of an HMM profile. *Length* specifies the number of match states in the model. All other required parameters are set to the default values.

Model = `hmmprofstruct(Length, Field1, Field1Value, Field2, Field2Value, ...)` returns an HMM profile structure using the specified parameters. All other required parameters are set to default values.

NewModel = `hmmprofstruct(Model, Field1, Field1Value, Field2, Field2Value, ...)` returns an updated HMM profile

structure using the specified parameters. All other parameters are taken from the input *Model*.

HMM Profile Structure

The MATLAB structure *Model* contains the following fields, which are the required and optional parameters of an HMM profile. All probability values are in the [0 1] range.

Field	Description
ModelLength	Integer specifying the length of the profile (number of MATCH states).
Alphabet	String specifying the alphabet used in the model. Choices are 'AA' (default) or 'NT'. <hr/> Note AlphaLength is 20 for 'AA' and 4 for 'NT'. <hr/>
MatchEmission	Symbol emission probabilities in the MATCH states. Either of the following: <ul style="list-style-type: none">• A matrix of size ModelLength-by-AlphabetLength, where each row corresponds to the emission distribution for a specific MATCH state. Defaults to uniform distributions.• A structure containing residue counts, such as returned by <code>aaccount</code> or <code>basecount</code>.

Field	Description
InsertEmission	<p>Symbol emission probabilities in the INSERT state.</p> <p>Either of the following:</p> <ul style="list-style-type: none"> • A matrix of size <code>ModelLength</code>-by-<code>AlphaLength</code>, where each row corresponds to the emission distribution for a specific INSERT state. Defaults to uniform distributions. • A structure containing residue counts, such as returned by <code>aaccount</code> or <code>basecount</code>.
NullEmission	<p>Symbol emission probabilities in the MATCH and INSERT states for the NULL model.</p> <p>Either of the following:</p> <ul style="list-style-type: none"> • A 1-by-<code>AlphaLength</code> row vector. Defaults to a uniform distribution. • A structure containing residue counts, such as returned by <code>aaccount</code> or <code>basecount</code>. <hr/> <p>Note The NULL model is used to compute the log-odds ratio at every state and avoid overflow when propagating the probabilities through the model.</p> <hr/> <p>Note NULL probabilities are also known as the background probabilities.</p>

hmmprofstruct

Field	Description
BeginX	<p>BEGIN state transition probabilities.</p> <p>Format is a 1-by-(ModelLength + 1) row vector:</p> $[B \rightarrow D1 \ B \rightarrow M1 \ B \rightarrow M2 \ B \rightarrow M3 \ \dots \ B \rightarrow Mend]$ <hr/> <p>Note If necessary, <code>hmmprofstruct</code> will normalize the data such that the sum of the transition probabilities from the BEGIN state equals 1:</p> $\text{sum}(\text{Model.BeginX}) = 1$ <p>For fragment profiles:</p> $\text{sum}(\text{Model.BeginX}(3:\text{end})) = 0$ <hr/> <p>Default is [0.01 0.99 0 0 ... 0].</p>

Field	Description
MatchX	<p>MATCH state transition probabilities.</p> <p>Format is a 4-by-(ModelLength - 1) matrix:</p> <pre data-bbox="762 427 1322 548"> [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] </pre> <hr/> <p>Note If necessary, hmmprofstruct will normalize the data such that the sum of the transition probabilities from every MATCH state equals 1:</p> $\text{sum}(\text{Model.MatchX}) = [1 \ 1 \ \dots \ 1]$ <p>For fragment profiles:</p> $\text{sum}(\text{Model.MatchX}(4,:)) = 0$ <hr/> <p>Default is <code>repmat([0.998 0.001 0.001 0],ModelLength-1,1)</code>.</p>

hmmprofstruct

Field	Description
InsertX	<p>INSERT state transition probabilities.</p> <p>Format is a 2-by-(ModelLength - 1) matrix:</p> <pre>[I1->M2 I2->M3 ... I[end-1]->Mend; I1->I1 I2->I2 ... I[end-1]->I[end-1]]</pre> <hr/> <p>Note If necessary, <code>hmmprofstruct</code> will normalize the data such that the sum of the transition probabilities from every INSERT state equals 1:</p> $\text{sum}(\text{Model.InsertX}) = [1 \ 1 \ \dots \ 1]$ <hr/> <p>Default is <code>repmat([0.5 0.5],ModelLength-1,1)</code>.</p>

Field	Description
DeleteX	<p>DELETE state transition probabilities.</p> <p>Format is a 2-by-(ModelLength - 1) matrix:</p> <pre data-bbox="762 427 1292 484">[D1->M2 D2->M3 ... D[end-1]->Mend ; D1->D2 D2->D3 ... D[end-1]->Dend]</pre> <hr/> <p>Note If necessary, hmmprofstruct will normalize the data such that the sum of the transition probabilities from every DELETE state equals 1:</p> $\text{sum}(\text{Model.DeleteX}) = [1 \ 1 \ \dots \ 1]$ <hr/> <p>Default is <code>repmat([0.5 0.5],ModelLength-1,1)</code>.</p>

hmmprofstruct

Field	Description
FlankingInsertX	<p data-bbox="676 317 1286 378">Flanking insert states (N and C) used for LOCAL profile alignment.</p> <p data-bbox="676 397 1006 425">Format is a 2-by-2 matrix:</p> $\begin{bmatrix} N \rightarrow B & C \rightarrow T \\ N \rightarrow N & C \rightarrow C \end{bmatrix}$ <hr/> <p data-bbox="676 591 1233 713">Note If necessary, hmmprofstruct will normalize the data such that the sum of the transition probabilities from Flanking Insert states equals 1:</p> $\text{sum}(\text{Model.FlankingInsertsX}) = [1 \ 1]$ <hr/> <p data-bbox="676 911 1132 939">Note To force global alignment use:</p> $\text{Model.FlankingInsertsX} = [1 \ 1; 0 \ 0]$ <hr/> <p data-bbox="676 1104 1139 1131">Default is [0.01 0.01; 0.99 0.99].</p>

Field	Description
LoopX	<p>Loop states transition probabilities used for multiple hits alignment.</p> <p>Format is a 2-by-2 matrix:</p> <pre data-bbox="762 458 951 517">[E->C J->B ; E->J J->J]</pre> <hr/> <p>Note If necessary, hmmprofstruct will normalize the data such that the sum of the transition probabilities from Loop states equals 1:</p> $\text{sum}(\text{Model}.\text{LoopX}) = [1 \ 1]$ <hr/> <p>Default is [0.5 0.01; 0.5 0.99].</p>

Field	Description
NullX	<p>Null transition probabilities used to provide scores with log-odds values also for state transitions.</p> <p>Format is a 2-by-1 column vector:</p> $\begin{bmatrix} G \rightarrow F \\ G \rightarrow G \end{bmatrix}$ <hr/> <p>Note If necessary, <code>hmmprofstruct</code> will normalize the data such that the sum of the transition probabilities from Null states equals 1:</p> $\text{sum}(\text{Model.NullX}) = 1$ <hr/> <p>Default is <code>[0.01; 0.99]</code>.</p>
IDNumber	Optional. User-assigned identification number.
Description	Optional. User-assigned description of the model.

HMM Profile Model

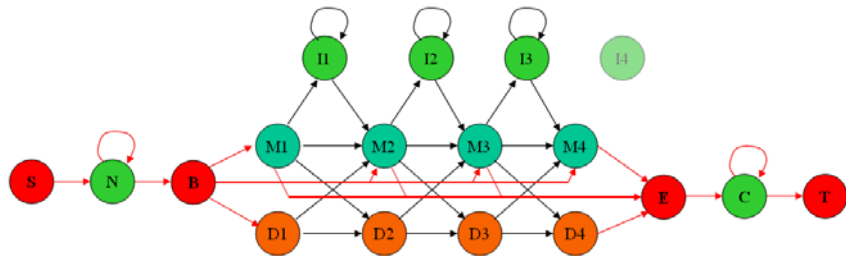
An HMM profile model is a common statistical tool for modeling structured sequences composed of symbols. These symbols include randomness in both the output (emission of symbols) and the state transitions of the process. Markov models are generally represented by state diagrams.

The following figure is a state diagram for an HMM profile of length four. INSERT, MATCH, and DELETE states are in the center section.

- INSERT state represents the excess of one or more symbols in the target sequence that are not included in the profile.

- MATCH state means that the target sequence is aligned to the profile at the specific location.
- DELETE state represents a gap or symbol absence in the target sequence (also known as a silent state because it does not emit any symbols).

Flanking states (S, N, B, E, C, T) are used for proper modeling of the ends of the sequence, either for global, local or fragment alignment of the profile. S, B, E, and T are silent, while N and C are used to insert symbols at the flanks.



Examples

Creating an HMM Profile Structure

Create an HMM profile structure with 100 MATCH states, using the amino acid alphabet.

```
hmmProfile = hmmprofstruct(100, 'Alphabet', 'AA')
```

Editing an HMM Profile Structure

- 1 Use the `pfamhmmread` function to create an HMM profile structure from `pf00002.1s`, a PFAM HMM-formatted file included with the software.

```
hmm02 = pfamhmmread('pf00002.1s');
```

hmmprofstruct

- 2 Modify the HMM profile structure to force a global alignment by setting the looping transition probabilities in the flanking insert states to zero.

```
hmm02 = hmmprofstruct(hmm02,'FlankingInsertX',[0 0;1 1]);  
hmm02.FlankingInsertX
```

```
ans =
```

```
0    0  
1    1
```

See Also

Bioinformatics Toolbox functions: `aaccount`, `basecount`, `gethmmprof`, `hmmprofalign`, `hmmprofestimate`, `hmmprofgenerate`, `hmmprofmerge`, `pfamhmmread`, `showhmmprof`

Purpose

Look up Illumina BeadStudio target (probe) sequence and annotation information

Syntax

```
AnnotStruct = ilmnblookup(AnnotationFile, ID)  
AnnotStruct = ilmnblookup(AnnotationFile,  
ID, 'LookUpField',  
    LookUpFieldValue)
```

Arguments

AnnotationFile String specifying a file name or a path and file name of an Illumina annotation file (CSV, BGX, or TXT format). If you specify only a file name, that file must be on the MATLAB search path or in the current directory.

Note Currently, the Bioinformatics Toolbox software supports annotation files associated with only these Illumina whole-genome arrays:

- HumanRef-8_V3
- HumanWG-6_V3
- HumanRef-8_V2
- HumanWG-6_V2
- MouseRef-8_V2
- MouseWG-6_V2
- MouseRef-8_V1
- MouseWG-6_V1
- RatRef-12_V1

Tip You can download Illumina annotation files, such as `HumanRef-8_V3_0_R0_11282963_A.bgx`, from the Illumina Web site.

ID String or cell array of strings representing a unique identifier(s) for one or more targets (probes) on an Illumina microarray.

Tip By default, *ID* must match the *Search_key* field in *AnnotationFile*. However, you can use an identifier that corresponds to any of the fields in *AnnotationFile*, then set the 'LookUpField' property appropriately. For example, if you want to look up annotation information for the targets (probes) on chromosome 7 only, set *ID* to '7', then set *LookUpFieldValue* to 'Chromosome'. For a list of all fields in *AnnotationFile*, see the following tables.

LookUpFieldValue Field in *AnnotationFile* where *ilmnbslookup* looks for the specified *ID*. Default is the *Search_key* field.

Tip Set this property so that it corresponds to the *ID* you use as input.

Return Values

AnnotStruct Structure containing the probe sequence and annotation information for one or more targets (probes) specified by *ID*, and by *AnnotationFile*, an Illumina annotation file.

AnnotStruct contains the same fields as *AnnotationFile*. The fields are described in the following two tables.

ilmnbslookup

Description

AnnotStruct = `ilmnbslookup(AnnotationFile, ID)` returns *AnnotStruct*, a structure containing probe sequence and annotation information for one or more targets (probes) specified by *ID*, and by *AnnotationFile*, an Illumina annotation file (CSV, BGX, or TXT format).

AnnotStruct contains the same fields as *AnnotationFile*. The fields are described in the following two tables.

Structure Created from Illumina CSV Annotation File

Field	Description
Search_key	Internal identifier for the target, useful for custom design array
Target	Unique identifier for the target
ProbeId	Illumina probe identifier
Gid	GenBank identifier for the gene
Transcript	Illumina internal transcript identifier
Accession	GenBank accession number for the gene
Symbol	Typically, the gene symbol
Type	Probe type
Start	Starting position of the probe sequence in the GenBank record
Probe_Sequence	Sequence of the probe
Definition	Definition field from the GenBank record
Ontology	Gene Ontology terms associated with the gene
Synonym	Synonyms for the gene (from the GenBank record)

Structure Created from a BGX or TXT Annotation File

Field	Description
Accession	GenBank accession number for the gene
Array_Address_Id	Decoder identifier
Chromosome	Chromosome on which the gene is located
Cytoband	Cytogenetic banding region of the chromosome on which the gene associated with the target is located
Definition	Definition field from the GenBank record
Entrez_Gene_ID	Entrez Gene database identifier for the gene
GI	GenBank identifier for the gene
ILMN_Gene	Illuminainternal gene symbol
Obsolete_Probe_Id	Probe identifier before BGX annotation files
Ontology_Component	Gene Ontology cellular components associated with the gene
Ontology_Function	Gene Ontology molecular functions associated with the gene
Ontology_Process	Gene Ontology biological processes associated with the gene
Probe_Chr_Orientation	Orientation of the probe on the NCBI genome build
Probe_Coordinates	Genomic position of the probe on the NCBI genome build
Probe_Id	Illuminaprobe identifier
Probe_Sequence	Sequence of the probe

Structure Created from a BGX or TXT Annotation File (Continued)

Field	Description
Probe_Start	Start position of the probe relative to the 5' end of the source transcript sequence
Probe_Type	Information about what the probe is targeting
Protein_Product	NCBI protein accession number
RefSeq_ID	Identifier from the NCBI RefSeq database
Reporter_Composite_map	Information associated with control probes
Reporter_Group_Name	Information associated with control probes
Reporter_Group_id	Information associated with control probes
Search_Key	Internal identifier for the target, useful for custom design array
Source	Source from which the transcript sequence was obtained
Source_Reference_ID	Source's identifier
Species	Species associated with the gene
Symbol	Typically, the gene symbol
Synonyms	Synonyms for the gene (from the GenBank record)
Transcript	Illumina internal transcript identifier
Unigene_ID	Identifier from the NCBI UniGene database

AnnotStruct = `ilmnbslookup(AnnotationFile, ID, 'LookUpField', LookUpFieldValue)` looks for *ID* in the annotation file in the field specified by *LookUpFieldValue*. Default is the `Search_key` field.

Examples

Note The gene expression file, `TumorAdjacent-probe-raw.txt`, and the annotation file, `HumanRef-8_V3_0_R0_11282963_A.bgx`, used in the following examples are not provided with the Bioinformatics Toolbox software.

Look Up Annotation Information for a Single Target (Probe)

- 1 Read the contents of a tab-delimited file exported from the Illumina BeadStudio software into a MATLAB structure.

```
ilmnStruct = ilmnbread('TumorAdjacent-probe-raw.txt')
```

```
ilmnStruct =
```

```

        Header: [1x1 struct]
      TargetID: {22184x1 cell}
   ColumnNames: {1x37 cell}
         Data: [22184x37 double]
TextColumnNames: {1x23 cell}
      TextData: {22184x23 cell}
```

- 2 Find the number of the `Search_key` column in the `TextColumnNames` cell array, which is returned in the `ilmnStruct` structure by the `ilmnbread` function.

```
srchCol = find(strcmpi('Search_Key',ilmnStruct.TextColumnNames))
```

```
srchCol =
```

```
1
```

- 3** Use the output from step 2 to look up the probe sequence and annotation information for the 10th entry in the annotation file, HumanRef-8_V3_0_R0_11282963_A.bgx.

```
annotation = ilmnblookup('HumanRef-8_V3_0_R0_11282963_A.bgx',...
                        ilmStruct.TextData{10,srchCol})

annotation =

    Accession: 'NM_144670.2'
    Array_Address_Id: '0004050154'
    Chromosome: '12'
    Cytoband: '12p13.31b'
    Definition: 'Homo sapiens alpha-2-macroglobulin-like 1 (A2ML1), mRNA.'
    Entrez_Gene_ID: '144568'
    GI: '74271844'
    ILMN_Gene: 'A2ML1'
    Obsolete_Probe_Id: ''
    Ontology_Component: ''
    Ontology_Function: 'endopeptidase inhibitor activity [goid 4866] [evidence IEA]'
    Ontology_Process: ''
    Probe_Chromosome_Orientation: '+'
    Probe_Coordinates: '8920412-8920461'
    Probe_Id: 'ILMN_2136495'
    Probe_Sequence: 'TGTAATCGCAGCCCCTTGGAAAGCCAAGGCAGGAGAATCGCCTCAACACT'
    Probe_Start: '4889'
    Probe_Type: 'S'
    Protein_Product: 'NP_653271.2'
    RefSeq_ID: 'NM_144670.2'
    Reporter_Composite_map: ''
    Reporter_Group_Name: ''
    Reporter_Group_id: ''
    Search_Key: 'ILMN_17375'
    Source: 'RefSeq'
    Source_Reference_ID: 'NM_144670.2'
    Species: 'Homo sapiens'
    Symbol: 'A2ML1'
    Synonyms: [1x141 char]
```

```
Transcript: 'ILMN_17375'
Unigene_ID: ''
```

Look Up Annotation Information for a Subset of Targets (Probes)

Use the `ilmnbslookup` function with the `'LookupField'` property to look up the annotation information for all targets located on chromosome 12 in the annotation file, `HumanRef-8_V3_0_RO_11282963_A.bgx`.

```
chr12annotation = ilmnblookup('HumanRef-8_V3_0_RO_11282963_A.bgx',
                              '12', 'LookupField', 'Chromosome')
```

```
chr12annotation =
```

```

    Accession: {1x1186 cell}
  Array_Address_Id: {1x1186 cell}
    Chromosome: {1x1186 cell}
    Cytoband: {1x1186 cell}
    Definition: {1x1186 cell}
  Entrez_Gene_ID: {1x1186 cell}
        GI: {1x1186 cell}
    ILMN_Gene: {1x1186 cell}
  Obsolete_Probe_Id: {1x1186 cell}
  Ontology_Component: {1x1186 cell}
  Ontology_Function: {1x1186 cell}
  Ontology_Process: {1x1186 cell}
  Probe_Chr_Orientation: {1x1186 cell}
  Probe_Coordinates: {1x1186 cell}
        Probe_Id: {1x1186 cell}
    Probe_Sequence: {1x1186 cell}
    Probe_Start: {1x1186 cell}
    Probe_Type: {1x1186 cell}
  Protein_Product: {1x1186 cell}
    RefSeq_ID: {1x1186 cell}
  Reporter_Composite_map: ''
  Reporter_Group_Name: ''
  Reporter_Group_id: ''
    Search_Key: {1x1186 cell}
```

ilmnbslookup

```
Source: {1x1186 cell}
Source_Reference_ID: {1x1186 cell}
Species: {1x1186 cell}
Symbol: {1x1186 cell}
Synonyms: {1x1186 cell}
Transcript: {1x1186 cell}
Unigene_ID: {1x1186 cell}
```

The output structure indicates that there are 1,186 targets located on chromosome 12.

See Also

Bioinformatics Toolbox function: `ilmnbsread`

Purpose

Read gene expression data exported from Illumina BeadStudio software

Syntax

```
IlmnStruct = ilmnbread(File)  
IlmnStruct = ilmnbread(File, ...'Columns',  
ColumnsValue, ...)  
IlmnStruct = ilmnbread(File, ...'HeaderOnly',  
HeaderOnlyValue, ...)  
IlmnStruct = ilmnbread(File, ...'CleanColNames',  
CleanColNamesValue, ...)
```

Arguments

File

String specifying a file name or a path and file name of a tab-delimited file or comma-separated expression data file exported from Illumina BeadStudio software. If you specify only a file name, that file must be on the MATLAB search path or in the current directory.

Note Currently, the Bioinformatics Toolbox software supports gene expression data files created from only these Illumina whole-genome arrays:

- HumanRef-8_V3
- HumanWG-6_V3
- HumanRef-8_V2
- HumanWG-6_V2
- MouseRef-8_V2
- MouseWG-6_V2
- MouseRef-8_V1
- MouseWG-6_V1
- RatRef-12_V1

ColumnsValue

Cell array that specifies the column names to read. Default is all column names.

<i>HeaderOnlyValue</i>	Controls the population of only the Header, ColumnNames, and TextColumnName fields in <i>IlmnStruct</i> . Choices are true or false (default).
<i>CleanColNamesValue</i>	Controls the conversion of any ColumnNames containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are true or false (default).

Return Values

<i>IlmnStruct</i>	MATLAB structure containing data exported from Illumina BeadStudio software.
-------------------	--

Description

IlmnStruct = `ilmnbsread(File)` reads *File*, a tab-delimited or comma-separated expression data file exported from the Illumina BeadStudio software, and creates *IlmnStruct*, a MATLAB structure containing the following fields.

Field	Description
Header	String containing a description of the data.
TargetID	Cell array containing unique identifiers for targets on an Illumina gene expression microarray.
ColumnNames	Cell array containing names of the columns that contain numeric data in the tab-delimited file exported from the Illumina BeadStudio software.

Field	Description
Data	Matrix containing numeric microarray data for each target on an Illumina gene expression microarray. <hr/> Note ColumnNames and Data have the same number of columns. <hr/>
TextColumnNames	Cell array containing names of the columns that contain nonnumeric data in the tab-delimited file exported from the Illumina BeadStudio software. This field can be empty.
TextData	Cell array containing nonnumeric microarray data (such as annotations) for each target on an Illumina gene expression microarray. This field can be empty. <hr/> Note TextColumnNames and TextData have the same number of columns. <hr/>

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

`IlmnStruct = ilmnbread(File, ...'Columns', ColumnsValue, ...)` reads the data only from the columns specified by *ColumnsValue*, a cell array of column names. Default behavior is to read data from all columns.

IlmnStruct = `ilmnbsread(File, ...'HeaderOnly', HeaderOnlyValue, ...)` controls the population of only the Header, ColumnNames, and TextColumnNames fields in *IlmnStruct*. Choices are true or false (default).

IlmnStruct = `ilmnbsread(File, ...'CleanColNames', CleanColNamesValue, ...)` controls the conversion of any ColumnNames containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are true or false (default).

Tip Use the 'CleanColNames' property if you plan to use the ColumnNames field as variable names.

Examples

Note The gene expression file, TumorAdjacent-probe-raw.txt used in the following example is not provided with the Bioinformatics Toolbox software.

Read the contents of a tab-delimited file exported from the Illumina BeadStudio software into a MATLAB structure.

```
ilmnStruct = ilmnbread('TumorAdjacent-probe-raw.txt')
```

```
ilmnStruct =
```

```
    Header: [1x1 struct]
    TargetID: {22184x1 cell}
    ColumnNames: {1x37 cell}
           Data: [22184x37 double]
    TextColumnNames: {1x23 cell}
           TextData: {22184x23 cell}
```

ilmnbsread

See Also

Bioinformatics Toolbox functions: `affyread`, `agferead`, `celintensityread`, `galread`, `geoseriesread`, `geosoftread`, `gprread`, `ilmnbslookup`, `imageneread`, `magetfield`, `sptread`

Purpose Read microarray data from ImaGene Results file

Syntax

```
imagedata = imageneread('File')  
imagedata = imageneread(..., 'CleanColNames',  
CleanColNamesValue, ...)
```

Arguments

File ImaGene Results formatted file. Enter a file name or a path and file name.

CleanColNamesValue Controls the conversion of any ColumnNames containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are true or false (default).

Description

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

Field
HeaderAA
Data
Blocks
Rows
Columns
Fields
IDs
ColumnNames
Indices
Shape

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

`imagenedata = imageneread(..., 'CleanColNames', CleanColNamesValue, ...)` controls the conversion of any `ColumnNames` containing spaces or characters that cannot be used as MATLAB variable names, to valid MATLAB variable names. Choices are `true` or `false` (default).

The field `Indices` of the structure contains indices that you can use for plotting heat maps of the data with the function `image` or `imagesc`.

For more details on the ImaGene format and example data, see the ImaGene documentation.

Examples

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

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

- 2 Plot the signal mean.

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

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

```
cy5Data = imageneread('cy5.txt');
```

- 4 Create a loglog plot of the signal median from two ImaGene Results files.

```
sigMedianCol = find(strcmp('Signal Median', cy3Data.ColumnNames));  
cy3Median = cy3Data.Data(:, sigMedianCol);  
cy5Median = cy5Data.Data(:, sigMedianCol);  
maloglog(cy3Median, cy5Median, 'title', 'Signal Median');
```

See Also

Bioinformatics Toolbox functions: `gprread`, `ilmnbsread`, `maboxplot`, `mimage`, `sptread`

int2aa

Purpose Convert amino acid sequence from integer to letter representation

Syntax
SeqChar = int2aa(*SeqInt*)
SeqChar = int2aa(*SeqInt*, 'Case', *CaseValue*)

Arguments

SeqInt Row vector of integers specifying an amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes on page 2-558. Integers are arbitrarily assigned to IUB/IUPAC letters.

CaseValue String specifying the case of the returned string. Choices are 'upper' (default) or 'lower'.

Return Values

SeqChar Amino acid sequence specified by a string of single-letter codes.

Description

SeqChar = int2aa(*SeqInt*) converts *SeqInt*, a row vector of integers specifying an amino acid sequence, to *SeqChar*, a string of single-letter codes specifying the same amino acid sequence. For valid integers, see the table Mapping Amino Acid Integers to Letter Codes on page 2-558.

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

Mapping Amino Acid Integers to Letter Codes

Amino Acid	Integer	Code
Alanine	1	A
Arginine	2	R
Asparagine	3	N

Mapping Amino Acid Integers to Letter Codes (Continued)

Amino Acid	Integer	Code
Aspartic acid (Aspartate)	4	D
Cysteine	5	C
Glutamine	6	Q
Glutamic acid (Glutamate)	7	E
Glycine	8	G
Histidine	9	H
Isoleucine	10	I
Leucine	11	L
Lysine	12	K
Methionine	13	M
Phenylalanine	14	F
Proline	15	P
Serine	16	S
Threonine	17	T
Tryptophan	18	W
Tyrosine	19	Y
Valine	20	V
Asparagine or Aspartic acid (Aspartate)	21	B
Glutamine or Glutamic acid (Glutamate)	22	Z
Unknown amino acid (any amino acid)	23	X
Translation stop	24	*
Gap of indeterminate length	25	-
Unknown (any integer not in table)	0 or ≥ 26	?

int2aa

Examples

Convert an amino acid sequence from integer to letter representation.

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

```
s =
```

```
MATLAB
```

See Also

Bioinformatics Toolbox functions: `aa2int`, `aminolookup`, `int2nt`, `nt2int`

Purpose	Convert nucleotide sequence from integer to letter representation								
Syntax	<pre>SeqChar = int2nt(SeqInt) SeqChar = int2nt(SeqInt, ...'Alphabet', AlphabetValue, ...) SeqChar = int2nt(SeqInt, ...'Unknown', UnknownValue, ...) SeqChar = int2nt(SeqInt, ...'Case', CaseValue, ...)</pre>								
Arguments	<table><tr><td><i>SeqInt</i></td><td>Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes on page 2-562. Integers are arbitrarily assigned to IUB/IUPAC letters.</td></tr><tr><td><i>AlphabetValue</i></td><td>String specifying a nucleotide alphabet. Choices are:<ul style="list-style-type: none">• 'DNA' (default) — Uses the symbols A, C, G, and T.• 'RNA' — Uses the symbols A, C, G, and U.</td></tr><tr><td><i>UnknownValue</i></td><td>Character to represent unknown nucleotides, that is 0 or integers ≥ 17. Choices are any character other than the nucleotide characters A, C, G, T, and U and the ambiguous nucleotide characters N, R, Y, K, M, S, W, B, D, H, and V. Default is *.</td></tr><tr><td><i>CaseValue</i></td><td>String specifying the case of the returned character string. Choices are 'upper' (default) or 'lower'.</td></tr></table>	<i>SeqInt</i>	Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes on page 2-562. Integers are arbitrarily assigned to IUB/IUPAC letters.	<i>AlphabetValue</i>	String specifying a nucleotide alphabet. Choices are: <ul style="list-style-type: none">• 'DNA' (default) — Uses the symbols A, C, G, and T.• 'RNA' — Uses the symbols A, C, G, and U.	<i>UnknownValue</i>	Character to represent unknown nucleotides, that is 0 or integers ≥ 17 . Choices are any character other than the nucleotide characters A, C, G, T, and U and the ambiguous nucleotide characters N, R, Y, K, M, S, W, B, D, H, and V. Default is *.	<i>CaseValue</i>	String specifying the case of the returned character string. Choices are 'upper' (default) or 'lower'.
<i>SeqInt</i>	Row vector of integers specifying a nucleotide sequence. For valid integers, see the table Mapping Nucleotide Integers to Letter Codes on page 2-562. Integers are arbitrarily assigned to IUB/IUPAC letters.								
<i>AlphabetValue</i>	String specifying a nucleotide alphabet. Choices are: <ul style="list-style-type: none">• 'DNA' (default) — Uses the symbols A, C, G, and T.• 'RNA' — Uses the symbols A, C, G, and U.								
<i>UnknownValue</i>	Character to represent unknown nucleotides, that is 0 or integers ≥ 17 . Choices are any character other than the nucleotide characters A, C, G, T, and U and the ambiguous nucleotide characters N, R, Y, K, M, S, W, B, D, H, and V. Default is *.								
<i>CaseValue</i>	String specifying the case of the returned character string. Choices are 'upper' (default) or 'lower'.								
Return Values	<table><tr><td><i>SeqChar</i></td><td>Nucleotide sequence specified by a character string of codes.</td></tr></table>	<i>SeqChar</i>	Nucleotide sequence specified by a character string of codes.						
<i>SeqChar</i>	Nucleotide sequence specified by a character string of codes.								
Description	<i>SeqChar</i> = int2nt(<i>SeqInt</i>) converts <i>SeqInt</i> , a row vector of integers specifying a nucleotide sequence, to <i>SeqChar</i> , a string of codes specifying								

the same nucleotide sequence. For valid codes, see the table Mapping Nucleotide Integers to Letter Codes on page 2-562.

Mapping Nucleotide Integers to Letter Codes

Nucleotide	Integer	Code
Adenosine	1	A
Cytidine	2	C
Guanine	3	G
Thymidine	4	T
Uridine (if 'Alphabet' set to 'RNA')	4	U
Purine (A or G)	5	R
Pyrimidine (T or C)	6	Y
Keto (G or T)	7	K
Amino (A or C)	8	M
Strong interaction (3 H bonds) (G or C)	9	S
Weak interaction (2 H bonds) (A or T)	10	W
Not A (C or G or T)	11	B
Not C (A or G or T)	12	D
Not G (A or C or T)	13	H
Not T or U (A or C or G)	14	V
Any nucleotide (A or C or G or T or U)	15	N
Gap of indeterminate length	16	-
Unknown (any integer not in table)	0 or ≥ 17	* (default)

`SeqChar = int2nt(SeqInt, ...PropertyName', PropertyValue, ...)` calls `int2nt` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation

marks and is case insensitive. These property name/property value pairs are as follows:

`SeqChar = int2nt(SeqInt, ...'Alphabet', AlphabetValue, ...)` specifies a nucleotide alphabet. *AlphabetValue* can be 'DNA', which uses the symbols A, C, G, and T, or 'RNA', which uses the symbols A, C, G, and U. Default is 'DNA'.

`SeqChar = int2nt(SeqInt, ...'Unknown', UnknownValue, ...)` specifies the character to represent unknown nucleotides, that is 0 or integers ≥ 17 . *UnknownValue* can be any character other than the nucleotide characters A, C, G, T, and U and the ambiguous nucleotide characters N, R, Y, K, M, S, W, B, D, H, and V. Default is '*'.

`SeqChar = int2nt(SeqInt, ...'Case', CaseValue, ...)` specifies the case of the returned character string. *CaseValue* can be 'upper' (default) or 'lower'.

Examples

- Convert a nucleotide sequence from integer to letter representation.

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

```
s =  
ACTGCTAGC
```

- Convert a nucleotide sequence from integer to letter representation and define # as the symbol for unknown numbers 17 and greater.

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

```
s =  
ACT#CT#GC
```

See Also

Bioinformatics Toolbox functions: `aa2int`, `baselookup`, `int2aa`, `nt2int`

isoelectric

Purpose Estimate isoelectric point for amino acid sequence

Syntax

```
pI = isoelectric(SeqAA)  
[pI Charge] = isoelectric(SeqAA)  
isoelectric(..., 'PropertyName', PropertyValue, ...)  
isoelectric(..., 'PKVals', PKValsValue)  
isoelectric(..., 'Charge', ChargeValue)  
isoelectric(..., 'Chart', ChartValue)
```

Arguments

<i>SeqAA</i>	Amino acid sequence. Enter a character string or a vector of integers from the table Mapping Amino Acid Letter Codes to Integers on page 2-2. Examples: 'ARN' or [1 2 3].
<i>PKValsValue</i>	Property to provide alternative pK values.
<i>ChargeValue</i>	Property to select a specific pH for estimating charge. Enter a number between 0 and 14. The default value is 7.2.
<i>ChartValue</i>	Property to control plotting a graph of charge versus pH. Enter true or false.

Description

pI = isoelectric(*SeqAA*) returns the estimated isoelectric point (*pI*) for an amino acid sequence. The isoelectric point is the pH at which the protein has a net charge of zero

[*pI Charge*] = isoelectric(*SeqAA*) returns the estimated isoelectric point (*pI*) for an amino acid sequence and the estimated charge for a given pH (default is typical intracellular pH 7.2).

The estimates are skewed by the underlying assumptions that all amino acids are fully exposed to the solvent, that neighboring peptides have no influence on the pK of any given amino acid, and that the constitutive amino acids, as well as the N- and C-termini, are unmodified. Cysteine residues participating in disulfide bridges also affect the true pI and are not considered here. By default, isoelectric uses the EMBOSS amino

acid pK table, or you can substitute other values using the property PKVals.

- If the sequence contains ambiguous amino acid characters (b z * -), isoelectric ignores the characters and displays a warning message.

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

- If the sequence contains undefined amino acid characters (i j o), isoelectric ignores the characters and displays a warning message.

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

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

isoelectric(..., 'PKVals', *PKValsValue*) uses the alternative pK table stored in the text file *PKValsValues*. 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 *ChartValue* is true, returns a graph plotting the charge of the protein versus the pH of the solvent.

isoelectric

Example

```
% Get a sequence from PDB.
pdbSeq = getpdb('1CIV', 'SequenceOnly', true)
% Estimate its isoelectric point.
isoelectric(pdbSeq)

% Plot the charge against the pH for a short polypeptide sequence.
isoelectric('PQGGGGWGQPHGGGGWGQPHGGGGWGQGGSHSQG', 'CHART', true)

% Get the Rh blood group D antigen from NCBI and calculate
% its charge at pH 7.3 (typical blood pH).
gpSeq = getgenpept('AAB39602')
[pI Charge] = isoelectric(gpSeq, 'Charge', 7.38)
```

See Also

Bioinformatics Toolbox functions: `aaccount`, `molweight`

Purpose Read JCAMP-DX-formatted files

Syntax `JCAMPStruct = jcampread(File)`

Arguments

File

Either of the following:

- String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a JCAMP-DX-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the current directory.
- MATLAB character array that contains the text of a JCAMP-DX-formatted file.

Return Values

JCAMPStruct MATLAB structure containing information from a JCAMP-DX-formatted file.

Description

JCAMP-DX is a file format for infrared, NMR, and mass spectrometry data from the Joint Committee on Atomic and Molecular Physical Data (JCAMP). `jcampread` supports reading data from files saved with Versions 4.24, 5, or 6 of the JCAMP-DX format. For more details, see:

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

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

Field
Title

Field
DataType
DataClass
Origin
Owner
Blocks
Notes

The `Blocks` field of the structure is an array of structures corresponding to each set of data in the file. These structures have the following fields.

Field
XData
YData
XUnits
YUnits
Notes

Examples

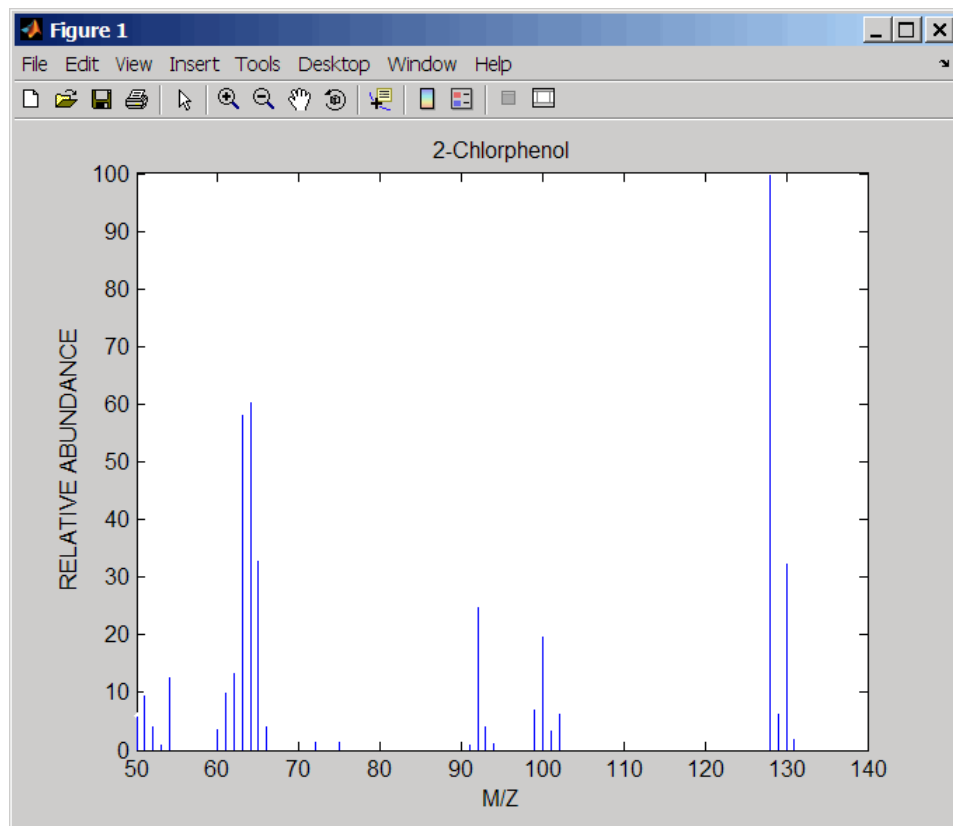
- 1 Download test data in the file `isa_ms1.dx` from:

<http://www.jcamp-dx.org/testdata.html>

- 2 Read a JCAMP-DX file (`isas_ms1.dx`) into the MATLAB software 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`, `mzcdfread`, `mzxmlread`

joinseq

Purpose Join two sequences to produce shortest supersequence

Syntax `SeqNT3 = joinseq(SeqNT1, SeqNT2)`

Arguments `SeqNT1, SeqNT2` Nucleotide sequences.

Description `SeqNT3 = joinseq(SeqNT1, SeqNT2)` creates a new sequence that is the shortest supersequence of `SeqNT1` and `SeqNT2`. If there is no overlap between the sequences, then `SeqNT2` is concatenated to the end of `SeqNT1`. If the length of the overlap is the same at both ends of the sequence, then the overlap at the end of `SeqNT1` and the start of `SeqNT2` is used to join the sequences.

If `SeqNT1` is a subsequence of `SeqNT2`, then `SeqNT2` is returned as the shortest supersequence and vice versa.

Examples

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

See Also MATLAB functions: `cat`, `strcat`, `strfind`

Purpose Classify data using nearest neighbor method

Syntax

```
Class = knnclassify(Sample, Training, Group)
Class = knnclassify(Sample, Training, Group, k)
Class = knnclassify(Sample, Training, Group, k, distance)
Class = knnclassify(Sample, Training, Group, k, distance,
                    rule)
```

Arguments

<i>Sample</i>	Matrix whose rows will be classified into groups. <i>Sample</i> must have the same number of columns as <i>Training</i> .
<i>Training</i>	Matrix used to group the rows in the matrix <i>Sample</i> . <i>Training</i> must have the same number of columns as <i>Sample</i> . Each row of <i>Training</i> belongs to the group whose value is the corresponding entry of <i>Group</i> .
<i>Group</i>	Vector whose distinct values define the grouping of the rows in <i>Training</i> .
<i>k</i>	The number of nearest neighbors used in the classification. Default is 1.

knnclassify

- distance* String specifying the distance metric. Choices are:
- 'euclidean' — Euclidean distance (default)
 - 'cityblock' — Sum of absolute differences
 - 'cosine' — One minus the cosine of the included angle between points (treated as vectors)
 - 'correlation' — One minus the sample correlation between points (treated as sequences of values)
 - 'hamming' — Percentage of bits that differ (suitable only for binary data)
- rule* String to specify the rule used to decide how to classify the sample. Choices are:
- 'nearest' — Majority rule with nearest point tie-break (default)
 - 'random' — Majority rule with random point tie-break
 - 'consensus' — Consensus rule

Description

Class = knnclassify(Sample, Training, Group) classifies the rows of the data matrix *Sample* into groups, based on the grouping of the rows of *Training*. *Sample* and *Training* must be matrices with the same number of columns. *Group* is a vector whose distinct values define the grouping of the rows in *Training*. Each row of *Training* belongs to the group whose value is the corresponding entry of *Group*. *knnclassify* assigns each row of *Sample* to the group for the closest row of *Training*. *Group* can be a numeric vector, a string array, or a cell array of strings. *Training* and *Group* must have the same number of rows. *knnclassify* treats NaNs or empty strings in *Group* as missing values, and ignores the corresponding rows of *Training*. *Class* indicates which group each row of *Sample* has been assigned to, and is of the same type as *Group*.

Class = knnclassify(Sample, Training, Group, k) enables you to specify *k*, the number of nearest neighbors used in the classification. Default is 1.

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

- 'euclidean' — Euclidean distance (default)
- 'cityblock' — Sum of absolute differences
- 'cosine' — One minus the cosine of the included angle between points (treated as vectors)
- 'correlation' — One minus the sample correlation between points (treated as sequences of values)
- 'hamming' — Percentage of bits that differ (suitable only for binary data)

`Class = knnclassify(Sample, Training, Group, k, distance, rule)` enables you to specify the rule used to decide how to classify the sample. Choices for *rule* are:

- 'nearest' — Majority rule with nearest point tie-break (default)
- 'random' — Majority rule with random point tie-break
- 'consensus' — Consensus rule

The default behavior is to use majority rule. That is, a sample point is assigned to the class the majority of the *k* nearest neighbors are from. Use 'consensus' to require a consensus, as opposed to majority rule. When using the 'consensus' option, points where not all of the *k* nearest neighbors are from the same class are not assigned to one of the classes. Instead the output `Class` for these points is NaN for numerical groups or '' for string named groups. When classifying to more than two groups or when using an even value for *k*, it might be necessary to break a tie in the number of nearest neighbors. Options are 'random', which selects a random tiebreaker, and 'nearest', which uses the nearest neighbor among the tied groups to break the tie. The default behavior is majority rule, with nearest tie-break.

knnclassify

Examples

Classifying Rows

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

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

```
sample =  
    0.9000    0.8000  
    0.1000    0.3000  
    0.2000    0.6000
```

```
training=[0 0;.5 .5;1 1]
```

```
training =  
    0    0  
    0.5000    0.5000  
    1.0000    1.0000
```

```
group = [1;2;3]
```

```
group =  
    1  
    2  
    3
```

```
class = knnclassify(sample, training, group)
```

```
class =  
    3  
    1  
    2
```

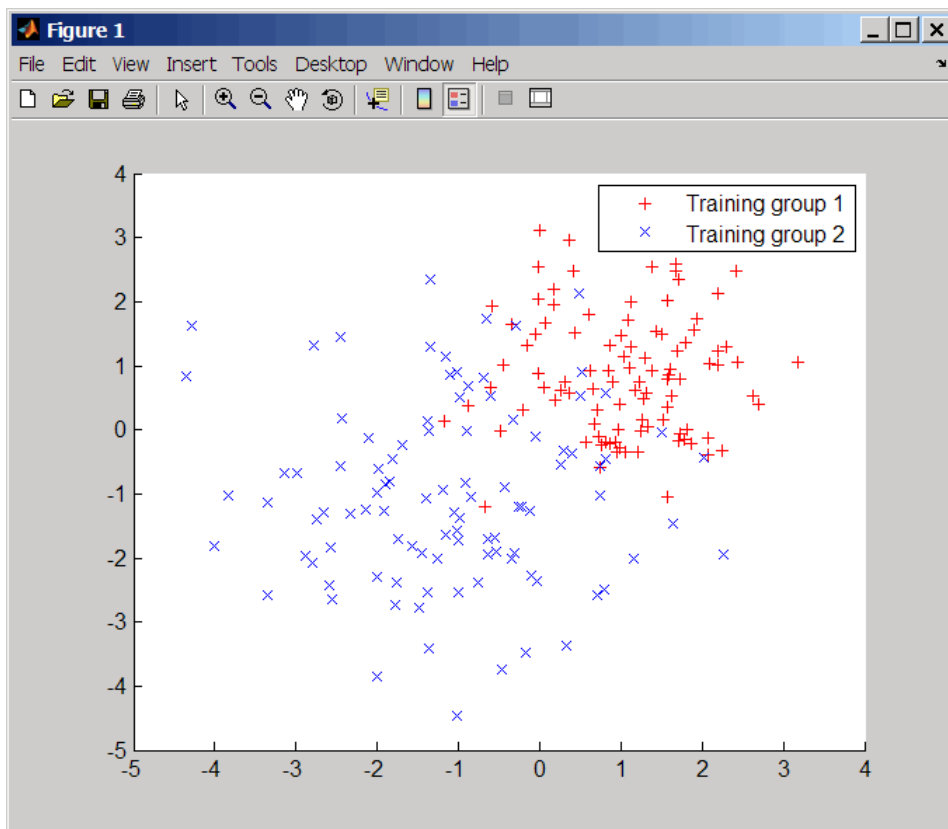
Row 1 of `sample` is closest to row 3 of `training`, so `class(1) = 3`. Row 2 of `sample` is closest to row 1 of `training`, so `class(2) = 1`. Row 3 of `sample` is closest to row 2 of `training`, so `class(3) = 2`.

Classifying Rows into One of Two Groups

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

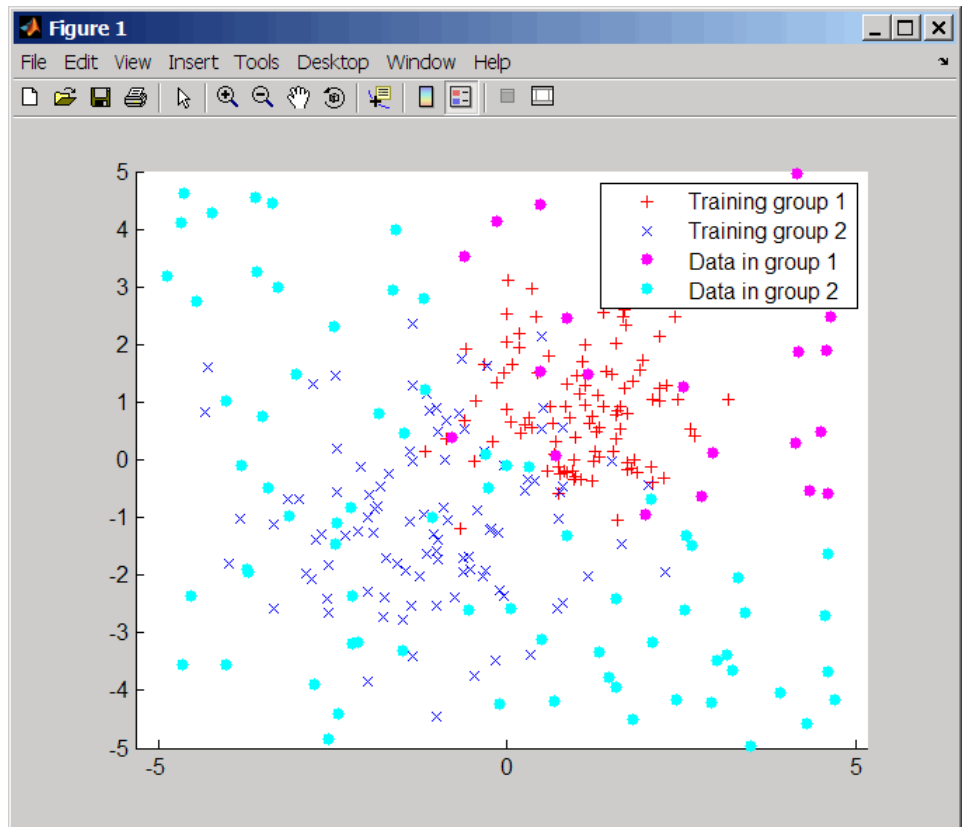
```
training = [mvnrnd([ 1  1], eye(2), 100); ...  
           mvnrnd([-1 -1], 2*eye(2), 100)];  
group = [repmat(1,100,1); repmat(2,100,1)];  
gscatter(training(:,1),training(:,2),group,'rb','+x');  
legend('Training group 1', 'Training group 2');  
hold on;
```

knnclassify



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

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

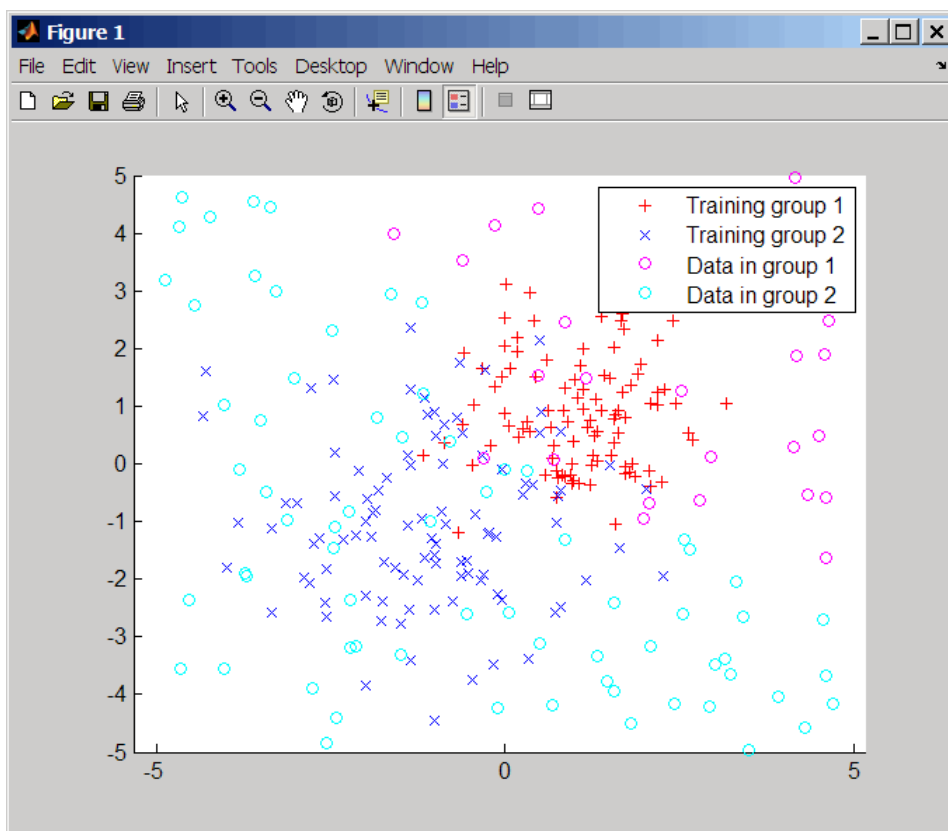



Classifying Rows Using the Three Nearest Neighbors

The following example uses the same data as in [Classifying Rows into One of Two Groups](#) on page 2-575, but classifies the rows of `sample` using three nearest neighbors instead of one.

```
gscatter(training(:,1),training(:,2),group,'rb','+x');  
hold on;  
c3 = knnclassify(sample, training, group, 3);  
gscatter(sample(:,1),sample(:,2),c3,'mc','o');  
legend('Training group 1','Training group 2','Data in group 1','Data in group 2');
```

knnclassify



If you compare this plot with the one in *Classifying Rows into One of Two Groups* on page 2-575, you see that some of the data points are classified differently using three nearest neighbors.

References

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

See Also

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

Statistics Toolbox function: `classify`

Purpose	Impute missing data using nearest-neighbor method
Syntax	<pre>knnimpute(Data) knnimpute(Data, k) knnimpute(..., 'Distance', DistanceValue, ...) knnimpute(..., 'DistArgs', DistArgsValue, ...) knnimpute(..., 'Weights', WeightsValues, ...) knnimpute(..., 'Median', MedianValue, ...)</pre>
Arguments	<p><i>Data</i></p> <p><i>k</i></p>
Description	<p><code>knnimpute(Data)</code> replaces NaNs in <i>Data</i> 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.</p> <p><code>knnimpute(Data, k)</code> replaces NaNs in <i>Data</i> with a weighted mean of the <i>k</i> nearest-neighbor columns. The weights are inversely proportional to the distances from the neighboring columns.</p> <p><code>knnimpute(..., 'PropertyName', PropertyValue, ...)</code> calls <code>knnimpute</code> with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each <i>PropertyName</i> must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:</p> <p><code>knnimpute(..., 'Distance', DistanceValue, ...)</code> computes nearest-neighbor columns using the distance metric <code>distfun</code>. The choices for <i>DistanceValue</i> are:</p>

'euclidean'	Euclidean distance (default).
'seuclidean'	Standardized Euclidean distance — each coordinate in the sum of squares is inversely weighted by the sample variance of that coordinate.
'cityblock'	City block distance.
'mahalanobis'	Mahalanobis distance.
'minkowski'	Minkowski distance with exponent 2.
'cosine'	One minus the cosine of the included angle.
'correlation'	One minus the sample correlation between observations, treated as sequences of values.
'hamming'	Hamming distance — the percentage of coordinates that differ.
'jaccard'	One minus the Jaccard coefficient — the percentage of nonzero coordinates that differ.
'chebychev'	Chebychev distance (maximum coordinate difference).
function handle	A handle to a distance function, specified using @, for example, @distfun.

See `pdist` for more details.

`knnimpute(..., 'DistArgs', DistArgsValue, ...)` passes arguments (*DistArgsValue*) to the function `distfun`. *DistArgsValue* can be a single value or a cell array of values.

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

`knnimpute(..., 'Median', MedianValue, ...)` when *MedianValue* is true, uses the median of the *k* nearest neighbors instead of the weighted mean.

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

```
knnimpute(A)
```

```
ans =
```

```

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

Example 2

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

```

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

References

[1] Speed, T. (2003). Statistical Analysis of Gene Expression Microarray Data (Chapman & Hall/CRC).

[2] Hastie, T., Tibshirani, R., Sherlock, G., Eisen, M., Brown, P., and Botstein, D. (1999). “Imputing missing data for gene expression arrays”, Technical Report, Division of Biostatistics, Stanford University.

[3] Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. (2001). Missing value estimation methods for DNA microarrays. *Bioinformatics* *17*(6), 520–525.

See Also

Statistics Toolbox function: `knnclassify`

MATLAB function: `isnan`

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

Purpose Create box plot for microarray data

Syntax

```

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

```

Arguments

<i>MAData</i>	DataMatrix object, numeric array, or a structure containing a field called <i>Data</i> . The values in the columns of <i>MAData</i> will be used to create box plots. If a DataMatrix object, the column names are used as labels in the box plot.
<i>ColumnName</i>	An array of column names corresponding to the data in <i>MAData</i> used as labels in the box plot.
<i>MAStruct</i>	A microarray data structure.
<i>FieldName</i>	A field within the microarray data structure, <i>MAStruct</i> . The values in the field <i>FieldName</i> will be used to create box plots.
<i>TitleValue</i>	A string to use as the title for the plot. The default title is <i>FieldName</i> .
<i>NotchValue</i>	Property to control the type of boxes drawn. Enter either <i>true</i> for notched boxes, or <i>false</i> , for square boxes. Default is <i>false</i> .

<i>OrientationValue</i>	Property to specify the orientation of the box plot. Enter 'Vertical' or 'Horizontal'. Default is 'Horizontal'.
<i>WhiskerLengthValue</i>	Property to specify the maximum length of the whiskers as a function of the interquartile range (IQR). The whisker extends to the most extreme data value within <i>WhiskerLengthValue</i> *IQR of the box. Default = 1.5. If <i>WhiskerLengthValue</i> equals 0, then <code>maboxplot</code> displays all data values outside the box, using the plotting symbol <code>Symbol</code> .
<i>BoxPlotValue</i>	A cell array of property name/property value pairs to pass to the Statistics Toolbox <code>boxplot</code> function, which creates the box plot. For valid pairs, see the <code>boxplot</code> function.

Description

`maboxplot(MAData)` displays a box plot of the values in the columns of *MAData*. *MAData* can be a `DataMatrix` object, numeric array, or a structure containing a field called `Data`, containing microarray data.

`maboxplot(MAData, ColumnName)` labels the box plot column names.

`maboxplot(MAStruct, FieldName)` displays a box plot of the values in the field *FieldName* in the microarray data structure *MAStruct*. If *MAStruct* is block based, `maboxplot` creates a box plot of the values in the field *FieldName* for each block.

Note If you provide *MAStruct*, without providing *FieldName*, `maboxplot` uses the `Signal` element in the `ColumnNames` field of *MAStruct*, if `Affymetrixdata`, or the first element in the in the `ColumnNames` field of *MAStruct*, otherwise.

H = `maboxplot(...)` returns the handle of the box plot axes.

`[H, HLines] = maboxplot(...)` returns the handles of the lines used to separate the different blocks in the image.

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

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

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

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

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

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

`maboxplot(..., 'BoxPlot', BoxPlotValue, ...)` allows you to specify arguments to pass to the `boxplot` function, which creates the box plot. *BoxPlotValue* is a cell array of property name/property value pairs. For valid pairs, see the `boxplot` function.

Examples

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

% Using a structure
```

maboxplot

```
geoStruct = getgeodata('GSM1768');
maboxplot(geoStruct, 'title', 'GSM1768');

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

See Also

Bioinformatics Toolbox functions: `magetfield`, `mimage`, `mairplot`, `maloglog`, `malowess`, `manorm`, `mavolcanoplot`

Statistics Toolbox function: `boxplot`

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

Syntax

```

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

```

Arguments

<i>PValues</i>	Either of the following: <ul style="list-style-type: none"> • Column vector of p-values for each feature (for example, gene) in a data set, such as returned by <code>mattest</code>. • <code>DataMatrix</code> object containing p-values for each feature (for example, gene) in a data set, such as returned by <code>mattest</code>.
<i>BHFDRValue</i>	Property to control the use of the linear step-up (LSU) procedure originally introduced by Benjamini and Hochberg, 1995. Choices are <code>true</code> or <code>false</code> (default).

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

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

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

Tip The series of values can be expressed by a colon operator with the form `[first:incr:last]`, where *first* is the first value in the series, *incr* is the increment, and *last* is the last value in the series.

Default *LambdaValue* is the series of values `[0.01:0.01:0.95]`.

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

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

- bootstrap (default)
- polynomial

ShowplotValue Property to display two plots:

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

Choices are true or false (default).

Return Values

FDR One of the following:

- Column vector of positive FDR (pFDR) values (if *PValues* is a column vector).
- `DataMatrix` object containing positive FDR (pFDR) values and the same row names as *PValues* (if *PValues* is a `DataMatrix` object).

Q Column vector of q-values.

Pi0 Estimated true null hypothesis, $\hat{\pi}_0$.

R2 Square of the correlation coefficient.

Description

FDR = `mafdr(PValues)` computes a positive FDR (pFDR) value for each value in *PValues*, a column vector or `DataMatrix` object containing p-values for each gene in two microarray data sets, using a procedure

introduced by Storey, 2002. *FDR* is a column vector or a *DataMatrix* object containing positive FDR (pFDR) values.

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

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

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

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

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

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

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

- A single value that is > 0 and < 1 .

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

Tip The series of values can be expressed by a colon operator with the form `[first:incr:last]`, where *first* is the first value in the series, *incr* is the increment, and *last* is the last value in the series.

Default *LambdaValue* is the series of values `[0.01:0.01:0.95]`.

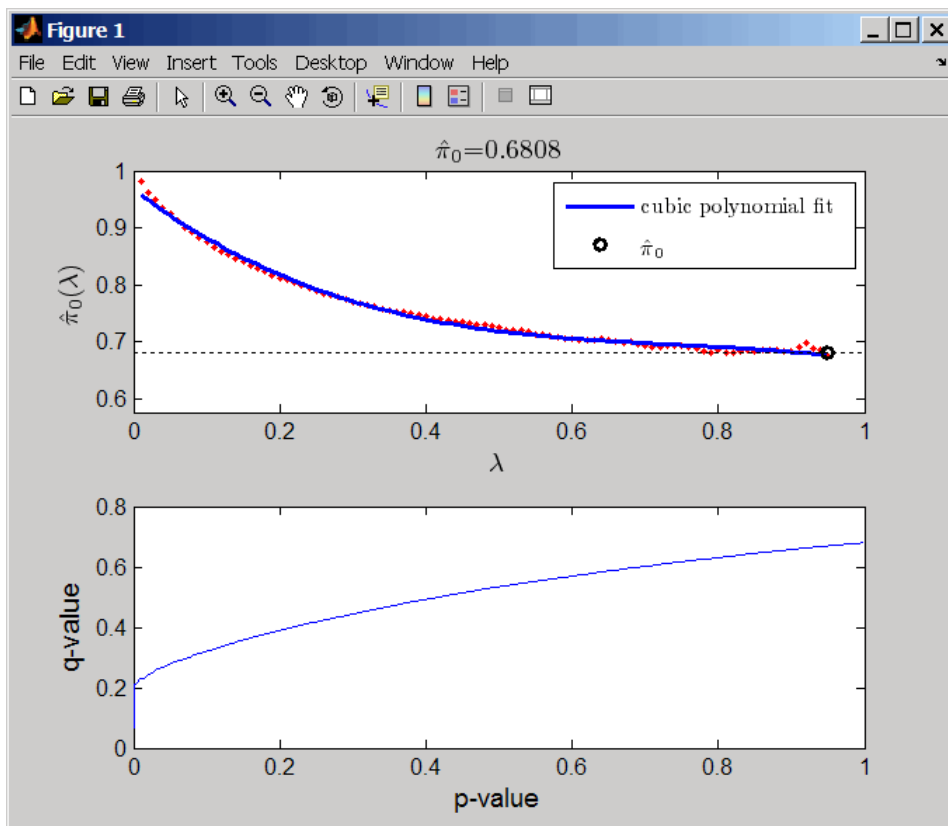
Note If *LambdaValue* is set to a single value, the `Method` property is ignored.

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

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

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

Choices are `true` or `false` (default).



Examples

- 1 Load the MAT-file, included with the Bioinformatics Toolbox software, that contains Affymetrix data from a prostate cancer study, specifically probe intensity data from Affymetrix HG-U133A GeneChip arrays. The two variables in the MAT-file, `dependentData` and `independentData`, are two matrices of gene expression values from two experimental conditions.

```
load prostatecancerexpdata
```


- 2 Use the `mattest` function to calculate p-values for the gene expression values in the two matrices.

```
pvalues = mattest(dependentData, independentData, 'permute', true);
```

- 3 Use the `mafdr` function to calculate positive FDR values and q-values for the gene expression values in the two matrices and plot the data.

```
[fdr, q] = mafdr(pvalues, 'showplot', true);
```

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

References

- [1] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. *Clinical Cancer Research* *11*, 6823–6834.
- [2] Storey, J.D. (2002). A direct approach to false discovery rates. *Journal of the Royal Statistical Society* *64(3)*, 479–498.
- [3] Storey, J.D., and Tibshirani, R. (2003). Statistical significance for genomewide studies. *Proc Nat Acad Sci* *100(16)*, 9440–9445.
- [4] Storey, J.D., Taylor, J.E., and Siegmund, D. (2004). Strong control conservative point estimation and simultaneous conservative consistency of false discovery rates: A unified approach. *Journal of the Royal Statistical Society* *66*, 187–205.
- [5] Benjamini, Y., and Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society* *57*, 289–300.

mafdr

See Also

Bioinformatics Toolbox functions: `affygcrrma`, `affyrma`, `gcrma`, `mairplot`, `maloglog`, `mapcaplot`, `matteest`, `mavolcanoplot`, `rmasummary`

Purpose Extract data from microarray structure

Syntax `magetfield(MAStruct, FieldName)`

Arguments

<i>MAStruct</i>	Microarray structure.
<i>FieldName</i>	A column in <i>MAStruct</i> .

Description `magetfield(MAStruct, FieldName)` extracts data for *FieldName*, a column in *MAStruct*, microarray structure.

The benefit of this function is to hide the details of extracting a column of data from a structure created with one of the microarray reader functions (`gprread`, `agferead`, `sptread`, `imageneread`).

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
cy3data = magetfield(maStruct, 'F635 Median');
cy5data = magetfield(maStruct, 'F532 Median');
mairplot(cy3data,cy5data,'title','R vs G IR plot');
```

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

mimage

Purpose Spatial image for microarray data

Syntax

```
mimage(X, FieldName)
H = mimage(...)
[H, HLines] = mimage(...)
mimage(..., 'PropertyName', PropertyValue,...)
mimage(..., 'Title', TitleValue)
mimage(..., 'ColorBar', ColorBarValue)
mimage(..., 'HandleGraphicsPropertyName' PropertyValue)
```

Arguments

<i>X</i>	A microarray data structure.
<i>FieldName</i>	A field in the microarray data structure <i>X</i> .
<i>TitleValue</i>	A string to use as the title for the plot. The default title is <i>FieldName</i> .
<i>ColorBarValue</i>	Property to control displaying a color bar in the Figure window. Enter either <code>true</code> or <code>false</code> . The default value is <code>false</code> .

Description

`mimage(X, FieldName)` displays an image of field *FieldName* from microarray data structure *X*. Microarray data can be GenPix Results (GPR) format. After creating the image, click a data point to display the value and ID, if known.

`H = mimage(...)` returns the handle of the image.

`[H, HLines] = mimage(...)` returns the handles of the lines used to separate the different blocks in the image.

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

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

`mimage(..., 'ColorBar', ColorBarValue)`, when *ColorBarValue* is true, a color bar is shown. If *ColorBarValue* is false, no color bar is shown. The default is for the color bar to be shown.

`mimage(..., '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 `mimage(..., 'color' 'r')`.

Examples

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

See Also

Bioinformatics Toolbox functions: `maboxplot`, `magetfield`, `mairplot`, `maloglog`, `malowess`

MATLAB function: `imagesc`

mainvarsetnorm

Purpose Perform rank invariant set normalization on gene expression values from two experimental conditions or phenotypes

Syntax

```
NormDataY = mainvarsetnorm(DataX, DataY)
NormDataY = mainvarsetnorm(..., 'Thresholds',
ThresholdsValue, ...)
NormDataY = mainvarsetnorm(..., 'Exclude',
ExcludeValue, ...)
NormDataY = mainvarsetnorm(..., 'Percentile',
PercentileValue, ...)
NormDataY = mainvarsetnorm(..., 'Iterate',
IterateValue, ...)
NormDataY = mainvarsetnorm(..., 'Method', MethodValue, ...)
NormDataY = mainvarsetnorm(..., 'Span', SpanValue, ...)
NormDataY = mainvarsetnorm(..., 'Showplot', ShowplotValue,
...)
```

Arguments

<i>DataX</i>	Vector of gene expression values from a single experimental condition or phenotype, where each row corresponds to a gene. These data points are used as the baseline.
<i>DataY</i>	Vector of gene expression values from a single experimental condition or phenotype, where each row corresponds to a gene. These data points will be normalized using the baseline.

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

ThresholdsValue is a 1-by-2 vector [LT , HT], where LT is the threshold for the lowest average rank and HT is threshold for the highest average rank. Values must be between 0 and 1. Default is [0.03, 0.07].

ExcludeValue Property to filter the invariant set of data points, by excluding the data points whose average rank (between $DataX$ and $DataY$) is in the highest N ranked averages or lowest N ranked averages.

PercentileValue Property to stop the iteration process when the number of data points in the invariant set reaches N percent of the total number of input data points. Default is 1.

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

mainvarsetnorm

IterateValue Property to control the iteration process for determining the invariant set of data points. Enter `true` to repeat the process until either no more data points are eliminated, or a predetermined percentage of data points (*PercentileValue*) is reached. Enter `false` to perform only one iteration of the process. Default is `true`.

Tip Select `false` for smaller data sets, typically less than 200 data points.

MethodValue Property to select the smoothing method used to normalize the data. Enter `'lowess'` or `'runmedian'`. Default is `'lowess'`.

SpanValue Property to set the window size for the smoothing method. If *SpanValue* is less than 1, the window size is that percentage of the number of data points. If *SpanValue* is equal to or greater than 1, the window size is of size *SpanValue*. Default is 0.05, which corresponds to a window size equal to 5% of the total number of data points in the invariant set.

ShowplotValue Property to control the plotting of a pair of M-A scatter plots (before and after normalization). M is the ratio between *DataX* and *DataY*. A is the average of *DataX* and *DataY*. Enter `true` to create the pair of M-A scatter plots. Default is `false`.

Description

$NormDataY = mainvarsetnorm(DataX, DataY)$ normalizes the values in *DataY*, a vector of gene expression values, to a reference vector, *DataX*, using the invariant set method. *NormDataY* is a vector of normalized gene expression values from *DataY*.

Specifically, `mainvarsetnorm`:

- Determines the proportional rank difference (*prd*) for each pair of ranks, *RankX* and *RankY*, from the two vectors of gene expression values, *DataX* and *DataY*.

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

- Determines the invariant set of data points by selecting data points whose proportional rank differences (*prd*) are below *threshold*, which is a predetermined threshold for a given data point (defined by the *ThresholdsValue* property). It optionally repeats the process until either no more data points are eliminated, or a predetermined percentage of data points is reached.

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

- Uses the invariant set of data points to calculate the lowess or running median smoothing curve, which is used to normalize the data in *DataY*.

Note If *DataX* or *DataY* contains NaN values, then *NormDataY* will also contain NaN values at the corresponding positions.

Tip `mainvarsetnorm` is useful for correcting for dye bias in two-color microarray data.

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

mainvarsetnorm

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

ThresholdsValue is a 1-by-2 vector [*LT*, *HT*], where *LT* is the threshold for the lowest average rank and *HT* is threshold for the highest average rank. Values must be between 0 and 1. Default is [0.03, 0.07].

NormDataY = mainvarsetnorm(..., 'Exclude', *ExcludeValue*, ...) filters the invariant set of data points, by excluding the data points whose average rank (between *DataX* and *DataY*) is in the highest *N* ranked averages or lowest *N* ranked averages.

NormDataY = mainvarsetnorm(..., 'Percentile', *PercentileValue*, ...) stops the iteration process when the number of data points in the invariant set reaches *N* percent of the total number of input data points. Default is 1.

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

NormDataY = mainvarsetnorm(..., 'Iterate', *IterateValue*, ...) controls the iteration process for determining the invariant set of data points. When *IterateValue* is true, mainvarsetnorm repeats the process until either no more data points are eliminated, or a predetermined percentage of data points (*PercentileValue*) is reached. When *IterateValue* is false, performs only one iteration of the process. Default is true.

Tip Select `false` for smaller data sets, typically less than 200 data points.

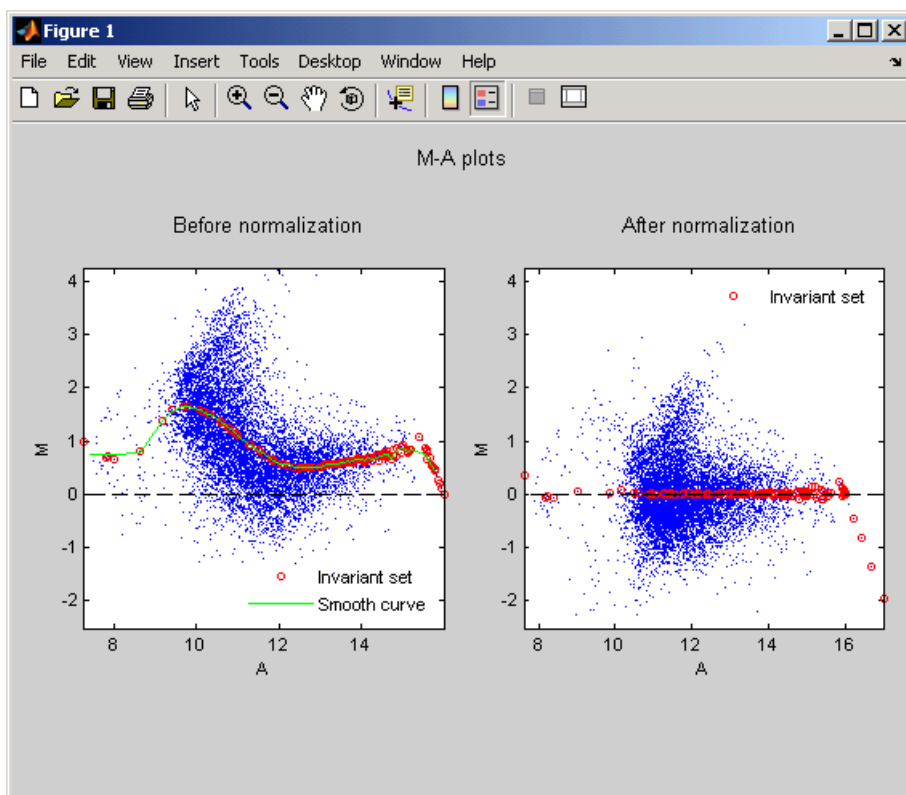
`NormDataY = mainvarsetnorm(..., 'Method', MethodValue, ...)` selects the smoothing method for normalizing the data. When *MethodValue* is 'lowess', `mainvarsetnorm` uses the lowess method. When *MethodValue* is 'runmedian', `mainvarsetnorm` uses the running median method. Default is 'lowess'.

`NormDataY = mainvarsetnorm(..., 'Span', SpanValue, ...)` sets the window size for the smoothing method. If *SpanValue* is less than 1, the window size is that percentage of the number of data points. If *SpanValue* is equal to or greater than 1, the window size is of size *SpanValue*. Default is 0.05, which corresponds to a window size equal to 5% of the total number of data points in the invariant set.

`NormDataY = mainvarsetnorm(..., 'Showplot', ShowplotValue, ...)` determines whether to plot a pair of M-A scatter plots (before and after normalization). M is the ratio between *DataX* and *DataY*. A is the average of *DataX* and *DataY*. When *ShowplotValue* is true, `mainvarsetnorm` plots the M-A scatter plots. Default is false.

The following example illustrates how `mainvarsetnorm` can correct for dye bias or scanning differences between two channels of data from a two-color microarray experiment. Under perfect experimental conditions, data points with equal expression values would fall along the $M = 0$ line, which represents a gene expression ratio of 1. However, dye bias caused the measured values in one channel to be higher than the other channel, as seen in the Before Normalization plot. Normalization corrected the variance, as seen in the After Normalization plot.

mainvarsetnorm



Examples

The following example extracts data from a GPR file and creates two column vectors of gene expression values from different experimental conditions. It then normalizes one of the data sets.

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

References

[1] Tseng, G.C., Oh, Min-Kyu, Rohlin, L., Liao, J.C., and Wong, W.H. (2001) Issues in cDNA microarray analysis: quality filtering, channel

normalization, models of variations and assessment of gene effects. *Nucleic Acids Research*. 29, 2549-2557.

[2] Hoffmann, R., Seidl, T., and Dugas, M. (2002) Profound effect of normalization on detection of differentially expressed genes in oligonucleotide microarray data analysis. *Genome Biology*. 3(7): research 0033.1-0033.11.

See Also

affyinvarsetnorm, malowess, manorm, quantilenorm

mairplot

Purpose

Create intensity versus ratio scatter plot of microarray data

Syntax

```
mairplot(DataX, DataY)
[Intensity, Ratio] = mairplot(DataX, DataY)
[Intensity, Ratio, H] = mairplot(DataX, DataY)
... = mairplot(..., 'Type', TypeValue, ...)
... = mairplot(..., 'LogTrans', LogTransValue, ...)
... = mairplot(..., 'FactorLines', FactorLinesValue, ...)
... = mairplot(..., 'Title', TitleValue, ...)
... = mairplot(..., 'Labels', LabelsValue, ...)
... = mairplot(..., 'Normalize', NormalizeValue, ...)
... = mairplot(..., 'LowessOptions', LowessOptionsValue, ...)
... = mairplot(..., 'Showplot', ShowplotValue, ...)
... = mairplot(..., 'PlotOnly', PlotOnlyValue, ...)
```

Arguments

<i>DataX, DataY</i>	DataMatrix object or vector of gene expression values where each row corresponds to a gene. For example, in a two-color microarray experiment, <i>DataX</i> could be cy3 intensity values and <i>DataY</i> could be cy5 intensity values.
<i>TypeValue</i>	String that specifies the plot type. Choices are 'IR' (plots \log_{10} of the product of the <i>DataX</i> and <i>DataY</i> intensities versus \log_2 of the intensity ratios) or 'MA' (plots $(1/2)\log_2$ of the product of the <i>DataX</i> and <i>DataY</i> intensities versus \log_2 of the intensity ratios). Default is 'IR'.
<i>LogTransValue</i>	Controls the conversion of data in <i>X</i> and <i>Y</i> from natural scale to \log_2 scale. Set <i>LogTransValue</i> to false, when the data is already \log_2 scale. Default is true, which assumes the data is natural scale.

FactorLinesValue Adds lines to the plot showing a factor of N change. Default is 2, which corresponds to a level of 1 and -1 on a \log_2 scale.

Tip You can also change the factor lines interactively, after creating the plot.

TitleValue String that specifies a title for the plot.

LabelsValue Cell array of labels for the data. If labels are defined, then clicking a point on the plot shows the label corresponding to that point.

NormalizeValue Controls the display of lowess normalized ratio values. Enter `true` to display to lowess normalized ratio values. Default is `false`.

Tip You can also normalize the data from the MAIR Plot window, after creating the plot.

LowessOptionsValue Cell array of one, two, or three property name/value pairs in any order that affect the lowess normalization. Choices for property name/value pairs are:

- 'Order', *OrderValue*
- 'Robust', *RobustValue*
- 'Span', *SpanValue*

For more information on the preceding property name/value pairs, see `malowess`.

ShowplotValue

Controls the display of the scatter plot. Choices are `true` (default) or `false`.

PlotOnlyValue

Controls the display of the scatter plot without user interface components. Choices are `true` or `false` (default).

Note If you set the 'PlotOnly' property to `true`, you can still display labels for data points by clicking a data point, and you can still adjust the horizontal fold change lines by click-dragging the lines.

Return Values*Intensity*

DataMatrix object or vector containing intensity values for the microarray gene expression data, calculated as:

- \log_{10} of the product of the *DataX* and *DataY* intensities (when Type is 'IR')
- $(1/2)\log_2$ of the product of the *DataX* and *DataY* intensities (when Type is 'MA')

Note If *DataX* or *DataY* is a DataMatrix object, then *Intensity* is also a DataMatrix object with the same properties.

Ratio

DataMatrix object or vector containing ratios of the microarray gene expression data, calculated as $\log_2(\text{DataX} ./ \text{DataY})$.

Note If *DataX* or *DataY* is a DataMatrix object, then *Ratio* is also a DataMatrix object with the same properties.

H

Handle of the plot.

Description

`mairplot(DataX, DataY)` creates a scatter plot that plots \log_{10} of the product of the *DataX* and *DataY* intensities versus \log_2 of the intensity ratios.

`[Intensity, Ratio] = mairplot(DataX, DataY)` returns the intensity and ratio values. If you set 'Normalize' to true, the returned ratio values are normalized.

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

mairplot

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

`... = mairplot(..., 'Type', TypeValue, ...)` specifies the plot type. Choices are 'IR' (plots \log_{10} of the product of the *DataX* and *DataY* intensities versus \log_2 of the intensity ratios) or 'MA' (plots $(1/2)\log_2$ of the product of the *DataX* and *DataY* intensities versus \log_2 of the intensity ratios). Default is 'IR'.

`... = mairplot(..., 'LogTrans', LogTransValue, ...)` controls the conversion of data in *X* and *Y* from natural to \log_2 scale. Set *LogTransValue* to `false`, when the data is already \log_2 scale. Default is `true`, which assumes the data is natural scale.

`... = mairplot(..., 'FactorLines', FactorLinesValue, ...)` adds lines to the plot showing a factor of *N* change. Default is 2, which corresponds to a level of 1 and -1 on a \log_2 scale.

Tip You can also change the factor lines interactively, after creating the plot.

`... = mairplot(..., 'Title', TitleValue, ...)` specifies a title for the plot.

`... = mairplot(..., 'Labels', LabelsValue, ...)` specifies a cell array of labels for the data. If labels are defined, then clicking a point on the plot shows the label corresponding to that point.

`... = mairplot(..., 'Normalize', NormalizeValue, ...)` controls the display of lowess normalized ratio values. Enter `true` to display to lowess normalized ratio values. Default is `false`.

Tip You can also normalize the data from the MAIR Plot window, after creating the plot.

`... = mairplot(..., 'LowessOptions', LowessOptionsValue, ...)` lets you specify up to three property name/value pairs (in any order) that affect the lowess normalization. Choices for property name/value pairs are:

- 'Order', *OrderValue*
- 'Robust', *RobustValue*
- 'Span', *SpanValue*

For more information on the previous three property name/value pairs, see the `malowess` function.

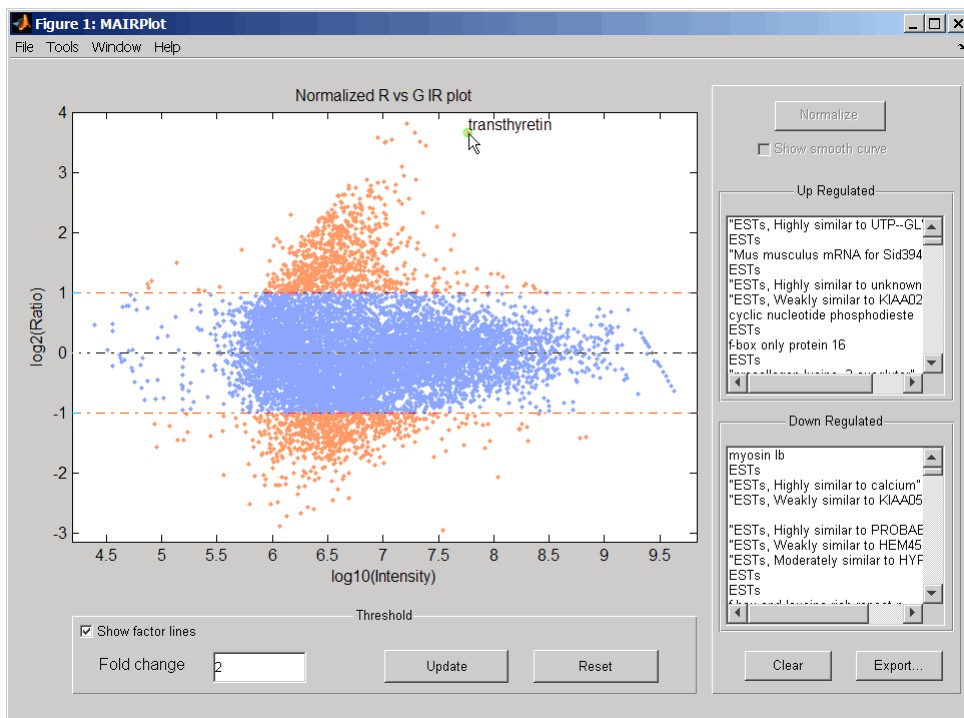
`... = mairplot(..., 'Showplot', ShowplotValue, ...)` controls the display of the scatter plot. Choices are `true` (default) or `false`.

`... = mairplot(..., 'PlotOnly', PlotOnlyValue, ...)` controls the display of the scatter plot without user interface components. Choices are `true` or `false` (default).

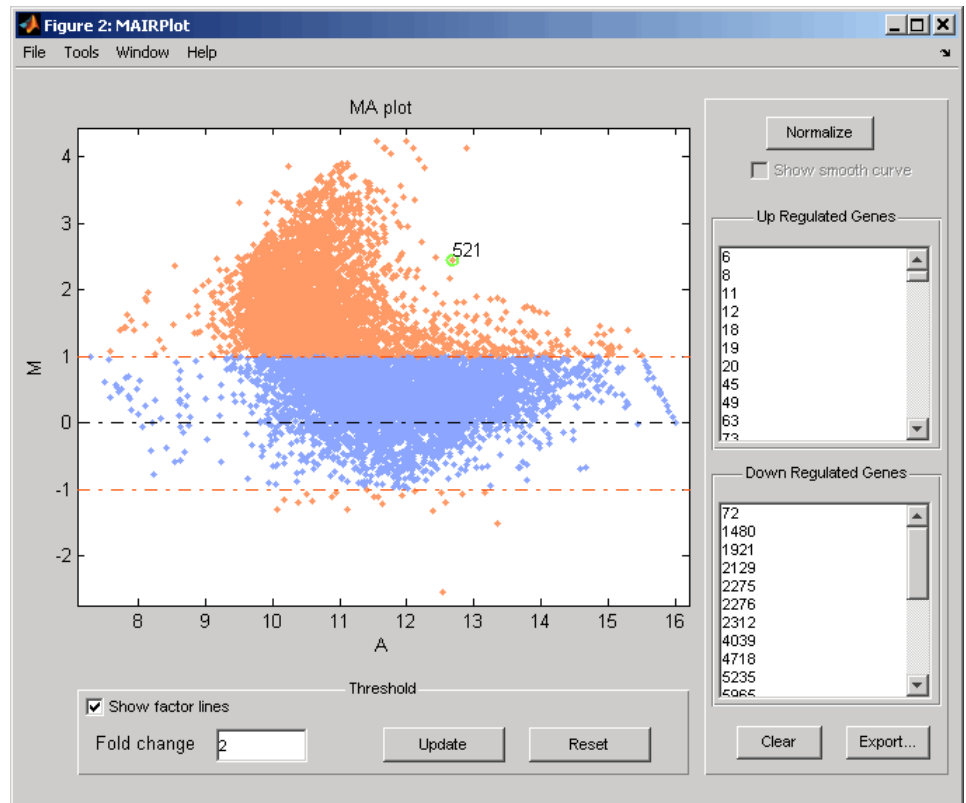
Note If you set the 'PlotOnly' property to `true`, you can still display labels for data points by clicking a data point, and you can still adjust the horizontal fold change lines by click-dragging the lines.

Following is an IR plot of normalized data.

mairplot



Following is an MA plot of unnormalized data.



The intensity versus ratio scatter plot displays the following:

- \log_{10} (Intensity) versus \log_2 (Ratio) scatter plot of genes.
- Two horizontal fold change lines at a fold change level of 2, which corresponds to a ratio of 1 and -1 on a \log_2 (Ratio) scale. (Lines will be at different fold change levels, if you used the 'FactorLines' property.)
- Data points for genes that are considered differentially expressed (outside of the fold change lines) appear in orange.

After you display the intensity versus ratio scatter plot, you can interactively do the following:

- Adjust the horizontal fold change lines by click-dragging one line or entering a value in the **Fold Change** text box, then clicking **Update**.
- Display labels for data points by clicking a data point.
- Select a gene from the **Up Regulated** or **Down Regulated** list to highlight the corresponding data point in the plot. Press and hold **Ctrl** or **Shift** to select multiple genes.
- Zoom the plot by selecting **Tools > Zoom In** or **Tools > Zoom Out**.
- View lists of significantly up-regulated and down-regulated genes, and optionally, export the gene labels and indices to a structure in the MATLAB Workspace by clicking **Export**.
- Normalize the data by clicking the **Normalize** button, then selecting whether to show the normalized plot in a separate window. If you show the normalized plot in a separate window, the **Show smooth curve** check box becomes available in the original (unnormalized) plot.

Tip To select different lowess normalization options before normalizing, select **Tools > Set LOWESS Normalization Options**, then enter options in the Options for LOWESS dialog box.

Examples

- 1 Use the `gprread` function to create a structure containing microarray data.

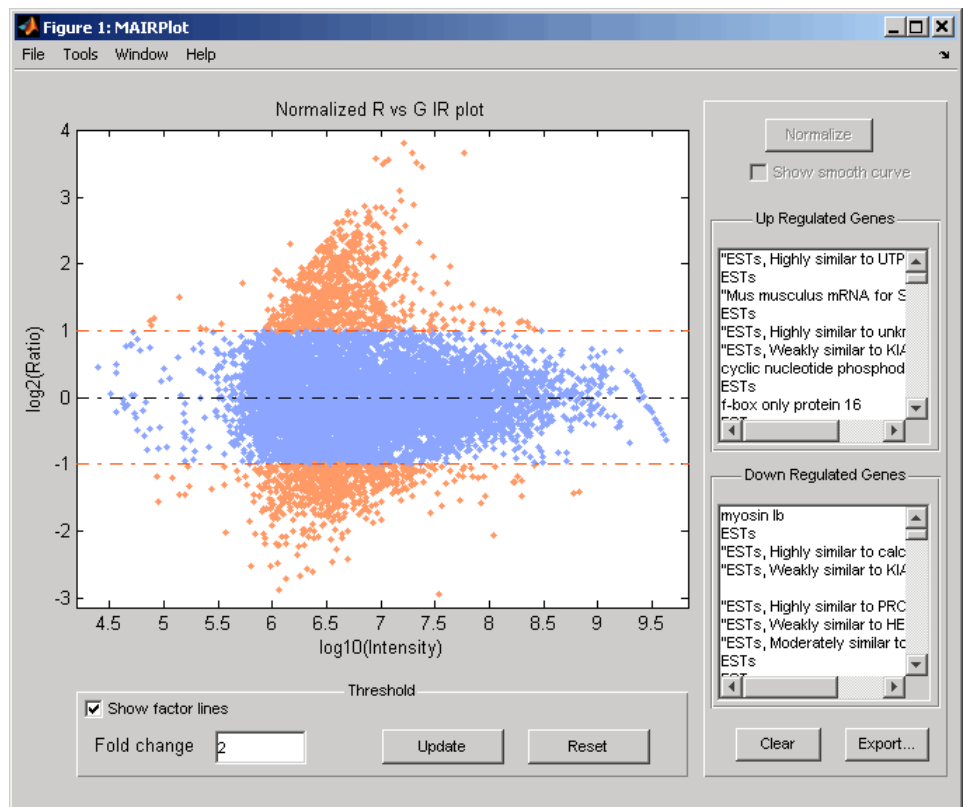
```
maStruct = gprread('mouse_a1wt.gpr');
```

- 2 Use the `magetfield` function to extract the green (cy3) and red (cy5) signals from the structure.

```
cy3data = magetfield(maStruct, 'F635 Median');  
cy5data = magetfield(maStruct, 'F532 Median');
```

- 3 Create an intensity versus ratio scatter plot of the cy3 and cy5 data. Normalize the data and add a title and labels:

```
mairplot(cy3data, cy5data, 'Normalize', true, ...  
        'Title', 'Normalized R vs G IR plot', ...  
        'Labels', maStruct.Names)
```

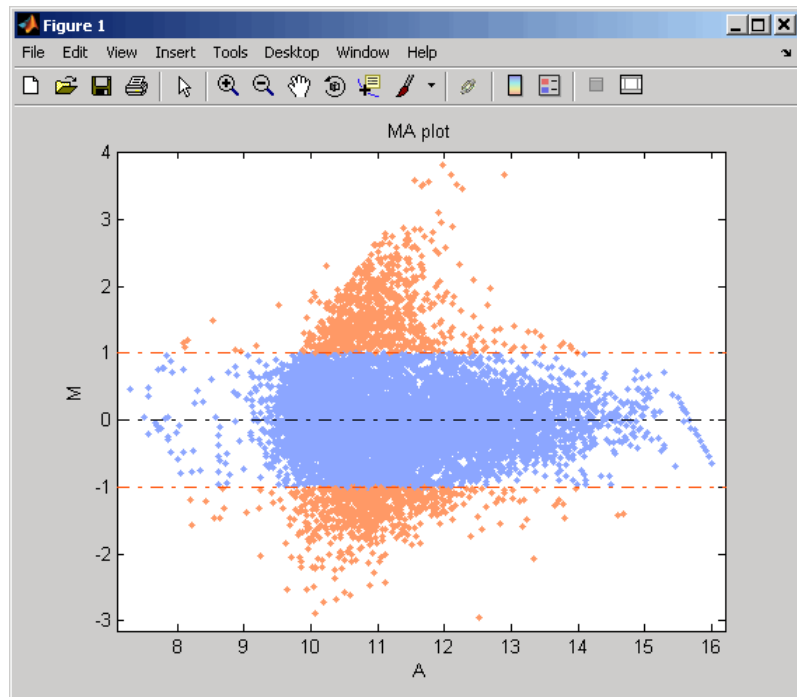


- 4 Return intensity values and ratios without displaying the plot.

```
[intensities, ratios] = mairplot(cy3data, cy5data, 'Showplot', false);
```

- 5 Create a normalized MA plot of the cy3 and cy5 data without the user interface components.

```
mairplot(cy3data, cy5data, 'Normalize', true, ...
         'Type', 'MA', 'PlotOnly', true)
```

References

- [1] Quackenbush, J. (2002). Microarray Data Normalization and Transformation. *Nature Genetics Suppl.* 32, 496–501.
- [2] Dudoit, S., Yang, Y.H., Callow, M.J., and Speed, T.P. (2002). Statistical Methods for Identifying Differentially Expressed Genes in Replicated cDNA Microarray Experiments. *Statistica Sinica* 12, 111–139.

See Also

Bioinformatics Toolbox functions: `maboxplot`, `magetfield`, `maimage`, `mainvarsetnorm`, `maloglog`, `malowess`, `manorm`, `matteest`, `maivolcanoplot`

maloglog

Purpose Create loglog plot of microarray data

Syntax

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

Arguments

<i>X, Y</i>	<code>DataMatrix</code> object or numeric array of microarray expression values from a single experimental condition.
<i>N</i>	Property to add two lines to the plot showing a factor of <i>N</i> change.
<i>TitleValue</i>	A string to use as the title for the plot.
<i>LabelsValue</i>	A cell array of labels for the data in <i>X</i> and <i>Y</i> . If you specify <i>LabelsValue</i> , then clicking a data point in the plot shows the label corresponding to that point.

Description

`maloglog(X, Y, 'PropertyName', PropertyValue...)` creates a loglog scatter plot of *X* versus *Y*. *X* and *Y* are `DataMatrix` objects or numeric arrays of microarray expression values from two different experimental conditions.

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

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

`maloglog(..., 'Labels', LabelsValues)` allows you to specify a cell array of labels for the data. If *LabelsValues* is defined, then clicking a data point in the plot shows the label corresponding to that point.

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

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

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
Red = magetfield(maStruct,'F635 Median');
Green = magetfield(maStruct,'F532 Median');
maloglog(Red,Green,'title','Red vs Green');
% Add factorlines and labels
figure
maloglog(Red,Green,'title','Red vs Green',...
         'FactorLines',2,'LABELS',maStruct.Names);
% Now create a normalized plot
figure
maloglog(manorm(Red),manorm(Green),'title',...
         'Normalized Red vs Green','FactorLines',2,...
         'LABELS',maStruct.Names);
```

See Also

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

MATLAB function `loglog`

Purpose Smooth microarray data using Lowess method

Syntax

```
YSmooth = malowess(X, Y)  
YSmooth = malowess(X, Y, ...'Order', OrderValue, ...)  
YSmooth = malowess(X, Y, ...'Robust', RobustValue, ...)  
YSmooth = malowess(X, Y, ...'Span', SpanValue, ...)
```

Arguments

<i>X</i> , <i>Y</i>	DataMatrix object or numeric vector containing scatter data.
<i>OrderValue</i>	Property to select the order of the algorithm. Enter either 1 (linear fit) or 2 (quadratic fit). The default order is 1.
<i>RobustValue</i>	Property to select a robust fit. Enter either true or false.
<i>SpanValue</i>	Property to specify the window size. The default value is 0.05 (5% of total points in <i>X</i>)

Description

YSmooth = malowess(*X*, *Y*) smooths scatter data in *X* and *Y* using the Lowess smoothing method. The default window size is 5% of the length of *X*. *YSmooth* is a numeric vector or, if *Y* is a DataMatrix object, also a DataMatrix object with the same properties as *Y*.

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

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

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

YSmooth = malowess(*X*, *Y*, ...'Span', *SpanValue*, ...) modifies the window size for the smoothing function. If *SpanValue* is less than 1, the window size is taken to be a fraction of the number of points in the data. If *SpanValue* is greater than 1, the window is of size *SpanValue*.

Examples

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

See Also

Bioinformatics Toolbox functions: `affyinvarsetnorm`, `maboxplot`, `magetfield`, `maimage`, `mainvarsetnorm`, `mairplot`, `maloglog`, `manorm`, `quantilenorm`

Statistics Toolbox function: `robustfit`

manorm

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, ...)
```

Arguments

X Numeric array or `DataMatrix` object of microarray data.

MAStruct Microarray structure.

FieldName Field.

Description

XNorm = manorm(*X*) scales the values in each column of *X*, a numeric array or `DataMatrix` object of microarray data, by dividing by the mean column intensity. *XNorm* is a vector, matrix, or `DataMatrix` object of normalized microarray data.

XNorm = manorm(*MAStruct*, *FieldName*) scales the data in *MAStruct*, a microarray structure, for a field specified by *FieldName*, for each block or print-tip by dividing each block by the mean column intensity. The output is a matrix with each column corresponding to the normalized data for each block.

[*XNorm*, *ColVal*] = manorm(...) returns the values used to normalize the data.

manorm(..., 'PropertyName', *PropertyValue*, ...) calls manorm with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each

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

`manorm(..., 'Method', MethodValue, ...)` allows you to choose the method for scaling or centering the data. *MethodValue* can be 'Mean' (default), 'Median', 'STD' (standard deviation), 'MAD' (median absolute deviation), or a function handle. If you pass a function handle, then the function should ignore NaNs and must return a single value per column of the input data.

`manorm(..., 'Extra_Args', Extra_ArgsValue, ...)` allows you to pass extra arguments to the function *MethodValue*. *Extra_ArgsValue* must be a cell array.

`manorm(..., 'LogData', LogDataValue, ...)`, when *LogDataValue* is true, works with log ratio data in which case the mean (or *MethodValue*) of each column is subtracted from the values in the columns, instead of dividing the column by the normalizing value.

`manorm(..., 'Percentile', PercentileValue, ...)` only uses the percentile (*PercentileValue*) of the data preventing large outliers from skewing the normalization. If *PercentileValue* is a vector containing two values, then the range from the *PercentileValue(1)* percentile to the *PercentileValue(2)* percentile is used. The default value is 100, that is to use all the data in the data set.

`manorm(..., 'Global', GlobalValue, ...)` when *GlobalValue* is true, normalizes the values in the data set by the global mean (or *MethodValue*) of the data, as opposed to normalizing each column or block of the data independently.

`manorm(..., 'StructureOutput', StructureOutputValue, ...)`, when *StructureOutputValue* is true, the input data is a structure returns the input structure with an additional data field for the normalized data.

`manorm(..., 'NewColumnName', NewColumnNameValue, ...)`, when using `StructureOutput`, allows you to specify the name of the column that is appended to the list of `ColumnNames` in the structure. The default behavior is to prefix 'Block Normalized' to the `FieldName` string.

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
% Extract some data of interest.
Red = magetfield(maStruct,'F635 Median');
Green = magetfield(maStruct,'F532 Median');
% Create a log-log plot.
maloglog(Red,Green,'factorlines',true)
% Center the data.
normRed = manorm(Red);
normGreen = manorm(Green);
% Create a log-log plot of the centered data.
figure
maloglog(normRed,normGreen,'title','Normalized','factorlines',true)

% Alternatively, you can work directly with the structure
normRedBs = manorm(maStruct,'F635 Median - B635');
normGreenBs = manorm(maStruct,'F532 Median - B532');
% Create a log-log plot of the centered data. This includes some
% zero values so turn off the warning.
figure
w = warning('off','Bioinfo:maloglog:ZeroValues');
warning('off','Bioinfo:maloglog:NegativeValues');
maloglog(normRedBs,normGreenBs,'title',...
          'Normalized Background-Subtracted Median Values',...
          'factorlines',true)
warning(w);
```

See Also

Bioinformatics Toolbox functions: `affyinvarsetnorm`, `maboxplot`, `magetfield`, `mainvarsetnorm`, `mairplot`, `maloglog`, `malowess`, `quantilenorm`, `rmasummary`

Purpose Create Principal Component Analysis (PCA) plot of microarray data

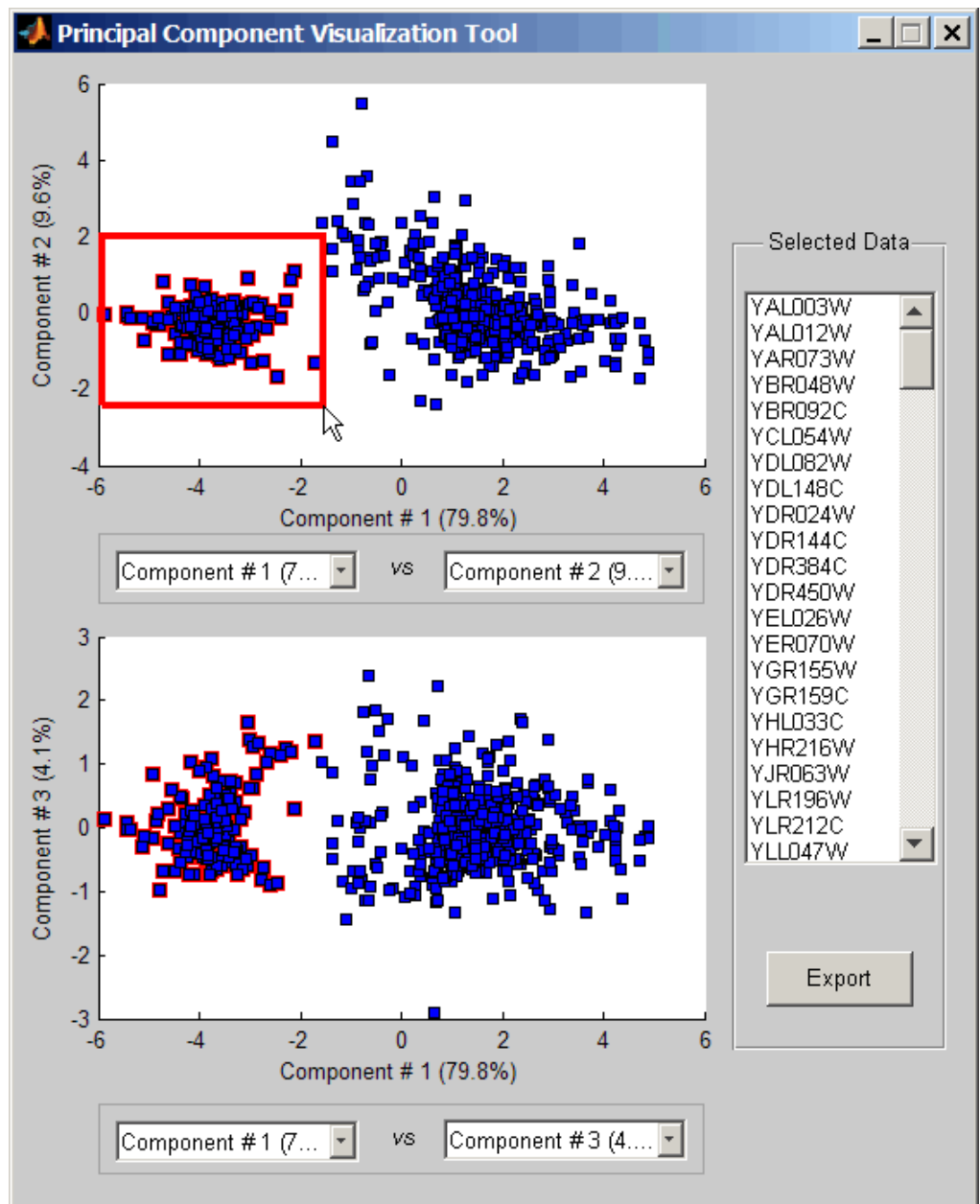
Syntax `mapcaplot(Data)`
`mapcaplot(Data, Label)`

Arguments

<i>Data</i>	DataMatrix object or numeric array containing microarray expression profile data. If a DataMatrix object, the row names are used as labels in the plot, unless you provide labels with the second input <i>Label</i> .
<i>Label</i>	Cell array of strings representing labels for the data points in the plot.

Description `mapcaplot(Data)` creates 2-D scatter plots of principal components of *Data*, a DataMatrix object or numeric array containing microarray expression profile data.

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



Once you plot the principal components, you can:

- Select principal components for the x and y axes from the drop-down list boxes below each scatter plot.
- Click a data point to display its label.
- Select a subset of data points by click-dragging a box around them. This will highlight the points in the selected region and the corresponding points in the other axes. The labels of the selected data points appear in the list box.
- Select a label in the list box to highlight the corresponding data point in the plot. Press and hold **Ctrl** or **Shift** to select multiple data points.
- Export the gene labels and indices to a structure in the MATLAB workspace by clicking **Export**.

Examples

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

See Also

Bioinformatics Toolbox functions: `clustergram`, `mattest`, `mavolcanoplot`

Statistics Toolbox function: `princomp`

mattest

Purpose

Perform two-sample t-test to evaluate differential expression of genes from two experimental conditions or phenotypes

Syntax

```
PValues = mattest(DataX, DataY)  
[PValues, TScores] = mattest(DataX, DataY)  
[PValues, TScores, DFs] = mattest(DataX, DataY)  
... = mattest(..., 'Permute', PermuteValue, ...)  
... = mattest(..., 'Bootstrap', BootstrapValue, ...)  
... = mattest(..., 'Showhist', ShowhistValue, ...)  
... = mattest(..., 'Showplot', ShowplotValue, ...)  
... = mattest(..., 'Labels', LabelsValue, ...)
```

Arguments

<i>DataX</i> , <i>DataY</i>	<p>DataMatrix object or a matrix of gene expression values where each row corresponds to a gene and each column corresponds to a replicate. <i>DataX</i> and <i>DataY</i> must have the same number of rows and are assumed to be normally distributed in each class with equal variances.</p> <p><i>DataX</i> contains data from one experimental condition and <i>DataY</i> contains data from a different experimental condition. For example, <i>DataX</i> could be expression values from cancer cells, and <i>DataY</i> could be expression values from normal cells.</p>
<i>PermuteValue</i>	<p>Controls whether permutation tests are run, and if so, how many. Choices are <code>true</code>, <code>false</code> (default), or any integer greater than 2. If set to <code>true</code>, the number of permutations is 1000.</p>
<i>BootstrapValue</i>	<p>Controls whether bootstrap tests are run, and if so, how many. Choices are <code>true</code>, <code>false</code> (default), or any integer greater than 2. If set to <code>true</code>, the number of bootstrap tests is 1000.</p>

<i>ShowhistValue</i>	Controls the display of histograms of t-score distributions and p-value distributions. Choices are <code>true</code> or <code>false</code> (default).
<i>ShowplotValue</i>	Controls the display of a normal t-score quantile plot. Choices are <code>true</code> or <code>false</code> (default). In the t-score quantile plot, data points with t-scores $> (1 - 1/(2N))$ or $< 1/(2N)$ display with red circles. N is the total number of genes.
<i>LabelsValue</i>	Cell array of labels (typically gene names or probe set IDs) for each row in <i>DataX</i> and <i>DataY</i> . The labels display if you click a data point in the t-score quantile plot.

Return Values

<i>PValues</i>	One of the following: <ul style="list-style-type: none"> • Column vector of p-values for each gene in <i>DataX</i> and <i>DataY</i> (if both inputs are matrices). • <code>DataMatrix</code> object with row names the same as the first input <code>DataMatrix</code> object and a column name of p-values (if at least one input is a <code>DataMatrix</code> object).
<i>TScores</i>	Column vector of t-scores for each gene in <i>DataX</i> and <i>DataY</i> .
<i>DFs</i>	Column vector containing the degree of freedom for each gene in <i>DataX</i> and <i>DataY</i> .

Description

PValues = `mattest(DataX, DataY)` compares the gene expression profiles in *DataX* and *DataY* and returns a p-value for each gene. *DataX* and *DataY* are either a `DataMatrix` object or a matrix of gene expression values, in which each row corresponds to a gene, and each column corresponds to a replicate. *DataX* contains data from

one experimental condition and *DataY* contains data from another experimental condition. *DataX* and *DataY* must have the same number of rows and are assumed to be normally distributed in each class with equal variances. *PValues* is a column vector of p-values for each gene, or, if at least one of the inputs is a *DataMatrix* object, a *DataMatrix* object with row names the same as the first input *DataMatrix* object and a column name of p-values.

`[PValues, TScores] = mattest(DataX, DataY)` also returns a t-score for each gene in *DataX* and *DataY*. *TScores* is a column vector of t-scores for each gene.

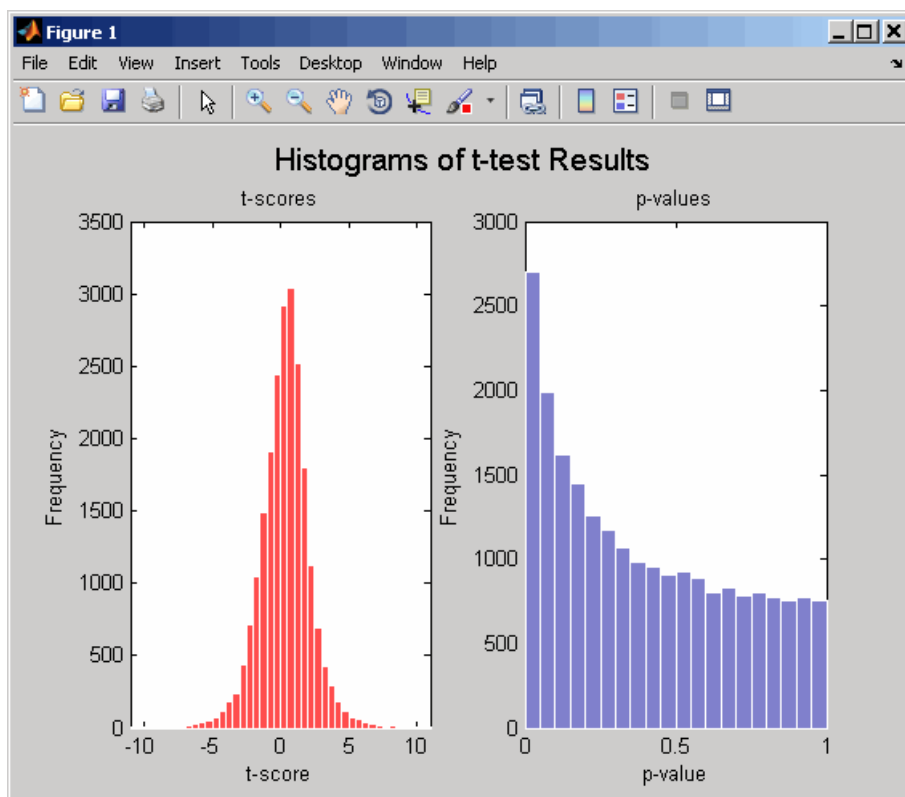
`[PValues, TScores, DFs] = mattest(DataX, DataY)` also returns *DFs*, a column vector containing the degree of freedom for each gene across both data sets, *DataX* and *DataY*.

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

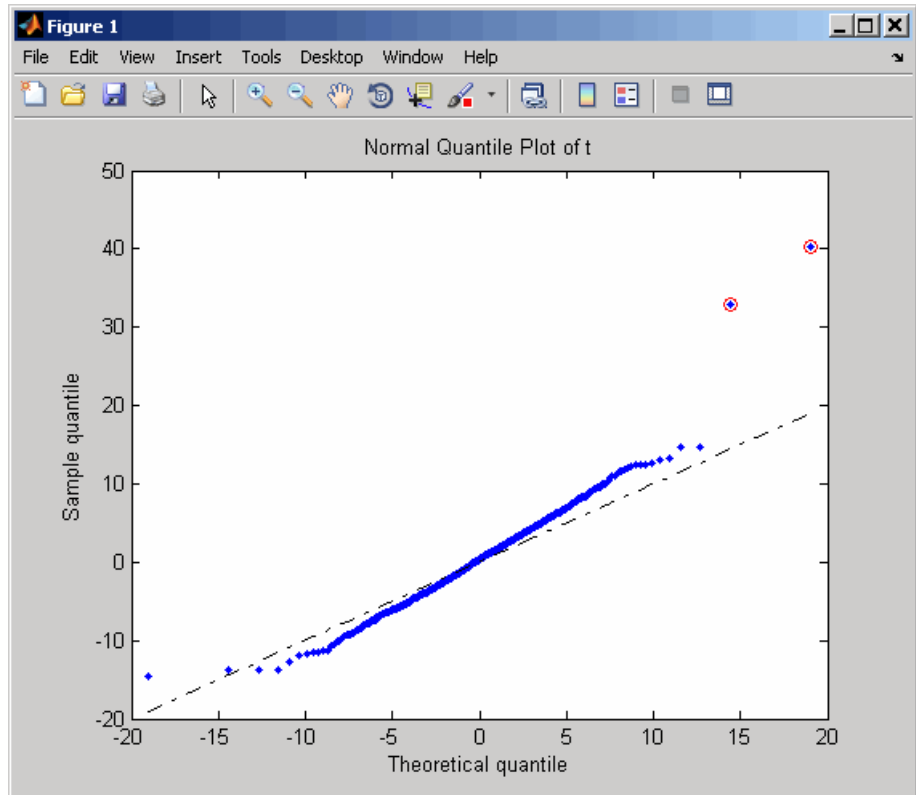
`... = mattest(..., 'Permute', PermuteValue, ...)` controls whether permutation tests are run, and if so, how many. *PermuteValue* can be true, false (default), or any integer greater than 2. If set to true, the number of permutations is 1000.

`... = mattest(..., 'Bootstrap', BootstrapValue, ...)` controls whether bootstrap tests are run, and if so, how many. *BootstrapValue* can be true, false (default), or any integer greater than 2. If set to true, the number of bootstrap tests is 1000.

`... = mattest(..., 'Showhist', ShowhistValue, ...)` controls the display of histograms of t-score distributions and p-value distributions. When *ShowhistValue* is true, `mattest` displays histograms. Default is false.



... = `mattest(..., 'Showplot', ShowplotValue, ...)` controls the display of a normal t-score quantile plot. When `ShowplotValue` is true, `mattest` displays a quantile-quantile plot. Default is false. In the t-score quantile plot, the black diagonal line represents the sample quantile being equal to the theoretical quantile. Data points of genes considered to be differentially expressed lie farther away from this line. Specifically, data points with t-scores $> (1 - 1/(2N))$ or $< 1/(2N)$ display with red circles. N is the total number of genes.



`... = mattest(..., 'Labels', LabelsValue, ...)` controls the display of labels when you click a data point in the t-score quantile plot. *LabelsValue* is a cell array of labels (typically gene names or probe set IDs) for each row in *DataX* and *DataY*.

Examples

- 1 Load the MAT-file, included with the Bioinformatics Toolbox software, that contains Affymetrix data from a prostate cancer study, specifically probe intensity data from Affymetrix HG-U133A GeneChip arrays. The two variables in the MAT-file, *dependentData* and *independentData*, are two matrices of gene expression values from two experimental conditions.


```
load prostatecancerexpdata
```

- 2 Calculate the p-values and t-scores for the gene expression values in the two matrices and display a normal t-score quantile plot.

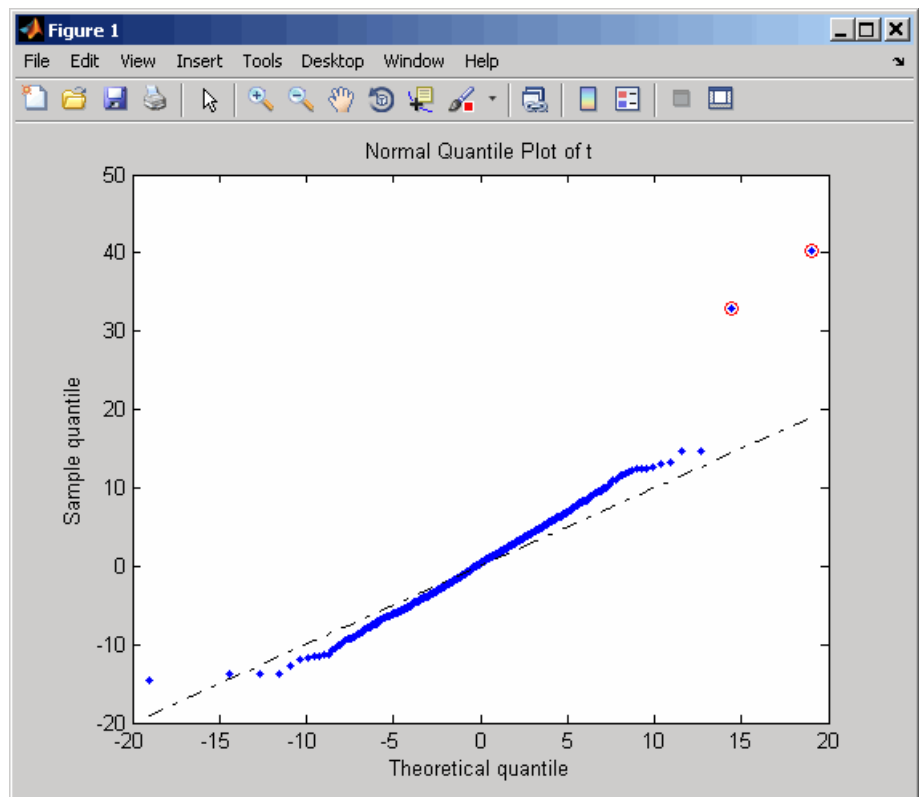
```
[pvalues,tcores] = mattest(dependentData, independentData,...  
                           'showplot',true);
```

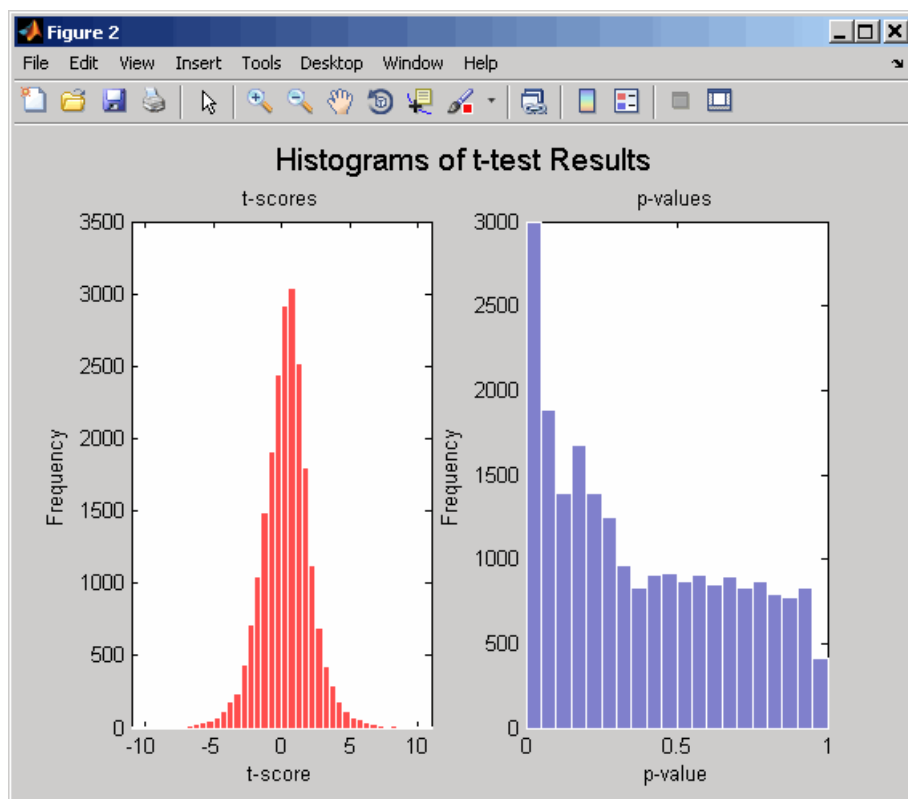
- 3 Calculate the p-values and t-scores again using permutation tests (1000 permutations) and displaying histograms of t-score distributions and p-value distributions.

```
[pvalues,tcores] = mattest(dependentData,independentData,...  
                           'permute',true,'showhist',true,...  
                           'showplot',true);
```

- 4 Calculate the p-values and t-scores again using bootstrap tests (2000 tests) and displaying histograms of t-score distributions and p-value distributions.

```
[pvalues,tcores] = mattest(dependentData,independentData,...  
                           'bootstrap',2000,'showhist',true,...  
                           'showplot',true);
```





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

References

- [1] Huber, W., von Heydebreck, A., Sültmann, H., Poustka, A., and Vingron, M. (2002). Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics 18 (Suppl. 1)*, S96–S104.
- [2] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan,

W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. *Clinical Cancer Research* *11*, 6823–6834.

See Also

Bioinformatics Toolbox functions: `affygcRMA`, `affyRMA`, `maboxplot`, `mafdr`, `mainvarsetnorm`, `mairplot`, `maloglog`, `malowess`, `manorm`, `mavolcanoplot`, `rmasummary`

Purpose Create significance versus gene expression ratio (fold change) scatter plot of microarray data

Syntax

```
mavolcanoplot(DataX, DataY, PValues)
SigStructure = mavolcanoplot(DataX, DataY, PValues)
... mavolcanoplot(..., 'Labels', LabelsValue, ...)
... mavolcanoplot(..., 'LogTrans', LogTransValue, ...)
... mavolcanoplot(..., 'PCutoff', PCutoffValue, ...)
... mavolcanoplot(..., 'Foldchange', FoldchangeValue, ...)
... mavolcanoplot(..., 'PlotOnly', PlotOnlyValue, ...)
```

Arguments

<i>DataX, DataY</i>	<i>DataMatrix</i> object, matrix, or vector of gene expression values from a single experimental condition. If a <i>DataMatrix</i> object or a matrix, each row is a gene, each column is a sample, and an average expression value is calculated for each gene.
---------------------	--

Note If the values in *DataX* or *DataY* are natural scale, use the `LogTrans` property to convert them to \log_2 scale.

<i>PValues</i>	<p>Either of the following:</p> <ul style="list-style-type: none"> • Column vector of p-values for each feature (for example, gene) in a data set, such as returned by <code>mattest</code>. • <i>DataMatrix</i> object containing p-values for each feature (for example, gene) in a data set, such as returned by <code>mattest</code>.
----------------	---

mavolcanoplot

<i>LabelsValue</i>	Cell array of labels (typically gene names or probe set IDs) for the data. After creating the plot, you can click a data point to display the label associated with it. If you do not provide a <i>LabelsValue</i> , data points are labeled with row numbers from <i>DataX</i> and <i>DataY</i> .
<i>LogTransValue</i>	Property to control the conversion of data in <i>DataX</i> and <i>DataY</i> from natural scale to \log_2 scale. Enter <code>true</code> to convert data to \log_2 scale, or <code>false</code> . Default is <code>false</code> , which assumes data is already \log_2 scale.
<i>PCutoffValue</i>	Lets you specify a cutoff p-value to define data points that are statistically significant. This value is displayed graphically as a horizontal line on the plot. Default is <code>0.05</code> , which is equivalent to 1.3010 on the $-\log_{10}$ (p-value) scale.

Note You can also change the p-value cutoff interactively after creating the plot.

FoldchangeValue Lets you specify a ratio fold change to define data points that are differentially expressed. Default is 2, which corresponds to a ratio of 1 and -1 on a \log_2 (ratio) scale.

Note You can also change the fold change interactively after creating the plot.

PlotOnlyValue Controls the display of the volcano plot without user interface components. Choices are `true` or `false` (default).

Note If you set the 'PlotOnly' property to `true`, you can still display labels for data points by clicking a data point, and you can still adjust vertical fold change lines and the horizontal p-value cutoff line by click-dragging the lines.

Return Values

SigStructure Structure containing information for genes that are considered to be both statistically significant (above the p-value cutoff) and significantly differentially expressed (outside of the fold change values). The fields are listed below.

Description

`mavolcanoplot(DataX, DataY, PValues)` creates a scatter plot of gene expression data, plotting significance versus fold change of gene expression ratios of two data sets, *DataX* and *DataY*. It plots significance as the $-\log_{10}$ (p-value) from the input, *PValues*. *DataX* and *DataY* can be vectors, matrices, or `DataMatrix` objects. *PValues* is a column vector or `DataMatrix` object.

mavolcanoplot

SigStructure = mavolcanoplot(*DataX*, *DataY*, *PValues*) returns a structure containing information for genes that are considered to be both statistically significant (above the p-value cutoff) and significantly differentially expressed (outside of the fold change values). The fields within *SigStructure* are sorted by p-value and include:

- Name
- PCutoff
- FCThreshold
- GeneLabels
- PValues
- FoldChanges

Note The fields *PValues* and *FoldChanges* will be either vectors or *DataMatrix* objects depending on the type of input *PValues*.

... mavolcanoplot(..., '*PropertyName*', *PropertyValue*, ...) defines optional properties that use property name/value pairs in any order. These property name/value pairs are as follows:

... mavolcanoplot(..., 'Labels', *LabelsValue*, ...) lets you provide a cell array of labels (typically gene names or probe set IDs) for the data. After creating the plot, you can click a data point to display the label associated with it. If you do not provide a *LabelsValue*, data points are labeled with row numbers from *DataX* and *DataY*.

... mavolcanoplot(..., 'LogTrans', *LogTransValue*, ...) controls the conversion of data from *DataX* and *DataY* to \log_2 scale. When *LogTransValue* is true, mavolcanoplot converts data from natural to \log_2 scale. Default is false, which assumes the data is already \log_2 scale.

... mavolcanoplot(..., 'PCutoff', *PCutoffValue*, ...) lets you specify a p-value cutoff to define data points that are statistically

significant. This value displays graphically as a horizontal line on the plot. Default is 0.05, which is equivalent to 1.3010 on the $-\log_{10}$ (p-value) scale.

Note You can also change the p-value cutoff interactively after creating the plot.

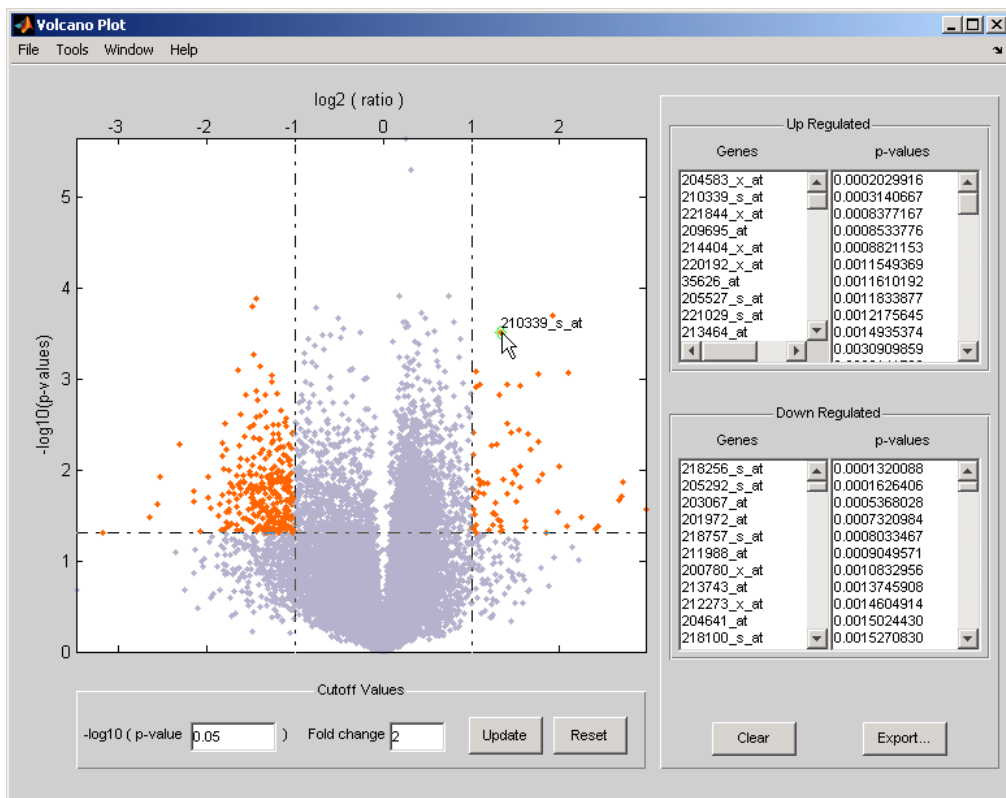
`... mavolcanoplot(..., 'Foldchange', FoldchangeValue, ...)` lets you specify a ratio fold change to define data points that are differentially expressed. Fold changes display graphically as two vertical lines on the plot. Default is 2, which corresponds to a ratio of 1 and -1 on a \log_2 (ratio) scale.

Note You can also change the fold change interactively after creating the plot.

`... mavolcanoplot(..., 'PlotOnly', PlotOnlyValue, ...)` controls the display of the volcano plot without user interface components. Choices are `true` or `false` (default).

Note If you set the 'PlotOnly' property to `true`, you can still display labels for data points by clicking a data point, and you can still adjust vertical fold change lines and the horizontal p-value cutoff line by click-dragging the lines.

mavolcanoplot



The volcano plot displays the following:

- $-\log_{10}(\text{p-value})$ versus $\log_2(\text{ratio})$ scatter plot of genes
- Two vertical fold change lines at a fold change level of 2, which corresponds to a ratio of 1 and -1 on a $\log_2(\text{ratio})$ scale. (Lines will be at different fold change levels, if you used the 'Foldchange' property.)
- One horizontal line at the 0.05 p-value level, which is equivalent to 1.3010 on the $-\log_{10}(\text{p-value})$ scale. (The line will be at a different p-value level, if you used the 'PCutoff' property.)

- Data points for genes that are considered both statistically significant (above the p-value line) and differentially expressed (outside of the fold changes lines) appear in orange.

After you display the volcano scatter plot, you can interactively:

- Adjust the vertical fold change lines by click-dragging one line or entering a value in the **Fold Change** text box.
- Adjust the horizontal p-value cutoff line by click-dragging or entering a value in the **p-value Cutoff** text box.
- Display labels for data points by clicking a data point.
- Select a gene from the **Up Regulated** or **Down Regulated** list to highlight the corresponding data point in the plot. Press and hold **Ctrl** or **Shift** to select multiple genes.
- Zoom the plot by selecting **Tools > Zoom In** or **Tools > Zoom Out**.
- View lists of significantly up-regulated and down-regulated genes and their associated p-values, and optionally, export the labels, p-values, and fold changes to a structure in the MATLAB Workspace by clicking **Export**.

Examples

- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains Affymetrix data variables, including `dependentData` and `independentData`, two matrices of gene expression values from two experimental conditions.

```
load prostatecancerexpdata
```

- 2 Use the `mattest` function to calculate p-values for the gene expression values in the two matrices.

```
pvalues = mattest(dependentData, independentData);
```

- 3 Using the two matrices, the `pvalues` calculated by `mattest`, and the `probesetIDs` column vector of labels provided, use `mavolcanoplot` to

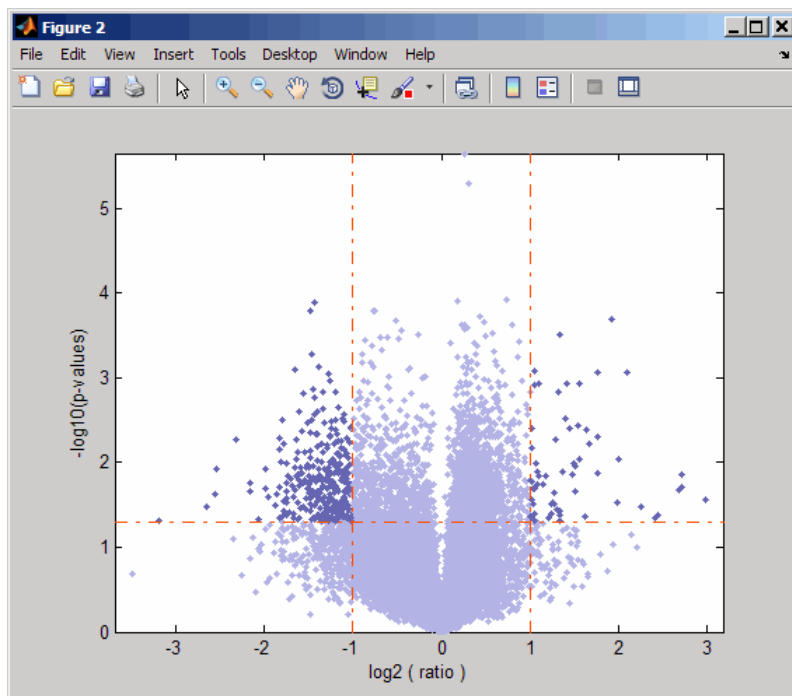
mavolcanoplot

create a significance versus gene expression ratio scatter plot of the microarray data from the two experimental conditions.

```
mavolcanoplot(dependentData, independentData, pvalues,...  
'Labels', probesetIDs)
```

4 View the volcano plot without the user interface components.

```
mavolcanoplot(dependentData, independentData, pvalues,...  
'Labels', probesetIDs,'Plotonly', true)
```



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

References

[1] Cui, X., Churchill, G.A. (2003). Statistical tests for differential expression in cDNA microarray experiments. *Genome Biology* 4, 210.

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

See Also

Bioinformatics Toolbox functions: `maboxplot`, `maimage`, `mainvarsetnorm`, `mairplot`, `maloglog`, `malowess`, `manorm`, `mapcaplot`, `mattest`

molweight

Purpose Calculate molecular weight of amino acid sequence

Syntax `molweight(SeqAA)`

Arguments

`SeqAA` Amino acid sequence. Enter a character string or a vector of integers from the tableAmino Acid Lookup on page 2-91. Examples: 'ARN', [1 2 3]. You can also enter a structure with the field Sequence.

Description `molweight(SeqAA)` calculates the molecular weight for the amino acid sequence `SeqAA`.

Examples

1 Get an amino acid sequence from the NCBI GenPept database.

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

2 Calculate the molecular weight of the sequence.

```
rhodopsinMW = molweight(rhodopsin)
```

```
rhodopsinMW =
```

```
3.8892e+004
```

See Also

Bioinformatics Toolbox functions: `aaccount`, `atomiccomp`, `isoelectric`, `proteinplot`

Purpose Display and manipulate 3-D molecule structure

Syntax `molviewer`
 `molviewer(File)`
 `molviewer(pdbID)`
 `molviewer(pdbStruct)`
 `FigureHandle = molviewer(...)`

Arguments

<i>File</i>	<p>String specifying one of the following:</p> <ul style="list-style-type: none">• File name of a file on the MATLAB search path or in the MATLAB Current Directory• Path and file name• URL pointing to a file (URL must begin with a protocol such as http://, ftp://, or file://) <p>The referenced file is a molecule model file, such as a Protein Data Bank (PDB)-formatted file (ASCII text file). Valid file types include:</p> <ul style="list-style-type: none">• PDB• MOL (MDL)• SDF• XYZ• SMOL• JVXL• CIF/mmCIF
<i>pdbID</i>	<p>String specifying a unique identifier for a protein structure record in the PDB database.</p> <hr/> <p>Note Each structure in the PDB database is represented by a four-character alphanumeric identifier. For example, 4hhb is the identifier for hemoglobin.</p> <hr/>
<i>pdbStruct</i>	<p>A structure containing a field for each PDB record, such as returned by the <code>getpdb</code> or <code>pdbread</code> function.</p>

Return Values

FigureHandle Figure handle to a Molecule Viewer window.

Description

molviewer opens a blank Molecule Viewer window. You can display 3-D molecular structures by selecting **File > Open**, **File > Load PDB ID**, or **File > Open URL**.

molviewer(*File*) reads the data in a molecule model file, *File*, and opens a Molecule Viewer window displaying the 3-D molecular structure for viewing and manipulation.

molviewer(*pdbID*) retrieves the data for a protein structure record, *pdbID*, from the PDB database and opens a Molecule Viewer window displaying the 3-D molecular structure for viewing and manipulation.

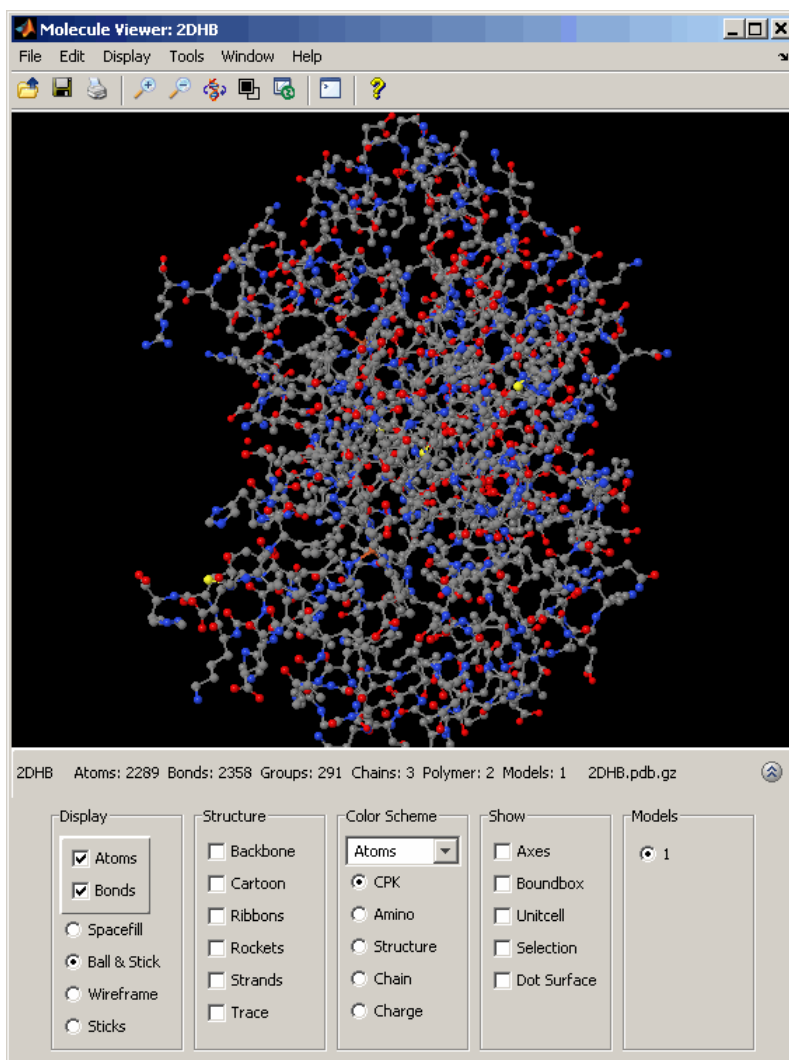
molviewer(*pdbStruct*) reads the data from *pdbStruct*, a structure containing a field for each PDB record, and opens a Molecule Viewer window displaying a 3-D molecular structure for viewing and manipulation.

FigureHandle = molviewer(...) returns the figure handle to the Molecule Viewer window.

Tip You can pass the *FigureHandle* to the evalrasmolscript function, which sends RasMol script commands to the Molecule Viewer window.

Tip If you receive any errors related to memory or Java™ heap space, try increasing your Java heap space as described at:





<http://www.mathworks.com/support/solutions/data/1-18I2C.html>

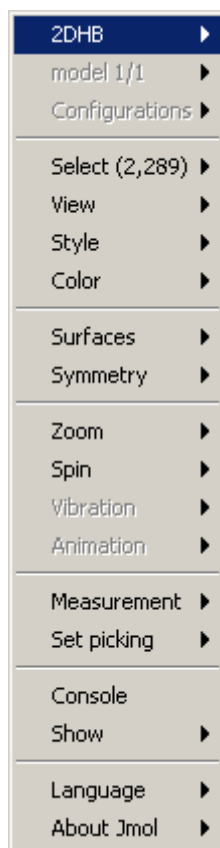


After displaying the 3-D molecule structure, you can:

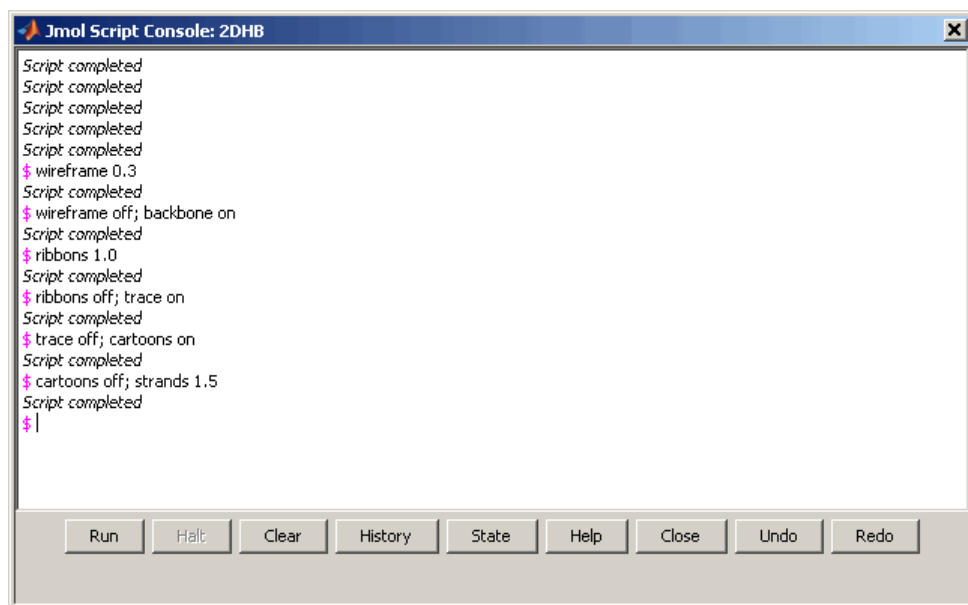
- Click-drag the molecule to spin, rotate, and view it from different angles.
- Hover the mouse over a subcomponent of the molecule to display an identification label for it.
- Zoom the plot by turning the mouse scroll wheel or clicking the following buttons:



- Spin the molecule by clicking .
- Change the background color between black and white by clicking .
- Reset the molecule position by clicking .
- Show or hide the Control Panel by clicking .
- Manipulate and annotate the 3-D structure by selecting options in the Control Panel or, for a complete list of options, by right-clicking the Molecule Viewer window to select commands:



- Display the Jmol Script Console by clicking .



Examples

View the acetylsalicylic acid (aspirin) molecule, whose structural information is contained in the Elsevier MDL molecule file `aspirin.mol`.

```
molviewer('aspirin.mol')
```

View the H5N1 influenza virus hemagglutinin molecule, whose structural information is located at www.rcsb.org/pdb/files/2FK0.pdb.gz.

```
molviewer('http://www.rcsb.org/pdb/files/2FK0.pdb.gz')
```

View the molecule with a PDB identifier of 2DHB.

```
molviewer('2DHB')
```

View the molecule with a PDB identifier of 4hhb, and create a figure handle for the molecule viewer.

molviewer

```
FH = molviewer('4hhb')
```

Use the `getpdb` function to retrieve protein structure data from the PDB database and create a MATLAB structure. Then view the protein molecule.

```
pdbstruct = getpdb('1vqx')  
molviewer(pdbstruct)
```

See Also

Bioinformatics Toolbox functions: `evalrasmolscript`, `getpdb`, `pdbread`, `pdbsuperpose`, `pdbtransform`, `pdwrite`

Purpose

Align peaks in mass spectrum to reference peaks

Syntax

```
IntensitiesOut = msalign(MZ, Intensities, RefMZ)  
... = msalign(..., 'Weights', WeightsValue, ...)  
... = msalign(..., 'Range', RangeValue, ...)  
... = msalign(..., 'WidthOfPulses',  
WidthOfPulsesValue, ...)  
... = msalign(..., 'WindowSizeRatio', WindowSizeRatioValue,  
...)  
... = msalign(..., 'Iterations', IterationsValue, ...)  
... = msalign(..., 'GridSteps', GridStepsValue, ...)  
... = msalign(..., 'SearchSpace', SearchSpaceValue, ...)  
... = msalign(..., 'ShowPlot', ShowPlotValue, ...)  
[IntensitiesOut, RefMZOut] = msalign(...,  
'Group', GroupValue,  
...)
```

Arguments

MZ

Vector of mass/charge (m/z) values for a spectrum or set of spectra. The number of elements in the vector equals n or the number of rows in the matrix *Intensities*.

Intensities

Either of the following:

- Column vector of intensity values for a spectrum, where each row corresponds to an m/z value.
- Matrix of intensity values for a set of mass spectra that share the same m/z range, where each row corresponds to an m/z value, and each column corresponds to a spectrum.

The number of rows equals n or the number of elements in vector *MZ*.

<i>RefMZ</i>	Vector of m/z values of known reference masses in a sample spectrum.
	<hr/> Tip For reference peaks, select compounds that do not undergo structural transformation, such as phosphorylation. Doing so will increase the accuracy of your alignment and allow you to detect compounds that do exhibit structural transformations among the sample spectra. <hr/>
<i>WeightsValue</i>	Vector of positive values, with the same number of elements as <i>RefMZ</i> . The default vector is <code>ones(size(RefMZ))</code> .
<i>RangeValue</i>	Two-element vector, in which the first element is negative and the second element is positive, that specifies the lower and upper limits of a range, in m/z units, relative to each peak. No peak will shift beyond these limits. Default is <code>[-100 100]</code> .
<i>WidthOfPulsesValue</i>	Positive value that specifies the width, in m/z units, for all the Gaussian pulses used to build the correlating synthetic spectrum. The point of the peak where the Gaussian pulse reaches 60.65% of its maximum is set to the width specified by <i>WidthOfPulsesValue</i> . Default is 10.

<i>WindowSizeRatioValue</i>	Positive value that specifies a scaling factor that determines the size of the window around every alignment peak. The synthetic spectrum is compared to the sample spectrum only within these regions, which saves computation time. The size of the window is given in m/z units by <i>WidthOfPulsesValue</i> * <i>WindowSizeRatioValue</i> . Default is 2.5, which means at the limits of the window, the Gaussian pulses have a value of 4.39% of their maximum.
<i>IterationsValue</i>	Positive integer that specifies the number of refining iterations. At every iteration, the search grid is scaled down to improve the estimates. Default is 5.
<i>GridStepsValue</i>	Positive integer that specifies the number of steps for the search grid. At every iteration, the search area is divided by <i>GridStepsValue</i> ² . Default is 20.
<i>SearchSpaceValue</i>	String that specifies the type of search space. Choices are: <ul style="list-style-type: none">• 'regular' — Default. Evenly spaced lattice.• 'latin' — Random Latin hypercube with <i>GridStepsValue</i>² samples.

ShowPlotValue

Controls the display of a plot of an original and aligned spectrum over the reference masses specified by *RefMZ*. Choices are *true*, *false*, or *I*, an integer specifying the index of a spectrum in *Intensities*. If set to *true*, the first spectrum in *Intensities* is plotted. Default is:

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

GroupValue

Controls the creation of *RefMZOut*, a new vector of m/z values to be used as reference masses for aligning the peaks. This vector is created by adjusting the values in *RefMZ*, based on the sample data from multiple spectra in *Intensities*, such that the overall shifting and scaling of the peaks is minimized. Choices are *true* or *false* (default).

Tip Set *GroupValue* to *true* only if *Intensities* contains data for a large number of spectra, and you are not confident of the m/z values used for your reference peaks in *RefMZ*. Leave *GroupValue* set to *false* if you are confident of the m/z values used for your reference peaks in *RefMZ*.

Return Values*IntensitiesOut*

Either of the following:

- Column vector intensity values for a spectrum, where each row corresponds to an m/z value.
- Matrix of intensity values for a set of mass spectra that share the same mass/charge (m/z) range, where each row corresponds to an m/z value, and each column corresponds to a spectrum.

The intensity values represent a shifting and scaling of the data.

RefMZOut

Vector of m/z values of reference masses, calculated from *RefMZ* and the sample data from multiple spectra in *Intensities*, when *GroupValue* is set to true.

Description

IntensitiesOut = `msalign(MZ, Intensities, RefMZ)` aligns the peaks in a raw mass spectrum or spectra, represented by *Intensities* and *MZ*, to reference peaks, provided by *RefMZ*. First, it creates a synthetic spectrum from the reference peaks using Gaussian pulses centered at the m/z values specified by *RefMZ*. Then, it shifts and scales the m/z scale to find the maximum alignment between the input spectrum or spectra and the synthetic spectrum. (It uses an iterative multiresolution grid search until it finds the best scale and shift factors for each spectrum.) Once the new m/z scale is determined, the corrected spectrum or spectra are created by resampling their intensities at the original m/z values, creating *IntensitiesOut*, a vector or matrix of corrected intensity values. The resampling method preserves the shape of the peaks.

Note The `msalign` function works best with three to five reference peaks (marker masses) that you know will appear in the spectrum. If you use a single reference peak (internal standard), there is a possibility of aligning sample peaks to the incorrect reference peaks as `msalign` both scales and shifts the *MZ* vector. If using a single reference peak, you might need to only shift the *MZ* vector. To do this, use `IntensitiesOut = interp1(MZ, Intensities, MZ - (ReferenceMass - ExperimentalMass))`. For more information, see [Aligning Mass Spectrum with One Reference Peak](#) on page 2-665.

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

`... = msalign(..., 'Weights', WeightsValue, ...)` specifies the relative weight for each mass in *RefMZ*, the vector of reference m/z values. *WeightsValue* is a vector of positive values, with the same number of elements as *RefMZ*. The default vector is `ones(size(RefMZ))`, which means each reference peak is weighted equally, so that more intense reference peaks have a greater effect in the alignment algorithm. If you have a less intense reference peak, you can increase its weight to emphasize it more in the alignment algorithm.

`... = msalign(..., 'Range', RangeValue, ...)` specifies the lower and upper limits of the range, in m/z units, relative to each peak. No peak will shift beyond these limits. *RangeValue* is a two-element vector, in which the first element is negative and the second element is positive. Default is `[-100 100]`.

Note Use these values to tune the robustness of the algorithm. Ideally, you should keep the range within the maximum expected shift. If you try to correct larger shifts by increasing the limits, you increase the possibility of picking incorrect peaks to align to the reference masses.

`... = msalign(..., 'WidthOfPulses', WidthOfPulsesValue, ...)` specifies the width, in m/z units, for all the Gaussian pulses used to build the correlating synthetic spectrum. The point of the peak where the Gaussian pulse reaches 60.65% of its maximum is set to the width specified by *WidthOfPulsesValue*. Choices are any positive value. Default is 10. *WidthOfPulsesValue* may also be a function handle. The function is evaluated at the respective m/z values and returns a variable width for the pulses. Its evaluation should give reasonable values between 0 and `max(abs(Range))`; otherwise, the function returns an error.

Note Tuning the spread of the Gaussian pulses controls a tradeoff between robustness (wider pulses) and precision (narrower pulses). However, the spread of the pulses is unrelated to the shape of the observed peaks in the spectrum. The purpose of the pulse spread is to drive the optimization algorithm.

`... = msalign(..., 'WindowSizeRatio', WindowSizeRatioValue, ...)` specifies a scaling factor that determines the size of the window around every alignment peak. The synthetic spectrum is compared to the sample spectrum only within these regions, which saves computation time. The size of the window is given in m/z units by `WidthOfPulsesValue * WindowSizeRatioValue`. Choices are any positive value. Default is 2.5, which means at the limits of the window, the Gaussian pulses have a value of 4.39% of their maximum.

`... = msalign(..., 'Iterations', IterationsValue, ...)`
specifies the number of refining iterations. At every iteration, the search grid is scaled down to improve the estimates. Choices are any positive integer. Default is 5.

`... = msalign(..., 'GridSteps', GridStepsValue, ...)`
specifies the number of steps for the search grid. At every iteration, the search area is divided by GridStepsValue^2 . Choices are any positive integer. Default is 20.

`... = msalign(..., 'SearchSpace', SearchSpaceValue, ...)`
specifies the type of search space. Choices are:

- 'regular' — Default. Evenly spaced lattice.
- 'latin' — Random Latin hypercube with GridStepsValue^2 samples.

`... = msalign(..., 'ShowPlot', ShowPlotValue, ...)` controls the display of a plot of an original and aligned spectrum over the reference masses specified by *RefMZ*. Choices are `true`, `false`, or *I*, an integer specifying the index of a spectrum in *Intensities*. If set to `true`, the first spectrum in *Intensities* is plotted. Default is:

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

`[IntensitiesOut, RefMZOut] = msalign(..., 'Group', GroupValue, ...)` controls the creation of *RefMZOut*, a new vector of m/z values to be used as reference masses for aligning the peaks. This vector is created by adjusting the values in *RefMZ*, based on the sample data from multiple spectra in *Intensities*, such that the overall shifting and scaling of the peaks is minimized. Choices are `true` or `false` (default).

Tip Set *GroupValue* to true only if *Intensities* contains data for a large number of spectra, and you are not confident of the m/z values used for your reference peaks in *RefMZ*. Leave *GroupValue* set to false if you are confident of the m/z values used for your reference peaks in *RefMZ*.

Examples

Aligning Mass Spectrum with Three or More Reference Peaks

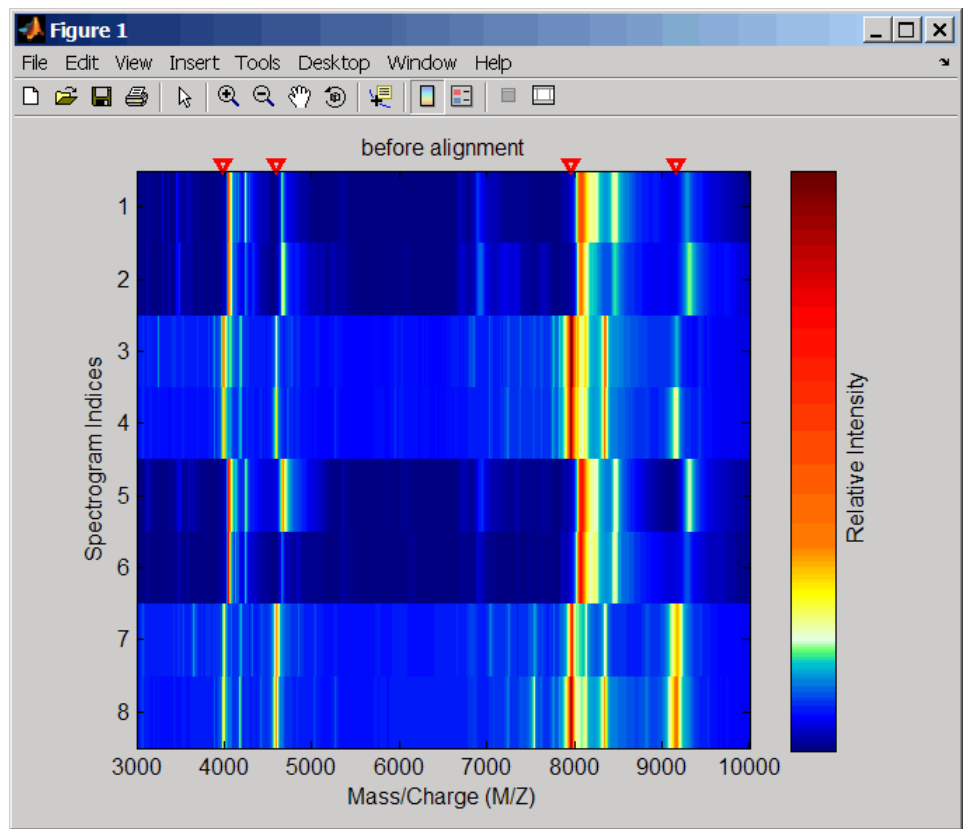
- 1 Load sample data, reference masses, and parameter data for synthetic peak width.

```
load sample_lo_res
R = [3991.4 4598 7964 9160];
W = [60 100 60 100];
```

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

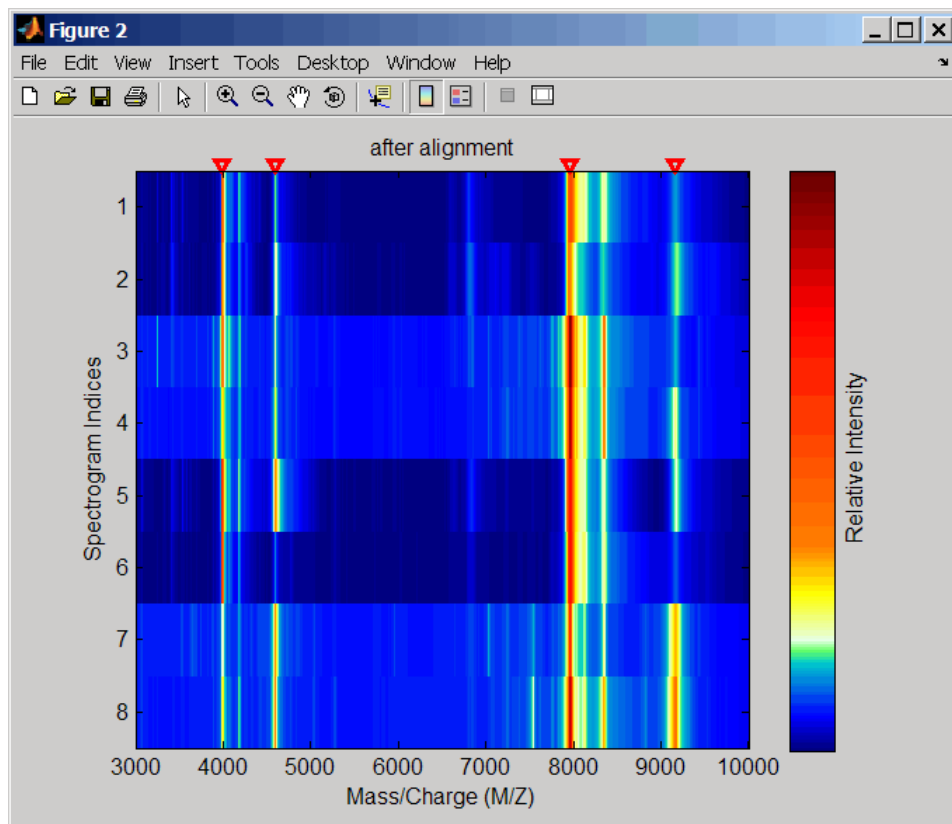
```
msheatmap(MZ_lo_res,Y_lo_res,'markers',R,'range',[3000 10000])
title('before alignment')
```

msalign



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

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

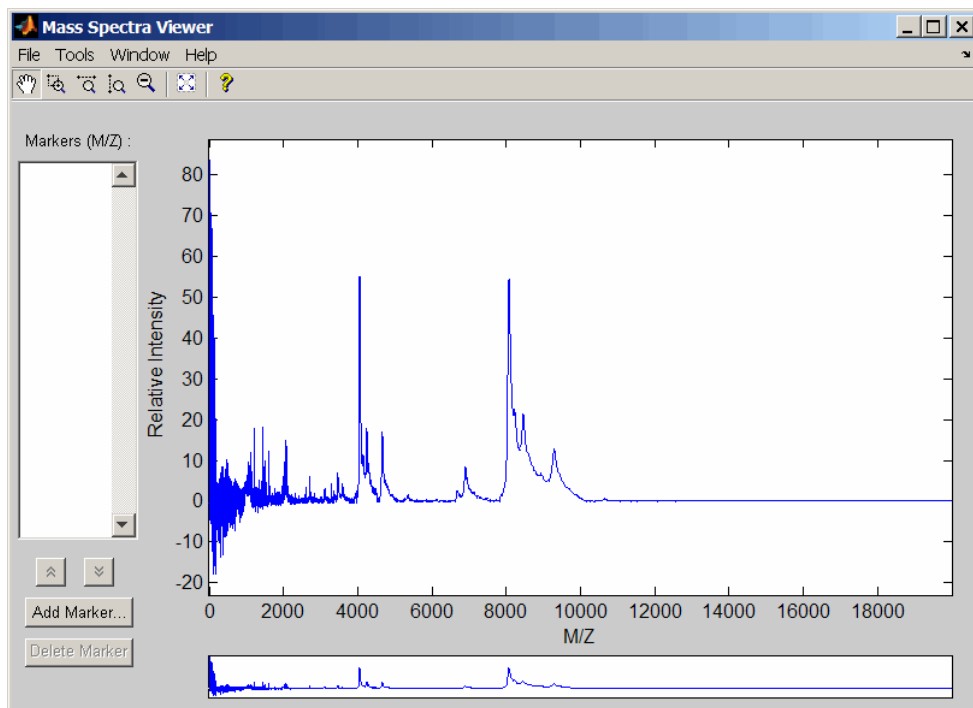
Aligning Mass Spectrum with One Reference Peak


It is not recommended to use the `msalign` function if you have only one reference peak. Instead, use the following procedure, which shifts the `MZ` vector, but does not scale it.

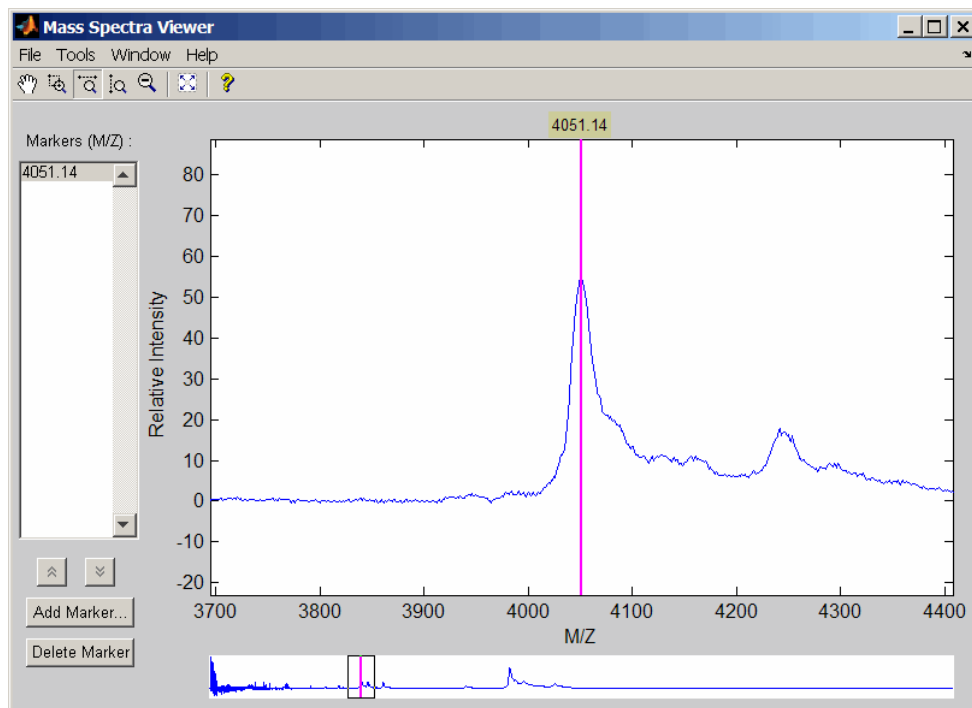
- 1 Load sample data and view the first sample spectrum.

```
load sample_lo_res
MZ = MZ_lo_res;
Y = Y_lo_res(:,1);
```

msviewer(MZ, Y)



- 2 Use the tall peak around 4000 m/z as the reference peak. To determine the reference peak's m/z value, click , and then click-drag to zoom in on the peak. Right-click in the center of the peak, and then click **Add Marker** to label the peak with its m/z value.



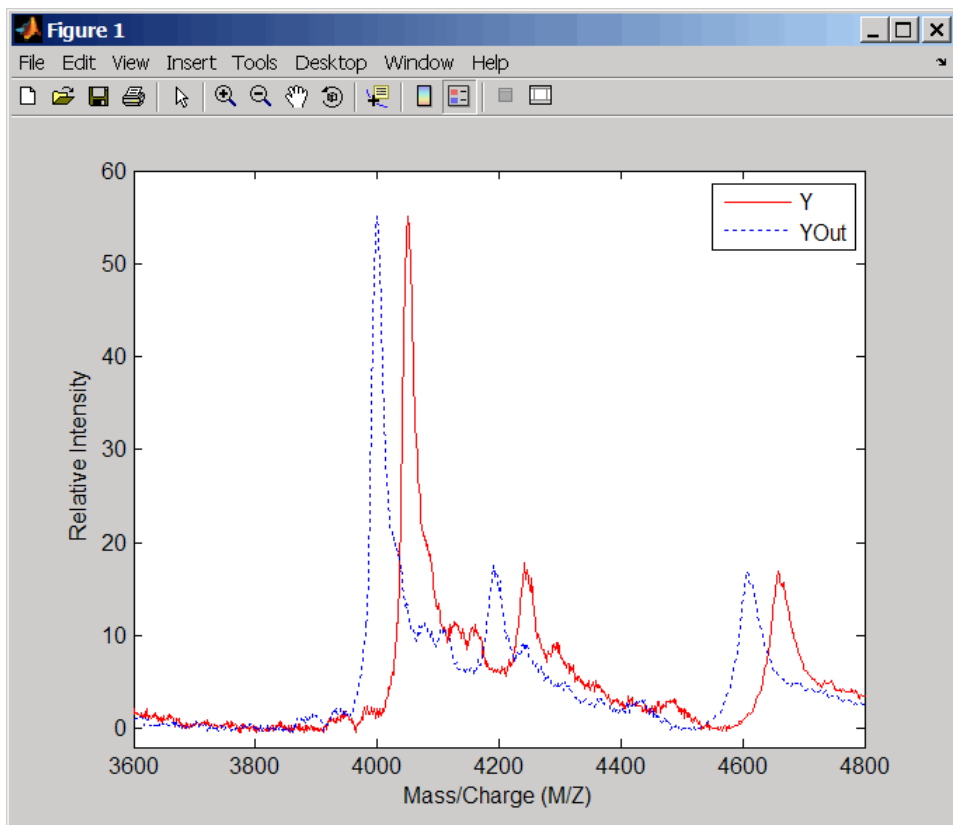
- 3** Shift a spectrum by the difference between RP, the known reference mass of 4000 m/z, and SP, the experimental mass of 4051.14 m/z.

```
RP = 4000;  
SP = 4051.14;  
YOut = interp1(MZ, Y, MZ - (RP - SP));
```

- 4** Plot the original spectrum in red and the shifted spectrum in blue and zoom in on the reference peak.

```
plot(MZ, Y, 'r', MZ, YOut, 'b:');  
xlabel('Mass/Charge (M/Z)');  
ylabel('Relative Intensity');
```

```
legend('Y','YOut')  
axis([3600 4800 -2 60])
```



References

[1] Monchamp, P., Andrade-Cetto, L., Zhang, J.Y., and Henson, R. (2007) Signal Processing Methods for Mass Spectrometry. In Systems Bioinformatics: An Engineering Case-Based Approach, G. Alterovitz and M.F. Ramoni, eds. (Artech House Publishers).

See Also

Bioinformatics Toolbox functions: msbackadj, msheatmap, mspalign, mspeaks, msresample, msviewer

Purpose

Correct baseline of mass spectrum

Syntax

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

Arguments

<i>MZ</i>	Range of mass/charge ions. Enter a vector with the range of ions in the spectra.
<i>Y</i>	Ion intensity vector with the same length as the mass/charge vector (<i>MZ</i>). <i>Y</i> can also be a matrix with several spectra that share the same mass/charge (<i>MZ</i>) range.

Description

`Yout = msbackadj(MZ, Y)` adjusts the variable baseline of a raw mass spectrum by following three steps:

- 1** Estimates the baseline within multiple shifted windows of width 200 m/z
- 2** Regresses the varying baseline to the window points using a spline approximation

3 Adjusts the baseline of the spectrum (Y)

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

`Yout = msbackadj(MZ, Y, ...'WindowSize', WindowSizeValue, ...)` specifies the width for the shifting window. *WindowSizeValue* can also be a function handle. The function is evaluated at the respective MZ values and returns a variable width for the windows. This option is useful for cases where the resolution of the signal is dissimilar at different regions of the spectrogram. The default value is 200 (baseline point estimated for windows with a width of 200 m/z).

Note The result of this algorithm depends on carefully choosing the window size and the step size. Consider the width of your peaks in the spectrum and the presence of possible drifts. If you have wider peaks toward the end of the spectrum, you may want to use variable parameters.

`Yout = msbackadj(MZ, Y, ...'StepSize', StepSizeValue, ...)` specifies the steps for the shifting window. The default value is 200 m/z (baseline point is estimated for windows placed every 200 m/z). *StepSizeValue* may also be a function handle. The function is evaluated at the respective m/z values and returns the distance between adjacent windows.

`Yout = msbackadj(MZ, Y, ...'RegressionMethod', RegressionMethodValue, ...)` specifies the method to regress the window estimated points to a soft curve. Enter 'pchip' (shape-preserving piecewise cubic interpolation), 'linear' (linear interpolation), or 'spline' (spline interpolation). The default value is 'pchip'.

`Yout = msbackadj(MZ, Y, ...'EstimationMethod', EstimationMethodValue, ...)` specifies the method for finding the likely baseline value in every window. Enter 'quantile' (quantile value is set to 10%) or 'em' (assumes a doubly stochastic model). With em, every sample is the independent and identically distributed (i.i.d.) draw of any of two normal distributed classes (background or peaks). Because the class label is hidden, the distributions are estimated with an Expectation-Maximization algorithm. The ultimate baseline value is the mean of the background class.

`Yout = msbackadj(MZ, Y, ...'SmoothMethod', SmoothMethodValue, ...)` specifies the method for smoothing the curve of estimated points and eliminating the effects of possible outliers. Enter 'none', 'lowess' (linear fit), 'loess' (quadratic fit), 'rloess' (robust linear), or 'rloess' (robust quadratic fit). Default is 'none'.

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

`Yout = msbackadj(MZ, Y, ...'PreserveHeights', PreserveHeightsValue, ...)`, when *PreserveHeightsValue* is true, sets the baseline subtraction mode to preserve the height of the tallest peak in the signal. The default value is false and peak heights are not preserved.

`Yout = msbackadj(MZ, Y, ...'ShowPlot', ShowPlotValue, ...)` plots the baseline estimated points, the regressed baseline, and the original spectrum. When `msbackadj` is called without output arguments, the spectra are plotted unless *ShowPlotValue* is false. When *ShowPlotValue* is true, only the first spectrum in *Y* is plotted. *ShowPlotValue* can also contain an index to one of the spectra in *Y*.

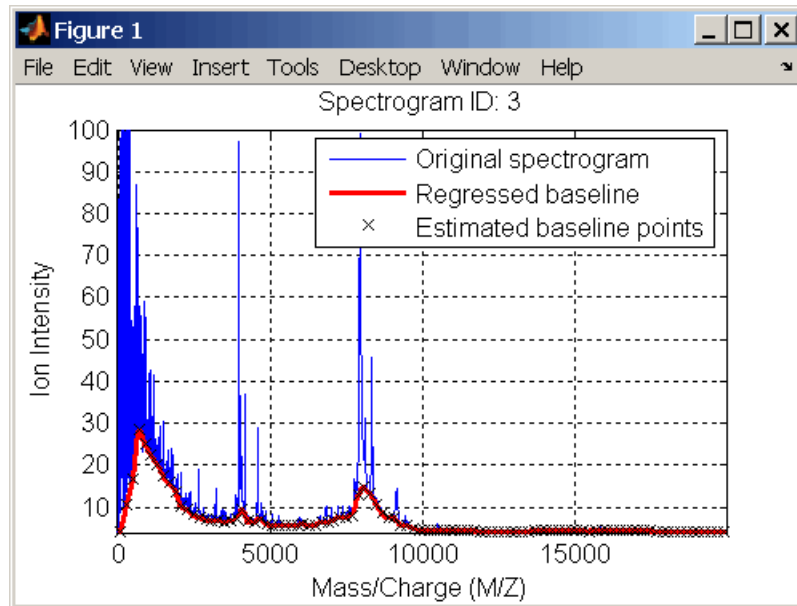
Examples

- 1 Load sample data.

```
load sample_lo_res
```

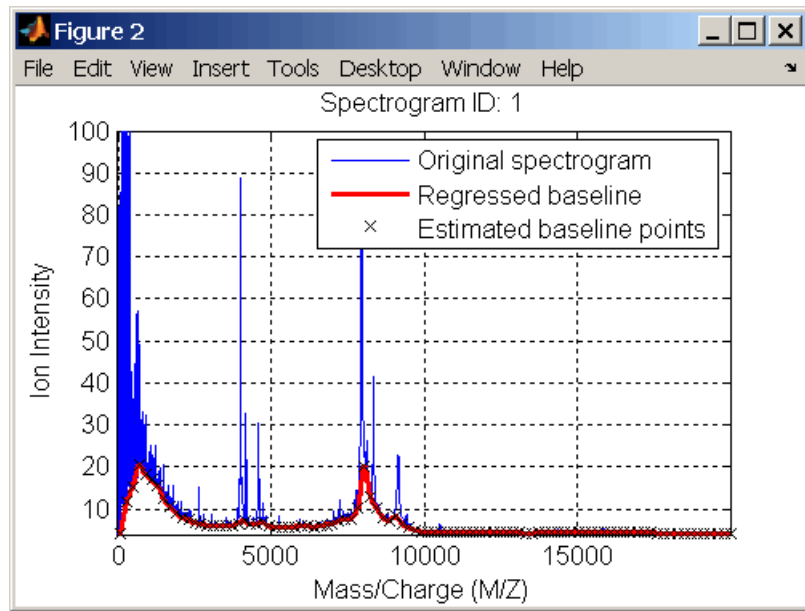
- 2 Adjust the baseline for a group of spectra and show only the third spectrum and its estimated background.

```
YB = msbackadj(MZ_lo_res,Y_lo_res,'SHOWPLOT',3);
```



- 3** Plot the estimated baseline for the fourth spectrum in `Y_lo_res` using an anonymous function to describe an `m/z` dependent parameter.

```
wf = @(mz) 200 + .001 .* mz;  
msbackadj(MZ_lo_res,Y_lo_res(:,4),'STEPsize',wf);
```


**See Also**

Bioinformatics Toolbox functions: `msalign`, `mslowess`, `msheatmap`, `msnorm`, `mspeaks`, `msresample`, `mssgolay`, `msviewer`

msdotplot

Purpose

Plot set of peak lists from LC/MS or GC/MS data set

Syntax

```
msdotplot(Peaks, Times)  
msdotplot(FigHandle, Peaks, Times)  
msdotplot(..., 'Quantile', QuantileValue)  
PlotHandle = msdotplot(...)
```

Arguments

Peaks Cell array of peak lists, where each element is a two-column matrix with m/z values in the first column and ion intensity values in the second column. Each element corresponds to a spectrum or retention time.

Tip You can use the `mzxml2peaks` function to create the *Peaks* cell array.

Times Vector of retention times associated with an LC/MS or GC/MS data set. The number of elements in *Times* equals the number of elements in the cell array *Peaks*.

Tip You can use the `mzxml2peaks` function to create the *Times* vector.

FigHandle Handle to an open Figure window such as one created by the `msheatmap` function.

QuantileValue Value that specifies a percentage. When peaks are ranked by intensity, only those that rank above this percentage are plotted. Choices are any value ≥ 0 and ≤ 1 . Default is 0. For example, setting *QuantileValue* = 0 plots all peaks, and setting *QuantileValue* = 0.8 plots only the 20% most intense peaks.

Return Values

PlotHandle Handle to the line series object (figure plot).

Description

`msdotplot(Peaks, Times)` plots a set of peak lists from a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set represented by *Peaks*, a cell array of peak lists, where each element is a two-column matrix with *m/z* values in the first column and ion intensity values in the second column, and *Times*, a vector of retention times associated with the spectra. *Peaks* and *Times* have the same number of elements. The data is plotted into any existing figure generated by the `msheatmap` function; otherwise, the data is plotted into a new Figure window.

`msdotplot(FigHandle, Peaks, Times)` plots the set of peak lists into the axes contained in an open Figure window with the handle *FigHandle*.

Tip This syntax is useful to overlay a dot plot on top of a heat map of mass spectrometry data created with the `msheatmap` function.

`msdotplot(..., 'Quantile', QuantileValue)` plots only the most intense peaks, specifically those in the percentage above the specified *QuantileValue*. Choices are any value ≥ 0 and ≤ 1 . Default is 0. For example, setting *QuantileValue* = 0 plots all peaks, and setting *QuantileValue* = 0.8 plots only the 20% most intense peaks.

PlotHandle = `msdotplot(...)` returns a handle to the line series object (figure plot). You can use this handle as input to the `get` function to display a list of the plot's properties. You can use this handle as input to the `set` function to change the plot's properties, including showing and hiding points.

Examples

- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains LC/MS data variables, including `peaks` and `ret_time`. `peaks` is a cell array of peak lists, where each element is a two-column

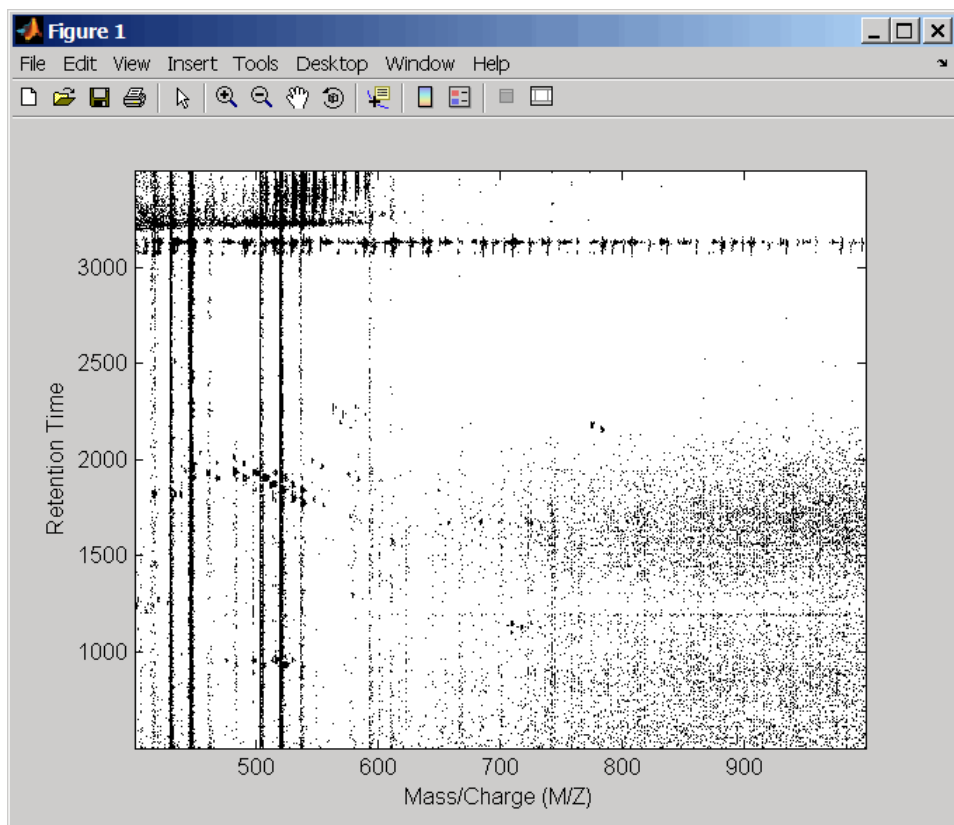
msdotplot

matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time. `ret_time` is a column vector of retention times associated with the LC/MS data set.

```
load lcmsdata
```

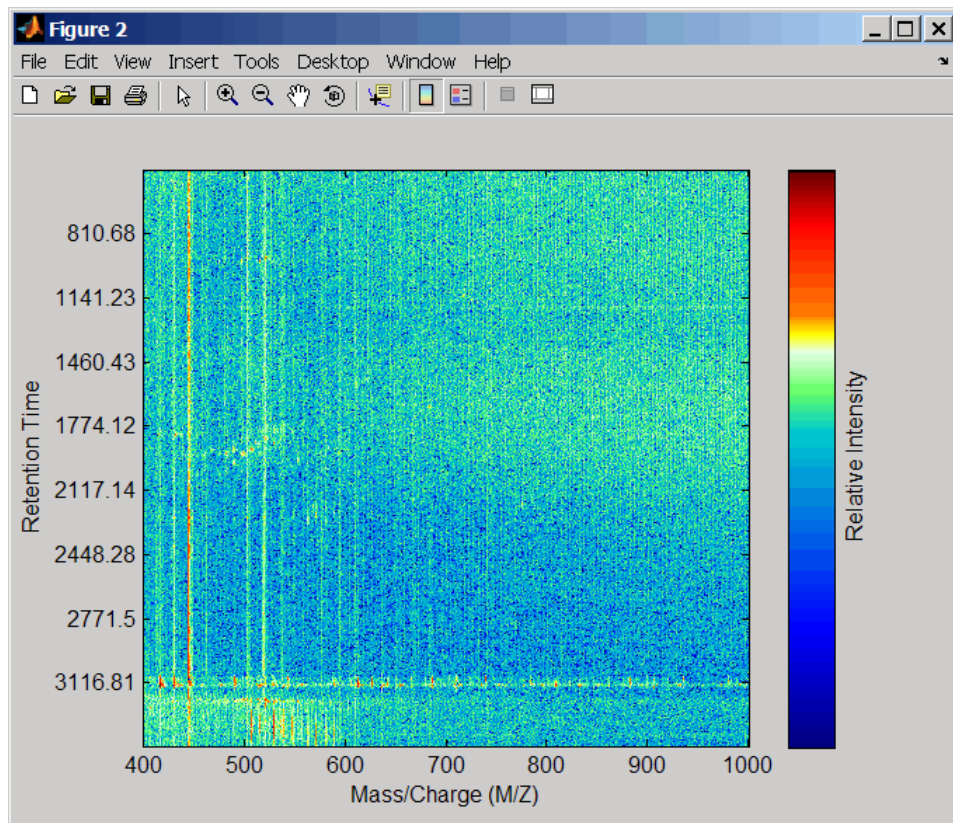
- 2 Create a dot plot with only the 5% most intense peaks.

```
msdotplot(peaks,ret_time,'Quantile',0.95)
```



- 3 Resample the data, then create a heat map of the LC/MS data.

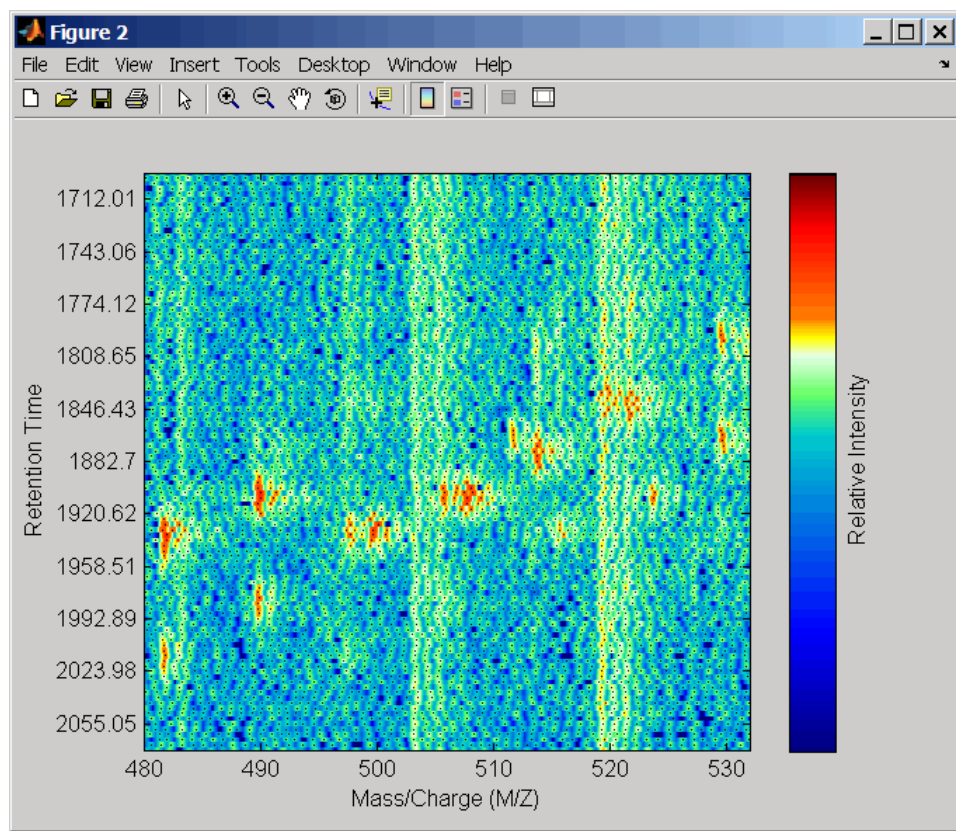
```
[MZ,Y] = mspresample(peaks,5000);  
msheatmap(MZ,ret_time,log(Y))
```



- 4 Overlay the dot plot on the heat map, and then zoom in to see the detail.

```
msdotplot(peaks,ret_time)  
axis([480 532 375 485])
```

msdotplot



See Also

Bioinformatics Toolbox functions: `msheatmap`, `mssalign`, `mspeaks`, `msspresample`, `mzcdf2peaks`, `mzcdfread`, `mzxml2peaks`, `mzxmlread`

Purpose

Create pseudocolor image of set of mass spectra

Syntax

```
msheatmap(MZ, Intensities)
msheatmap(MZ, Times, Intensities)
msheatmap(..., 'Midpoint', MidpointValue, ...)
msheatmap(..., 'Range', RangeValue, ...)
msheatmap(..., 'Markers', MarkersValue, ...)
msheatmap(..., 'SpecIdx', SpecIdxValue, ...)
msheatmap(..., 'Group', GroupValue, ...)
msheatmap(..., 'Resolution', ResolutionValue, ...)
```

Arguments

MZ

Column vector of common mass/charge (m/z) values for a set of spectra. The number of elements in the vector equals the number of rows in the matrix *Intensities*.

Note You can use the `mppresample` function to create the *MZ* vector.

Times

Column vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. The number of elements in the vector equals the number of columns in the matrix *Intensities*. The retention times are used to label the y-axis of the heat map.

Tip You can use the `mzxm12peaks` function to create the *Times* vector.

Intensities

Matrix of intensity values for a set of mass spectra that share the same m/z range. Each row corresponds to an m/z value, and each column corresponds to a spectrum or retention time. The number of rows equals the number of elements in vector *MZ*. The number of columns equals the number of elements in vector *Times*.

Note You can use the `msppresample` function to create the *Intensities* matrix.

- MidpointValue* Value specifying a quantile of the ion intensity values to fall below the midpoint of the colormap, meaning they do not represent peaks. `msheatmap` uses a custom colormap where cool colors represent nonpeak regions, white represents the midpoint, and warm colors represent peaks. Choices are any value ≥ 0 and ≤ 1 . Default is:
- 0.99 — For LC/MS or GC/MS data or when input *T* is provided. This means that 1% of the pixels are warm colors and represent peaks.
 - 0.95 — For non-LC/MS or non-GC/MS data or when input *T* is not provided. This means that 5% of the pixels are warm colors and represent peaks.

Tip You can also change the midpoint interactively after creating the heat map by right-clicking the color bar, selecting **Interactive Colormap Shift**, and then click-dragging the cursor vertically on the color bar. This technique is useful when comparing multiple heat maps.

- RangeValue* 1-by-2 vector specifying the m/z range for the x-axis of the heat map. *RangeValue* must be within $[\min(MZ) \max(MZ)]$. Default is the full range $[\min(MZ) \max(MZ)]$.
- MarkersValue* Vector of m/z values to mark on the top horizontal axis of the heat map. Default is `[]`.

SpecIdxValue

Either of the following:

- Vector of values with the same number of elements as columns (spectra) in the matrix *Intensities*.
- Cell array of strings with the same number of elements as columns (spectra) in the matrix *Intensities*.

Each value or string specifies a label for the corresponding spectrum. These values or strings are used to label the *y*-axis of the heat map.

Note If input *Times* is provided, it is assumed that *Intensities* contains LC/MS or GC/MS data, and *SpecIdxValue* is ignored.

- GroupValue*
- Either of the following:
- Vector of values with the same number of elements as rows in the matrix *Intensities*
 - Cell array of strings with the same number of elements as rows (spectra) in the matrix *Intensities*

Each value or string specifies a group to which the corresponding spectrum belongs. The spectra are sorted and combined into groups along the *y*-axis in the heat map.

Note If input *Times* is provided, it is assumed that *Intensities* contains LC/MS or GC/MS data, and *GroupValue* is ignored.

- ResolutionValue* Value specifying the horizontal resolution of the heat map image. Increase this value to enhance details. Decrease this value to reduce memory usage. Default is:
- 0.5 — When *MZ* contains > 2,500 elements.
 - 0.05 — When *MZ* contains ≤ 2,500 elements.

Description

`msheatmap(MZ, Intensities)` displays a pseudocolor heat map image of the intensities for the spectra in matrix *Intensities*.

`msheatmap(MZ, Times, Intensities)` displays a pseudocolor heat map image of the intensities for the spectra in matrix *Intensities*, using the retention times in vector *Times* to label the *y*-axis.

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

msheatmap

`msheatmap(..., 'Midpoint', MidpointValue, ...)` specifies a quantile of the ion intensity values to fall below the midpoint of the colormap, meaning they do not represent peaks. `msheatmap` uses a custom colormap where cool colors represent nonpeak regions, white represents the midpoint, and warm colors represent peaks. Choices are any value between 0 and 1. Default is:

- 0.99 — For LC/MS or GC/MS data or when input *T* is provided. This means that 1% of the pixels are warm colors and represent peaks.
- 0.95 — For non-LC/MS or non-GC/MS data or when input *T* is not provided. This means that 5% of the pixels are warm colors and represent peaks.

Tip You can also change the midpoint interactively after creating the heat map by right-clicking the color bar, selecting **Interactive Colormap Shift**, then click-dragging the cursor vertically on the color bar. This technique is useful when comparing multiple heat maps.

`msheatmap(..., 'Range', RangeValue, ...)` specifies the m/z range for the x-axis of the heat map. *RangeValue* is a 1-by-2 vector that must be within $[\min(MZ) \max(MZ)]$. Default is the full range $[\min(MZ) \max(MZ)]$.

`msheatmap(..., 'Markers', MarkersValue, ...)` places markers along the top horizontal axis of the heat map for the m/z values specified in the vector *MarkersValue*. Default is `[]`.

`msheatmap(..., 'SpecIdx', SpecIdxValue, ...)` labels the spectra along the y-axis in the heat map. The labels are specified by *SpecIdxValue*, a vector of values or cell array of strings. The number of values or strings is the same as the number of columns (spectra) in the matrix *Intensities*. Each value or string specifies a label for the corresponding spectrum.

`msheatmap(..., 'Group', GroupValue, ...)` sorts and combines spectra into groups along the y-axis in the heat map. The groups are

specified by *GroupValue*, a vector of values or cell array of strings. The number of values or strings is the same as the number of rows in the matrix *Intensities*. Each value or string specifies a group to which the corresponding spectrum belongs.

`msheatmap(..., 'Resolution', ResolutionValue, ...)` specifies the horizontal resolution of the heat map image. Increase this value to enhance details. Decrease this value to reduce memory usage. Default is:

- 0.5 — When *MZ* contains > 2,500 elements.
- 0.05 — When *MZ* contains <= 2,500 elements.

Examples

SELDI-TOF Data

- 1 Load SELDI-TOF sample data.

```
load sample_lo_res
```

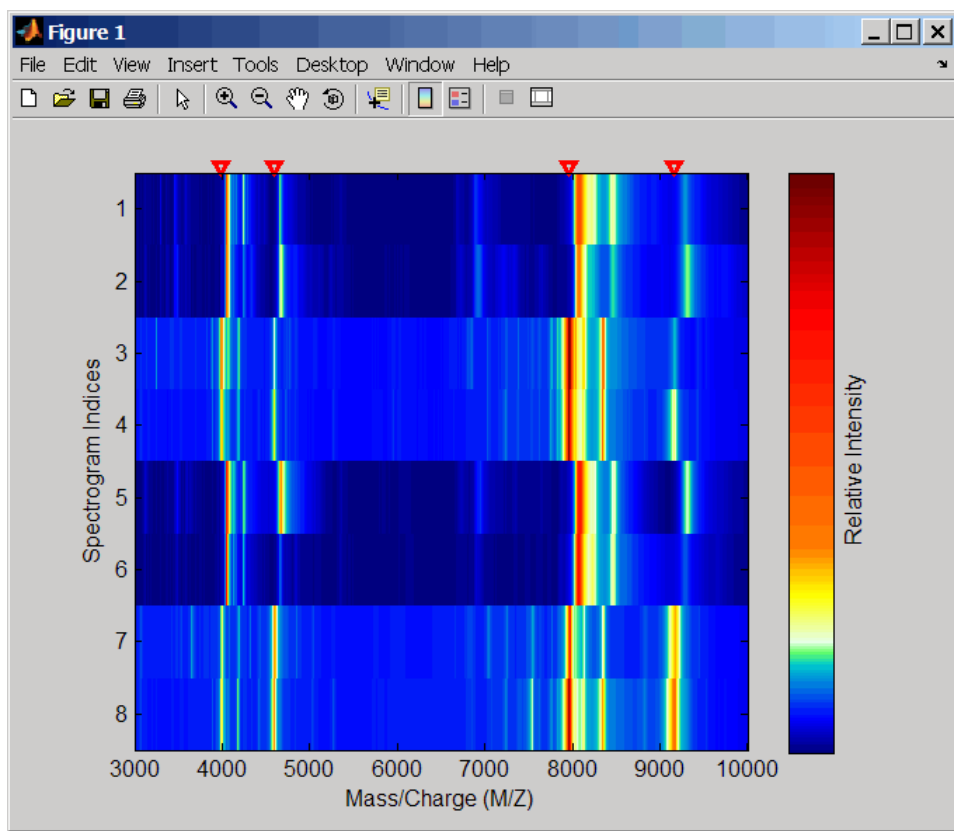
- 2 Create a vector of four m/z values to mark along the top horizontal axis of the heat map.

```
M = [3991.4 4598 7964 9160];
```

- 3 Display the heat map with m/z markers and a limited m/z range.

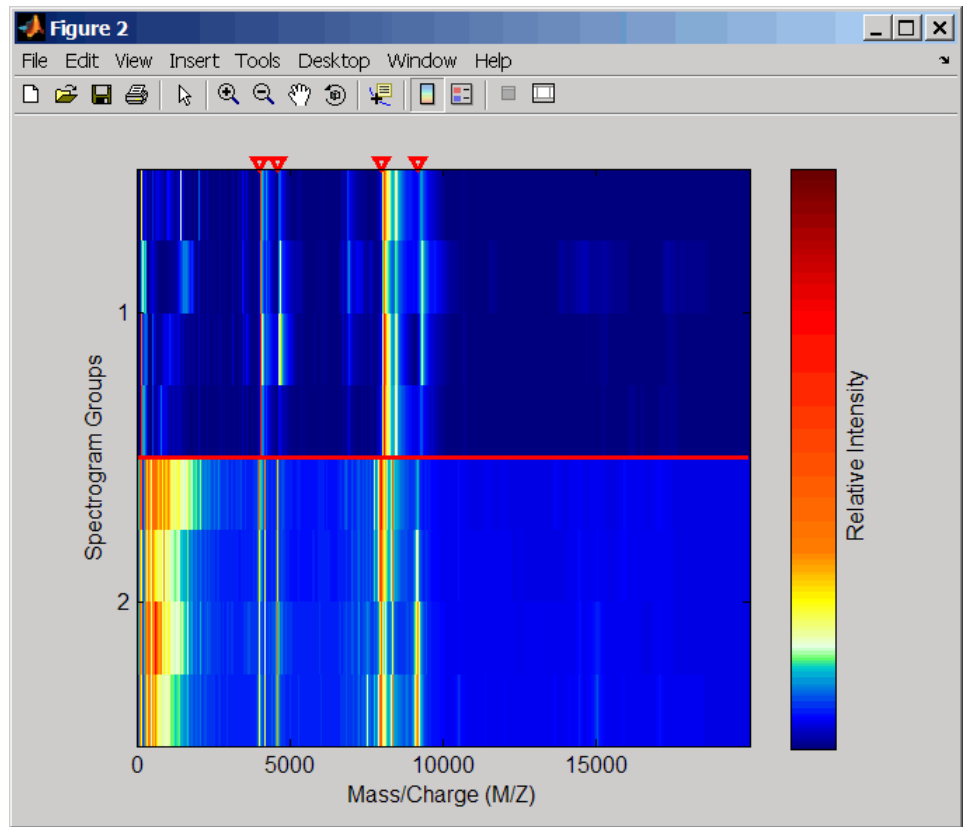
```
msheatmap(MZ_lo_res,Y_lo_res,'markers',M,'range',[3000 10000])
```

msheatmap



- 4** Display the heat map again grouping each spectrum into one of two groups.

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



Liquid Chromatography/Mass Spectrometry (LC/MS) Data

- 1 Load LC/MS sample data.

```
load lcmsdata
```

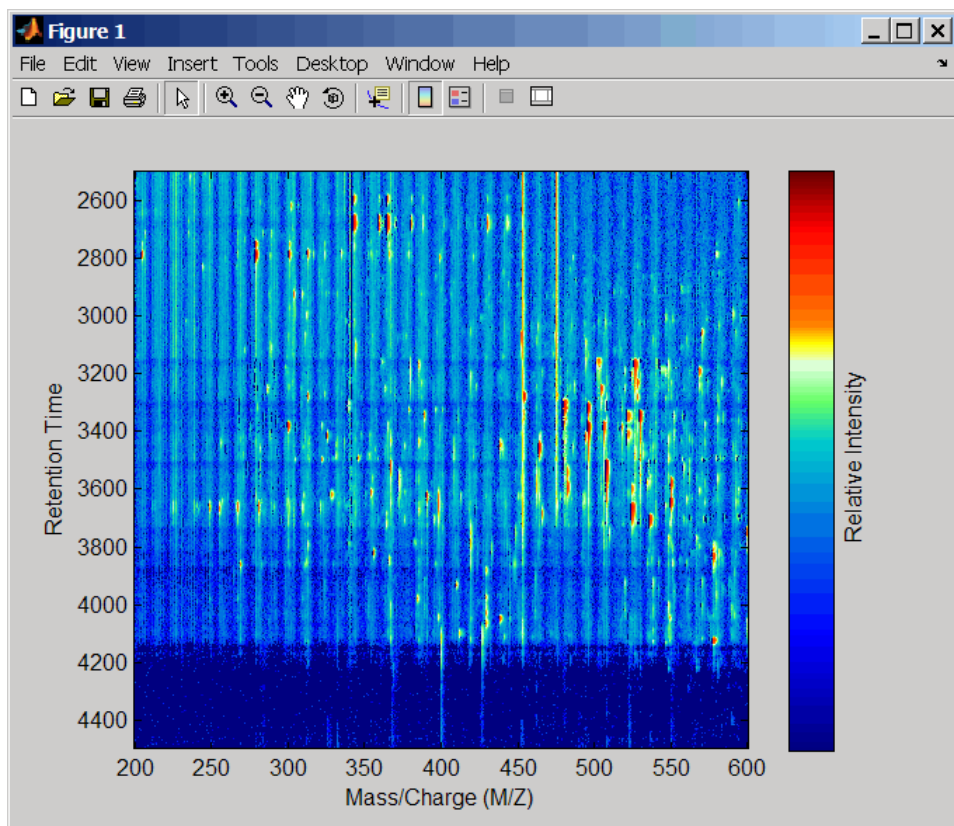
- 2 Resample the peak lists to create a vector of m/z values and a matrix of intensity values.

```
[MZ, Intensities] = mspresample(peaks, 5000);
```

msheatmap

- 3 Display the heat map showing mass spectra at different retention times.

```
msheatmap(MZ, ret_time, log(Intensities))
```



See Also

Bioinformatics Toolbox functions: `msalign`, `msbackadj`, `msdotplot`, `mslowess`, `msnorm`, `mssalign`, `msresample`, `mssgolay`, `msviewer`

Purpose

Smooth mass spectrum using nonparametric method

Syntax

```
Yout = mslowess(MZ, Y, 'PropertyName', PropertyValue...)
mslowess(..., 'Order', OrderValue)
mslowess(..., 'Span', SpanValue)
mslowess(..., 'Kernel', KernelValue)
mslowess(..., 'RobustIterations', RobustIterationsValue)
mslowess(..., 'ShowPlot', ShowPlotValue)
```

Arguments

MZ	Mass/charge vector with the range of ions in the spectra.
Y	Ion intensity vector with the same length as the mass/charge vector (MZ). Y can also be a matrix with several spectra that share the same mass/charge (MZ) range.

Description

`Yout = mslowess(MZ, Y, 'PropertyName', PropertyValue...)` smoothes a mass spectrum (Y) using a locally weighted linear regression (lowess) method with a default span of 10 samples.

Note 1) `mslowess` assumes that a mass/charge vector (MZ) might not be uniformly spaced. Therefore, the sliding window for smoothing is centered using the closest samples in terms of the MZ value and not in terms of the MZ indices.

2) When the vector MZ does not have repeated values or NaNs, the algorithm is approximately twice as fast.

`mslowess(..., 'Order', OrderValue)` specifies the order (*OrderValue*) of the Lowess smoother. Enter 1 (linear polynomial fit or Lowess), 2 (quadratic polynomial fit or Loess), or 0 (equivalent to a weighted local mean estimator and presumably faster because only a

mean computation is performed instead of a least squares regression). The default value is 1.

Note Curve Fitting Toolbox software also refers to Lowess smoothing of order 2 as Loess smoothing.

`mslowess(..., 'Span', SpanValue)` specifies the window size for the smoothing kernel. If *SpanValue* is greater than 1, the window is equal to *SpanValue* number of samples independent of the 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)	$(1 - (\text{dist}/\text{dmax})^3)^3$
'gaussian'	$\exp(-2*\text{dist}/\text{dmax})^2$
'linear'	$1 - \text{dist}/\text{dmax}$

`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*.

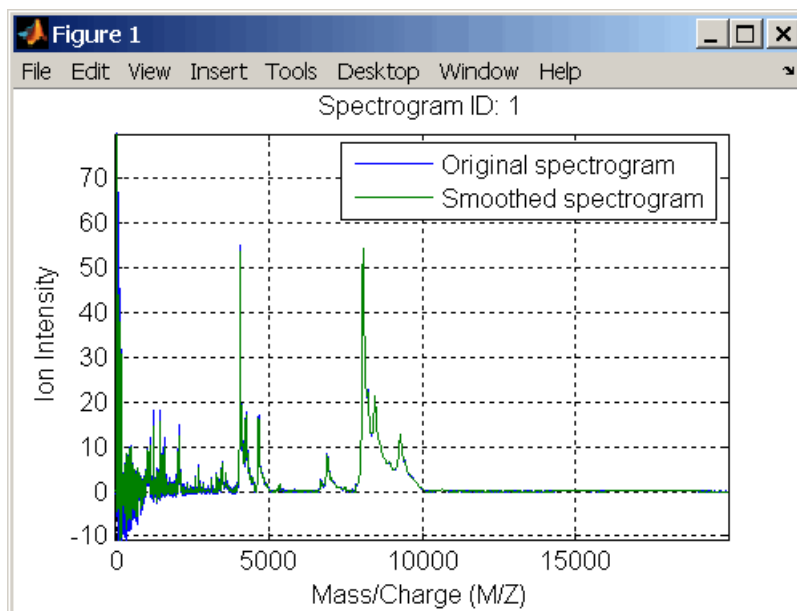
Example

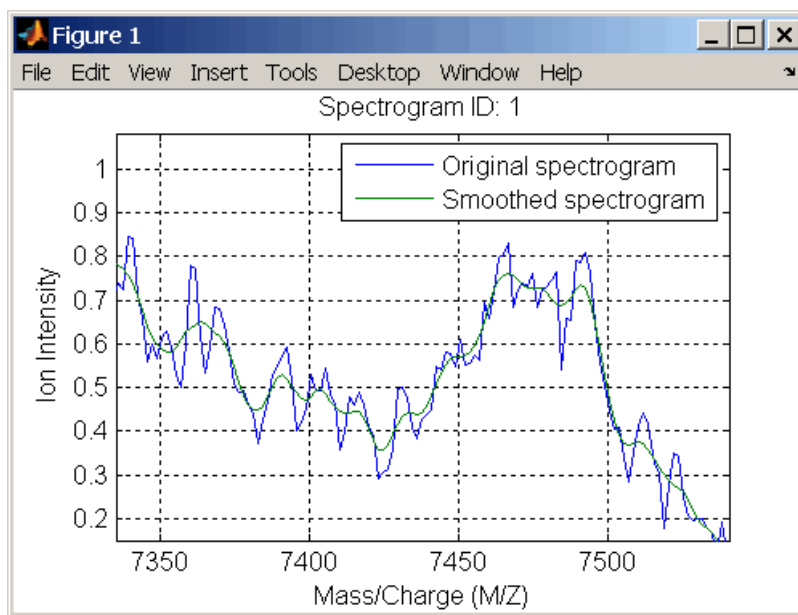
- 1 Load sample data.

```
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);
```





See Also

Bioinformatics Toolbox functions: `msalign`, `msbackadj`, `msheatmap`, `msheatmap`, `msnorm`, `mspeaks`, `msresample`, `mssgolay`, `msviewer`

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

<i>MZ</i>	Mass/charge vector with the range of ions in the spectra.
<i>Y</i>	Ion intensity vector with the same length as the mass/charge vector (<i>MZ</i>). <i>Y</i> can also be a matrix with several spectra that share the same mass/charge (<i>MZ</i>) range.

Description

Yout = msnorm(*MZ*, *Y*) normalizes a group of mass spectra by standardizing the area under the curve (AUC) to the group median.

[*Yout*, *NormParameters*] = msnorm(...) returns a structure with the parameters to normalize another group of spectra.

msnorm(*MZ*, *NewY*, *NormParameters*) uses the parameter information from a previous normalization (*NormParameters*) to normalize a new set of spectra (*NewY*) with the *MZ* positions and output scale from the previous normalization. *NormParameters* is a structure created by msnorm. If a consensus proportion (*ConsensusValue*) was given in the previous normalization, no new *MZ* positions are selected, and normalization is performed using the same *MZ* positions.

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

`msnorm(..., 'Quantile', QuantileValue)` specifies a 1-by-2 vector with the quantile limits for reducing the set of MZ values. For example, when *QuantileValue* is `[0.9 1]`, only the largest 10% of ion intensities in every spectrum are used to compute the AUC. When *QuantileValue* is a scalar, the scalar value represents the lower quantile limit and the upper quantile limit is set to 1. The default value is `[0 1]` (use the whole area under the curve, AUC).

`msnorm(..., 'Limits', LimitsValue)` specifies a 1-by-2 vector with an MZ range for picking normalization points. This parameter is useful to eliminate low-mass noise from the AUC calculation. The default value is `[1, max(MZ)]`.

`msnorm(..., 'Consensus', ConsensusValue)` selects MZ positions with a consensus rule to include an MZ position into the AUC. Its ion intensity must be within the quantile limits of at least part (*ConsensusValue*) of the spectra in *Y*. The same MZ positions are used to normalize all the spectrums. Enter a scalar between 0 and 1.

Use the `Consensus` property to eliminate low-intensity peaks and noise from the normalization.

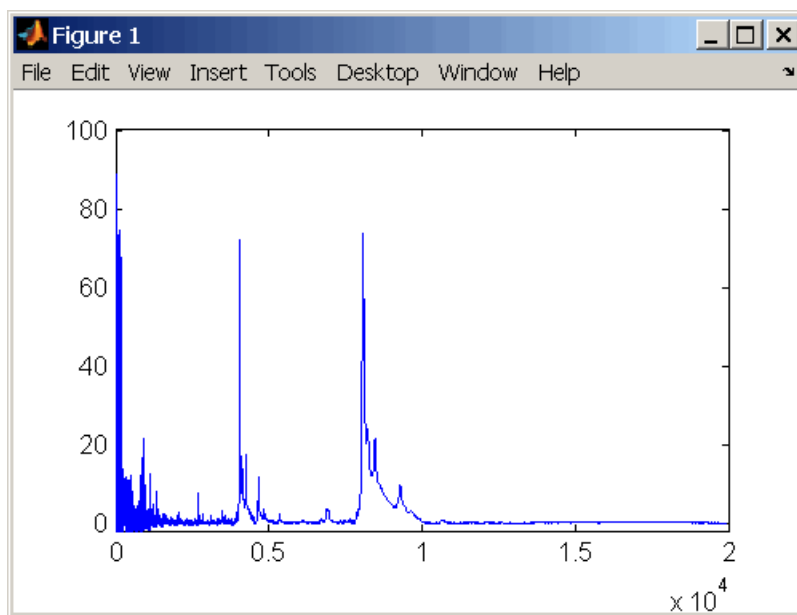
`msnorm(..., 'Method', MethodValue)` selects a method for normalizing the AUC of every spectrum. Enter either `'Median'` (default) or `'Mean'`.

`msnorm(..., 'Max', MaxValue)`, after individually normalizing every spectrum, scales each spectrum to an overall maximum intensity (`Max`). `Max` is a scalar. If omitted, no postscaling is performed. If *QuantileValue* is `[1 1]`, then a single point (peak height of the tallest peak) is normalized to `Max`.

Example 1

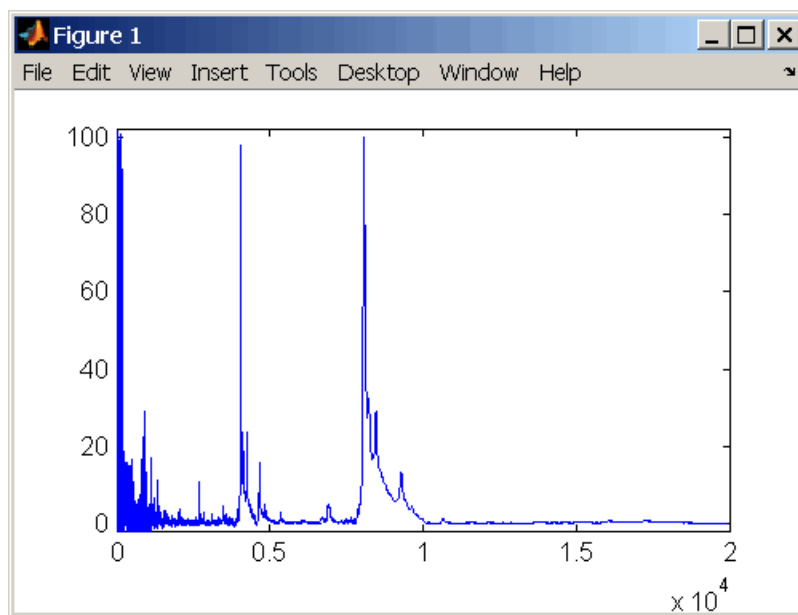
1 Load sample data and plot one of the spectra.

```
load sample_lo_res;
Y = Y_lo_res(:, [1 2 5 6]);
MZ = MZ_lo_res;
plot(MZ, Y(:, 4));
```



- 2 Normalize the AUC of every spectrum to its median, eliminating low-mass noise, and post-rescaling such that the maximum intensity is 100.

```
Y1 = msnorm(MZ,Y,'Limits',[1000 inf],'Max',100);  
plot(MZ, Y1(:, 4));
```



- 3 Normalize the ion intensity of every spectrum to the maximum intensity of the single highest peak from any of the spectra in the range above 100 m/z.

```
Y2 = msnorm(MZ,Y,'QUANTILE',[1 1],'LIMITS',[1000 inf]);
```

Example 2

- 1 Select MZ regions where the intensities are within the third quartile in at least 90% of the spectrograms.

```
[Y3,S] = msnorm(MZ,Y,'Quantile',[0.5 0.75],'Consensus',0.9);
```

- 2 Use the same MZ regions to normalize another set of spectrograms.

```
Y4 = msnorm(MZ,Y,S);
```

See Also

Bioinformatics Toolbox functions: `msalign`, `msbackadj`, `msheatmap`, `mslowess`, `msresample`, `mssgolay`, `msviewer`

Purpose

Align mass spectra from multiple peak lists from LC/MS or GC/MS data set

Syntax

```
[CMZ, AlignedPeaks] = msalign(Peaks)
[CMZ, AlignedPeaks] = msalign(Peaks, ...'Quantile',
QuantileValue, ...)
[CMZ, AlignedPeaks] = msalign(Peaks,
... 'EstimationMethod',
EstimationMethodValue, ...)
[CMZ, AlignedPeaks] = msalign(Peaks,
... 'CorrectionMethod',
CorrectionMethodValue, ...)
[CMZ, AlignedPeaks] = msalign(Peaks, ... 'ShowEstimation',
ShowEstimationValue, ...)
```

Arguments

Peaks

Cell array of peak lists from a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. Each element in the cell array is a two-column matrix with m/z values in the first column and ion intensity values in the second column. Each element corresponds to a spectrum or retention time.

Note You can use the `mzxm12peaks` function or the `mspeaks` function to create the *Peaks* cell array.

QuantileValue

Value that determines which peaks are selected by the estimation method to create *CMZ*, the vector of common m/z values. Choices are any value ≥ 0 and ≤ 1 . Default is 0.95.

EstimationMethodValue String specifying the method to estimate *CMZ*, the vector of common mass/charge (m/z) values. Choices are:

- **histogram** — Default method. Peak locations are clustered using a kernel density estimation approach. The peak ion intensity is used as a weighting factor. The center of all the clusters conform to the *CMZ* vector.
- **regression** — Takes a sample of the distances between observed significant peaks and regresses the inter-peak distance to create the *CMZ* vector with similar inter-element distances.

CorrectionMethodValue String specifying the method to align each peak list to the *CMZ* vector. Choices are:

- *nearest-neighbor* — Default method. For each common peak in the *CMZ* vector, its counterpart in each peak list is the peak that is closest to the common peak's *m/z* value.
- *shortest-path* — For each common peak in the *CMZ* vector, its counterpart in each peak list is selected using the shortest path algorithm.

ShowEstimationValue Controls the display of an assessment plot relative to the estimation method and the vector of common mass/charge (*m/z*) values. Choices are *true* or *false*. Default is either:

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

Return Values

CMZ Vector of common mass/charge (*m/z*) values estimated by the *malign* function.

AlignedPeaks Cell array of peak lists, with the same form as *Peaks*, but with corrected *m/z* values in the first column of each matrix.

Description

[*CMZ*, *AlignedPeaks*] = *malign*(*Peaks*) aligns mass spectra from multiple peak lists (centroided data), by first estimating *CMZ*, a vector of common mass/charge (*m/z*) values estimated by considering the peaks in all spectra in *Peaks*, a cell array of peak lists, where each element corresponds to a spectrum or retention time. It then aligns the peaks

in each spectrum to the values in *CMZ*, creating *AlignedPeaks*, a cell array of aligned peak lists.

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

`[CMZ, AlignedPeaks] = malign(Peaks, ...'Quantile', QuantileValue, ...)` determines which peaks are selected by the estimation method to create *CMZ*, the vector of common m/z values. Choices are a scalar between 0 and 1. Default is 0.95.

`[CMZ, AlignedPeaks] = malign(Peaks, ...'EstimationMethod', EstimationMethodValue, ...)` specifies the method used to estimate *CMZ*, the vector of common mass/charge (m/z) values. Choices are:

- **histogram** — Default method. Peak locations are clustered using a kernel density estimation approach. The peak ion intensity is used as a weighting factor. The center of all the clusters conform to the *CMZ* vector.
- **regression** — Takes a sample of the distances between observed significant peaks and regresses the inter-peak distance to create the *CMZ* vector with similar inter-element distances.

`[CMZ, AlignedPeaks] = malign(Peaks, ...'CorrectionMethod', CorrectionMethodValue, ...)` specifies the method used to align each peak list to the *CMZ* vector. Choices are:

- **nearest-neighbor** — Default method. For each common peak in the *CMZ* vector, its counterpart in each peak list is the peak that is closest to the common peak's m/z value.

- **shortest-path** — For each common peak in the *CMZ* vector, its counterpart in each peak list is selected using the shortest path algorithm.

`[CMZ, AlignedPeaks] = mspalign(Peaks, ... 'ShowEstimation', ShowEstimationValue, ...)` controls the display of an assessment plot relative to the estimation method and the estimated vector of common mass/charge (*m/z*) values. Choices are `true` or `false`. Default is either:

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

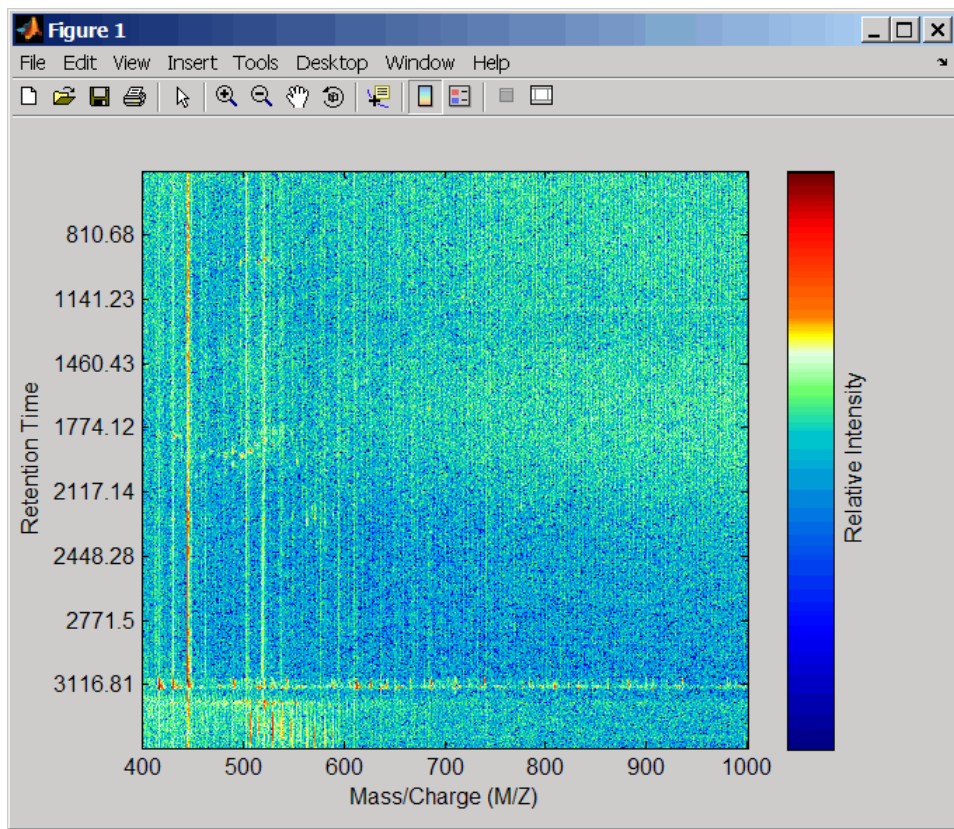
Examples

- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains liquid chromatography/mass spectrometry (LC/MS) data variables, including peaks and `ret_time`. `peaks` is a cell array of peak lists, where each element is a two-column matrix of *m/z* values and ion intensity values, and each element corresponds to a spectrum or retention time. `ret_time` is a column vector of retention times associated with the LC/MS data set.

```
load lcmsdata
```

- 2 Resample the unaligned data, display it in a heat map, and then overlay a dot plot.

```
[MZ,Y] = msppresample(peaks,5000);  
msheatmap(MZ,ret_time,log(Y))
```



```
mshotplot(peaks,ret_time)
```

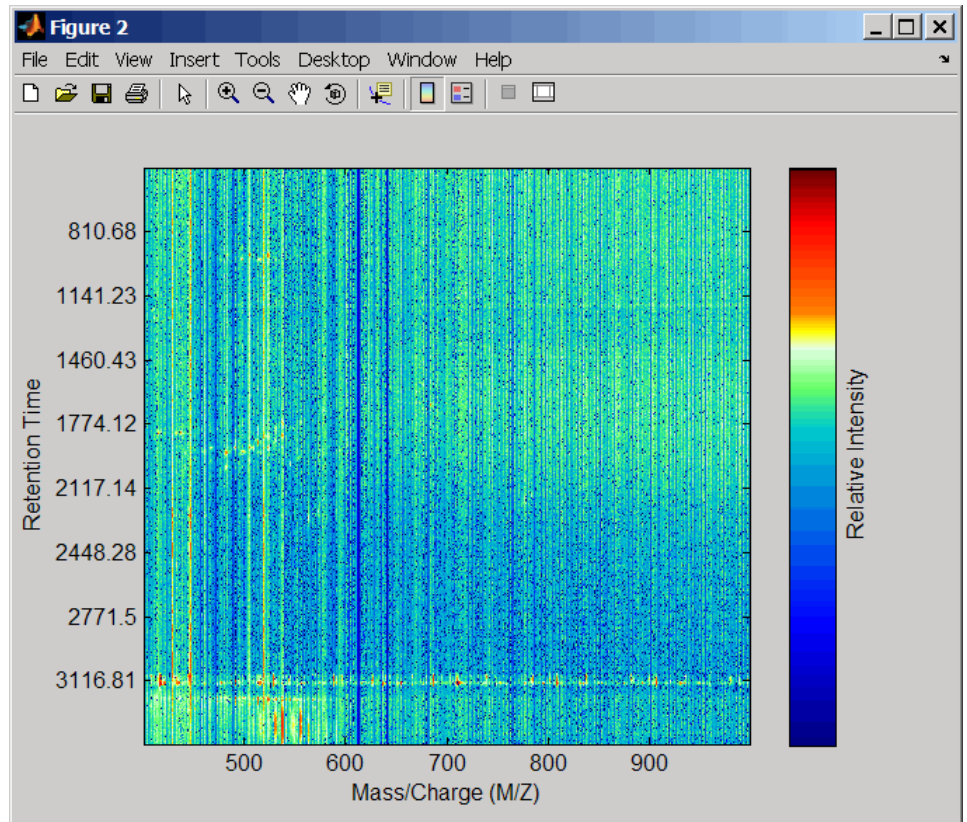
- 3 Align the peak lists from the mass spectra using the default estimation and correction methods.

```
[CMZ, aligned_peaks] = mspalign(peaks);
```

- 4 Resample the unaligned data, display it in a heat map, and then overlay a dot plot.

```
[M22,Y2] = mspresample(aligned_peaks,5000);
```

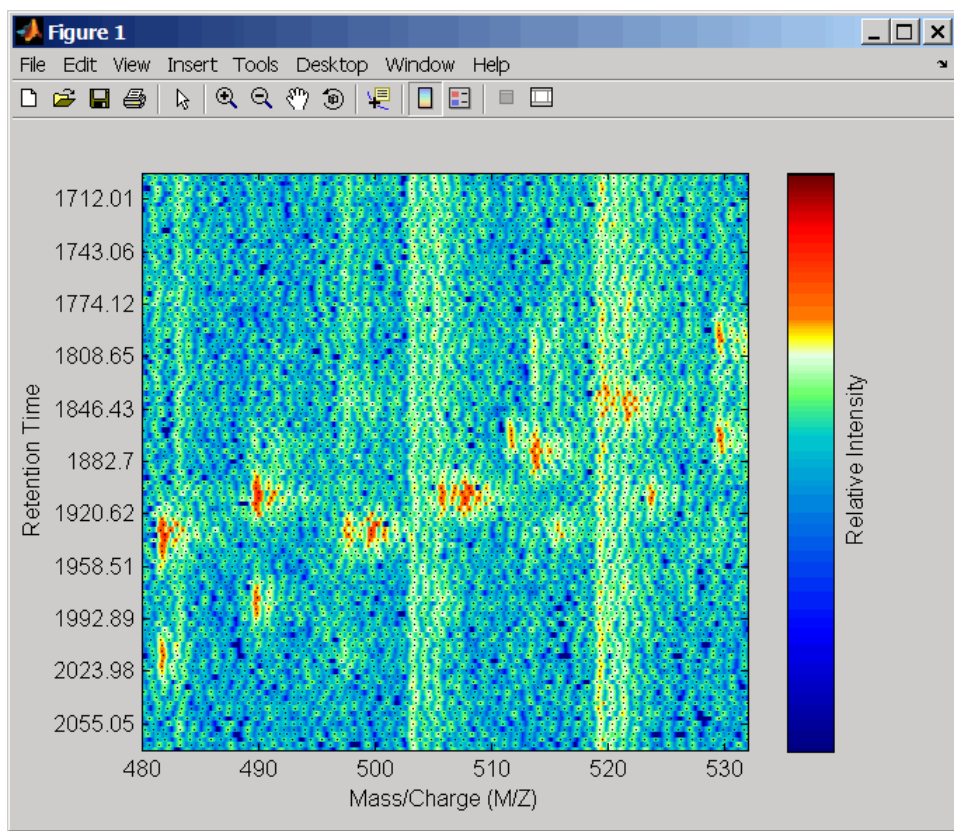
```
msheatmap(MZ2,ret_time,log(Y2))
```

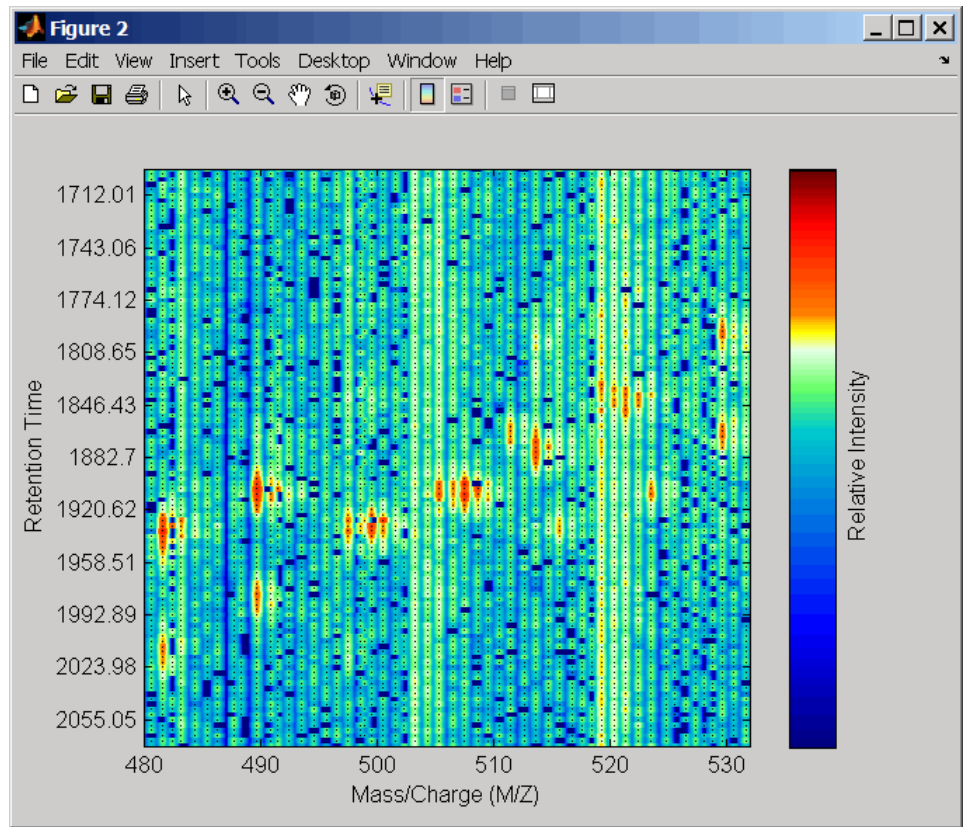


```
msdotplot(aligned_peaks,ret_time)
```

- 5 Link the axes of the two heat plots and zoom in to observe the detail to compare the unaligned and aligned LC/MS data sets.

```
linkaxes(findobj(0,'Tag','MSHeatMap'))  
axis([480 532 375 485])
```





References

- [1] Jeffries, N. (2005) Algorithms for alignment of mass spectrometry proteomic data. *Bioinformatics* 21:14, 3066–3073.
- [2] Purvine, S., Kolker, N., and Kolker, E. (2004) Spectral Quality Assessment for High-Throughput Tandem Mass Spectrometry Proteomics. *OMICS: A Journal of Integrative Biology* 8:3, 255–265.

See Also

Bioinformatics Toolbox functions: `msalign`, `msdotplot`, `msheatmap`, `mspeaks`, `msppresample`, `mzcdf2peaks`, `mzxml2peaks`

mspeaks

Purpose

Convert raw mass spectrometry data to peak list (centroided data)

Syntax

```
Peaks = mspeaks(MZ, Intensities)
Peaks = mspeaks(MZ, Intensities, ...'Base', BaseValue, ...)
Peaks = mspeaks(MZ, Intensities, ...'Levels', LevelsValue,
    ...)
Peaks = mspeaks(MZ, Intensities, ...'NoiseEstimator',
    NoiseEstimatorValue, ...)
Peaks = mspeaks(MZ, Intensities, ...'Multiplier',
    MultiplierValue, ...)
Peaks = mspeaks(MZ, Intensities, ...'Denoising',
    DenoisingValue, ...)
Peaks = mspeaks(MZ, Intensities, ...'PeakLocation',
    PeakLocationValue, ...)
Peaks = mspeaks(MZ, Intensities, ...'FWHHFilter',
    FWHHFilterValue, ...)
Peaks = mspeaks(MZ, Intensities,
    ...'OverSegmentationFilter',
    OverSegmentationFilterValue, ...)
Peaks = mspeaks(MZ, Intensities, ...'HeightFilter',
    HeightFilterValue, ...)
Peaks = mspeaks(MZ, Intensities, ...'ShowPlot',
    ShowPlotValue, ...)
```

Arguments*MZ*

Vector of mass/charge (m/z) values for a set of spectra. The number of elements in the vector equals n or the number of rows in matrix *Intensities*.

Intensities

Matrix of intensity values for a set of mass spectra that share the same mass/charge (m/z) range. Each row corresponds to an m/z value, and each column corresponds to a spectrum or retention time. The number of rows equals n or the number of elements in vector *MZ*.

BaseValue

An integer between 2 and 20 that specifies the wavelet base. Default is 4.

LevelsValue

An integer between 1 and 12 that specifies the number of levels for the wavelet decomposition. Default is 10.

NoiseEstimatorValue

String or scalar that specifies the method to estimate the threshold, T , to filter out noisy components in the first high-band decomposition (y_h). Choices are:

- **mad** — Default. Median absolute deviation, which calculates $T = \text{sqrt}(2 * \log(n)) * \text{mad}(y_h) / 0.6745$, where n = the number of rows in the *Intensities* matrix.
- **std** — Standard deviation, which calculates $T = \text{std}(y_h)$.
- A positive real value.

MultiplierValue

A positive real value that specifies the threshold multiplier constant. Default is 1.0.

DenoisingValue

Controls the use of wavelet denoising to smooth the signal. Choices are **true** (default) or **false**.

Note If your data has previously been smoothed, for example, with the `mslowess` or `mssgolay` function, it is not necessary to use wavelet denoising. Set this property to **false**.

<i>PeakLocationValue</i>	Value that specifies the proportion of the peak height that selects the points used to compute the centroid mass of the respective peak. The value must be ≥ 0 and ≤ 1 . Default is 1.0.
<i>FWHHFilterValue</i>	Positive real value that specifies the minimum full width at half height (FWHH), in m/z units, for reported peaks. Peaks with FWHH below this value are not included in the output list <i>Peaks</i> . Default is 0.
<i>OverSegmentationFilterValue</i>	Positive real value that specifies the minimum distance, in m/z units, between neighboring peaks. When the signal is not smoothed appropriately, multiple maxima can appear to represent the same peak. By increasing this filter value, oversegmented peaks are joined into a single peak. Default is 0.

HeightFilterValue Positive real value that specifies the minimum height for reported peaks. Default is 0.

ShowPlotValue Controls the display of a plot of the original and the smoothed signal, with the peaks included in the output matrix *Peaks* marked. Choices are `true`, `false`, or *I*, an integer specifying the index of a spectrum in *Intensities*. If set to `true`, the first spectrum in *Intensities* is plotted. Default is:

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

Return Values

Peaks Two-column matrix where each row corresponds to a peak. The first column contains mass/charge (m/z) values, and the second column contains ion intensity values.

Description

Peaks = `mspeaks(MZ, Intensities)` finds relevant peaks in raw mass spectrometry data, and creates *Peaks*, a two-column matrix, containing the m/z value and ion intensity for each peak.

`mspeaks` finds peaks by first smoothing the signal using undecimated wavelet transform with Daubechies coefficients, then assigning peak locations, and lastly, eliminating peaks that do not satisfy specified criteria.

Peaks = `mspeaks(MZ, Intensities, ... 'PropertyName', PropertyValue, ...)` calls `mspeaks` with optional properties that

use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

Peaks = mspeaks(*MZ*, *Intensities*, ...'Base', *BaseValue*, ...) specifies the wavelet base. *BaseValue* must be an integer between 2 and 20. Default is 4.

Peaks = mspeaks(*MZ*, *Intensities*, ...'Levels', *LevelsValue*, ...) specifies the number of levels for the wavelet decomposition. *LevelsValue* must be an integer between 1 and 12. Default is 10.

Peaks = mspeaks(*MZ*, *Intensities*, ...'NoiseEstimator', *NoiseEstimatorValue*, ...) specifies the method to estimate the threshold, T , to filter out noisy components in the first high-band decomposition (y_h). Choices are:

- mad — Default. Median absolute deviation, which calculates $T = \sqrt{2 \cdot \log(n)} \cdot \text{mad}(y_h) / 0.6745$, where n = the number of rows in the *Intensities* matrix.
- std — Standard deviation, which calculates $T = \text{std}(y_h)$.
- A positive real value.

Peaks = mspeaks(*MZ*, *Intensities*, ...'Multiplier', *MultiplierValue*, ...) specifies the threshold multiplier constant. *MultiplierValue* must be a positive real value. Default is 1.0.

Peaks = mspeaks(*MZ*, *Intensities*, ...'Denoising', *DenoisingValue*, ...) controls the use of wavelet denoising to smooth the signal. Choices are true (default) or false.

Note If your data has previously been smoothed, for example, with the `mslowess` or `mssgolay` function, it is not necessary to use wavelet denoising. Set this property to `false`.

Peaks = `mspeaks(MZ, Intensities, ...'PeakLocation', PeakLocationValue, ...)` specifies the proportion of the peak height that selects the points used to compute the centroid mass of the respective peak. *PeakLocationValue* must be a value ≥ 0 and ≤ 1 . Default is 1.0.

Note When *PeakLocationValue* = 1.0, the peak location is exactly at the maximum of the peak, while when *PeakLocationValue* = 0, the peak location is computed with all the points from the closest minimum to the left of the peak to the closest minimum to the right of the peak.

Peaks = `mspeaks(MZ, Intensities, ...'FWHHFilter', FWHHFilterValue, ...)` specifies the minimum full width at half height (FWHH), in m/z units, for reported peaks. Peaks with FWHH below this value are not included in the output list *Peaks*. *FWHHFilterValue* must be a positive real value. Default is 0.

Peaks = `mspeaks(MZ, Intensities, ...'OverSegmentationFilter', OverSegmentationFilterValue, ...)` specifies the minimum distance, in m/z units, between neighboring peaks. When the signal is not smoothed appropriately, multiple maxima can appear to represent the same peak. By increasing this filter value, oversegmented peaks are joined into a single peak. *OverSegmentationFilterValue* must be a positive real value. Default is 0.

Peaks = `mspeaks(MZ, Intensities, ...'HeightFilter', HeightFilterValue, ...)` specifies the minimum height for reported

peaks. Peaks with heights below this value are not included in the output list *Peaks*. *HeightFilterValue* must be a positive real value. Default is 0.

Peaks = mspeaks(*MZ*, *Intensities*, ... 'ShowPlot', *ShowPlotValue*, ...) controls the display of a plot of the original and the smoothed signal, with the peaks included in the output matrix *Peaks* marked. Choices are true, false, or *I*, an integer specifying the index of a spectrum in *Intensities*. If set to true, the first spectrum in *Intensities* is plotted. Default is either:

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

Examples

- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains mass spectrometry data variables, including *MZ_lo_res*, a vector of *m/z* values for a set of spectra, and *Y_lo_res*, a matrix of intensity values for a set of mass spectra that share the same *m/z* range.

```
load sample_lo_res
```

- 2 Adjust the baseline of the eight spectra stored in *Y_lo_res*.

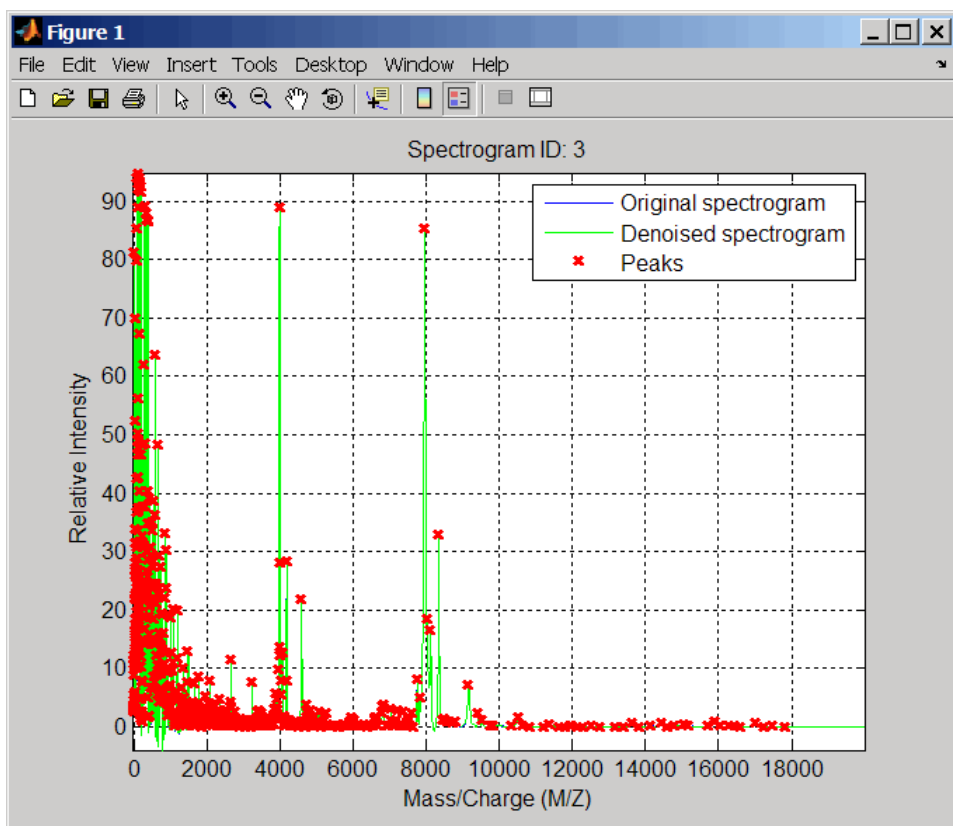
```
YB = msbackadj(MZ_lo_res,Y_lo_res);
```

- 3 Convert the raw mass spectrometry data to a peak list by finding the relevant peaks in each spectrum.

```
P = mspeaks(MZ_lo_res,YB);
```

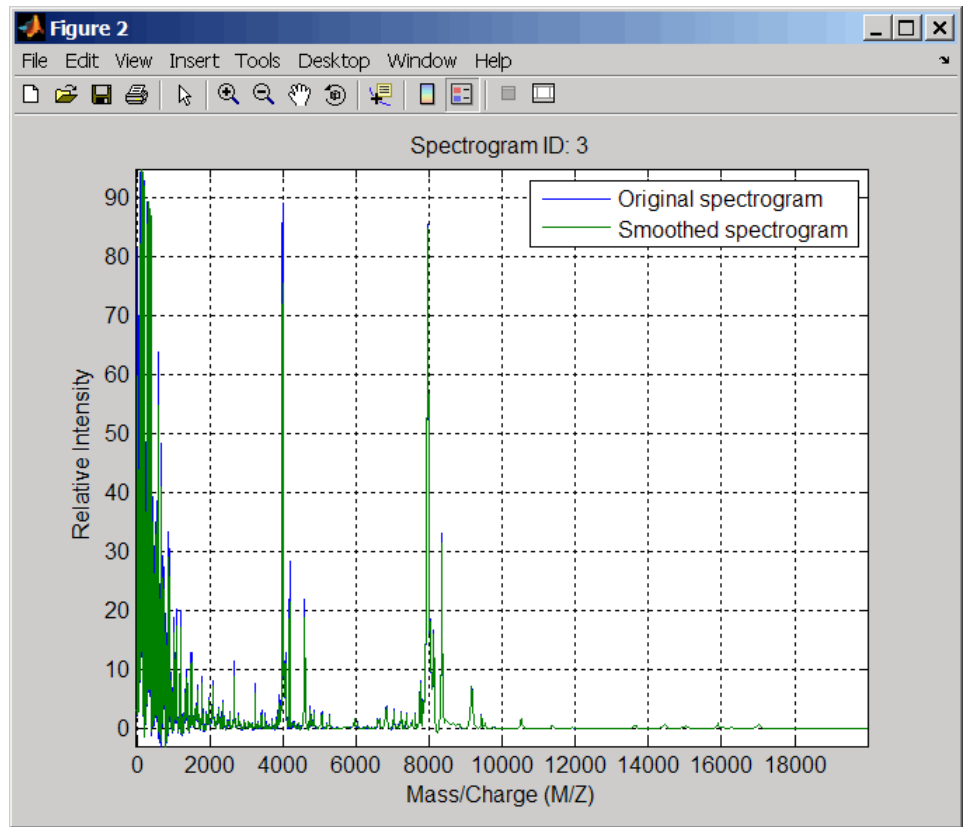
- 4 Plot the third spectrum in *YB*, the matrix of baseline-corrected intensity values, with the detected peaks marked.

```
P = mspeaks(MZ_lo_res,YB,'SHOWPLOT',3);
```

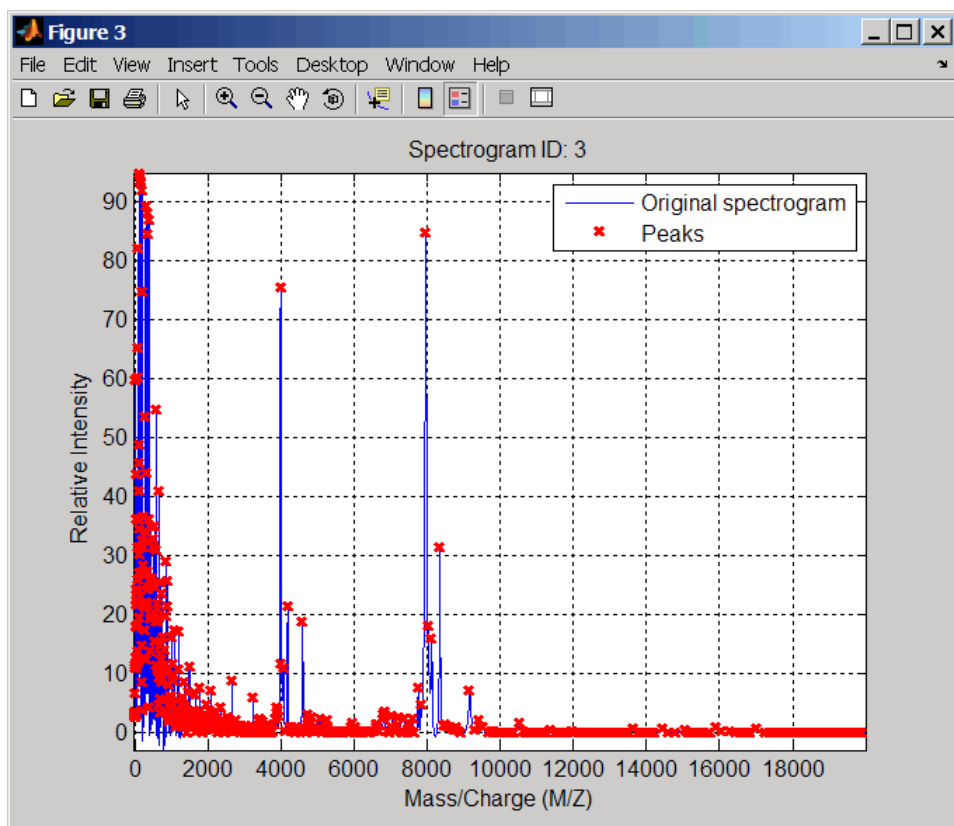


- 5 Smooth the signal using the `mslowess` function. Then convert the smoothed data to a peak list by finding relevant peaks and plot the third spectrum.

```
YS = mslowess(MZ_lo_res,YB,'SHOWPLOT',3);
```

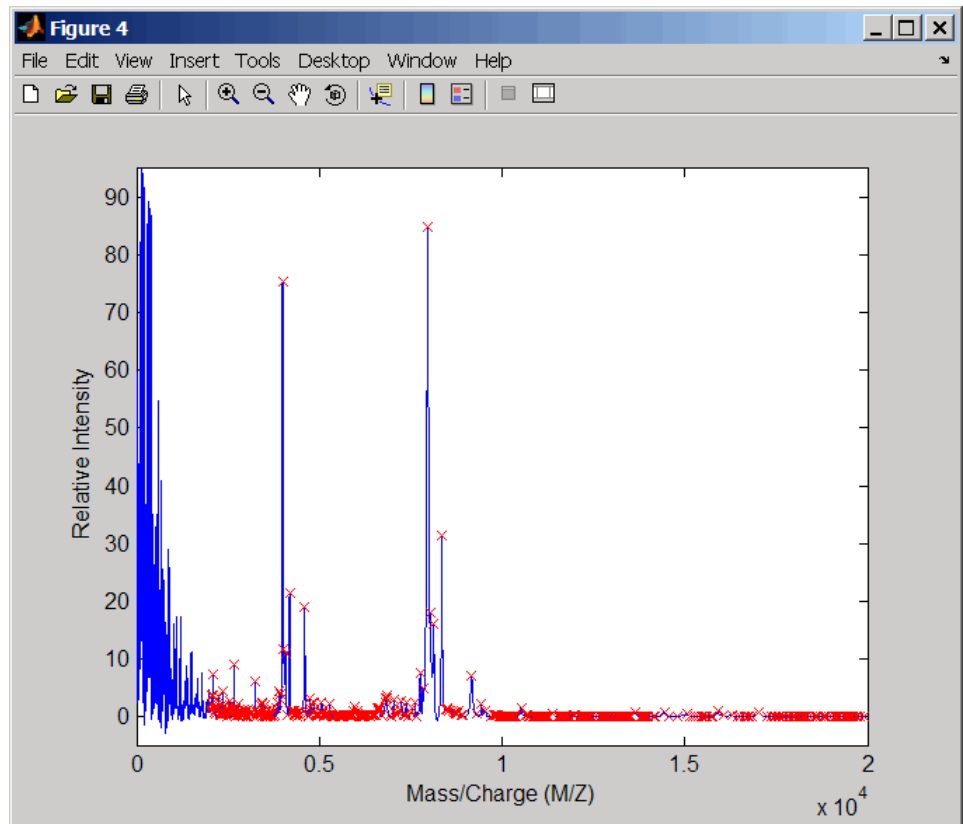


```
P = mspeaks(MZ_lo_res,YS, 'DENOISING',false,'SHOWPLOT',3);
```



- 6 Use the `cellfun` function to remove all peaks with m/z values less than 2000 from the eight peaks lists in output `P`. Then plot the peaks of the third spectrum (in red) over its smoothed signal (in blue).

```
Q = cellfun(@(p) p(p(:,1)>2000,:),P,'UniformOutput',false);
figure
plot(MZ_lo_res,YS(:,3),'b',Q{3}(:,1),Q{3}(:,2),'rx')
xlabel('Mass/Charge (M/Z)')
ylabel('Relative Intensity')
axis([0 20000 -5 95])
```



References

- [1] Morris, J.S., Coombes, K.R., Koomen, J., Baggerly, K.A., and Kobayash, R. (2005) Feature extraction and quantification for mass spectrometry in biomedical applications using the mean spectrum. *Bioinformatics* 21:9, 1764–1775.
- [2] Yasui, Y., Pepe, M., Thompson, M.L., Adam, B.L., Wright, G.L., Qu, Y., Potter, J.D., Winget, M., Thornquist, M., and Feng, Z. (2003) A data-analytic strategy for protein biomarker discovery: profiling of

high-dimensional proteomic data for cancer detection. *Biostatistics* 4:3, 449–463.

[3] Donoho, D.L., and Johnstone, I.M. (1995) Adapting to unknown smoothness via wavelet shrinkage. *J. Am. Statist. Asso.* 90, 1200–1224.

[4] Strang, G., and Nguyen, T. (1996) *Wavelets and Filter Banks* (Wellesley: Cambridge Press).

[5] Coombes, K.R., Tsavachidis, S., Morris, J.S., Baggerly, K.A., Hung, M.C., and Kuerer, H.M. (2005) Improved peak detection and quantification of mass spectrometry data acquired from surface-enhanced laser desorption and ionization by denoising spectra with the undecimated discrete wavelet transform. *Proteomics* 5(16), 4107–4117.

See Also

Bioinformatics Toolbox functions: `msbackadj`, `msdotplot`, `mslowess`, `mssalign`, `msspresample`, `mssgolay`

Purpose

Resample mass spectrometry signal while preserving peaks

Syntax

```
[MZ, Intensities] = msppresample(Peaks, N)
[MZ, Intensities] = msppresample(Peaks, N,
... 'Range', RangeValue, ...)
[MZ, Intensities] = msppresample(Peaks, N, ... 'FWHH',
FWHHValue, ...)
[MZ, Intensities] = msppresample(Peaks, N, ... 'ShowPlot',
ShowPlotValue, ...)
```

Arguments

Peaks

Either of the following:

- Two-column matrix, where the first column contains mass/charge (m/z) values and the second column contains ion intensity values.
- Cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time.

Note You can use the `mzxm12peaks` function or the `mspeaks` function to create the *Peaks* matrix or cell array.

N

Integer specifying the number of equally spaced points (m/z values) in the resampled signal.

RangeValue

1-by-2 vector specifying the minimum and maximum m/z values for the output matrix *Intensities*. *RangeValue* must be within $[\min(\text{inputMZ}) \max(\text{inputMZ})]$, where *inputMZ* is the concatenated m/z values from the input *Peaks*. Default is the full range $[\min(\text{inputMZ}) \max(\text{inputMZ})]$.

FWHHValue Value that specifies the full width at half height (FWHH) in m/z units. The FWHH is used to convert each peak to a Gaussian shaped curve. Default is $\text{median}(\text{diff}(\text{inputMZ}))/2$, where *inputMZ* is the concatenated m/z values from the input *Peaks*. The default is a rough approximation of resolution observed in the input data, *Peaks*.

Tip To ensure that the resolution of the peaks is preserved, set *FWHHValue* to half the distance between the two peaks of interest that are closest to each other.

ShowPlotValue Controls the display of a plot of an original and resampled spectrum. Choices are `true`, `false`, or *I*, an integer specifying the index of a spectrum in *Intensities*. If set to `true`, the first spectrum in *Intensities* is plotted. Default is:

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

Return Values

MZ Vector of equally spaced, common mass/charge (m/z) values for a set of spectra. The number of elements in the vector equals *N* or the number of rows in matrix *Intensities*.

Intensities Matrix of reconstructed intensity values for a set of mass spectra that share the same mass/charge (m/z) range. Each row corresponds to an m/z value, and each column corresponds to a spectrum or retention time. The number of rows equals *N* or the number of elements in vector *MZ*.

Description

`[MZ, Intensities] = msppresample(Peaks, N)` resamples *Peaks*, a mass spectrometry peak list, by converting centroided peaks to a semicontinuous, raw signal that preserves peak information. The resampled signal has *N* equally spaced points. Output *MZ* is a vector of *N* elements specifying the equally spaced, common m/z values for the spectra. Output *Intensities* is a matrix of reconstructed intensity values for a set of mass spectra that share the same m/z range. Each row corresponds to an m/z value, and each column corresponds to a spectrum or retention time. The number of rows equals *N*.

`msppresample` uses a Gaussian kernel to reconstruct the signal. The ion intensity at any given m/z value is taken from the maximum intensity of any contributing (overlapping) peaks.

Tip `msppresample` is useful to prepare a set of spectra for imaging functions such as `msheatmap` and preprocessing functions such as `msbackadj` and `msnorm`.

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

`[MZ, Intensities] = msppresample(Peaks, N, ... 'Range', RangeValue, ...)` specifies an m/z range for the output matrix *Intensities* using the minimum and maximum m/z values specified in the 1-by-2 vector *RangeValue*. *RangeValue* must be within `[min(inputMZ) max(inputMZ)]`, where *inputMZ* is the concatenated m/z values from the input *Peaks*. Default is the full range `[min(inputMZ) max(inputMZ)]`

`[MZ, Intensities] = msppresample(Peaks, N, ... 'FWHH', FWHHValue, ...)` sets the full width at half height (FWHH) in m/z units. The FWHH is used to convert each peak

to a Gaussian shaped curve. Default is `median(diff(inputMZ))/2`, where `inputMZ` is the concatenated m/z values from the input `Peaks`. The default is a rough approximation of resolution observed in the input data, `Peaks`.

Tip To ensure that the resolution of the peaks is preserved, set `FWHHValue` to half the distance between the two peaks of interest that are closest to each other.

`[MZ, Intensities] = msppresample(Peaks, N, ... 'ShowPlot', ShowPlotValue, ...)` controls the display of a plot of an original and resampled spectrum. Choices are `true`, `false`, or `I`, an integer specifying the index of a spectrum in `Intensities`. If set to `true`, the first spectrum in `Intensities` is plotted. Default is:

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

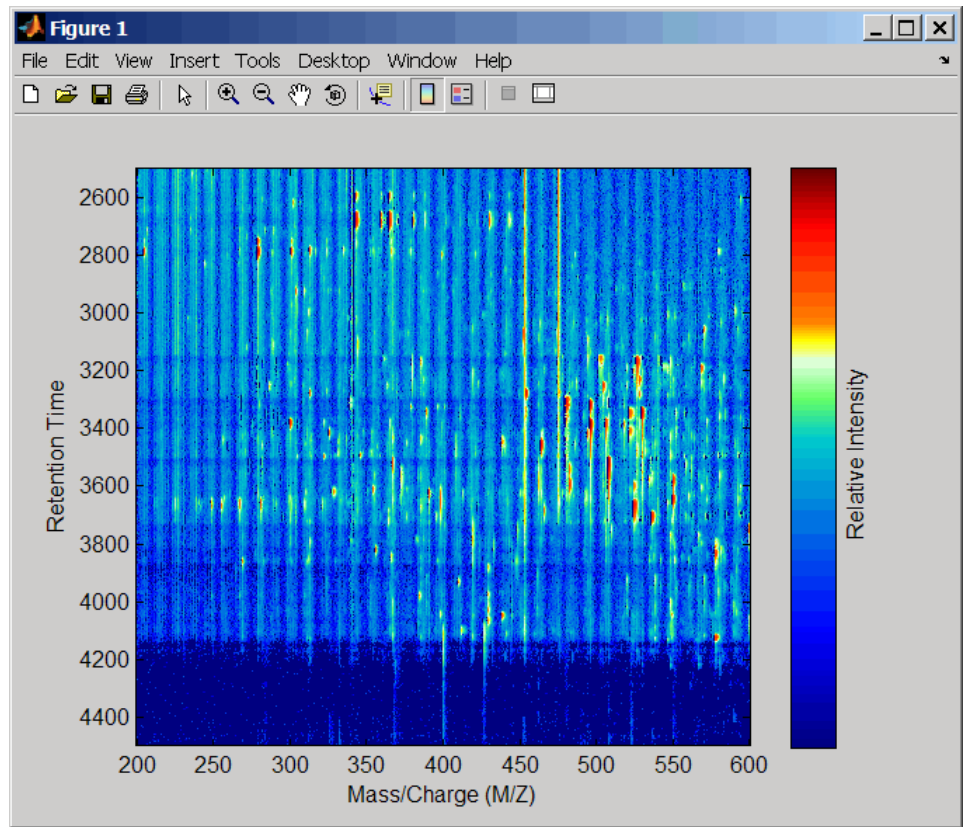
Examples

- 1 Load a MAT-file, included with the Bioinformatics Toolbox software, which contains liquid chromatography/mass spectrometry (LC/MS) data variables, including `peaks`, a cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time.

```
load lcmsdata
```

- 2 Resample the data, specifying 5000 m/z values in the resampled signal. Then create a heat map of the LC/MS data.

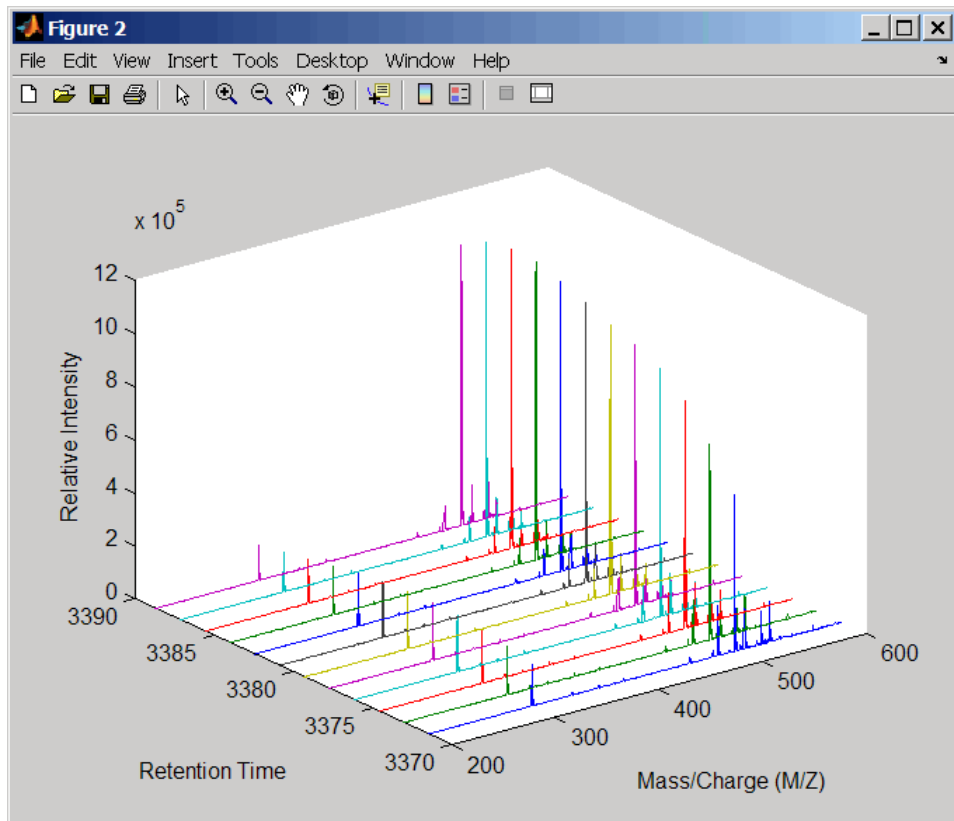
```
[MZ,Y] = msppresample(peaks,5000);  
msheatmap(MZ,ret_time,log(Y))
```



3 Plot the reconstructed profile spectra between two retention times.

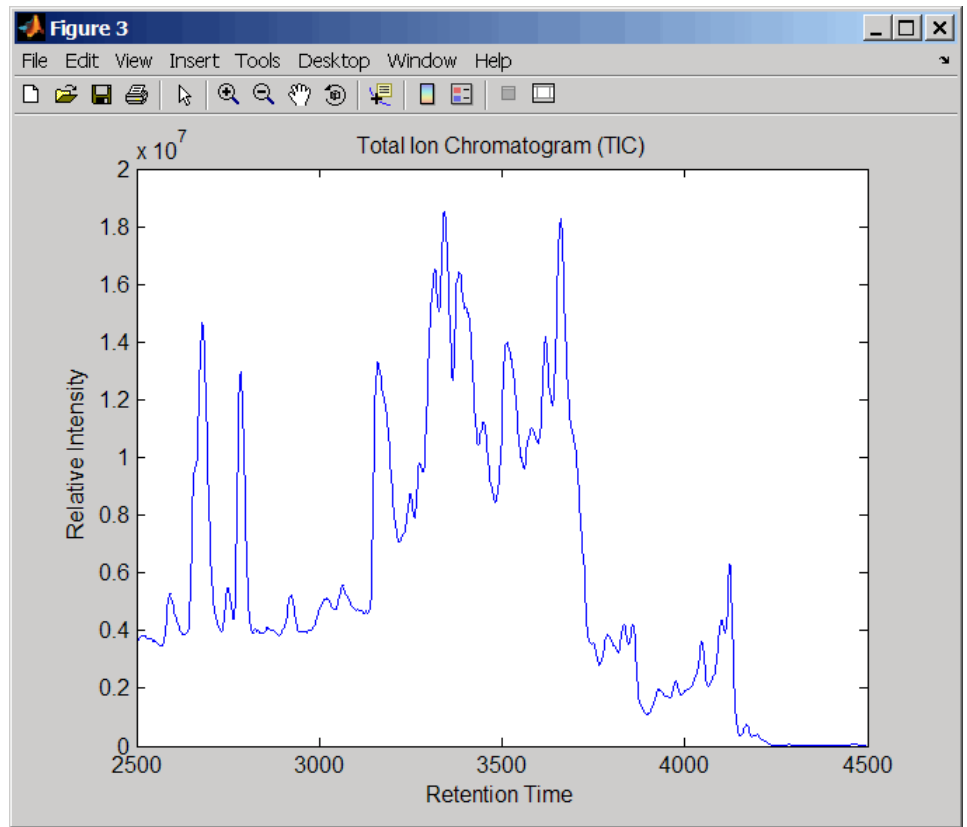
```
figure
t1 = 3370;
t2 = 3390;
h = find(ret_time>t1 & ret_time<t2);
[MZ,Y] = msppresample(peaks(h),10000);
plot3(repmat(MZ,1,numel(h)),repmat(ret_time(h)',10000,1),Y)
xlabel('Mass/Charge (M/Z)')
ylabel('Retention Time')
```

```
zlabel('Relative Intensity')
```



4 Resample the data to plot the Total Ion Chromatogram (TIC).

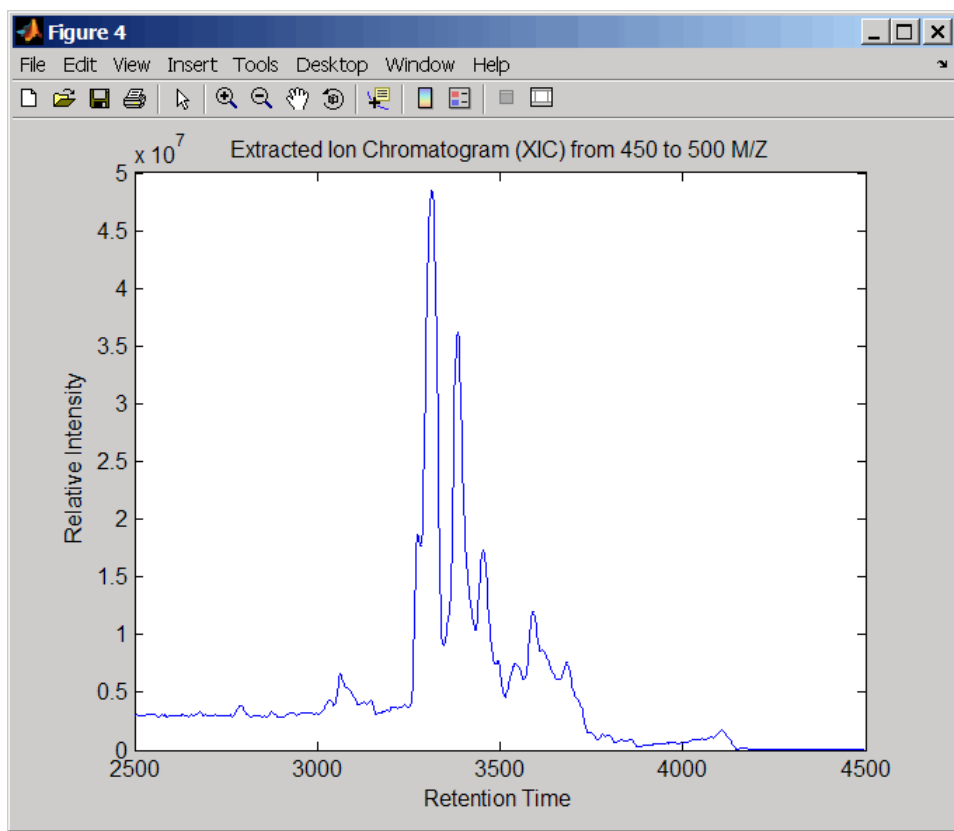
```
figure  
[MZ,Y] = msppresample(peaks,5000);  
plot(ret_time,sum(Y))  
title('Total Ion Chromatogram (TIC)')  
xlabel('Retention Time')  
ylabel('Relative Intensity')
```



- 5 Resample the data to plot the Extracted Ion Chromatogram (XIC) in the 450 to 500 m/z range.

```
figure
[MZ,Y] = m sppresample(peaks,5000,'Range',[450 500]);
plot(ret_time,sum(Y))
title('Extracted Ion Chromatogram (XIC) from 450 to 500 M/Z')
xlabel('Retention Time')
ylabel('Relative Intensity')
```

msppresample



See Also

Bioinformatics Toolbox functions: `msdotplot`, `mspeaks`, `mssalign`, `msresample`, `mzcdf2peaks`, `mzcdfread`, `mzxml2peaks`, `mzxmlread`

Purpose

Resample mass spectrometry signal

Syntax

```
[MZout, Yout] = msresample(MZ, Y, N)
msresample(..., 'Uniform', UniformValue, ...)
msresample(..., 'Range', RangeValue, ...)
msresample(..., 'RangeWarnOff', RangeWarnOffValue, ...)
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 cutoff 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, ...)` calls `msresample` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

`msresample(..., 'Uniform', UniformValue, ...)`, when *UniformValue* is true, 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 $[\min(MZ) \max(MZ)]$. Default value is the full range $[\min(MZ) \max(MZ)]$. When *RangeValue* values go beyond the values in *MZ*, `msresample` extrapolates the signal with zeros and returns a warning message.

`msresample(..., 'RangeWarnOff', RangeWarnOffValue, ...)` controls the return of a warning message when *RangeValue* values go beyond the values in *MZ*. *RangeWarnOffValue* can be true or false (default).

`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*.

Tip LC/MS data analysis requires extended amounts of memory from the operating system. If you receive any errors related to memory, increase the virtual memory (or swap space) of your operating system or set the 3 GB switch (32-bit Windows® XP only) as described at:

<http://www.mathworks.com/support/tech-notes/1100/1107.html>

Examples

- 1 Load mass spectrometry data and extract m/z and intensity value vectors.

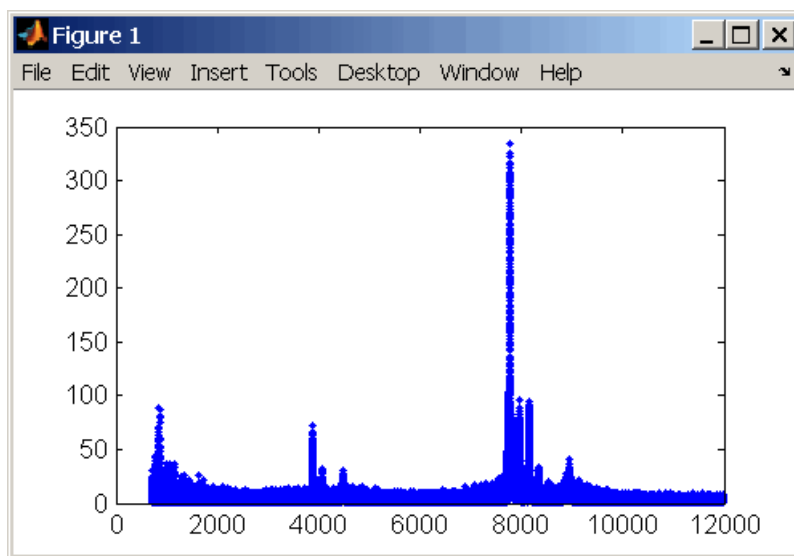
```
load sample_hi_res;  
mz = MZ_hi_res;  
y = Y_hi_res;
```

- 2 Plot original data to a lower resolution.

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

A figure displays.

msresample



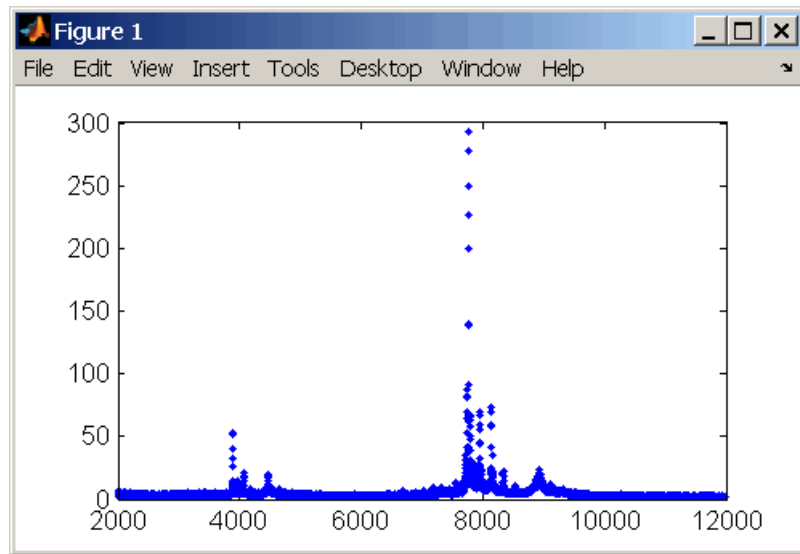
3 Resample data.

```
[mz1,y1] = msresample(mz, y, 10000, 'range',[2000 max(mz)]);
```

4 Plot resampled data.

```
plot(mz1,y1,'.')
```

A figure displays with the down sampled data.



See Also

Bioinformatics Toolbox functions: `msalign`, `msbackadj`, `msheatmap`, `mslowess`, `msnorm`, `mppresample`, `mssgolay`, `msviewer`

mssgolay

Purpose Smooth mass spectrum with least-squares polynomial

Syntax

```
Yout = mssgolay(MZ, Y)  
mssgolay(..., 'PropertyName', PropertyValue,...)  
mssgolay(..., 'Span', SpanValue)  
mssgolay(..., 'Degree', DegreeValue)  
mssgolay(..., 'ShowPlot', ShowPlotValue)
```

Arguments

<i>MZ</i>	Mass/charge vector with the range of ions in the spectra.
<i>Y</i>	Ion intensity vector with the same length as the mass/charge vector (<i>MZ</i>). <i>Y</i> can also be a matrix with several spectra that share the same mass/charge (<i>MZ</i>) range.

Description *Yout* = mssgolay(*MZ*, *Y*) smoothes a raw mass spectrum (*Y*) using a least squares digital polynomial filter (Savitzky and Golay filters). The default span or frame is 15 samples.

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

mssgolay(..., '*Span*', *SpanValue*) modifies the frame size for the smoothing function. If *SpanValue* is greater than 1, the window is the size of *SpanValue* in samples independent of the *MZ* vector. Higher values will smooth the signal more with an increase in computation time. If *SpanValue* is less than 1, the window size is a fraction of the number of points in the data (*MZ*). For example, if *SpanValue* is 0.05, the window size is equal to 5% of the number of points in *MZ*.

Note 1) The original algorithm by Savitzky and Golay assumes a uniformly spaced mass/charge vector (MZ), while `mssgolay` also allows one that is not uniformly spaced. Therefore, the sliding frame for smoothing is centered using the closest samples in terms of the MZ value and not in terms of the MZ index.

2) When the vector MZ does not have repeated values or NaNs, the algorithm is approximately twice as fast.

3) When the vector MZ is evenly spaced, the least-squares fitting is performed once so that the spectrum is filtered with the same coefficients, and the speed of the algorithm increases considerably.

4) If the vector MZ is evenly spaced and `SpanValue` is even, `Span` is 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

Bioinformatics Toolbox functions: `msalign`, `msbackadj`, `msheatmap`, `mslowess`, `msnorm`, `mspeaks`, `msresample`, `msviewer`

msviewer

Purpose Explore mass spectrum or set of mass spectra

Syntax
`msviewer(MZ, Y)`
`msviewer(..., 'Markers', MarkersValue)`
`msviewer(..., 'Group', GroupValue)`

Arguments

MZ	Mass/charge vector with the range of ions in the spectra.
Y	Ion intensity vector with the same length as the mass/charge vector (MZ). Y can also be a matrix with several spectra that share the same mass/charge (MZ) range.

Description `msviewer(MZ, Y)` creates a GUI to display and explore a mass spectrum (Y).

`msviewer(..., 'Markers', MarkersValue)` specifies a list of marker positions from the mass/charge vector (MZ) for exploration and easy navigation. Enter a column vector with MZ values.

`msviewer(..., 'Group', GroupValue)` specifies a class label for every spectrum with a different color for every class. Enter a column vector of size [numSpectra x 1] with integers. The default value is [numSpectra].

MSViewer GUI features include the following:

- Plot mass spectra. The spectra are plotted with different colors according to their class labels.
- An overview displays a full spectrum, and a box indicates the region that is currently displayed in the main window.
- Five different zoom in options, one zoom out option, and a reset view option resize the spectrum.
- Add/focus/move/delete marker operations

- Import/Export markers from/to MATLAB workspace
- Print and preview the spectra plot
- Print the spectra plot to a MATLAB Figure window

MSViewer has five components:

- Menu bar: **File**, **Tools**, **Window**, and **Help**
- Toolbar: Zoom XY, Zoom X, Zoom Y, Reset view, Zoom out, and Help
- Main window: display the spectra
- Overview window: display the overview of a full spectrum (the average of all spectra in display)
- Marker control panel: a list of markers, Add marker, Delete marker, up and down buttons

Examples

- 1 Load and plot sample data

```
load sample_lo_res
msviewer(MZ_lo_res, Y_lo_res)
```

- 2 Add a marker by pointing to a mass peak, right-clicking, and then clicking **Add Marker**.

- 3 From the **File** menu, select

- **Import Markers from Workspace** — Opens the Import Markers From MATLAB Workspace dialog. The dialog should display a list of double Mx1 or 1xM variables. If the selected variable is out of range, the viewer displays an error message
- **Export Markers to Workspace** — Opens the Export Markers to MATLAB Workspace dialog. You can enter a variable name for the markers. All markers are saved. If there is no marker available, this menu item should be disabled.
- **Print to Figure** — Prints the spectra plot in the main display to a MATLAB Figure window

- 4** From the **Tools** menu, click
 - **Add Marker** — Opens the Add Marker dialog. Enter an m/z marker.
 - **Delete Marker** — Removes the currently selected m/z marker from the **Markers** (m/z) list.
 - **Next Marker** or **Previous Marker** — Moves the selection up and down the **Markers** (m/z) list.
 - **Zoom XY, Zoom X, Zoom Y, or Zoom Out** — Changes the cursor from an arrow to crosshairs. Left-click and drag a rectangle box over an area and then release the mouse button. The display zooms the area covered by the box.
- 5** Move the cursor to the range window at the bottom. Click and drag the view box to a new location.

See Also

Bioinformatics Toolbox functions: `msalign`, `msbackadj`, `mslowess`, `msnorm`, `msheatmap`, `msresample`, `mssgolay`

Purpose

Align multiple sequences using progressive method

Syntax

```
SeqsMultiAligned = multialign(Seqs)
SeqsMultiAligned = multialign(Seqs, Tree)
multialign(..., 'PropertyName', PropertyValue, ...)
multialign(..., 'Weights', WeightsValue)
multialign(..., 'ScoringMatrix', ScoringMatrixValue)
multialign(..., 'SMInterp', SMInterpValue)
multialign(..., 'GapOpen', GapOpenValue)
multialign(..., 'ExtendGap', ExtendGapValue)
multialign(..., 'DelayCutoff', DelayCutoffValue)
multialign(..., 'JobManager', JobManagerValue)
multialign(..., 'WaitInQueue', WaitInQueueValue)
multialign(..., 'Verbose', VerboseValue)
multialign(..., 'ExistingGapAdjust',
ExistingGapAdjustValue)
multialign(..., 'TerminalGapAdjust',
TerminalGapAdjustValue)
```

Arguments

Seqs

Vector of structures with the fields 'Sequence' for the residues and 'Header' or 'Name' for the labels.

Seqs may also be a cell array of strings or a char array.

SeqsMultiAligned

Vector of structures (same as *Seqs*) but with the field 'Sequence' updated with the alignment.

When *Seqs* is a cell or char array, *SeqsMultiAligned* is a char array with the output alignment following the same order as the input.

multialign

<i>Tree</i>	Phylogenetic tree calculated with either of the functions <code>seqlinkage</code> or <code>seqneighjoin</code> .
<i>WeightsValue</i>	Property to select the sequence weighting method. Enter either 'THG' (default) or 'equal'.
<i>ScoringMatrixValue</i>	Property to select or specify the scoring matrix. Enter an [MxM] matrix or [MxMxN] array of matrixes withN user-defined scoring matrices. <i>ScoringMatrixValue</i> may also be a cell array of strings with matrix names. The default is the BLOSUM80 to BLOSUM30 series for amino acids or a fixed matrix NUC44 for nucleotides. When passing your own series of scoring matrices make sure all of them share the same scale.
<i>SMInterpValue</i>	Property to specify whether linear interpolation of the scoring matrices is on or off. When <code>false</code> , scoring matrix is assigned to a fixed range depending on the distances between the two profiles (or sequences) being aligned. Default is <code>true</code> .
<i>GapOpenValue</i>	Scalar or a function specified using @. If you enter a function, <code>multialign</code> passes four values to the function: the average score for two matched residues (<code>sm</code>), the average score for two mismatched residues (<code>sx</code>), and, the length of both profiles or sequences (<code>len1</code> , <code>len2</code>). Default is <code>@(sm,sx,len1,len2) 5*sm</code> .

<i>ExtendGapValue</i>	Scalar or a function specified using @. IF you enter a function, <code>multialign</code> passes four values to the function: the average score for two matched residues (<code>sm</code>), the average score for two mismatched residues (<code>sx</code>), and the length of both profiles or sequences (<code>len1</code> , <code>len2</code>). Default is <code>@(sm,sx,len1,len2) sm/4</code> .
<i>DelayCutoffValue</i>	Property to specify the threshold delay of divergent sequences. The default is unity where sequences with the closest sequence farther than the median distance are delayed.
<i>JobManagerValue</i>	JobManager object representing an available distributed MATLAB resource. Enter a jobmanager object returned by the Parallel Computing Toolbox™ function <code>findResource</code> .
<i>WaitInQueueValue</i>	Property to control waiting for a distributed MATLAB resource to be available. Enter either <code>true</code> or <code>false</code> . The default value is <code>false</code> .
<i>VerboseValue</i>	Property to control displaying the sequences with sequence information. Default value is <code>false</code> .
<i>ExistingGagAdjustValue</i>	Property to control automatic adjustment based on existing gaps. Default value is <code>true</code> .
<i>TerminalGapAdjustValue</i>	Property to adjust the penalty for opening a gap at the ends of the sequence. Default value is <code>false</code> .

Description

SeqsMultiAligned = `multialign(Seqs)` performs a progressive multiple alignment for a set of sequences (*Seqs*). Pairwise distances between sequences are computed after pairwise alignment with the Gonnet scoring matrix and then by counting the proportion of sites at which each pair of sequences are different (ignoring gaps). The guide tree is calculated by the neighbor-joining method assuming equal variance and independence of evolutionary distance estimates.

SeqsMultiAligned = `multialign(Seqs, Tree)` uses a tree (*Tree*) as a guide for the progressive alignment. The sequences (*Seqs*) should have the same order as the leaves in the tree (*Tree*) or use a field ('Header' or 'Name') to identify the sequences.

`multialign(..., 'PropertyName', PropertyValue, ...)` enters optional arguments as property name/value pairs.

`multialign(..., 'Weights', WeightsValue)` selects the sequence weighting method. Weights emphasize highly divergent sequences by scaling the scoring matrix and gap penalties. Closer sequences receive smaller weights.

Values of the property Weights:

- 'THG' (default) — Thompson-Higgins-Gibson method using the phylogenetic tree branch distances weighted by their thickness.
- 'equal' — Assigns same weight to every sequence.

`multialign(..., 'ScoringMatrix', ScoringMatrixValue)` selects the scoring matrix (*ScoringMatrixValue*) for the progressive alignment. Match and mismatch scores are interpolated from the series of scoring matrices by considering the distances between the two profiles or sequences being aligned. The first matrix corresponds to the smallest distance and the last matrix to the largest distance. Intermediate distances are calculated using linear interpolation.

`multialign(..., 'SMInterp', SMInterpValue)`, when *SMInterpValue* is false, turns off the linear interpolation of the scoring matrices. Instead, each supplied scoring matrix is assigned to

a fixed range depending on the distances between the two profiles or sequences being aligned.

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

`multialign(..., 'ExtendGap', ExtendGapValue)` specifies the initial penalty for extending a gap.

`multialign(..., 'DelayCutoff', DelayCutoffValue)` specifies a threshold to delay the alignment of divergent sequences whose closest neighbor is farther than

$(\textit{DelayCutoffValue}) * (\text{median patristic distance between sequences})$

`multialign(..., 'JobManager', JobManagerValue)` distributes pairwise alignments into a cluster of computers using the Parallel Computing Toolbox software.

`multialign(..., 'WaitInQueue', WaitInQueueValue)` when *WaitInQueueValue* is true, waits in the job manager queue for an available worker. When *WaitInQueueValue* is false (default) and there are no workers immediately available, `multialign` errors out. Use this property with the Parallel Computing Toolbox software and the `multialign` property `WaitInQueue`.

`multialign(..., 'Verbose', VerboseValue)`, when *VerboseValue* is true, turns on verbosity.

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

`multialign(..., 'ExistingGapAdjust', ExistingGapAdjustValue)`, if *ExistingGapAdjustValue* is false, turns off the automatic adjustment based on existing gaps of the position-specific penalties for opening a gap.

When *ExistingGapAdjustValue* is true, for every profile position, `profalign` proportionally lowers the penalty for opening a gap toward

multialign

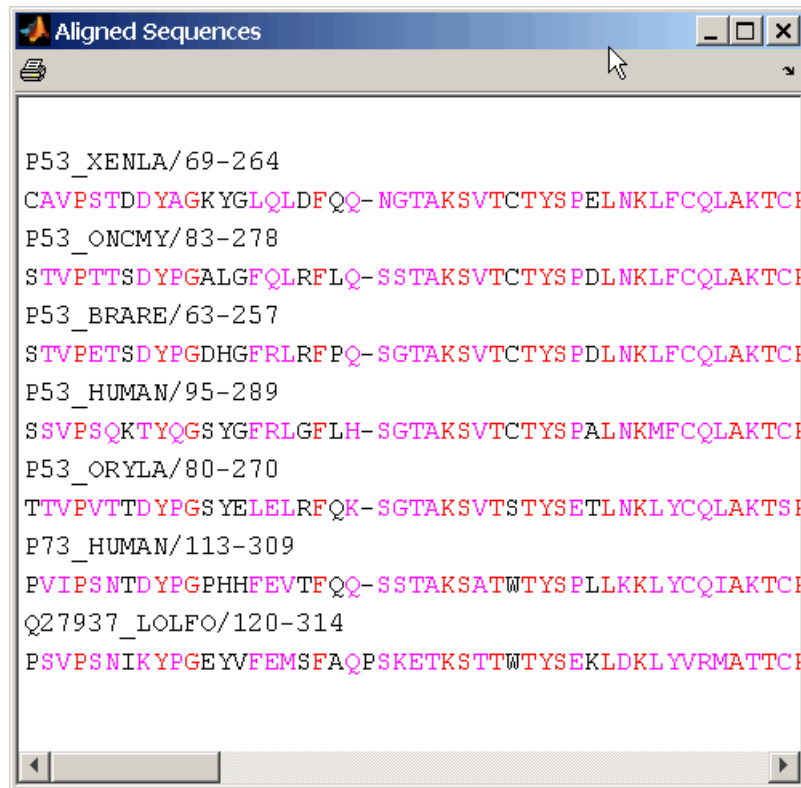
the penalty of extending a gap based on the proportion of gaps found in the contiguous symbols and on the weight of the input profile.

`multialign(..., 'TerminalGapAdjust', TerminalGapAdjustValue)`, when *TerminalGapAdjustValue* is `true`, adjusts the penalty for opening a gap at the ends of the sequence to be equal to the penalty for extending a gap.

Example 1

1 Align seven cellular tumor antigen p53 sequences.

```
p53 = fastaread('p53samples.txt')
ma = multialign(p53, 'verbose', true)
showalignment(ma)
```



- 2 Use an UPGMA phylogenetic tree instead as a guiding tree.

```

dist = seqpdist(p53,'ScoringMatrix',gonnet);
tree = seqlinkage(dist,'UPGMA',p53)
  
```

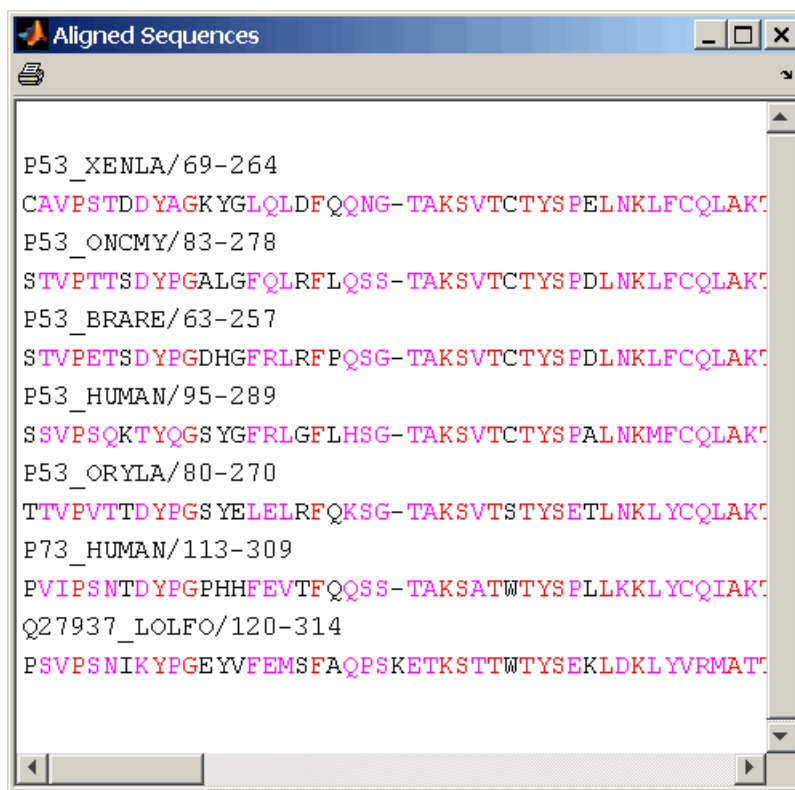
Phylogenetic tree object with 7 leaves (6 branches)

- 3 Score the progressive alignment with the PAM family.

```

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

multialign



Example 2

- 1 Enter an array of sequences.

```
seqs = {'CACGTAACATCTC', 'ACGACGTAACATCTTCT', 'AAACGTAACATCTCGC'};
```

- 2 Promote terminations with gaps in the alignment.

```
multialign(seqs, 'terminalGapAdjust', true)
```

```
ans =
--CACGTAACATCTC--
ACGACGTAACATCTTCT
-AAACGTAACATCTCGC
```


3 Compare alignment without termination gap adjustment.

```
multialign(seqs)
```

```
ans =  
CA--CGTAACATCT--C  
ACGACGTAACATCTTCT  
AA-ACGTAACATCTCGC
```

See Also

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

multialignread

Purpose	Read multiple sequence alignment file	
Syntax	<pre>S = multialignread(<i>File</i>) [<i>Headers</i>, <i>Sequences</i>] = multialignread(<i>File</i>) ... = multialignread(<i>File</i>, 'IgnoreGaps', <i>IgnoreGapsValue</i>)</pre>	
Arguments	<i>File</i>	Multiple sequence alignment file specified by one of the following: <ul style="list-style-type: none">• File name or path and file name• URL pointing to a file• MATLAB character array that contains the text of a multiple sequence alignment file You can read common multiple sequence alignment file types, such as ClustalW (.aln), GCG (.msf), and PHYLIP.
	<i>IgnoreGapsValue</i>	Controls removing gap symbols, such as '-' or '.', from the sequences. Choices are true or false (default).
Return Values	<i>S</i>	MATLAB structure array containing the following fields: <ul style="list-style-type: none">• Header — Header information from the file.• Sequence — Amino acid or nucleotide sequences.
	<i>Headers</i>	Cell array containing the header information from the file.
	<i>Sequences</i>	Cell array containing the amino acid or nucleotide sequences.

Description

`S = multialignread(File)` reads a multiple sequence alignment file. The file contains multiple sequence lines that start with a sequence header followed by an optional number (not used by `multialignread`) and a section of the sequence. The multiple sequences are broken into blocks with the same number of blocks for every sequence. To view an example multiple sequence alignment file, type `open aagag.aln` at the MATLAB command line.

The output, `S`, is a structure array where `S.Header` contains the header information and `S.Sequence` contains the amino acid or nucleotide sequences.

`[Headers, Sequences] = multialignread(File)` reads the file into separate variables, `Headers` and `Sequences`, which are cell arrays containing header information and amino acid or nucleotide sequences, respectively.

`... = multialignread(File, 'IgnoreGaps', IgnoreGapsValue)` controls the removal of any gap symbol, such as '-' or '.', from the sequences. Choices are `true` or `false` (default).

Examples

Read a multiple sequence alignment of the gag polyprotein for several HIV strains.

```
gagaa = multialignread('aagag.aln')
```

```
gagaa =
```

```
1x16 struct array with fields:
```

```
Header
```

```
Sequence
```

See Also

Bioinformatics Toolbox functions: `fastaread`, `gethmmalignment`, `multialign`, `multialignviewer`, `multialignwrite`, `seqconsensus`, `seqdisp`, `seqprofile`

multialignviewer

Purpose Open viewer for multiple sequence alignments

Syntax `multialignviewer(Alignment)`
`multialignviewer(Alignment, 'Alphabet', AlphabetValue)`

Arguments

Alignment A structure with a field *Sequence*, a character array, or a file name.

AlphabetValue String specifying the alphabet type for the sequences. Choices are 'AA' for amino acids or 'NT' for nucleotides. Default is 'AA'.

Description

The `multialignviewer` function provides an interactive graphical user interface (GUI) for viewing multiple sequence alignments. For examples of using this GUI, see “Multiple Sequence Alignment Viewer”.

`multialignviewer(Alignment)` loads a group of previously multiple aligned sequences into the viewer. *Alignment* is a structure with a field *Sequence*, a character array, or a file name.

`multialignviewer(Alignment, 'Alphabet', AlphabetValue)` specifies the alphabet type for the sequences. *AlphabetValue* can be 'AA' for amino acids or 'NT' for nucleotides. Default is 'AA'. If *AlphabetValue* is not specified, `multialignviewer` guesses the alphabet type.

Examples

```
multialignviewer('aagag.aln')
```

See Also

Bioinformatics Toolbox functions: `fastaread`, `gethmmalignment`, `multialign`, `multialignread`, `multialignwrite`, `seqtool`

Purpose Write multiple alignment to file using ClustalW ALN format

Syntax

```
multialignwrite(File, Alignment)  
multialignwrite(..., 'Header', HeaderValue, ...)  
multialignwrite(..., 'WriteCount', WriteCountValue, ...)
```

Arguments

<i>File</i>	String specifying either a file name or a path and file name for saving the ClustalW ALN-formatted data. If you specify only a file name, the file is saved to the MATLAB Current Directory.
<i>Alignment</i>	An alignment, such as returned by the <code>multialign</code> function, represented by either a: <ul style="list-style-type: none">• Vector of structures, each containing the fields <code>Header</code> and <code>Sequence</code>.• Character array
<i>HeaderValue</i>	String that specifies the first line of the file. Default is 'MATLAB multiple sequence alignment'.
<hr/> Tip Use the 'Header' property if your file header needs to be a specific format for a third-party software application. <hr/>	
<i>WriteCountValue</i>	Determines whether to add the residue counts to the end of each line. Choices are <code>true</code> (default) or <code>false</code> .

Description `multialignwrite(File, Alignment)` writes the contents of *Alignment* to *File*, a ClustalW ALN-formatted file.

multialignwrite

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

`multialignwrite(..., 'Header', HeaderValue, ...)` specifies the first line of the file. Default is 'MATLAB multiple sequence alignment'.

Tip Use the 'Header' property if your file header needs to be a specific format for a third-party software application.

`multialignwrite(..., 'WriteCount', WriteCountValue, ...)` determines whether to add the residue counts to the end of each line. Choices are `true` (default) or `false`.

Note Below the columns of the ClustalW ALN-formatted file, symbols may appear that denote:

- * — Residues or nucleotides in the column are identical in all sequences in the alignment.
- : — Conserved substitutions exist in the column for all sequences in the alignment.
- . — Semiconserved substitutions exist in the column for all sequences in the alignment.

For more information on these symbols, see <http://www.ebi.ac.uk/help/formats.html#aln>.

For more information on the groups of residues considered conserved and semiconserved, see section 12 of "Changes since version 1.6" at http://web.mit.edu/clustalw_v1.83/README.

Examples

- 1 Use the `fastaread` function to read `p53samples.txt`, a FASTA-formatted file included with the Bioinformatics Toolbox software, which contains seven cellular tumor antigen p53 sequences.

```
p53 = fastaread('p53samples.txt')
```

```
p53 =
```

```
7x1 struct array with fields:
```

```
Header
```

```
Sequence
```

- 2 Use the `multialign` function to align the seven cellular tumor antigen p53 sequences.

```
ma = multialign(p53,'verbose',true);
```

- 3 Write the alignment to a file named `p53.aln`.

multialignwrite

```
multialignwrite('p53.aln',ma)
```

See Also

Bioinformatics Toolbox functions: `fastawrite`, `gethmmalignment`, `multialign`, `multialignread`, `multialignviewer`, `phytreewrite`, `seqconsensus`, `seqdisp`, `seqprofile`

Purpose	Convert mzCDF structure to peak list
Syntax	<code>[Peaks, Times] = mzcdf2peaks(mzCDFStruct)</code>
Arguments	<p><i>mzCDFStruct</i> MATLAB structure containing information from a netCDF file, such as one created by the <code>mzcdfread</code> function. Its fields correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with the <code>_attributes</code> string. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are <code>mass_values</code> and <code>intensity_values</code> fields.</p>
Return Values	<p><i>Peaks</i> Either of the following:</p> <ul style="list-style-type: none">• Two-column matrix, where the first column contains mass/charge (m/z) values and the second column contains ion intensity values.• Cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time. <p><i>Times</i> Scalar or vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. If <i>Times</i> is a vector, the number of elements equals the number of peak lists contained in <i>Peaks</i>.</p>
Description	<code>[Peaks, Times] = mzcdf2peaks(mzCDFStruct)</code> extracts peak information from <i>mzCDFStruct</i> , a MATLAB structure containing

information from a netCDF file, such as one created by the `mzcdfread` function, and creates *Peaks*, a single matrix or a cell array of matrices containing mass/charge (m/z) values and ion intensity values, and *Times*, a scalar or vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set.

mzCDFStruct contains fields that correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with the `_attributes` string. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are `mass_values` and `intensity_values` fields.

Examples

In the following example, the file `results.cdf` is not provided.

- 1 Use the `mzcdfread` function to read a netCDF file into the MATLAB software as a structure. Then extract the peak information from the structure.

```
mzcdf_struct = mzcdfread('results.cdf');  
[peaks,time] = mzcdf2peaks(mzcdf_struct)
```

```
peaks =
```

```
    [7008x2 single]  
    [7008x2 single]  
    [7008x2 single]  
    [7008x2 single]
```

```
time =
```

```
    8.3430  
   12.6130  
   16.8830  
   21.1530
```

- 2 Create a color map containing a color for each peak list (retention time).

```
colors = hsv(numel(peaks));
```

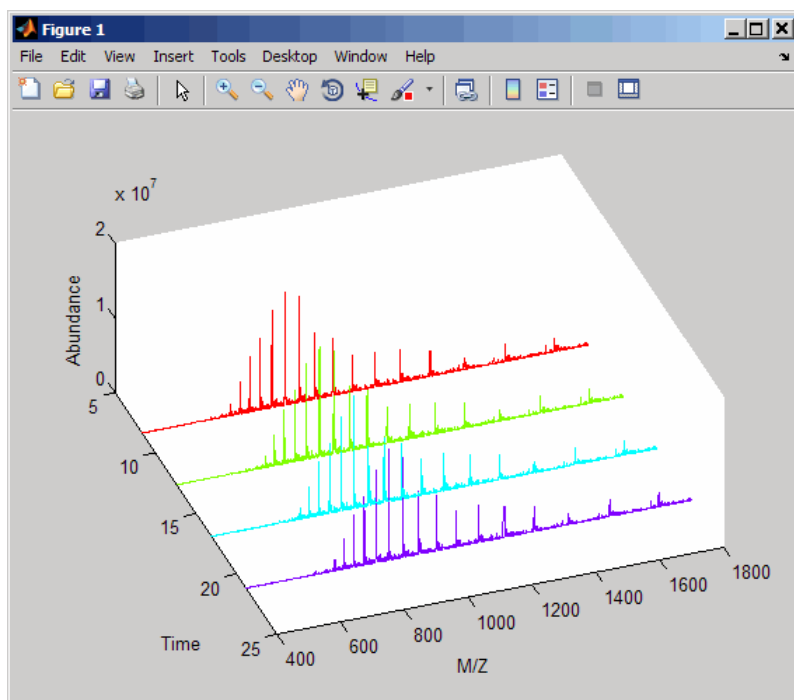
- 3 Create a 3-D figure of the peaks and add labels to it.

```
figure
hold on

for i = 1:numel(peaks)
    t = repmat(time(i),size(peaks{i},1),1);
    plot3(t,peaks{i}(:,1),peaks{i}(:,2),'color',colors(i,:))
end

view(70,60)
xlabel('Time')
ylabel(mzcdf_struct.mass_axis_label)
zlabel(mzcdf_struct.intensity_axis_label)
```

mzcdf2peaks



See Also

Bioinformatics Toolbox functions: `mshotplot`, `mshalign`, `mshpresample`, `mzcdfread`

Purpose Return information about netCDF file containing mass spectrometry data

Syntax `InfoStruct = mzcdinfo(File)`

Arguments

<i>File</i>	String containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data and conforms to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications.
	If you specify only a file name, that file must be on the MATLAB search path or in the current directory.

Return Values

<i>InfoStruct</i>	MATLAB structure containing information from a netCDF file. It includes the fields in the following table.
-------------------	--

Description `InfoStruct = mzcdinfo(File)` returns a MATLAB structure, *InfoStruct*, containing summary information about a netCDF file, *File*.

File is a string containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data. The file must conform to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications.

InfoStruct includes the following fields.

Field	Description
Filename	Name of the netCDF file.

Field	Description
FileTimeStamp	Date time stamp of the netCDF file.
FileSize	Size of the file in bytes.
NumberOfScans	Number of scans in the file.
StartTime	Run start time.
EndTime	Run end time.
TimeUnits	Units for time.
GlobalMassMin	Minimum m/z value in all scans.
GlobalMassMax	Maximum m/z value in all scans.
GlobalIntensityMin	Minimum intensity value in all scans.
GlobalIntensityMax	Maximum intensity value in all scans.
ExperimentType	Indicates if data is raw or centroided.

Note If any of the associated attributes are not in the netCDF file (because they are optional in the specifications), the value for that field will be set to N/A or NaN.

Examples

In the following example, the file `results.cdf` is not provided.

Return a MATLAB structure containing summary information about a netCDF file.

```
info = mzcdfinfo('results.cdf')
```

```
info =
```

```
Filename: 'results.cdf'  
FileTimeStamp: '19930703134354-700'  
FileSize: 339892  
NumberOfScans: 4  
StartTime: 8.3430  
EndTime: 21.1530  
TimeUnits: 'N/A'  
GlobalMassMin: 399.9990  
GlobalMassMax: 1.8000e+003  
GlobalIntensityMin: NaN  
GlobalIntensityMax: NaN  
ExperimentType: 'Continuum Mass Spectrum'
```

See AlsoBioinformatics Toolbox function: `mzcdfread`

mzcdfread

Purpose

Read mass spectrometry data from netCDF file

Syntax

```
mzCDFStruct = mzcdfread(File)  
mzCDFStruct = mzcdfread(File, ...'TimeRange',  
    TimeRangeValue,  
    ...)  
mzCDFStruct = mzcdfread(File, ...'ScanIndices',  
    ScanIndicesValue, ...)  
mzCDFStruct = mzcdfread(File, ...'Verbose', VerboseValue,  
    ...)
```


Arguments*File*

String containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data and conforms to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications.

If you specify only a file name, that file must be on the MATLAB search path or in the current directory.

TimeRangeValue

Two-element numeric array [Start End] that specifies the time range in *File* for which to read spectra. Default is to read spectra from all times [0 Inf].

Tip Time units are indicated in the netCDF global attributes. For summary information about the time ranges in a netCDF file, use the `mzcdfinfo` function.

Note If you specify a *TimeRangeValue*, you cannot specify a *ScanIndicesValue*.

ScanIndicesValue Positive integer, vector of integers, or a two-element numeric array [Start_Ind End_Ind] that specifies a scan, multiple scans, or a range of scans in *File* to read. Start_Ind and End_Ind are each positive integers indicating a scan index number. Start_Ind must be less than End_Ind. Default is to read all scans.

Tip For information about the scan indices in a netCDF file, check the NumberOfScans field in the structure returned by the `mzcdfinfo` function.

Note If you specify a *ScanIndicesValue*, you cannot specify *TimeRangeValue*.

VerboseValue Controls the display of the progress of the reading of *File*. Choices are true (default) or false.

Return Values

mzCDFStruct MATLAB structure containing mass spectrometry information from a netCDF file. Its fields correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with the `_attributes` string. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are `mass_values` and `intensity_values` fields.

Description

`mzCDFStruct = mzcdfread(File)` reads a netCDF file, *File*, and then creates a MATLAB structure, *mzCDFStruct*.

File is a string containing a file name, or a path and file name, of a netCDF file that contains mass spectrometry data. The file must conform to the ANDI/MS or the ASTM E2077-00 (2005) standard specification or earlier specifications.

mzCDFStruct contains fields that correspond to the variables and global attributes in a netCDF file. If a netCDF variable contains local attributes, an additional field is created, with the name of the field being the variable name appended with the `_attributes` string. The number and names of the fields will vary, depending on the mass spectrometer software, but typically there are `mass_values` and `intensity_values` fields.

Tip LC/MS data analysis requires extended amounts of memory from the operating system. If you receive any errors related to memory, try the following:

- Increase the virtual memory (or swap space) of your operating system or set the 3 GB switch (32-bit Windows XP only) as described at:

<http://www.mathworks.com/support/tech-notes/1100/1107.html>

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

mzCDFStruct = `mzcdfread(File, ...'TimeRange', TimeRangeValue, ...)` specifies the range of time in *File* to read. *TimeRangeValue* is a two-element numeric array [Start End]. Default is to read spectra from all times [0 Inf].

Tip Time units are indicated in the netCDF global attributes. For summary information about the time ranges in a netCDF file, use the `mzcdfinfo` function.

Note If you specify a *TimeRangeValue*, you cannot specify *ScanIndicesValue*.

`mzCDFStruct = mzcdfread(File, ...'ScanIndices', ScanIndicesValue, ...)` specifies a scan, multiple scans, or range of scans in *File* to read. *ScanIndicesValue* is a positive integer, vector of integers, or a two-element numeric array [Start_Ind End_Ind]. Start_Ind and End_Ind are each positive integers indicating a scan index number. Start_Ind must be less than End_Ind. Default is to read all scans.

Tip For information about the scan indices in a netCDF file, check the `NumberOfScans` field in the structure returned by the `mzcdfinfo` function.

Note If you specify a *ScanIndicesValue*, you cannot specify a *TimeRangeValue*.

`mzCDFStruct = mzcdfread(File, ...'Verbose', VerboseValue, ...)` controls the progress display when reading *File*. Choices are true (default) or false.

Examples

In the following example, the file `results.cdf` is not provided.

- 1 Read a netCDF file into the MATLAB software as a structure.

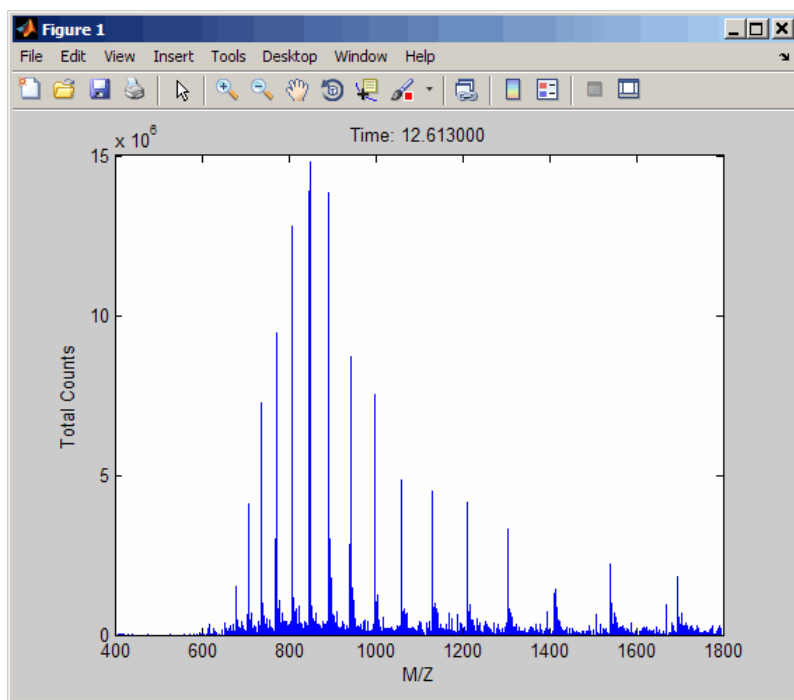
```
out = mzcdfread('results.cdf');
```

- 2 View the second scan in the netCDF file by creating separate variables containing the intensity and m/z values, and then plotting these values. Add a title and x - and y -axis labels using fields in the output structure.

```
idx1 = out.scan_index(2)+1;
idx2 = out.scan_index(3);
y = out.intensity_values(idx1:idx2);
z = out.mass_values(idx1:idx2);
stem(z,y,'marker','none')

title(sprintf('Time: %f',out.scan_acquisition_time(2)))
xlabel(out.mass_axis_units)
ylabel(out.intensity_axis_units)
```

mzcdfread



See Also

Bioinformatics Toolbox functions: `jcampread`, `mzcdf2peaks`, `mzcdfinfo`, `mzxmlread`

Purpose	Convert mzXML structure to peak list
Syntax	<pre>[Peaks, Times] = mzxml2peaks(mzXMLStruct) [Peaks, Times] = mzxml2peaks(mzXMLStruct, 'Levels', LevelsValue)</pre>
Arguments	<p><i>mzXMLStruct</i> MATLAB structure containing information from an mzXML file, such as one created by the <code>mzxmlread</code> function. It includes the fields shown in the table below.</p> <p><i>LevelsValue</i> Positive integer or vector of integers that specifies the level(s) of spectra in <i>mzXMLStruct</i> to convert, assuming the spectra are from tandem MS data sets. Default is 1, which converts only the first-level spectra, that is, spectra containing precursor ions. Setting <i>LevelsValue</i> to 2 converts only the second-level spectra, which are the fragment spectra (created from a precursor ion).</p>
Return Values	<p><i>Peaks</i> Either of the following:</p> <ul style="list-style-type: none">• Two-column matrix, where the first column contains mass/charge (m/z) values and the second column contains ion intensity values.• Cell array of peak lists, where each element is a two-column matrix of m/z values and ion intensity values, and each element corresponds to a spectrum or retention time. <p><i>Times</i> Vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. The number of elements in <i>Times</i> equals the number of elements in <i>Peaks</i>.</p>

Description

`[Peaks, Times] = mzxml2peaks(mzXMLStruct)` extracts peak information from `mzXMLStruct`, a MATLAB structure containing information from an mzXML file, such as one created by the `mzxmlread` function, and creates `Peaks`, a cell array of matrices containing mass/charge (m/z) values and ion intensity values, and `Times`, a vector of retention times associated with a liquid chromatography/mass spectrometry (LC/MS) or gas chromatography/mass spectrometry (GC/MS) data set. `mzXMLStruct` includes the following fields:

Field	Description
scan	Structure array containing the data pertaining to each individual scan, such as mass spectrometry level, total ion current, polarity, precursor mass (when it applies), and the spectrum data.
index	Structure containing indices to the positions of scan elements in the XML document.
mzXML	Structure containing: <ul style="list-style-type: none">• Information in the root element of the mzXML schema, such as instrument details, experiment details, and preprocessing method• URLs pointing to schemas for the individual scans• Indexing approach• Digital signature calculated for the current instance of the document

`[Peaks, Times] = mzxml2peaks(mzXMLStruct, 'Levels', LevelsValue)` specifies the level(s) of the spectra in `mzXMLStruct` to convert, assuming the spectra are from tandem MS data sets. Default is 1, which converts only the first-level spectra, that is, spectra containing precursor ions. Setting `LevelsValue` to 2 converts only the second-level spectra, which are the fragment spectra (created from a precursor ion).

Examples

Note In the following example, the file `results.mzxml` is not provided. Sample mzXML files can be found at:

- Open Proteomics Database
 - Peptide Atlas Repository at the Institute for Systems Biology (ISB)
 - The Sashimi Project
-

- 1 Use the `mzxmlread` function to read an mzXML file into the MATLAB software as structure. Then extract the peak information of only the first-level ions from the structure.

```
mzxml_struct = mzxmlread('results.mzxml');  
[peaks,time] = mzxml2peaks(mzxml_struct);
```

- 2 Create a dot plot of the LC/MS data.

```
msdotplot(peaks,time)
```

See Also

Bioinformatics Toolbox functions: `msdotplot`, `mssalign`, `mssppresample`, `mzxmlread`

mzxmlinfo

Purpose Return information about mzXML file

Syntax `InfoStruct = mzxmlinfo(File)`
`InfoStruct = mzxmlinfo(File, 'NumOfLevels',
NumOfLevelsValue)`

Arguments

<i>File</i>	String containing a file name, or a path and file name, of an mzXML file that conforms to the mzXML 2.1 specification or earlier specifications. If you specify only a file name, that file must be on the MATLAB search path or in the current directory.
<i>NumOfLevelsValue</i>	Controls the return of <code>NumOfLevels</code> , an additional field in <i>InfoStruct</i> , that contains the number of mass spectrometry (MS) levels of spectra in <i>File</i> . Choices are <code>true</code> or <code>false</code> (default).

Return Values

<i>InfoStruct</i>	MATLAB structure containing information from an mzXML file. It includes the fields shown in the table below.
-------------------	--

Description `InfoStruct = mzxmlinfo(File)` returns a MATLAB structure, *InfoStruct*, containing summary information about an mzXML file, *File*.

File is a string containing a file name, or a path and file name, of an mzXML file. The file must conform to the mzXML 2.1 specification or earlier specifications. You can view the mzXML 2.1 specification at:

http://sashimi.sourceforge.net/schema_revision/mzXML_2.1/Doc/mzXML_2.1_tutorial.pdf

InfoStruct includes the following fields.

Field	Description
Filename	Name of the mzXML file.
FileModDate	Modification date of the file.
FileSize	Size of the file in bytes.
NumberOfScans	Number of scans in the file.*
StartTime	Run start time.*
EndTime	Run end time.*
DataProcessingIntensityCutoff	Minimum mass/charge (m/z) intensity value.*
DataProcessingCentroided	Indicates if data is centroided.*
DataProcessingDeisotoped	Indicates if data is deisotoped.*
DataProcessingChargeDeconvoluted	Indicates if data is deconvoluted.*
DataProcessingSpotIntegration	For LC/MALDI experiments, indicates if peaks eluting over multiple spots have been integrated into a single spot.*

* — These fields contain N/A if the mzXML file does not include the associated attributes. The associated attributes are optional in the mzXML file, per the mzXML 2.1 specification.

InfoStruct = `mzxmlinfo(File, 'NumOfLevels', NumOfLevelsValue)` controls the return of `NumOfLevels`, an additional field in *mzXMLInfo*, that contains the number of mass spectrometry levels of spectra in *File*. Choices are `true` or `false` (default).

Examples

Note In the following example, the file `results.mzxml` is not provided. Sample mzXML files can be found at:

- Open Proteomics Database
 - Peptide Atlas Repository at the Institute for Systems Biology (ISB)
 - The Sashimi Project
-

Return a MATLAB structure containing summary information about an mzXML file.

```
info = mzxmlinfo('results.mzxml');

info =

    Filename: 'results.mzxml'
  FileModDate: '07-May-2008 13:39:12'
    FileSize: 10607
  NumberOfScans: 2
    StartTime: 'PT0.00683333S'
    EndTime: 'PT200.036S'
  DataProcessingIntensityCutoff: 'N/A'
  DataProcessingCentroided: 'false'
  DataProcessingDeisotoped: 'N/A'
  DataProcessingChargeDeconvoluted: 'N/A'
  DataProcessingSpotIntegration: 'N/A'
```

Return a MATLAB structure containing summary information, including the number of mass spectrometry levels, about an mzXML file.

```
info = mzxmlinfo('results.mzxml','numoflevels',true);

info =

    Filename: 'results.mzxml'
```

```
FileModDate: '07-May-2008 13:39:12'  
  FileSize: 10607  
  NumberOfScans: 2  
    StartTime: 'PT0.00683333S'  
    EndTime: 'PT200.036S'  
DataProcessingIntensityCutoff: 'N/A'  
  DataProcessingCentroided: 'false'  
  DataProcessingDeisotoped: 'N/A'  
DataProcessingChargeDeconvoluted: 'N/A'  
  DataProcessingSpotIntegration: 'N/A'  
    NumberOfMSLevels: 2
```

See AlsoBioinformatics Toolbox function: `mzxmlread`

mzxmlread

Purpose Read data from mzXML file

Syntax

```
mzXMLStruct = mzxmlread(File)
mzXMLStruct = mzxmlread(File, ...'Levels',
LevelsValue, ...)
mzXMLStruct = mzxmlread(File, ...'TimeRange',
TimeRangeValue,
...)
mzXMLStruct = mzxmlread(File, ...'ScanIndices',
ScanIndicesValue, ...)
mzXMLStruct = mzxmlread(File, ...'Verbose', VerboseValue,
...)
```

Arguments

<i>File</i>	String containing a file name, or a path and file name, of an mzXML file that conforms to the mzXML 2.1 specification or earlier specifications. If you specify only a file name, that file must be on the MATLAB search path or in the current directory.
<i>LevelsValue</i>	Positive integer or vector of integers specifying the level(s) of spectra in <i>File</i> to read. Default is to read all levels of spectra.

Tip For summary information about the levels of spectra in an mzXML file, use the `mzxmlinfo` function.

Note If you specify a *LevelsValue*, you cannot specify *TimeRangeValue* or *ScanIndicesValue*.

TimeRangeValue Two-element numeric array [Start End] that specifies the range of time in *File* to read. Start is a scalar between the startTime and endTime attributes of the msRun element in *File*. End is a scalar between Start and the endTime attribute of the msRun element in *File*. Default is to read spectra from all times.

Tip For summary information about the time ranges in an mzXML file, use the `mzxmlinfo` function.

Note If you specify a *TimeRangeValue*, you cannot specify *LevelsValue* or *ScanIndicesValue*.

ScanIndicesValue Positive integer, vector of integers, or a two-element numeric array [*Start_Ind* *End_Ind*] that specifies a scan, multiple scans, or a range of scans in *File* to read. *Start_Ind* and *End_Ind* are each positive integers indicating a scan number in the *num* attribute of the *msRun* element in *File*. *Start_Ind* must be less than *End_Ind*. Default is to read all scans.

Tip For summary information about the scan indices in an mzXML file, use the `mzxmlinfo` function.

Note If you specify a *ScanIndicesValue*, you cannot specify *LevelsValue* or *TimeRangeValue*.

VerboseValue Controls the display of the progress of the reading of *File*. Choices are `true` (default) or `false`.

Return Values

mzXMLStruct MATLAB structure containing information from an mzXML file. It includes the fields shown in the table below.

Description

`mzXMLStruct = mzxmlread(File)` reads an mzXML file, *File*, and then creates a MATLAB structure, *mzXMLStruct*.

File is a string containing a file name, or a path and file name, of an mzXML file. The file must conform to the mzXML 2.1 specification or earlier specifications. You can view the mzXML 2.1 specification at:

http://sashimi.sourceforge.net/schema_revision/mzXML_2.1/Doc/mzXML_2.1_tutorial.pdf

mzXMLStruct includes the following fields.

Field	Description
scan	Structure array containing the data pertaining to each individual scan, such as mass spectrometry level, total ion current, polarity, precursor mass (when it applies), and the spectrum data.
index	Structure containing indices to the positions of scan elements in the XML document.
mzXML	Structure containing: <ul style="list-style-type: none">• Information in the root element of the mzXML schema, such as instrument details, experiment details, and preprocessing method• URLs pointing to schemas for the individual scans• Indexing approach• Digital signature calculated for the current instance of the document

Tip LC/MS data analysis requires extended amounts of memory from the operating system. If you receive any errors related to memory or Java heap space, try the following:

- Increase the virtual memory (or swap space) of your operating system or set the 3 GB switch (32-bit Windows XP only) as described at:

<http://www.mathworks.com/support/tech-notes/1100/1107.html>

- Increase your Java heap space as described at:

<http://www.mathworks.com/support/solutions/data/1-18I2C.html>

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

`mzXMLStruct = mzxmlread(File, ...'Levels', LevelsValue, ...)` specifies the level(s) of spectra in *File* to read. *LevelsValue* is a positive integer or vector of integers. Default is to read all levels of spectra.

Tip For summary information about the levels of spectra in an mzXML file, use the `mzxmlinfo` function.

Note If you specify a *LevelsValue*, you cannot specify *TimeRangeValue* or *ScanIndicesValue*.

`mzXMLStruct = mzxmlread(File, ...'TimeRange', TimeRangeValue, ...)` specifies the range of time in *File* to read. *TimeRangeValue* is two-element numeric array [Start End]. Start is a scalar between the `startTime` and `endTime` attributes of the `msRun` element in *File*. End is a scalar between Start and the `endTime` attribute of the `msRun` element in *File*. Default is to read spectra from all times.

Tip For summary information about the time ranges in an mzXML file, use the `mzxmlinfo` function.

Note If you specify a *TimeRangeValue*, you cannot specify *LevelsValue* or *ScanIndicesValue*.

mzXMLStruct = `mzxmlread(File, ...'ScanIndices', ScanIndicesValue, ...)` specifies a scan, multiple scans, or range of scans in *File* to read. *ScanIndicesValue* is a positive integer, vector of integers, or a two-element numeric array [Start_Ind End_Ind]. Start_Ind and End_Ind are each positive integers indicating a scan number in the num attribute of the msRun element in *File*. Start_Ind must be less than End_Ind. Default is to read all scan.

Tip For summary information about the scan indices in an mzXML file, use the `mzxmlinfo` function.

Note If you specify a *ScanIndicesValue*, you cannot specify *LevelsValue* or *TimeRangeValue*.

mzXMLStruct = `mzxmlread(File, ...'Verbose', VerboseValue, ...)` controls the display of the progress of the reading of *File*. Choices are true (default) or false.

Examples

Note In the following example, the file `results.mzxml` is not provided. Sample mzXML files can be found at:

- Open Proteomics Database
 - Peptide Atlas Repository at the Institute for Systems Biology (ISB)
 - The Sashimi Project
-

mzxmlread

- 1 Read an mzXML file into the MATLAB software as a structure.

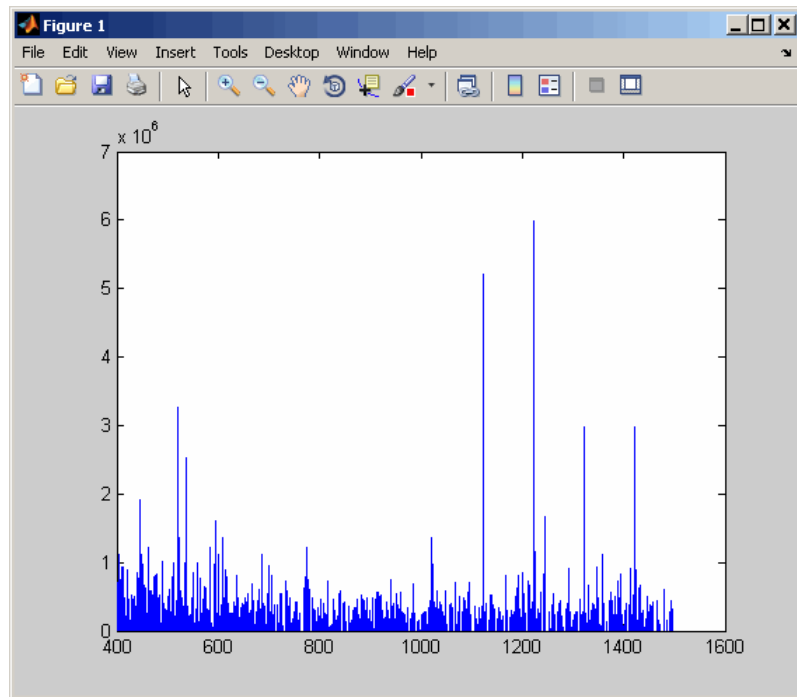
```
out = mzxmlread('results.mzxml')
```

```
out =
```

```
    mzXML: [1x1 struct]  
    index: [1x1 struct]  
    scan: [2000x1 struct]
```

- 2 View the first scan in the mzXML file by creating separate variables containing the mass and charge values respectively, and then plotting these values.

```
m = out.scan(1).peaks.mz(1:2:end);  
z = out.scan(1).peaks.mz(2:2:end);  
stem(m,z,'marker','none')
```

**See Also**

Bioinformatics Toolbox functions: `jcampread`, `mzcdfread`, `mzxm12peaks`, `mzxmlinfo`

MATLAB function: `xmlread`

nmercount

Purpose Count n-mers in nucleotide or amino acid sequence

Syntax
Nmer = nmercount(*Seq*, *Length*)
Nmer = nmercount(*Seq*, *Length*, *C*)

Arguments

<i>Seq</i>	One of the following: <ul style="list-style-type: none">• String of codes specifying a nucleotide sequence or amino acid sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers on page 2-794 or the table Mapping Amino Acid Letter Codes to Integers on page 2-2.• MATLAB structure containing a <i>Sequence</i> field that contains a nucleotide sequence or an amino acid sequence, such as returned by <i>fastaread</i>, <i>emblread</i>, <i>getembl</i>, <i>genbankread</i>, <i>getgenbank</i>, <i>getgenpept</i>, <i>genpeptread</i>, <i>getpdb</i>, or <i>pdbread</i>.
<i>Length</i>	Integer specifying the length of n-mer to count.

Return Values

<i>Nmer</i>	Cell array containing the n-mer counts in <i>Seq</i> .
-------------	--

Description

Nmer = nmercount(*Seq*, *Length*) counts the n-mers or patterns of a specific length in *Seq*, a nucleotide sequence or amino acid sequence, and returns the n-mer counts in a cell array.

Nmer = nmercount(*Seq*, *Length*, *C*) returns only the n-mers with cardinality of at least *C*.

Examples

1 Use the *getgenpept* function to retrieve the amino acid sequence for the human insulin receptor.

```
S = getgenpept('AAA59174', 'SequenceOnly', true);
```

- 2** Count the number of four-mers in the amino acid sequence and display the first 20 rows in the cell array.

```
nmers = nmercount(S,4);  
nmers(1:20,:)
```

```
ans =  
    'APES'    [2]  
    'DFRD'    [2]  
    'ESLK'    [2]  
    'FRDL'    [2]  
    'GNYS'    [2]  
    'LKEL'    [2]  
    'SHCQ'    [2]  
    'SLKD'    [2]  
    'SVRI'    [2]  
    'TDYL'    [2]  
    'TSLA'    [2]  
    'TVIN'    [2]  
    'VING'    [2]  
    'VPLD'    [2]  
    'YALV'    [2]  
    'AAAA'    [1]  
    'AAP'     [1]  
    'AAEI'    [1]  
    'AAEL'    [1]  
    'AAFP'    [1]
```

See Also

Bioinformatics Toolbox functions: `aaccount`, `basecount`, `codoncount`, `dimercount`

nt2aa

Purpose

Convert nucleotide sequence to amino acid sequence

Syntax

```
SeqAA = nt2aa(SeqNT)
SeqAA = nt2aa(..., 'Frame', FrameValue, ...)
SeqAA = nt2aa(..., 'GeneticCode', GeneticCodeValue, ...)
SeqAA = nt2aa(..., 'AlternativeStartCodons',
               AlternativeStartCodonsValue, ...)
```


Arguments

SeqNT

One of the following:

- Character string of codes specifying a nucleotide sequence
- MATLAB structure containing a `Sequence` field that contains a nucleotide sequence, such as returned by `fastaread`, `emblread`, `getembl`, `genbankread`, or `getgenbank`

Valid characters include:

- A
- C
- G
- T
- U
- hyphen (-)

Note Hyphens are valid only if the codon to which it belongs represents a gap, that is, the codon contains all hyphens.
Example: ACT---TGA

Tip Do not use a sequence with hyphens if you specify 'all' for *FrameValue*.

FrameValue

Integer or string specifying a reading frame in the nucleotide sequence. Choices are 1, 2, 3, or 'all'. Default is 1.

If *FrameValue* is 'all', then *SeqAA* is a 3-by-1 cell array.

GeneticCodeValue

Integer or string specifying a genetic code number or code name from the table Genetic Code on page 2-787. Default is 1 or 'Standard'.

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

AlternativeStartCodonsValue Controls the translation of alternative codons. Choices are true (default) or false.

Return Values

SeqAA

Amino acid sequence specified by a character string of single-letter codes.

Description

SeqAA = nt2aa(*SeqNT*) converts a nucleotide sequence, specified by *SeqNT*, to an amino acid sequence, returned in *SeqAA*, using the standard genetic code.

SeqAA = nt2aa(*SeqNT*, ...'*PropertyName*', *PropertyValue*, ...) calls nt2aa with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

SeqAA = nt2aa(..., '*Frame*', *FrameValue*, ...) converts a nucleotide sequence for a specific reading frame to an amino acid sequence. Choices are 1, 2, 3, or 'all'. Default is 1. If *FrameValue* is 'all', then output *SeqAA* is a 3-by-1 cell array.

`SeqAA = nt2aa(..., 'GeneticCode', GeneticCodeValue, ...)` specifies a genetic code to use when converting a nucleotide sequence to an amino acid sequence. *GeneticCodeValue* can be an integer or string specifying a code number or code name from the table Genetic Code on page 2-787. Default is 1 or 'Standard'. The amino acid to nucleotide codon mapping for the Standard genetic code is shown in the table Standard Genetic Code on page 2-788.

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

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

If this option is set to `false`, then an alternative start codon at the start of a sequence is translated to its corresponding amino acid in the genetic code that you specify, which might not necessarily be methionine. For example, in the human mitochondrial genetic code, AUA and AUU are known to be alternative start codons.

For more information about alternative start codons, see:

www.ncbi.nlm.nih.gov/Taxonomy/Uutils/wprintgc.cgi?mode=t#SG1

Genetic Code

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

Genetic Code (Continued)

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

Standard Genetic Code

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

Standard Genetic Code (Continued)

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

Standard Genetic Code (Continued)

Amino Acid Name	Amino Acid Code	Nucleotide Codon
Glutamine or Glutamic acid (Glutamate)	Z	Random codon from E and Q
Unknown amino acid (any amino acid)	X	Random codon
Translation stop	*	TAA TAG TGA
Gap of indeterminate length	-	---
Unknown character (any character or symbol not in table)	?	???

Examples

Converting the ND1 Gene

- 1 Use the `getgenbank` function to retrieve the nucleotide sequence for the human mitochondrion from the GenBank database.

```
mitochondria = getgenbank('NC_001807', 'SequenceOnly', true);
```

- 2 Extract the sequence for the ND1 gene from the nucleotide sequence.

```
ND1gene = mitochondria (3308:4261);
```

- 3 Convert the ND1 gene on the human mitochondria genome to an amino acid sequence using the Vertebrate Mitochondrial genetic code.

```
protein1 = nt2aa(ND1gene, 'GeneticCode', 2);
```

- 4 Use the `getgenpept` function to retrieve the same amino acid sequence from the GenPept database.

```
protein2 = getgenpept('NP_536843', 'SequenceOnly', true);
```

- 5 Use the `isequal` function to compare the two amino acid sequences.

```
isequal (protein1, protein2)
```

```
ans =
```

```
1
```

Converting the ND2 Gene

- 1 Use the `getgenbank` function to retrieve the nucleotide sequence for the human mitochondrion from the GenBank database.

```
mitochondria = getgenbank('NC_001807', 'SequenceOnly', true);
```

- 2 Extract the sequence for the ND2 gene from the nucleotide sequence.

```
ND2gene = mitochondria (4471:5511);
```

- 3 Convert the ND2 gene on the human mitochondria genome to an amino acid sequence using the Vertebrate Mitochondrial genetic code.

```
protein1 = nt2aa(ND2gene, 'GeneticCode', 2);
```

Note In the ND2gene nucleotide sequence, the first codon is ATT, which is translated to M, while the subsequent ATT codons are translated to I. If you set 'AlternativeStartCodons' to false, then the first ATT codon is translated to I, the corresponding amino acid in the Vertebrate Mitochondrial genetic code.

- 4** Use the `getgenpept` function to retrieve the same amino acid sequence from the GenPept database.

```
protein2 = getgenpept('NP_536844', 'SequenceOnly', true);
```

- 5** Use the `isequal` function to compare the two amino acid sequences.

```
isequal (protein1, protein2)
```

```
ans =
```

```
1
```

See Also

Bioinformatics Toolbox functions: `aa2nt`, `aminolookup`, `baselookup`, `codonbias`, `dnds`, `dndsml`, `geneticcode`, `revgeneticcode`, `seqtool`

Purpose	Convert nucleotide sequence from letter to integer representation	
Syntax	<pre>SeqInt = nt2int(SeqChar) SeqInt = nt2int(SeqChar, ...'Unknown', UnknownValue, ...) SeqInt = nt2int(SeqChar, ...'ACGTOnly', ACGTOnlyValue, ...)</pre>	
Arguments	<i>SeqChar</i>	One of the following: <ul style="list-style-type: none">• String of codes specifying a nucleotide sequence. For valid letter codes, see the table Mapping Nucleotide Letter Codes to Integers on page 2-794. Integers are arbitrarily assigned to IUB/IUPAC letters.• MATLAB structure containing a <i>Sequence</i> field that contains a nucleotide sequence, such as returned by <i>fastaread</i>, <i>emblread</i>, <i>getembl</i>, <i>genbankread</i>, or <i>getgenbank</i>.
	<i>UnknownValue</i>	Integer to represent unknown nucleotides. Choices are integers ≥ 0 and ≤ 255 . Default is 0.
	<i>ACGTOnlyValue</i>	Controls the prohibition of ambiguous nucleotides. Choices are <i>true</i> or <i>false</i> (default). If <i>ACGTOnlyValue</i> is <i>true</i> , you can enter only the characters A, C, G, T, and U.
Return Values	<i>SeqInt</i>	Nucleotide sequence specified by a row vector of integers.
Description	<i>SeqInt</i> = nt2int(<i>SeqChar</i>) converts <i>SeqChar</i> , a string of codes specifying a nucleotide sequence, to <i>SeqInt</i> , a row vector of integers specifying the same nucleotide sequence. For valid codes, see the	

table Mapping Nucleotide Letter Codes to Integers on page 2-794. Unknown characters (characters not in the table) are mapped to 0. Gaps represented with hyphens are mapped to 16.

`SeqInt = nt2int(SeqChar, ...'PropertyName', PropertyValue, ...)` calls `nt2int` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`SeqInt = nt2int(SeqChar, ...'Unknown', UnknownValue, ...)` specifies an integer to represent unknown nucleotides. *UnknownValue* can be an integer ≥ 0 and ≤ 255 . Default is 0.

`SeqInt = nt2int(SeqChar, ...'ACGTOnly', ACGTOnlyValue, ...)` controls the prohibition of ambiguous nucleotides (N, R, Y, K, M, S, W, B, D, H, and V). Choices are `true` or `false` (default). If *ACGTOnlyValue* is `true`, you can enter only the characters A, C, G, T, and U.

Mapping Nucleotide Letter Codes to Integers

Nucleotide	Code	Integer
Adenosine	A	1
Cytidine	C	2
Guanine	G	3
Thymidine	T	4
Uridine (if 'Alphabet' set to 'RNA')	U	4
Purine (A or G)	R	5
Pyrimidine (T or C)	Y	6
Keto (G or T)	K	7
Amino (A or C)	M	8
Strong interaction (3 H bonds) (G or C)	S	9

Mapping Nucleotide Letter Codes to Integers (Continued)

Nucleotide	Code	Integer
Weak interaction (2 H bonds) (A or T)	W	10
Not A (C or G or T)	B	11
Not C (A or G or T)	D	12
Not G (A or C or T)	H	13
Not T or U (A or C or G)	V	14
Any nucleotide (A or C or G or T or U)	N	15
Gap of indeterminate length	-	16
Unknown (any character not in table)	*	0 (default)

Examples**Converting a Simple Sequence**

Convert a nucleotide sequence from letters to integers.

```
s = nt2int('ACTGCTAGC')
```

```
s =
    1    2    4    3    2    4    1    3    2
```

Converting a Random Sequence

- 1 Create a random character string to represent a nucleotide sequence.

```
SeqChar = randseq(20)
```

```
SeqChar =
```

```
TTATGACGTTATTCTACTTT
```

- 2 Convert the nucleotide sequence from letter to integer representation.

```
SeqInt = nt2int(SeqChar)
```

nt2int

SeqInt =

Columns 1 through 13

4 4 1 4 3 1 2 3 4 4 1 4 4

Columns 14 through 20

2 4 1 2 4 4 4

See Also

Bioinformatics Toolbox functions: [aa2int](#), [baselookup](#), [int2aa](#), [int2nt](#)

Purpose

Plot density of nucleotides along sequence

Syntax

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

Description

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

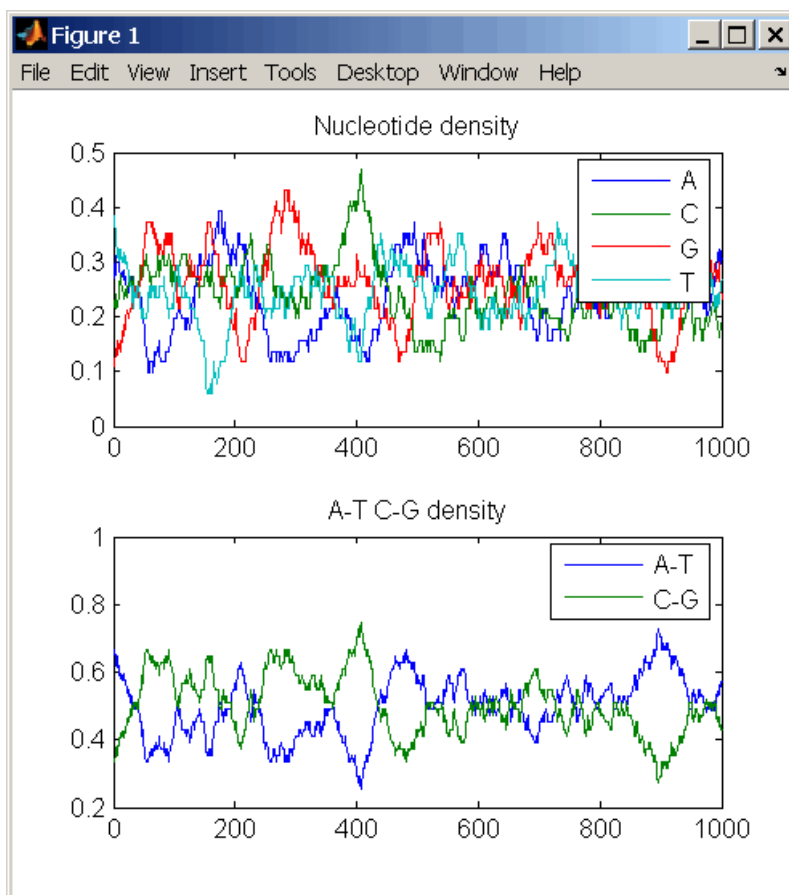
Density = ntdensity(SeqNT, 'PropertyName', PropertyValue) returns a MATLAB structure with the density of nucleotides A, C, G, and T.

ntdensity(..., 'Window', WindowValue) uses a window of length Window for the density calculation. The default value is length(SeqNT)/20.

[Density, HighCG] = ntdensity(..., 'CGThreshold', CGThresholdValue) returns indices for regions where the CG content of SeqNT is greater than CGThreshold. The default value for CGThreshold is 5.

Examples

```
s = randseq(1000, 'alphabet', 'dna');
ntdensity(s)
```



See Also

Bioinformatics Toolbox functions: `basecount`, `codoncount`, `cpgisland`, `dimercount`

MATLAB function: `filter`

Purpose	Return NUC44 scoring matrix for nucleotide sequences
Syntax	<pre>ScoringMatrix = nuc44 [ScoringMatrix, MatrixInfo] = nuc44</pre>
Description	<p><i>ScoringMatrix</i> = nuc44 returns the scoring matrix. The nuc44 scoring matrix uses ambiguous nucleotide codes and probabilities rounded to the nearest integer.</p> <p>Scale = 0.277316</p> <p>Expected score = -1.7495024, Entropy = 0.5164710 bits</p> <p>Lowest score = -4, Highest score = 5</p> <p>Order: A C G T R Y K M S W B D H V N</p> <p>[<i>ScoringMatrix</i>, <i>MatrixInfo</i>] = nuc44 returns a structure with information about the matrix with fields Name and Order.</p>

num2goid

Purpose Convert numbers to Gene Ontology IDs

Syntax `GOIDs = num2goid(X)`

Description `GOIDs = num2goid(X)` converts the numbers in `X` to strings with Gene Ontology IDs. IDs are a 7-digit number preceded by the prefix 'GO:'.

Examples Get the Gene Ontology IDs of the following numbers.

```
t = [5575 5622 5623 5737 5840 30529 43226 43228 ...  
     43229 43232 43234];  
ids = num2goid(t)
```

See Also Bioinformatics Toolbox functions: `geneont` (object constructor), `goannotread`
Bioinformatics Toolbox methods of `geneont` object: `getancestors`, `getdescendants`, `getmatrix`, `getrelatives`

Purpose

Globally align two sequences using Needleman-Wunsch algorithm

Syntax

```
Score = nwalgn(Seq1,Seq2)
[Score, Alignment] = nwalgn(Seq1,Seq2)
[Score, Alignment, Start] = nwalgn(Seq1,Seq2)
... = nwalgn(Seq1,Seq2, ...'Alphabet', AlphabetValue, ...)
... = nwalgn(Seq1,Seq2, ...'ScoringMatrix',
    ScoringMatrixValue, ...)
... = nwalgn(Seq1,Seq2, ...'Scale', ScaleValue, ...)
... = nwalgn(Seq1,Seq2, ...'GapOpen', GapOpenValue, ...)
... = nwalgn(Seq1,Seq2, ...'ExtendGap',
    ExtendGapValue, ...)
... = nwalgn(Seq1,Seq2, ...'Showscore',
    ShowscoreValue, ...)
```

Arguments

Seq1, Seq2

Amino acid or nucleotide sequences. Enter any of the following:

- Character string of letters representing amino acids or nucleotides, such as returned by `int2aa` or `int2nt`
- Vector of integers representing amino acids or nucleotides, such as returned by `aa2int` or `nt2int`
- Structure containing a `Sequence` field

Tip For help with letter and integer representations of amino acids and nucleotides, see [Amino Acid Lookup](#) on page 2-91 or [Nucleotide Lookup](#) on page 2-102.

AlphabetValue

String specifying the type of sequence. Choices are 'AA' (default) or 'NT'.

ScoringMatrixValue String specifying the scoring matrix to use for the global alignment. Choices for amino acid sequences are:

- 'PAM40'
- 'PAM250'
- 'DAYHOFF'
- 'GONNET'
- 'BLOSUM30' increasing by 5 up to 'BLOSUM90'
- 'BLOSUM62'
- 'BLOSUM100'

Default is:

- 'BLOSUM50' (when *AlphabetValue* equals 'AA')
- 'NUC44' (when *AlphabetValue* equals 'NT')

Note All of the above scoring matrices have a built-in scale factor that returns *Score* in bits.

ScaleValue Positive value that specifies the scale factor used to return *Score* in arbitrary units other than bits. For example, if you enter $\log(2)$ for *ScaleValue*, then *nwalign* returns *Score* in nats.

GapOpenValue Positive integer specifying the penalty for opening a gap in the alignment. Default is 8.

<i>ExtendGapValue</i>	Positive integer specifying the penalty for extending a gap. Default is equal to <i>GapOpenValue</i> .
<i>ShowscoreValue</i>	Controls the display of the scoring space and the winning path of the alignment. Choices are true or false (default).

Return Values

<i>Score</i>	Optimal global alignment score in bits.
<i>Alignment</i>	3-by-N character array showing the two sequences, <i>Seq1</i> and <i>Seq2</i> , in the first and third rows, and symbols representing the optimal global alignment for them in the second row.
<i>Start</i>	2-by-1 vector of indices indicating the starting point in each sequence for the alignment. Because this is a global alignment, <i>Start</i> is always [1;1].

Description

Score = `nwalgn(Seq1,Seq2)` returns the optimal global alignment score in bits. The scale factor used to calculate the score is provided by the scoring matrix.

`[Score, Alignment]` = `nwalgn(Seq1,Seq2)` returns a 3-by-N character array showing the two sequences, *Seq1* and *Seq2*, in the first and third rows, and symbols representing the optimal global alignment for them in the second row. The symbol | indicates amino acids or nucleotides that match exactly. The symbol : indicates amino acids or nucleotides that are related as defined by the scoring matrix (nonmatches with a zero or positive scoring matrix value).

`[Score, Alignment, Start]` = `nwalgn(Seq1,Seq2)` returns a 2-by-1 vector of indices indicating the starting point in each sequence for the alignment. Because this is a global alignment, *Start* is always [1;1].

... = nwalign(Seq1,Seq2, ...'*PropertyName*',
PropertyValue, ...) calls nwalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = nwalign(Seq1,Seq2, ...'*Alphabet*',
AlphabetValue, ...) specifies the type of sequences. Choices are 'AA' (default) or 'NT'.

... = nwalign(Seq1,Seq2,
...'*ScoringMatrix*', *ScoringMatrixValue*, ...) specifies the scoring matrix to use for the global alignment. Default is:

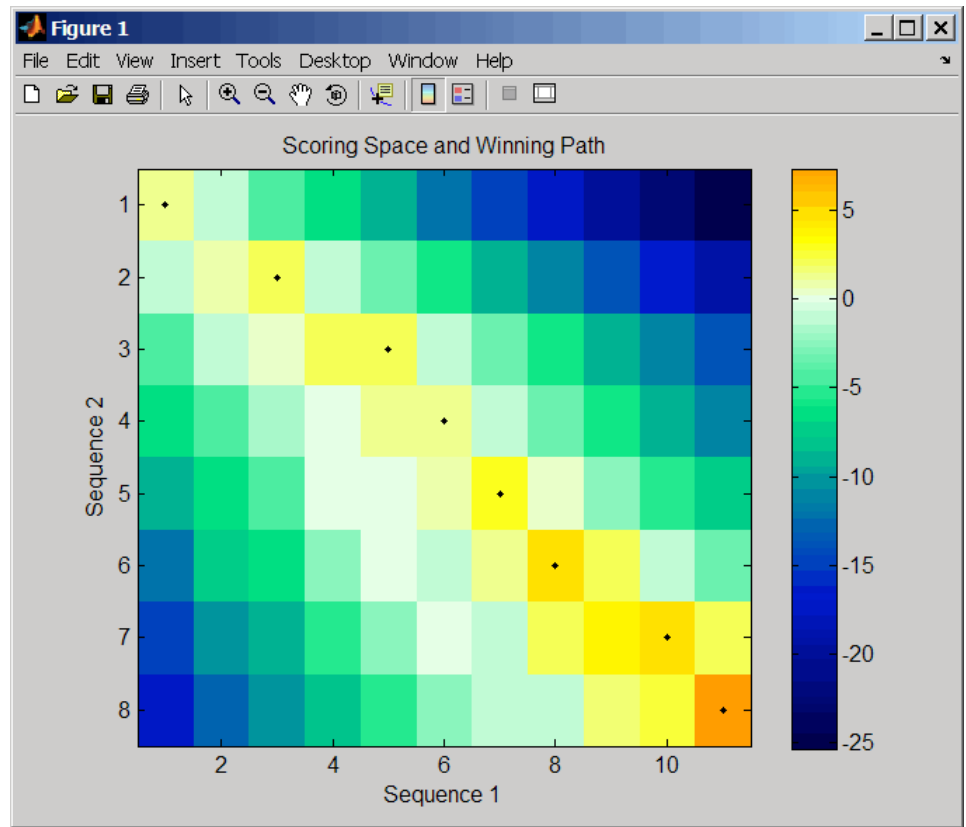
- 'BLOSUM50' (when *AlphabetValue* equals 'AA')
- 'NUC44' (when *AlphabetValue* equals 'NT')

... = nwalign(Seq1,Seq2, ...'*Scale*', *ScaleValue*, ...) specifies the scale factor used to return *Score* in arbitrary units other than bits. Choices are any positive value.

... = nwalign(Seq1,Seq2, ...'*GapOpen*', *GapOpenValue*, ...) specifies the penalty for opening a gap in the alignment. Choices are any positive integer. Default is 8.

... = nwalign(Seq1,Seq2, ...'*ExtendGap*',
ExtendGapValue, ...) specifies the penalty for extending a gap in the alignment. Choices are any positive integer. Default is equal to *GapOpenValue*.

... = nwalign(Seq1,Seq2, ...'*Showscore*',
ShowscoreValue, ...) controls the display of the scoring space and winning path of the alignment. Choices are true or false (default)



The scoring space is a heat map displaying the best scores for all the partial alignments of two sequences. The color of each $(n1, n2)$ coordinate in the scoring space represents the best score for the pairing of subsequences $Seq1(1:n1)$ and $Seq2(1:n2)$, where $n1$ is a position in $Seq1$ and $n2$ is a position in $Seq2$. The best score for a pairing of specific subsequences is determined by scoring all possible alignments of the subsequences by summing matches and gap penalties.

The winning path is represented by black dots in the scoring space and represents the pairing of positions in the optimal global alignment. The color of the last point (lower right) of the winning path represents the

optimal global alignment score for the two sequences and is the *Score* output returned by *nwalign*.

Tip The scoring space visually indicates if there are potential alternate winning paths, which is useful when aligning sequences with big gaps. Visual patterns in the scoring space can also indicate a possible sequence rearrangement.

Examples

- 1 Globally align two amino acid sequences using the BLOSUM50 (default) scoring matrix and the default values for the *GapOpen* and *ExtendGap* properties. Return the optimal global alignment score in bits and the alignment character array.

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

```
7.3333
```

```
Alignment =
```

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

- 2 Globally align two amino acid sequences specifying the PAM250 scoring matrix and a gap open penalty of 5.

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

```
Score =
```

```
2.3333
```

```
Alignment =
```

```

IGRHRHYHIG-G
:  ||  ||  |
-S--RY-IGRG

```

- 3** Globally align two amino acid sequences returning the *Score* in nat units (nats) by specifying a scale factor of $\log(2)$.

```
[Score, Alignment] = nwalgn('HEAGAWGHEE', 'PAWHEAE', 'Scale', log(2))
```

```
Score =
```

```
0.2310
```

```
Alignment =
```

```

HEAGAWGHE-E
  ||  ||  |
--P-AW-HEAE

```

References

[1] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). Biological Sequence Analysis (Cambridge University Press).

See Also

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

Purpose

Calculate sequence properties of DNA oligonucleotide

Syntax

```
SeqProperties = oligoprop(SeqNT)
SeqProperties = oligoprop(SeqNT, ...'Salt', SaltValue, ...)
SeqProperties = oligoprop(SeqNT, ...'Temp', TempValue, ...)
SeqProperties = oligoprop(SeqNT, ...'Primerconc',
    PrimerconcValue, ...)
SeqProperties = oligoprop(SeqNT, ...'HPBase', HPBaseValue,
    ...)
SeqProperties = oligoprop(SeqNT, ...'HPLoop', HPLoopValue,
    ...)
SeqProperties = oligoprop(SeqNT, ...'Dimerlength',
    DimerlengthValue, ...)
```

Arguments

<i>SeqNT</i>	DNA oligonucleotide sequence represented by any of the following: <ul style="list-style-type: none">• Character string containing the letters A, C, G, T, or N• Vector of integers containing the integers 1, 2, 3, 4, or 15• Structure containing a <code>Sequence</code> field that contains a nucleotide sequence
<i>SaltValue</i>	Value that specifies a salt concentration in moles/liter for melting temperature calculations. Default is 0.05 moles/liter.
<i>TempValue</i>	Value that specifies the temperature in degrees Celsius for nearest-neighbor calculations of free energy. Default is 25 degrees Celsius.
<i>PrimerconcValue</i>	Value that specifies the concentration in moles/liter for melting temperature calculations. Default is 50e-6 moles/liter.

HPBaseValue Value that specifies the minimum number of paired bases that form the neck of the hairpin. Default is 4 base pairs.

HPLoopValue Value that specifies the minimum number of bases that form the loop of a hairpin. Default is 2 bases.

DimerlengthValue Value that specifies the minimum number of aligned bases between the sequence and its reverse. Default is 4 bases.

Return Values

SeqProperties Structure containing the sequence properties for a DNA oligonucleotide.

Description

SeqProperties = `oligoprop(SeqNT)` returns the sequence properties for a DNA oligonucleotide as a structure with the following fields:

Field	Description
GC	Percent GC content for the DNA oligonucleotide. Ambiguous N characters in <i>SeqNT</i> are considered to potentially be any nucleotide. If <i>SeqNT</i> contains ambiguous N characters, GC is the midpoint value, and its uncertainty is expressed by GCdelta.
GCdelta	The difference between GC (midpoint value) and either the maximum or minimum value GC could assume. The maximum and minimum values are calculated by assuming all N characters are G/C or not G/C, respectively. Therefore, GCdelta defines the possible range of GC content.

Field	Description
Hairpins	H-by-length(<i>SeqNT</i>) matrix of characters displaying all potential hairpin structures for the sequence <i>SeqNT</i> . Each row is a potential hairpin structure of the sequence, with the hairpin forming nucleotides designated by capital letters. H is the number of potential hairpin structures for the sequence. Ambiguous N characters in <i>SeqNT</i> are considered to potentially complement any nucleotide.
Dimers	D-by-length(<i>SeqNT</i>) matrix of characters displaying all potential dimers for the sequence <i>SeqNT</i> . Each row is a potential dimer of the sequence, with the self-dimerizing nucleotides designated by capital letters. D is the number of potential dimers for the sequence. Ambiguous N characters in <i>SeqNT</i> are considered to potentially complement any nucleotide.
MolWeight	Molecular weight of the DNA oligonucleotide. Ambiguous N characters in <i>SeqNT</i> are considered to potentially be any nucleotide. If <i>SeqNT</i> contains ambiguous N characters, MolWeight is the midpoint value, and its uncertainty is expressed by MolWeightdelta.
MolWeightdelta	The difference between MolWeight (midpoint value) and either the maximum or minimum value MolWeight could assume. The maximum and minimum values are calculated by assuming all N characters are G or C, respectively. Therefore, MolWeightdelta defines the possible range of molecular weight for <i>SeqNT</i> .

Field	Description
Tm	<p>A vector with melting temperature values, in degrees Celsius, calculated by six different methods, listed in the following order:</p> <ul style="list-style-type: none">• Basic (Marmur et al., 1962)• Salt adjusted (Howley et al., 1979)• Nearest-neighbor (Breslauer et al., 1986)• Nearest-neighbor (SantaLucia Jr. et al., 1996)• Nearest-neighbor (SantaLucia Jr., 1998)• Nearest-neighbor (Sugimoto et al., 1996) <p>Ambiguous N characters in <i>SeqNT</i> are considered to potentially be any nucleotide. If <i>SeqNT</i> contains ambiguous N characters, Tm is the midpoint value, and its uncertainty is expressed by Tmdelta.</p>
Tmdelta	<p>A vector containing the differences between Tm (midpoint value) and either the maximum or minimum value Tm could assume for each of the six methods. Therefore, Tmdelta defines the possible range of melting temperatures for <i>SeqNT</i>.</p>

Field	Description
Thermo	<p>4-by-3 matrix of thermodynamic calculations. The rows correspond to nearest-neighbor parameters from:</p> <ul style="list-style-type: none"> • Breslauer et al., 1986 • SantaLucia Jr. et al., 1996 • SantaLucia Jr., 1998 • Sugimoto et al., 1996 <p>The columns correspond to:</p> <ul style="list-style-type: none"> • delta H — Enthalpy in kilocalories per mole, kcal/mol • delta S — Entropy in calories per mole-degrees Kelvin, cal/(K)(mol) • delta G — Free energy in kilocalories per mole, kcal/mol <p>Ambiguous N characters in <i>SeqNT</i> are considered to potentially be any nucleotide. If <i>SeqNT</i> contains ambiguous N characters, <i>Thermo</i> is the midpoint value, and its uncertainty is expressed by <i>Thermodelta</i>.</p>
Thermodelta	<p>4-by-3 matrix containing the differences between <i>Thermo</i> (midpoint value) and either the maximum or minimum value <i>Thermo</i> could assume for each calculation and method. Therefore, <i>Thermodelta</i> defines the possible range of thermodynamic values for <i>SeqNT</i>.</p>

SeqProperties = oligoprop(*SeqNT*, ...'*PropertyName*', *PropertyValue*, ...) calls oligoprop with optional properties that

use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

SeqProperties = oligoprop(*SeqNT*, ...'Salt', *SaltValue*, ...) specifies a salt concentration in moles/liter for melting temperature calculations. Default is 0.05 moles/liter.

SeqProperties = oligoprop(*SeqNT*, ...'Temp', *TempValue*, ...) specifies the temperature in degrees Celsius for nearest-neighbor calculations of free energy. Default is 25 degrees Celsius.

SeqProperties = oligoprop(*SeqNT*, ...'Primerconc', *PrimerconcValue*, ...) specifies the concentration in moles/liter for melting temperatures. Default is 50e-6 moles/liter.

SeqProperties = oligoprop(*SeqNT*, ...'HPBase', *HPBaseValue*, ...) specifies the minimum number of paired bases that form the neck of the hairpin. Default is 4 base pairs.

SeqProperties = oligoprop(*SeqNT*, ...'HPLoop', *HPLoopValue*, ...) specifies the minimum number of bases that form the loop of a hairpin. Default is 2 bases.

SeqProperties = oligoprop(*SeqNT*, ...'Dimerlength', *DimerlengthValue*, ...) specifies the minimum number of aligned bases between the sequence and its reverse. Default is 4 bases.

Examples

Calculating Properties for a DNA Sequence

- 1 Create a random sequence.

```
seq = randseq(25)
```

```
seq =
```

```
TAGCTTCATCGTTGACTTCTACTAA
```

- 2 Calculate sequence properties of the sequence.

```
S1 = oligoprop(seq)

S1 =

          GC: 36
      GCAlpha: 0
    Hairpins: [0x25 char]
      Dimers: 'tAGCTtcatcgttgacttctactaa'
    MolWeight: 7.5820e+003
MolWeightAlpha: 0
          Tm: [52.7640 60.8629 62.2493 55.2870 54.0293 61.0614]
      TmAlpha: [0 0 0 0 0 0]
        Thermo: [4x3 double]
    ThermoAlpha: [4x3 double]
```

3 List the thermodynamic calculations for the sequence.

```
S1.Thermo

ans =

-178.5000 -477.5700 -36.1125
-182.1000 -497.8000 -33.6809
-190.2000 -522.9000 -34.2974
-191.9000 -516.9000 -37.7863
```

Calculating Properties for a DNA Sequence with Ambiguous Characters

1 Calculate sequence properties of the sequence ACGTAGAGGACGTN.

```
S2 = oligoprop('ACGTAGAGGACGTN')

S2 =

          GC: 53.5714
      GCAlpha: 3.5714
    Hairpins: 'ACGTagaggACGTn'
```

```
Dimers: [3x14 char]
MolWeight: 4.3329e+003
MolWeightAlpha: 20.0150
Tm: [38.8357 42.2958 57.7880 52.4180 49.9633 55.1330]
TmAlpha: [1.4643 1.4643 10.3885 3.4633 0.2829 3.8074]
Thermo: [4x3 double]
ThermoAlpha: [4x3 double]
```

2 List the potential dimers for the sequence.

S2.Dimers

ans =

```
ACGTagaggacgtn
ACGTagaggACGTn
acgtagagGACGTN
```

References

- [1] Breslauer, K.J., Frank, R., Blöcker, H., and Marky, L.A. (1986). Predicting DNA duplex stability from the base sequence. *Proceedings of the National Academy of Science USA* *83*, 3746–3750.
- [2] Chen, S.H., Lin, C.Y., Cho, C.S., Lo, C.Z., and Hsiung, C.A. (2003). Primer Design Assistant (PDA): A web-based primer design tool. *Nucleic Acids Research* *31(13)*, 3751–3754.
- [3] Howley, P.M., Israel, M.A., Law, M., and Martin, M.A. (1979). A rapid method for detecting and mapping homology between heterologous DNAs. Evaluation of polyomavirus genomes. *The Journal of Biological Chemistry* *254(11)*, 4876–4883.
- [4] Marmur, J., and Doty, P. (1962). Determination of the base composition of deoxyribonucleic acid from its thermal denaturation temperature. *Journal Molecular Biology* *5*, 109–118.

[5] Panjkovich, A., and Melo, F. (2005). Comparison of different melting temperature calculation methods for short DNA sequences. *Bioinformatics* *21(6)*, 711–722.

[6] SantaLucia Jr., J., Allawi, H.T., and Seneviratne, P.A. (1996). Improved Nearest-Neighbor Parameters for Predicting DNA Duplex Stability. *Biochemistry* *35*, 3555–3562.

[7] SantaLucia Jr., J. (1998). A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proceedings of the National Academy of Science USA* *95*, 1460–1465.

[8] Sugimoto, N., Nakano, S., Yoneyama, M., and Honda, K. (1996). Improved thermodynamic parameters and helix initiation factor to predict stability of DNA duplexes. *Nucleic Acids Research* *24(22)*, 4501–4505.

[9] <http://www.basic.northwestern.edu/biotools/oligocalc.html> for weight calculations.

See Also

Bioinformatics Toolbox functions: `isoelectric`, `molweight`, `ntdensity`, `palindromes`, `randseq`

Purpose Determine optimal leaf ordering for hierarchical binary cluster tree

Syntax

```
Order = optimalleaforder(Tree, Dist)
Order = optimalleaforder(Tree, Dist, ...'Criteria',
CriteriaValue, ...)
Order = optimalleaforder(Tree, Dist, ...'Transformation',
TransformationValue, ...)
```

Arguments

<i>Tree</i>	Hierarchical binary cluster tree represented by an $(M - 1)$ -by-3 matrix, created by the linkage function, where M is the number of leaves.
<i>Dist</i>	Distance matrix, such as that created by the <code>pdist</code> function.

optimalleaforder

CriteriaValue String that specifies the optimization criteria. Choices are:

- `adjacent` (default) — Minimizes the sum of distances between adjacent leaves.
- `group` — Minimizes the sum of distances between every leaf and all other leaves in the adjacent cluster.

TransformationValue Either of the following:

- String that specifies the algorithm to transform the distances in *Dist* into similarity values. Choices are:
 - `linear` (default) — Similarity = $\max(\text{all distances}) - \text{distance}$
 - `quadratic` — Similarity = $(\max(\text{all distances}) - \text{distance})^2$
 - `inverse` — Similarity = $1/\text{distance}$
- A function handle created using `@` to a function that transforms the distances in *Dist* into similarity values. The function is typically a monotonic decreasing function within the range of the distance values. The function must accept a vector input and return a vector of the same size.

Return Values

Order Optimal leaf ordering for the hierarchical binary cluster tree represented by *Tree*.

Description

Order = `optimalleaforder(Tree, Dist)` returns the optimal leaf ordering for the hierarchical binary cluster tree represented by *Tree*, an $(M - 1)$ -by-3 matrix, created by the `linkage` function, where M is the number of leaves. Optimal leaf ordering of a binary tree maximizes the

similarity between adjacent elements (clusters or leaves) by flipping tree branches, but without dividing the clusters. The input *Dist* is a distance matrix, such as that created by the `pdist` function.

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

`Order = optimalleaforder(Tree, Dist, ...'Criteria', CriteriaValue, ...)` specifies the optimization criteria.

`Order = optimalleaforder(Tree, Dist, ...'Transformation', TransformationValue, ...)` specifies the algorithm to transform the distances in *Dist* into similarity values. The transformation is necessary because `optimalleaforder` maximizes the similarity between adjacent elements, which is comparable to minimizing the sum of distances between adjacent elements.

Examples

- 1 Use the `rand` function to create a 10-by-2 matrix of random values.

```
X = rand(10,2);
```

- 2 Use the `pdist` function to create a distance matrix containing the city block distances between the pairs of objects in matrix *X*.

```
Dist = pdist(X,'cityblock');
```

- 3 Use the `linkage` function to create a matrix, *Tree*, that represents a hierarchical binary cluster tree, from the distance matrix, *Dist*.

```
Tree = linkage(Dist,'average');
```

- 4 Use the `optimalleaforder` function to determine the optimal leaf ordering for the hierarchical binary cluster tree represented by *Tree*, using the distance matrix *Dist*.

```
order = optimalleaforder(Tree,Dist)
```

optimalleaforder

References

[1] Bar-Joseph, Z., Gifford, D.K., and Jaakkola, T.S. (2001). Fast optimal leaf ordering for hierarchical clustering. *Bioinformatics* 17, Suppl 1:S22–9. PMID: 11472989.

See Also

Bioinformatics Toolbox function: `clustergram`

Statistics Toolbox functions: `linkage`, `pdist`

Purpose Find palindromes in sequence

Syntax

```
[Position, Length] = palindromes(SeqNT, 'PropertyName',
    PropertyValue)
[Position, Length, Pal] = palindromes(SeqNT)
palindromes(..., 'Length', LengthValue)
palindromes(..., 'Complement', ComplementValue)
```

Description

[Position, Length] = palindromes(SeqNT, 'PropertyName', PropertyValue) finds all palindromes in sequence SeqNT with a length greater than or equal to 6, and returns the starting indices, Position, and the lengths of the palindromes, Length.

[Position, Length, Pal] = palindromes(SeqNT) also returns a cell array Pal of the palindromes.

palindromes(..., 'Length', LengthValue) finds all palindromes longer than or equal to Length. The default value is 6.

palindromes(..., 'Complement', ComplementValue) finds complementary palindromes if Complement is true, that is, where the elements match their complementary pairs A-T(or U) and C-G instead of an exact nucleotide match.

Examples

```
[p,l,s] = palindromes('GCTAGTAACGTATATATAAT')
```

```
p =
    11
    12
l =
     7
     7
s =
    'TATATAT'
    'ATATATA'
```

```
[pc,lc,sc] = palindromes('GCTAGTAACGTATATATAAT',...
    'Complement',true);
```

palindromes

Find the palindromes in a random nucleotide sequence.

```
a = randseq(100)

a =
TAGCTTCATCGTTGACTTCTACTAA
AAGCAAGCTCCTGAGTAGCTGGCCA
AGCGAGCTTGCTTGTGCCGGCTGC
GGCGGTTGTATCCTGAATAGCCAT

[pos,len,pal]=palindromes(a)

pos =
    74
len =
    6
pal =
    'GCGGCG'
```

See Also

Bioinformatics Toolbox functions: `seqrcomplement`, `seqshowwords`
MATLAB functions: `regexp`, `strfind`

Purpose

Return PAM scoring matrix

Syntax

```
ScoringMatrix = pam(N)
[ScoringMatrix, MatrixInfo] = pam(N)
... = pam(N, ...'Extended', ExtendedValue, ...)
... = pam(N, ...'Order', OrderValue, ...)
```

Arguments

<i>N</i>	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 *.
<i>ExtendedValue</i>	Property to add ambiguous characters to the scoring matrix. Enter either true or false. Default is false.
<i>OrderValue</i>	Property to control the order of amino acids in the scoring matrix. Enter a string with at least the 20 standard amino acids.

Description

ScoringMatrix = pam(*N*) returns a PAM scoring matrix for amino acid sequences.

[*ScoringMatrix*, *MatrixInfo*] = pam(*N*) returns a structure with information about the PAM matrix. The fields in the structure are Name, Scale, Entropy, Expected, and Order.

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

... = pam(*N*, ...'Extended', *ExtendedValue*, ...) if Extended is true, returns a scoring matrix with the 20 amino acid characters, the

ambiguous characters, and stop character (B, Z, X, *), . If `Extended` is `false`, only the standard 20 amino acids are included in the matrix.

`...` = `pam(N, ... 'Order', OrderValue, ...)` returns a PAM matrix ordered by the amino acid sequence in `Order`. If `Order` does not contain the extended characters B, Z, X, and *, then these characters are not returned.

PAM50 substitution matrix in 1/2 bit units, Expected score = -3.70, Entropy = 2.00 bits, Lowest score = -13, Highest score = 13.

PAM250 substitution matrix in 1/3 bit units, Expected score = -0.844, Entropy = 0.354 bits, Lowest score = -8, Highest score = 17.

Examples

Return a PAM matrix with `N = 50`.

```
PAM50 = pam(50)
```

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

See Also

Bioinformatics Toolbox functions: `blosum`, `dayhoff`, `gonnet`, `nalign`, `swalign`

Purpose Visualize intermolecular distances in Protein Data Bank (PDB) file

Syntax

```
pdbdistplot('PDBid')  
pdbdistplot('PDBid', Distance)
```

Arguments

PDBid	Unique identifier for a protein structure record. Each structure in the PDB is represented by a 4-character alphanumeric identifier. For example, 4hbb 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);
```

pdbdistplot

See Also

Bioinformatics Toolbox functions: `getpdb`, `molviewer`, `pdbread`, `proteinplot`, `ramachandran`

Purpose Read data from Protein Data Bank (PDB) file

Syntax
`PDBStruct = pdbread(File)`
`PDBStruct = pdbread(File, 'ModelNum', ModelNumValue)`

Arguments

<i>File</i>	Either of the following: <ul style="list-style-type: none"> • String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a Protein Data Bank (PDB)-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Directory. • MATLAB character array that contains the text of a PDB-formatted file.
<i>ModelNumValue</i>	Positive integer specifying a model in a PDB-formatted file.

Return Values

<i>PDBStruct</i>	MATLAB structure containing a field for each PDB record.
------------------	--

Description The Protein Data Bank (PDB) database is an archive of experimentally determined 3-D biological macromolecular structure data. For more information about the PDB format, see:

[http://www.wwpdb.org/documentation/format23/v2.3.html](http://www ww pdb .org /documentation /format23 /v2.3 .html)

`PDBStruct = pdbread(File)` reads the data from PDB-formatted text file *File* and stores the data in the MATLAB structure, *PDBStruct*, which contains a field for each PDB record. The following table summarizes

the possible PDB records and the corresponding fields in the MATLAB structure *PDBStruct*:

PDB Database Record	Field in the MATLAB Structure
HEADER	Header
OBSLTE	Obsolete
TITLE	Title
CAVEAT	Caveat
COMPND	Compound
SOURCE	Source
KEYWDS	Keywords
EXPDTA	ExperimentData
AUTHOR	Authors
REVDAT	RevisionDate
SPRSDE	Superseded
JRNL	Journal
REMARK 1	Remark1
REMARK <i>N</i>	Remark <i>n</i>
Note <i>N</i> equals 2 through 999.	Note <i>n</i> equals 2 through 999.
DBREF	DBReferences
SEQADV	SequenceConflicts
SEQRES	Sequence
FTNOTE	Footnote
MODRES	ModifiedResidues

PDB Database Record	Field in the MATLAB Structure
HET	Heterogen
HETNAM	HeterogenName
HETSYN	HeterogenSynonym
FORMUL	Formula
HELIX	Helix
SHEET	Sheet
TURN	Turn
SSBOND	SSBond
LINK	Link
HYDBND	HydrogenBond
SLTBRG	SaltBridge
CISPEP	CISPeptides
SITE	Site
CRYST1	Cryst1
ORIGXn	OriginX
SCALEn	Scale
MTRIXn	Matrix
TVECT	TranslationVector
MODEL	Model
ATOM	Atom
SIGATM	AtomSD
ANISOU	AnisotropicTemp
SIGUIJ	AnisotropicTempSD
TER	Terminal

PDB Database Record	Field in the MATLAB Structure
HETATM	HeterogenAtom
CONECT	Connectivity

PDBStruct = `pdbread(File, 'ModelNum', ModelNumValue)` reads only the model specified by *ModelNumValue* from the PDB-formatted text file *File* and stores the data in the MATLAB structure *PDBStruct*. If *ModelNumValue* does not correspond to an existing mode number in *File*, then `pdbread` reads the coordinate information of all the models.

The Sequence Field

The `Sequence` field is also a structure containing sequence information in the following subfields:

- `NumOfResidues`
- `ChainID`
- `ResidueNames` — Contains the three-letter codes for the sequence residues.
- `Sequence` — Contains the single-letter codes for the sequence residues.

Note If the sequence has modified residues, then the `ResidueNames` subfield might not correspond to the standard three-letter amino acid codes. In this case, the `Sequence` subfield will contain the modified residue code in the position corresponding to the modified residue. The modified residue code is provided in the `ModifiedResidues` field.

The Model Field

The `Model` field is also a structure or an array of structures containing coordinate information. If the MATLAB structure contains one model, the `Model` field is a structure containing coordinate information for that model. If the MATLAB structure contains multiple models, the `Model`

field is an array of structures containing coordinate information for each model. The Model field contains the following subfields:

- Atom
- AtomSD
- AnisotropicTemp
- AnisotropicTempSD
- Terminal
- HeterogenAtom

The Atom Field

The Atom field is also an array of structures containing the following subfields:

- AtomSerNo
- AtomName
- altLoc
- resName
- chainID
- resSeq
- iCode
- X
- Y
- Z
- occupancy
- tempFactor
- segID
- element

- charge
- AtomNameStruct — Contains three subfields: chemSymbol, remoteInd, and branch.

Examples

- 1 Use the `getpdb` function to retrieve structure information from the Protein Data Bank (PDB) for the nicotinic receptor protein with identifier 1abt, and then save the data to the PDB-formatted file `nicotinic_receptor.pdb` in the MATLAB Current Directory.

```
getpdb('1abt', 'ToFile', 'nicotinic_receptor.pdb');
```

- 2 Read the data from the `nicotinic_receptor.pdb` file into a MATLAB structure `pdbstruct`.

```
pdbstruct = pdbread('nicotinic_receptor.pdb');
```

- 3 Read only the second model from the `nicotinic_receptor.pdb` file into a MATLAB structure `pdbstruct_Model12`.

```
pdbstruct_Model12 = pdbread('nicotinic_receptor.pdb', 'ModelNum', 2);
```

- 4 View the atomic coordinate information in the model fields of both MATLAB structures `pdbstruct` and `pdbstruct_Model12`.

```
pdbstruct.Model
```

```
ans =
```

```
1x4 struct array with fields:
```

```
MDLSerNo
```

```
Atom
```

```
Terminal
```

```
pdbstruct_Model12.Model
```

```
ans =
```

```
MDLSerNo: 2
```



```
Atom: [1x1205 struct]  
Terminal: [1x2 struct]
```

- 5 Read the data from an URL into a MATLAB structure, `gf1_pdbstruct`.

```
gf1_pdbstruct = pdbread('http://www.rcsb.org/pdb/files/1gf1.pdb')
```

See Also

Bioinformatics Toolbox functions: `genpeptread`, `getpdb`, `molviewer`, `pdbdistplot`, `pdbsuperpose`, `pdbtransform`, `pdbwrite`

pdbsuperpose

Purpose Superpose 3-D structures of two proteins

Syntax

```
pdbsuperpose(PDB1, PDB2)
Dist = pdbsuperpose(PDB1, PDB2)
[Dist, RMSD] = pdbsuperpose(PDB1, PDB2)
[Dist, RMSD, Transf] = pdbsuperpose(PDB1, PDB2)
[Dist, RMSD, Transf, PBD2TX] = pdbsuperpose(PDB1, PDB2)
... = pdbsuperpose(..., 'ModelNum', ModelNumValue, ...)
... = pdbsuperpose(..., 'Scale', ScaleValue, ...)
... = pdbsuperpose(..., 'Translate', TranslateValue, ...)
... = pdbsuperpose(..., 'Reflection', ReflectionValue, ...)
... = pdbsuperpose(..., 'SeqAlign', SeqAlignValue, ...)
... = pdbsuperpose(..., 'Segment', SegmentValue, ...)
... = pdbsuperpose(..., 'Apply', ApplyValue, ...)
... = pdbsuperpose(..., 'Display', DisplayValue, ...)
```

Arguments*PDB1, PDB2*

Protein structures represented by any of the following:

- String specifying a unique identifier for a protein structure record in the Protein Data Bank (PDB) database.
- Variable containing a PDB-formatted MATLAB structure, such as returned by `getpdb` or `pdbread`.
- String specifying a file name or, a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Directory.

ModelNumValue

Two-element numeric array whose elements correspond to models in *PDB1* and *PDB2* respectively when *PDB1* or *PDB2* contains multiple models. It specifies the models to consider in the superposition. By default, the first model in each structure is considered.

ScaleValue

Specifies whether to include a scaling component in the linear transformation. Choices are `true` or `false` (default).

TranslateValue

Specifies whether to include a translation component in the linear transformation. Choices are `true` (default) or `false`.

ReflectionValue Specifies whether to include a reflection component in the linear transformation. Choices are:

- `true` — Include reflection component.
- `false` — Exclude reflection component.
- `'best'` — Default. May or may not include the reflection component, depending on the best fit solution.

SeqAlignValue Specifies whether to perform a local sequence alignment and then use only the portions of the structures corresponding to the segments that align to compute the linear transformation. Choices are `true` (default) or `false`.

Note If you set the `'SeqAlign'` property to `true`, you can also specify the following properties used by the `swalign` function:

- `'ScoringMatrix'`
- `'GapOpen'`
- `'ExtendGap'`

For more information on these properties, see `swalign`.

SegmentValue

Specifies the boundaries and the chain of two subsequences to consider for computing the linear transformation. *SegmentValue* is a cell array of strings with the following format:

```
{'start1-stop1:chain1',  
'start2-stop2:chain2'}
```

You can omit the boundaries to indicate the entire chain, such as in {'chain1', 'start2-stop2:chain2'}. You can specify only one pair of segments at any given time, and the specified segments are assumed to contain the same number of alpha carbon atoms.

ApplyValue

Specifies the extent to which the linear transformation should be applied. Choices are:

- 'all' — Default. Apply the linear transformation to the entire PDB2 structure.
- 'chain' — Apply the linear transformation to the specified chain only.
- 'segment' — Apply the linear transformation to the specified segment only.

DisplayValue

Specifies whether to display the original *PDB1* structure and the resulting transformed *PDB2TX* structure in the Molecule Viewer window using the `molviewer` function. Each structure is represented as a separate model. Choices are `true` (default) or `false`.

pdbsuperpose

Return Values

<i>Dist</i>	Value representing a dissimilarity measure given by the sum of the squared errors between <i>PDB1</i> and <i>PDB2</i> . For more information, see <code>procrustes</code> in the Statistics Toolbox documentation.
<i>RMSD</i>	Scalar representing the root mean square distance between the coordinates of the <i>PDB1</i> structure and the transformed <i>PDB2</i> structure, considering only the atoms used to compute the linear transformation.
<i>Transf</i>	Linear transformation computed to superpose the chain of <i>PDB2</i> to the chain of <i>PDB1</i> . <i>Transf</i> is a MATLAB structure with the following fields: <ul style="list-style-type: none">• T — Orthogonal rotation and reflection component.• b — Scale component.• c — Translation component.
	<hr/> Note Only alpha carbon atom coordinates are used to compute the linear transformation. <hr/>
	<hr/> Tip You can use the <i>Transf</i> output as input to the <code>pdbtransform</code> function. <hr/>
<i>PDB2TX</i>	PDB-formatted MATLAB structure that represents the coordinates in the transformed <i>PDB2</i> protein structure.

Description

`pdbsuperpose(PDB1, PDB2)` computes and applies a linear transformation to superpose the coordinates of the protein structure

represented in *PDB2* to the coordinates of the protein structure represented in *PDB1*. *PDB1* and *PDB2* are protein structures represented by any of the following:

- String specifying a unique identifier for a protein structure record in the PDB database.
- Variable containing a PDB-formatted MATLAB structure, such as returned by `getpdb` or `pdbread`.
- String specifying a file name or a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Directory.

Alpha carbon atom coordinates of single chains for each structure are considered to compute the linear transformation (translation, reflection, orthogonal rotation, and scaling). By default, the first chain in each structure is considered to compute the transformation, and the transformation is applied to the entire molecule. By default, the original *PDB1* structure and the resulting transformed *PDB2* structure are displayed as separate models in the Molecule Viewer window using the `molviewer` function.

`Dist = pdbsuperpose(PDB1, PDB2)` returns a dissimilarity measure given by the sum of the squared errors between *PDB1* and *PDB2*. For more information, see `procrustes`.

`[Dist, RMSD] = pdbsuperpose(PDB1, PDB2)` also returns *RMSD*, the root mean square distance between the coordinates of the *PDB1* structure and the transformed *PDB2* structure, considering only the atoms used to compute the linear transformation.

`[Dist, RMSD, Transf] = pdbsuperpose(PDB1, PDB2)` also returns *Transf*, the linear transformation computed to superpose the chain of *PDB2* to the chain of *PDB1*. *Transf* is a MATLAB structure with the following fields:

- `T` — Orthogonal rotation and reflection component.

- *b* — Scale component.
- *c* — Translation component.

Note Only alpha carbon atom coordinates are used to compute the linear transformation.

`[Dist, RMSD, Transf, PBD2TX] = pdbsuperpose(PDB1, PDB2)` also returns *PBD2TX*, a PDB-formatted MATLAB structure that represents the coordinates in the transformed *PDB2* protein structure.

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

`... = pdbsuperpose(..., 'ModelNum', ModelNumValue, ...)` specifies the models to consider in the superposition when *PDB1* or *PDB2* contains multiple models. *ModelNumValue* is a two-element numeric array whose elements correspond to the models in *PDB1* and *PDB2* respectively. By default, the first model in each structure is considered.

`... = pdbsuperpose(..., 'Scale', ScaleValue, ...)` specifies whether to include a scaling component in the linear transformation. Choices are `true` or `false` (default).

`... = pdbsuperpose(..., 'Translate', TranslateValue, ...)` specifies whether to include a translation component in the linear transformation. Choices are `true` (default) or `false`.

`... = pdbsuperpose(..., 'Reflection', ReflectionValue, ...)` specifies whether to include a reflection component in the linear transformation. Choices are `true` (include reflection component), `false` (exclude reflection component), or `'best'` (may or may not include the reflection component, depending on the best fit solution). Default is `'best'`.

`... = pdbsuperpose(..., 'SeqAlign', SeqAlignValue, ...)`
specifies whether to perform a local sequence alignment and then use only the portions of the structures corresponding to the segments that align to compute the linear transformation. Choices are `true` (default) or `false`.

Note If you set the 'SeqAlign' property to `true`, you can also specify the following properties used by the `swalign` function:

- 'ScoringMatrix'
- 'GapOpen'
- 'ExtendGap'

For more information on these properties, see `swalign`.

`... = pdbsuperpose(..., 'Segment', SegmentValue, ...)`
specifies the boundaries and the chain of two subsequences to consider for computing the linear transformation. *SegmentValue* is a cell array of strings with the following format: {'start1-stop1:chain1', 'start2-stop2:chain2'}. You can omit the boundaries to indicate the entire chain, such as in {'chain1', 'start2-stop2:chain2'}. You can specify only one pair of segments at any given time, and the specified segments are assumed to contain the same number of alpha carbon atoms.

`... = pdbsuperpose(..., 'Apply', ApplyValue, ...)` specifies the extent to which the linear transformation should be applied. Choices are 'all' (apply the linear transformation to the entire PDB2 structure), 'chain' (apply the linear transformation to the specified chain only), or 'segment' (apply the linear transformation to the specified segment only). Default is 'all'.

`... = pdbsuperpose(..., 'Display', DisplayValue, ...)`
specifies whether to display the original *PDB1* structure and the resulting transformed *PDB2TX* structure in the Molecule Viewer window

using the `molviewer` function. Each structure is represented as a separate model. Choices are `true` (default) or `false`.

Examples

Superposing Two Hemoglobin Structures

- 1 Use the `getpdb` function to retrieve protein structure data from the Protein Data Bank (PDB) database for two hemoglobin structures.

```
str1 = getpdb('1dke');  
str2 = getpdb('4hbb');
```

- 2 Superpose the first model of the two hemoglobin structures, applying the transformation to the entire molecule.

```
d = pdbsuperpose(str1, str2, 'model', [1 1], 'apply', 'all');
```

- 3 Superpose the two hemoglobin structures (each containing four chains), computing and applying the linear transformation chain by chain. Do not display the structures.

```
strtx = str2;  
chainList1 = {str1.Sequence.ChainID};  
chainList2 = {str2.Sequence.ChainID};  
for i = 1:4  
    [d(i), rmsd(i), tr(i), strtx] = pdbsuperpose(str1, strtx, ...  
        'segment', {chainList1{i}; chainList2{i}}, ...  
        'apply', 'chain', 'display', false);  
end
```

Superposing Two Chains of a Thioredoxin Structure

Superpose chain B on chain A of a thioredoxin structure (PDBID = `2trx`), and then apply the transformation only to chain B.

```
[d, rmsd, tr] = pdbsuperpose('2trx', '2trx', 'segment', {'A', 'B'}, ...  
    'apply', 'chain')  
  
d =
```

```

0.0028

rmsd =

0.6604

tr =

T: [3x3 double]
b: 1
c: [109x3 double]

```

Superposing Two Calmodulin Structures

Superpose two calmodulin structures according to the linear transformation obtained using two 20 residue-long segments.

```

pdbsuperpose('1a29', '1c11', 'segment', {'10-30:A', '10-30:A'})

ans =

0.1945

```

See Also

Bioinformatics Toolbox functions: `getpdb`, `molviewer`, `pdbread`, `pdbtransform`, `swalign`

Statistics Toolbox function: `procrustes`

pdbtransform

Purpose Apply linear transformation to 3-D structure of molecule

Syntax

```
pdbtransform(PDB, Transf)  
PDBTX = pdbtransform(PDB, Transf)  
... = pdbtransform(..., 'ModelNum', ModelNumValue, ...)  
... = pdbtransform(..., 'Segment', SegmentValue, ...)
```

Arguments

PDB

Protein structure represented by any of the following:

- String specifying a unique identifier for a protein structure record in the Protein Data Bank (PDB) database.
- Variable containing a PDB-formatted MATLAB structure, such as returned by `getpdb` or `pdbread`.
- String specifying a file name or a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Directory.

Transf

MATLAB structure representing a linear transformation, which is applied to the coordinates of the molecule represented by *PDB*. *Transf* contains the following fields:

- **T** — Orthogonal rotation and reflection component.
- **b** — Scale component.
- **c** — Translation component.

Tip You can use the *Transf* structure returned by the `pdbsuperpose` function as input.

pdbtransform

<i>ModelNumValue</i>	Positive integer that specifies the model to which to apply the transformation, when <i>PDB</i> contains multiple models. By default, the first model is considered.
<i>SegmentValue</i>	Specifies the extent to which the linear transformation is applied. <i>SegmentValue</i> can be either: <ul style="list-style-type: none">• 'all' — The transformation is applied to the entire PDB input.• String specifying the boundaries and the chain to consider. It uses either of the following formats: 'start-stop:chain' or 'chain'. Omitting the boundaries indicates the entire chain.

Return Values

<i>PDBTX</i>	Transformed PDB-formatted MATLAB structure.
--------------	---

Description

`pdbtransform(PDB, Transf)` applies the linear transformation specified in *Transf*, a MATLAB structure representing a linear transformation, to the coordinates of the molecule represented by *PDB*, which can be any of the following:

- String specifying a unique identifier for a protein structure record in the PDB database.
- Variable containing a PDB-formatted MATLAB structure, such as returned by `getpdb` or `pdbread`.
- String specifying a file name or a path and file name. The referenced file is a PDB-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Directory.

PDBTX = `pdbtransform(PDB, Transf)` returns *PDBTX*, the transformed PDB-formatted MATLAB structure.

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

`... = pdbtransform(..., 'ModelNum', ModelNumValue, ...)` specifies the model to which to apply the transformation, when *PDB* contains multiple models. *ModelNumValue* is a positive integer. By default, the first model is considered.

`... = pdbtransform(..., 'Segment', SegmentValue, ...)` specifies the extent to which the linear transformation is applied. *SegmentValue* can be either:

- 'all' — The transformation is applied to the entire PDB input.
- String specifying the boundaries and the chain to consider. It uses either of the following formats: 'start-stop:chain' or 'chain'. Omitting the boundaries indicates the entire chain.

Examples

- 1 Create a MATLAB structure that defines a linear transformation.

```
transf.T = eye(3);  transf.b = 1;  transf.c = [11.8 -2.8 -32.3];
```

- 2 Apply the linear transformation to chain B in the thioredoxin structure, with a PDB identifier of 2trx.

```
pdmtx = pdbtransform('2trx', transf, 'segment', 'B');
```

See Also

Bioinformatics Toolbox functions: `getpdb`, `molviewer`, `pdmbread`, `pdbsuperpose`

Statistics Toolbox function: `procrustes`

pdbwrite

Purpose Write to file using Protein Data Bank (PDB) format

Syntax `pdbwrite(File, PDBStruct)`
`PDBArray = pdbwrite(File, PDBStruct)`

Arguments

<i>File</i>	String specifying either a file name or a path and file name for saving the PDB-formatted data. If you specify only a file name, the file is saved to the MATLAB Current Directory.
-------------	---

Tip After you save the MATLAB structure to a local PDB-formatted file, you can use the `molviewer` function to display and manipulate a 3-D image of the structure.

<i>PDBStruct</i>	MATLAB structure containing 3-D protein structure coordinate data, created initially by using the <code>getpdb</code> or <code>pdbread</code> functions.
------------------	--

Note You can edit this structure to modify its 3-D protein structure data. The coordinate information is stored in the `Model` field of *PDBStruct*.

Return Values	<i>PDBArray</i>	Character array in which each row corresponds to a line in a PDB record.
----------------------	-----------------	--

Description `pdbwrite(File, PDBStruct)` writes the contents of the MATLAB structure *PDBStruct* to a PDB-formatted file (ASCII text file) whose path and file name are specified by *File*. In the output file, *File*, the

atom serial numbers are preserved. The atomic coordinate records are ordered according to their atom serial numbers.

Tip After you save the MATLAB structure to a local PDB-formatted file, you can use the `molviewer` function to display and manipulate a 3-D image of the structure.

`PDBArray = pdbwrite(File, PDBStruct)` saves the formatted PDB record, converted from the contents of the MATLAB structure `PDBStruct`, to `PDBArray`, a character array in which each row corresponds to a line in a PDB record.

Note You can edit `PDBStruct` to modify its 3-D protein structure data. The coordinate information is stored in the `Model` field of `PDBStruct`.

Examples

- 1 Use the `getpdb` function to retrieve structure information from the Protein Data Bank (PDB) for the green fluorescent protein with identifier 1GFL , and store the data in the MATLAB structure `gflstruct`.

```
gflstruct = getpdb('1GFL');
```

- 2 Find the *x*-coordinate of the first atom.

```
gflstruct.Model.Atom(1).X
```

```
ans =
```

```
-14.0930
```

- 3 Edit the *x*-coordinate of the first atom.

```
gflstruct.Model.Atom(1).X = -18;
```

Note Do not add or remove any Atom fields, because the `pdbwrite` function does not allow the number of elements in the structure to change.

- 4 Write the modified MATLAB structure `gflstruct` to a new PDB-formatted file `modified_gfl.pdb` in the Work directory on your C drive.

```
pdbwrite('c:\work\modified_gfl.pdb', gflstruct);
```

- 5 Use the `pdbread` function to read the modified PDB file into a MATLAB structure, then confirm that the *x*-coordinate of the first atom has changed.

```
modified_gflstruct = pdbread('c:\work\modified_gfl.pdb')
modified_gflstruct.Model.Atom(1).X
```

```
ans =
```

```
-18
```

See Also

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

Purpose Read data from PFAM HMM-formatted file

Syntax `HMMStruct = pfamhmmread(File)`

Arguments

File

Either of the following:

- String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a PFAM HMM-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the current directory.
- MATLAB character array that contains the text of a PFAM-HMM-formatted file.

Tip You can use the `gethmmprof` function with the `'ToFile'` property to retrieve HMM profile information from the PFAM database and create a PFAM HMM-formatted file.

Return Values

HMMStruct

MATLAB structure containing information from a PFAM HMM-formatted file.

Description

Note `pfamhmmread` reads version 2.0 HMMER file formats.

`HMMStruct = pfamhmmread(File)` reads *File*, a PFAM HMM-formatted file, and converts it to *HMMStruct*, a MATLAB structure containing the following fields corresponding to parameters of an HMM profile:

pfamhmmread

Field	Description
Name	The protein family name (unique identifier) of the HMM profile record in the PFAM database.
PfamAccessionNumber	The protein family accession number of the HMM profile record in the PFAM database.
ModelDescription	Description of the HMM profile.
ModelLength	The length of the profile (number of MATCH states).
Alphabet	The alphabet used in the model, 'AA' or 'NT'. <hr/> Note AlphaLength is 20 for 'AA' and 4 for 'NT'. <hr/>
MatchEmission	Symbol emission probabilities in the MATCH states. The format is a matrix of size ModelLength-by-AlphabetLength, where each row corresponds to the emission distribution for a specific MATCH state.
InsertEmission	Symbol emission probabilities in the INSERT state. The format is a matrix of size ModelLength-by-AlphabetLength, where each row corresponds to the emission distribution for a specific INSERT state.
NullEmission	Symbol emission probabilities in the MATCH and INSERT states for the NULL model. The format is a 1-by-AlphabetLength row vector. <hr/> Note NULL probabilities are also known as the background probabilities. <hr/>

Field	Description
BeginX	<p>BEGIN state transition probabilities.</p> <p>Format is a 1-by-(ModelLength + 1) row vector:</p> <p>[B->D1 B->M1 B->M2 B->M3 ... B->Mend]</p>
MatchX	<p>MATCH state transition probabilities.</p> <p>Format is a 4-by-(ModelLength - 1) matrix:</p> <p>[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]</p>
InsertX	<p>INSERT state transition probabilities.</p> <p>Format is a 2-by-(ModelLength - 1) matrix:</p> <p>[I1->M2 I2->M3 ... I[end-1]->Mend; I1->I1 I2->I2 ... I[end-1]->I[end-1]]</p>
DeleteX	<p>DELETE state transition probabilities.</p> <p>Format is a 2-by-(ModelLength - 1) matrix:</p> <p>[D1->M2 D2->M3 ... D[end-1]->Mend ; D1->D2 D2->D3 ... D[end-1]->Dend]</p>
FlankingInsertX	<p>Flanking insert states (N and C) used for LOCAL profile alignment.</p> <p>Format is a 2-by-2 matrix:</p> <p>[N->B C->T ; N->N C->C]</p>

pfamhmmread

Field	Description
LoopX	Loop states transition probabilities used for multiple hits alignment. Format is a 2-by-2 matrix: [E->C J->B ; E->J J->J]
NullX	Null transition probabilities used to provide scores with log-odds values also for state transitions. Format is a 2-by-1 column vector: [G->F ; G->G]

For more information on HMM profile models, see “HMM Profile Model” on page 2-536.

Examples

Read a URL pointing to a PFAM HMM-formatted file into a MATLAB structure.

```
site='http://pfam.sanger.ac.uk/';  
hmm = pfamhmmread([site 'family/gethmm?mode=ls&id=7tm_2'])  
  
hmm =  
  
        Name: '7tm_2'  
PfamAccessionNumber: 'PF00002.15'  
ModelDescription: '7 transmembrane receptor (Secretin family)'  
ModelLength: 293  
Alphabet: 'AA'  
MatchEmission: [293x20 double]  
InsertEmission: [293x20 double]  
NullEmission: [1x20 double]  
BeginX: [294x1 double]
```

```
MatchX: [292x4 double]
InsertX: [292x2 double]
DeleteX: [292x2 double]
FlankingInsertX: [2x2 double]
LoopX: [2x2 double]
NullX: [2x1 double]
```

Read a locally saved PFAM HMM-formatted file into a MATLAB structure.

```
pfamhmmread('pf00002.ls')
```

See Also

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

phytree

Purpose Create phytree object

Syntax

```
Tree = phytree(B)
Tree = phytree(B, D)
Tree = phytree(B, C)
Tree = phytree(BC)
Tree = phytree(..., N)
Tree = phytree
```

Arguments

- B* Numeric array of size [NUMBRANCHES X 2] in which every row represents a branch of the tree. It contains two pointers to the branch or leaf nodes, which are its children.
- C* Column vector with distances for every branch.
- D* Column vector with distances from every node to their parent branch.
- BC* Combined matrix with pointers to branches or leaves, and distances of branches.
- N* Cell array with the names of leaves and branches.

Description

Tree = `phytree(B)` creates an ultrametric phylogenetic tree object. In an ultrametric phylogenetic tree object, all leaves are the same distance from the root.

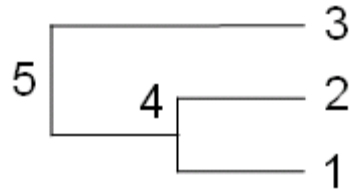
B is a numeric array of size [NUMBRANCHES X 2] in which every row represents a branch of the tree and it contains two pointers to the branch or leaf nodes, which are its children.

Leaf nodes are numbered from 1 to NUMLEAVES and branch nodes are numbered from NUMLEAVES + 1 to NUMLEAVES + NUMBRANCHES. Note that because only binary trees are allowed, NUMLEAVES = NUMBRANCHES + 1.

Branches are defined in chronological order (for example, $B(i, :) > \text{NUMLEAVES} + i$). As a consequence, the first row can only have pointers to leaves, and the last row must represent the root branch. Parent-child

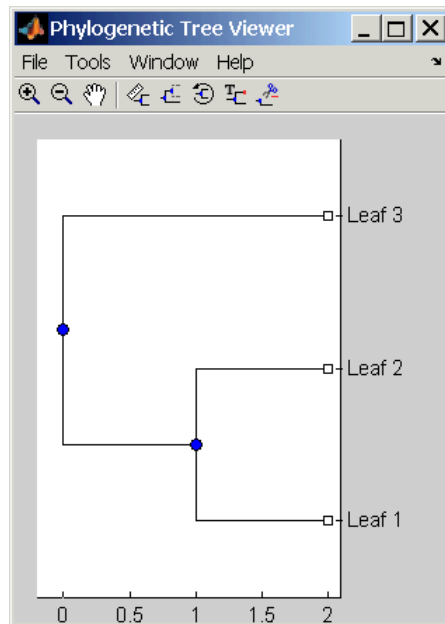
distances are set to 1, unless the child is a leaf and to satisfy the ultrametric condition of the tree its distance is increased.

Given a tree with three leaves and two branches as an example.



In the MATLAB Command Window, type

```
B = [1 2 ; 3 4]
tree = phytree(B)
view(tree)
```



Tree = `phytree(B, D)` creates an additive (ultrametric or nonultrametric) phylogenetic tree object with branch distances defined by *D*. *D* is a numeric array of size [NUMNODES X 1] with the distances of every child node (leaf or branch) to its parent branch equal to NUMNODES = NUMLEAVES + NUMBRANCHES. The last distance in *D* is the distance of the root node and is meaningless.

```
b = [1 2 ; 3 4 ]; d = [1 2 1.5 1 0]
view(phytree(b,d))
```

Tree = `phytree(B, C)` creates an ultrametric phylogenetic tree object with distances between branches and leaves defined by *C*. *C* is a numeric array of size [NUMBRANCHES X 1], which contains the distance from each branch to the leaves. In ultrametric trees, all of the leaves are at the same location (same distance to the root).

```
b = [1 2 ; 3 4]; c = [1 4]'
view(phytree(b,c))
```

Tree = `phytree(BC)` creates an ultrametric phylogenetic binary tree object with branch pointers in `BC(:, [1 2])` and branch coordinates in `BC(:,3)`. Same as `phytree(B,C)`.

Tree = `phytree(..., N)` specifies the names for the leaves and/or the branches. *N* is a cell of strings. If `NUMEL(N)==NUMLEAVES`, then the names are assigned chronologically to the leaves. If `NUMEL(N)==NUMBRANCHES`, the names are assigned to the branch nodes. If `NUMEL(N)==NUMLEAVES + NUMBRANCHES`, all the nodes are named. Unassigned names default to 'Leaf #' and/or 'Branch #' as required.

Tree = `phytree` creates an empty phylogenetic tree object.

Examples

Create a phylogenetic tree for a set of multiply aligned sequences.

```
Sequences = multialignread('aagag.aln')
distances = seqpdist(Sequences)
tree = seqlinkage(distances)
phytreetool(tree)
```

See Also

Bioinformatics Toolbox functions: `phytreeread`, `phytreetool`, `phytreewrite`, `seqlinkage`, `seqneighjoin`, `seqpdist`

Bioinformatics Toolbox object: `phytree` object

Bioinformatics Toolbox methods of `phytree` object: `get`, `getbyname`, `getcanonical`, `getmatrix`, `getnewickstr`, `pdist`, `plot`, `prune`, `reroot`, `select`, `subtree`, `view`, `weights`

phytreeread

Purpose Read phylogenetic tree file

Syntax `Tree = phytreeread(File)`

Arguments

File Newick-formatted tree files (ASCII text file). Enter a file name, a path and file name, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text for a file.

Tree phytree object created with the function `phytree`.

Description

`Tree = phytreeread(File)` reads a Newick formatted tree file and returns a phytree object in the MATLAB workspace with data from the file.

The NEWICK tree format can be found at

<http://evolution.genetics.washington.edu/phylip/newicktree.html>

Note This implementation only allows binary trees. Non-binary trees are translated into a binary tree with extra branches of length 0.

Examples

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

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `gethmmtree`, `phytreetool`, `phytreewrite`

Purpose View, edit, and explore phylogenetic tree data

Syntax `phytreetool(Tree)`
`phytreetool(File)`

Arguments

Tree Phytree object created with the functions `phytree` or `phytreeread`.

File Newick or ClustalW tree formatted file (ASCII text file) with phylogenetic tree data. Enter a file name, a path and file name, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text for a Newick file.

Description

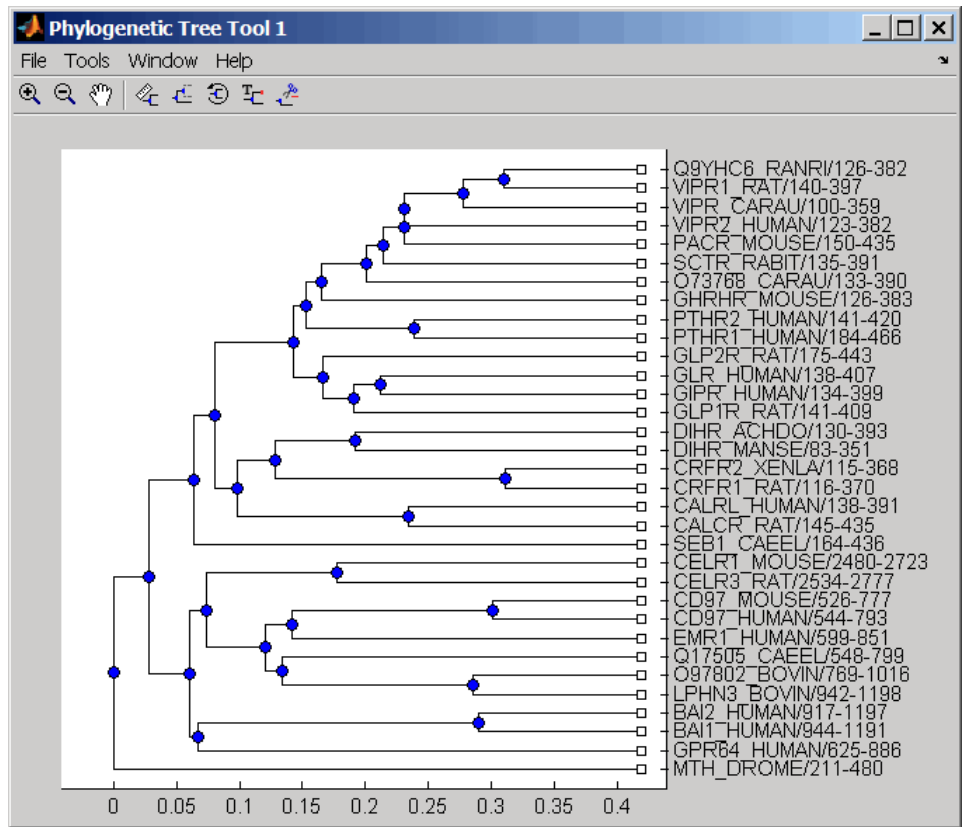
`phytreetool` is an interactive GUI that allows you to view, edit, and explore phylogenetic tree data. This GUI allows branch pruning, reordering, renaming, and distance exploring. It can also open or save Newick formatted files.

`phytreetool(Tree)` loads data from a `phytree` object in the MATLAB workspace into the GUI.

`phytreetool(File)` loads data from a Newick formatted file into the GUI.

Examples

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



See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreeread`, `phytreewrite`

Bioinformatics Toolbox methods of `phytree` object: `plot`, `view`

Purpose Write phylogenetic tree object to Newick-formatted file

Syntax `phytreewrite('File', Tree)`
`phytreewrite(Tree)`

Arguments

File Newick-formatted file. Enter either a file name or a path and file name supported by your operating system (ASCII text file).

Tree Phylogenetic tree object, either created with `phytree` (object constructor function) or imported using the `phytreeread` function.

Description

`phytreewrite('File', Tree)` copies the contents of a `phytree` object from the MATLAB workspace to a file. Data in the file uses the Newick format for describing trees.

The Newick tree format can be found at

<http://evolution.genetics.washington.edu/phylip/newicktree.html>

`phytreewrite(Tree)` opens the Save Phylogenetic Tree As dialog box for you to enter or select a file name.

Examples

Read tree data from a Newick-formatted file.

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

Remove all the mouse proteins

```
ind = getbyname(tr,'mouse');  
tr = prune(tr,ind);
```

phytreewrite

```
view(tr)
```

Write pruned tree data to a file.

```
phytreewrite('newtree.tree', tr)
```

See Also

Bioinformatics Toolbox functions: `multialignwrite`, `phytree` (object constructor), `phytreeread`, `phytreetool`, `seqlinkage`

Bioinformatics Toolbox object: `phytree` object

Bioinformatics Toolbox methods of `phytree` object: `getnewickstr`

Purpose	Create table of probe set library information	
Syntax	<i>ProbeInfo</i> = probelibraryinfo(<i>CELStruct</i> , <i>CDFStruct</i>)	
Arguments	<i>CELStruct</i>	Structure created by the <i>affyread</i> function from an Affymetrix CEL file.
	<i>CDFStruct</i>	Structure created by the <i>affyread</i> function from an Affymetrix CDF library file associated with the CEL file.
Return Values	<i>ProbeInfo</i>	<p>Three-column matrix with the same number of rows as the <i>Probes</i> field of the <i>CELStruct</i>.</p> <ul style="list-style-type: none"> • Column 1 — Probe set ID/name to which the probe belongs. (Probes that do not belong to a probe set in the CDF library file have probe set ID/name equal to 0.) • Column 2 — Contains the probe pair number. • Column 3 — Indicates if the probe is a perfect match (1) or mismatch (-1) probe.
	Description	<i>ProbeInfo</i> = probelibraryinfo(<i>CELStruct</i> , <i>CDFStruct</i>) creates a table of information linking the probe data from <i>CELStruct</i> , a structure created from an Affymetrix CEL file, with probe set information from <i>CDFStruct</i> , a structure created from an Affymetrix CDF file.

Note Affymetrix probe pair indexing is 0-based, while MATLAB software indexing is 1-based. The output from *probelibraryinfo* is 1-based.

Examples

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

```
http://www.affymetrix.com/support/technical/sample\_data/demo\_data.affx
```

After you download the demo data, you will need the Affymetrix Data Transfer Tool to extract the CEL file from a DTT file. You can download the Affymetrix Data Transfer Tool from:

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

The following example assumes that the `Ecoli-antisense-121502.CEL` file is stored on the MATLAB search path or in the current directory. It also assumes that the associated CDF library file, `Ecoli_ASv2.CDF`, is stored at `D:\Affymetrix\LibFiles\Ecoli`.

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

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

- 2 Read the contents of a CDF file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

- 3 Extract probe set library information.

```
ProbeInfo = probelibraryinfo(celStruct, cdfStruct);
```

- 4 Determine the probe set to which the 1104th probe belongs.

```
cdfStruct.ProbeSets(ProbeInfo(1104,1)).Name
```

```
ans =
```

```
thrA_b0002_at
```

See Also

Bioinformatics Toolbox functions: `affyread`, `celintensityread`, `probesetlink`, `probesetlookup`, `probesetplot`, `probesetvalues`

Purpose

Display probe set information on NetAffx Web site

Syntax

```
probesetlink(AffyStruct, PS)
URL = probesetlink(AffyStruct, PS)
probesetlink(AffyStruct, PS, ...'Source', SourceValue, ...)
probesetlink(AffyStruct, PS, ...'Browser',
BrowserValue, ...)
URL = probesetlink(AffyStruct, PS, ...'NoDisplay',
NoDisplayValue, ...)
```

Arguments

<i>AffyStruct</i>	Structure created by the <i>affyread</i> function from an Affymetrix CHP file or an Affymetrix CDF library file.
<i>PS</i>	Probe set index or the probe set ID/name.
<i>SourceValue</i>	Controls the linking to the data source (for example, GenBank or Flybase) for the probe set (instead of linking to the NetAffx Web site). Choices are true or false (default).

Note This property requires the GIN library file associated with the CHP or CDF file to be located in the same directory as the CDF library file.

<i>BrowserValue</i>	Controls the display of the probe set information in your system's default Web browser. Choices are true or false (default).
<i>NoDisplayValue</i>	Controls the return of <i>URL</i> without opening a Web browser. Choices are true or false (default).

Return Values

<i>URL</i>	URL for the probe set information.
------------	------------------------------------

probesetlink

Description

`probesetlink(AffyStruct, PS)` opens a Web Browser window displaying information on the NetAffx Web site about a probe set specified by *PS*, a probe set index or the probe set ID/name, and *AffyStruct*, a structure created from an Affymetrix CHP file or Affymetrix CDF library file.

URL = `probesetlink(AffyStruct, PS)` also returns the URL (linking to the NetAffx Web site) for the probe set information.

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

`probesetlink(AffyStruct, PS, ...'Source', SourceValue, ...)` controls the linking to the data source (for example, GenBank or Flybase) for the probe set (instead of linking to the NetAffx Web site). Choices are true or false (default).

Note The 'Source' property requires the GIN library file associated with the CHP or CDF file to be located in the same directory as the CDF library file.

`probesetlink(AffyStruct, PS, ...'Browser', BrowserValue, ...)` controls the display of the probe set information in your system's default Web browser. Choices are true or false (default).

URL = `probesetlink(AffyStruct, PS, ...'NoDisplay', NoDisplayValue, ...)` controls the return of the URL without opening a Web browser. Choices are true or false (default).

Note The NetAffx Web site requires you to register and provide a user name and password.

Examples

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

```
http://www.affymetrix.com/support/technical/sample\_data/demo\_data.affx
```

After you download the demo data, you will need the Affymetrix Data Transfer Tool to extract the CHP file from a DTT file. You can download the Affymetrix Data Transfer Tool from:

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

The following example assumes that the `Ecoli-antisense-121502.CHP` file is stored on the MATLAB search path or in the current directory. It also assumes that the associated CDF library file, `Ecoli_ASv2.CDF`, is stored at `D:\Affymetrix\LibFiles\Ecoli`.

- 1 Read the contents of a CHP file into a MATLAB structure.

```
chpStruct = affyread('Ecoli-antisense-121502.CHP',...  
                    'D:\Affymetrix\LibFiles\Ecoli');
```

- 2 Display information from the NetAffx Web site for the `argG_b3172_at` probe set.

```
probesetlink(chpStruct, 'argG_b3172_at')
```

See Also

Bioinformatics Toolbox functions: `affyread`, `celintensityread`, `probelibraryinfo`, `probesetlookup`, `probesetplot`, `probesetvalues`

probesetlookup

Purpose Look up information for Affymetrix probe set

Syntax *PSStruct* = probesetlookup(*AffyStruct*, *ID*)

Arguments *AffyStruct* Structure created by the `affyread` function from an Affymetrix CHP file or an Affymetrix CDF library file for expression assays.

ID String or cell array of strings specifying one or more probe set IDs/names or gene IDs.

Return Values *PSStruct* Structure or array of structures containing the following fields for a probe set:

- **Identifier** — Gene ID associated with the probe set
- **ProbeSetName** — Probe set ID/name
- **CDFIndex** — Index into the CDF structure for the probe set
- **GINIndex** — Index into the GIN structure for the probe set
- **Description** — Description of the probe set
- **Source** — Source(s) of the probe set
- **SourceURL** — Source URL(s) for the probe set

Description *PSStruct* = probesetlookup(*AffyStruct*, *ID*) returns a structure or an array of structures containing information for an Affymetrix probe set specified by *ID*, a string or cell array of strings specifying one or more probe set IDs/names or gene IDs, and by *AffyStruct*, a structure created from an Affymetrix CHP file or Affymetrix CDF library file for expression assays.

Note This function works with CHP files and CDF files for expression assays only. It requires that the GIN library file associated with the CHP file or CDF file to be located in the same directory as the CDF library file.

Examples

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

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

The following example assumes that the `Ecoli_ASv2.CDF` library file is stored at `D:\Affymetrix\LibFiles\Ecoli`.

- 1 Read the contents of a CDF library file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

- 2 Look up the gene ID (Identifier) associated with the `argG_b3172_at` probe set.

```
probesetlookup(cdfStruct, 'argG_b3172_at')
```

```
ans =
```

```

Identifier: '3315278'
ProbeSetName: 'argG_b3172_at'
CDFIndex: 5213
GINIndex: 3074
Description: [1x82 char]
Source: 'NCBI EColi Genome'
SourceURL: [1x74 char]
```

See Also

Bioinformatics Toolbox functions: `affyread`, `celintensityread`, `probelibraryinfo`, `probesetlink`, `probesetplot`, `probesetvalues`, `rmabackadj`

probesetplot

Purpose Plot Affymetrix probe set intensity values

Syntax

```
probesetplot(CELStruct, CDFStruct, PS)
probesetplot(CELStruct, CDFStruct, PS, ...'GeneName',
GeneNameValue, ...)
probesetplot(CELStruct, CDFStruct, PS, ...'Field',
FieldValue, ...)
probesetplot(CELStruct, CDFStruct, PS, ...'ShowStats',
ShowStatsValue, ...)
```

Arguments

<i>CELStruct</i>	Structure created by the affyread function from an Affymetrix CEL file.
<i>CDFStruct</i>	Structure created by the affyread function from an Affymetrix CDF library file associated with the CEL file.
<i>PS</i>	Probe set index or the probe set ID/name.
<i>GeneNameValue</i>	Controls whether the probe set name or the gene name is used for the title of the plot. Choices are true or false (default).

Note The 'GeneName' property requires the GIN library file associated with the CEL and CDF files to be located in the same directory as the CDF library file from which *CDFStruct* was created.

<i>FieldValue</i>	String specifying the type of data to plot. Choices are: <ul style="list-style-type: none"> • 'Intensity' (default) • 'StdDev' • 'Background' • 'Pixels' • 'Outlier'
<i>ShowStatsValue</i>	Controls whether the mean and standard deviation lines are included in the plot. Choices are true or false (default).

Description

`probesetplot(CELStruct, CDFStruct, PS)` plots the PM (perfect match) and MM (mismatch) intensity values for a specified probe set. *CELStruct* is a structure created by the `affyread` function from an Affymetrix CEL file. *CDFStruct* is a structure created by the `affyread` function from an Affymetrix CDF library file associated with the CEL file. *PS* is the probe set index or the probe set ID/name.

Note MATLAB software uses 1-based indexing for probe set numbers, while the Affymetrix CDF file uses 0-based indexing for probe set numbers. For example, `CDFStruct.ProbeSets(1)` has a `ProbeSetNumber` of 0 in the `ProbePairs` field.

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

probesetplot

`probesetplot(CELStruct, CDFStruct, PS, ...'GeneName', GeneNameValue, ...)` controls whether the probe set name or the gene name is used for the title of the plot. Choices are `true` or `false` (default).

Note The 'GeneName' property requires the GIN library file associated with the CEL and CDF files to be located in the same directory as the CDF library file from which *CDFStruct* was created.

`probesetplot(CELStruct, CDFStruct, PS, ...'Field', FieldValue, ...)` specifies the type of data to plot. Choices are:

- 'Intensity' (default)
- 'StdDev'
- 'Background'
- 'Pixels'
- 'Outlier'

`probesetplot(CELStruct, CDFStruct, PS, ...'ShowStats', ShowStatsValue, ...)` controls whether the mean and standard deviation lines are included in the plot. Choices are `true` or `false` (default).

Examples

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

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

After you download the demo data, you will need the Affymetrix Data Transfer Tool to extract the CEL file from a DTT file. You can download the Affymetrix Data Transfer Tool from:

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

The following example assumes that the `Ecoli-antisense-121502.CEL` file is stored on the MATLAB search path or in the current directory. It also assumes that the associated CDF library file, `Ecoli_ASv2.CDF`, is stored at `D:\Affymetrix\LibFiles\Ecoli`.

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

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

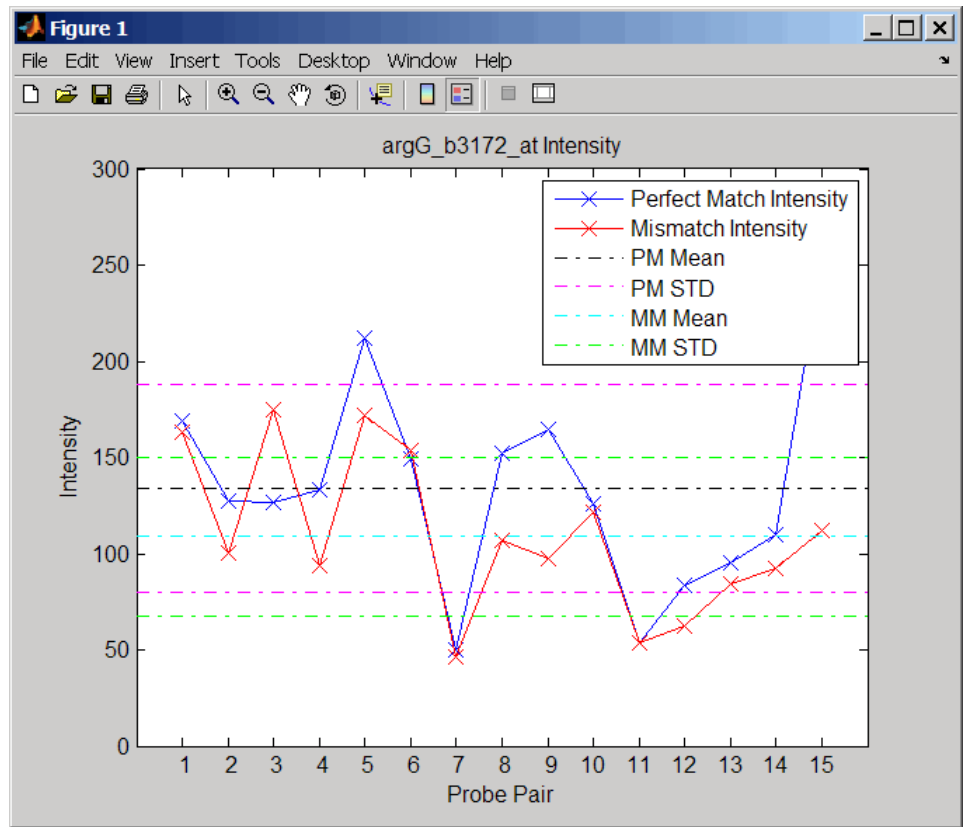
- 2 Read the contents of a CDF file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

- 3 Plot the PM and MM intensity values of the `argG_b3172_at` probe set, including the mean and standard deviation.

```
probesetplot(celStruct, cdfStruct, 'argG_b3172_at', 'showstats', true)
```

probesetplot



See Also

Bioinformatics Toolbox functions: `affyread`, `celintensityread`, `probesetlink`, `probesetlookup`, `probesetvalues`

Purpose

Create table of Affymetrix probe set intensity values

Syntax

```
PSValues = probesetvalues(CELStruct, CDFStruct, PS)  
PSValues = probesetvalues(CELStruct, CDFStruct, PS,  
    'Background', BackgroundValue)  
ColumnNames = probesetvalues
```

Arguments

CELStruct

Structure created by the `affyread` function from an Affymetrix CEL file.

CDFStruct

Structure created by the `affyread` function from an Affymetrix CDF library file associated with the CEL file.

probesetvalues

<i>PS</i>	Probe set index or the probe set ID/name.
<i>BackgroundValue</i>	Controls the background correction in the calculation. Choices are: <ul style="list-style-type: none">• true (default) — Background values from the <i>Background</i> field in the <i>PSValues</i> matrix are used to calculate the probe intensity values.• false — Background values are not calculated.• A vector of precalculated background values (such as returned by the <i>zonebackadj</i> function) whose length is equal to the number of probes in <i>CELStruct</i>. These background values are used to calculate the probe intensity values.

Tip Including background correction in the calculation of the probe intensity values can be slow. Therefore, setting 'Background' to **false** can speed up the calculation. However, the values returned in the 'Background' field of the *PSValues* matrix will be zero.

Return Values

<i>PSValues</i>	Twenty-column matrix with one row for each probe pair in the probe set.
<i>ColumnNames</i>	Cell array of strings containing the column names of the <i>PSValues</i> matrix. This is returned only when you call <i>probesetvalues</i> with no input arguments.

Description

PSValues = *probesetvalues*(*CELStruct*, *CDFStruct*, *PS*) creates a table of intensity values for *PS*, a probe set, from the probe-level data in *CELStruct*, a structure created by the *affyread* function from

an Affymetrix CEL file. *PS* is a probe set index or probe set ID/name from *CDFStruct*, a structure created by the *affyread* function from an Affymetrix CDF library file associated with the CEL file. *PSValues* is a twenty-column matrix with one row for each probe pair in the probe set. The columns correspond to the following fields.

Column	Field	Description
1	'ProbeSetNumber'	Number identifying the probe set to which the probe pair belongs.
2	'ProbePairNumber'	Index of the probe pair within the probe set.
3	'UseProbePair'	This field is for backward compatibility only and is not currently used.
4	'Background'	Estimated background of probe intensity values of the probe pair.
5	'PMPosX'	<i>x</i> -coordinate of the perfect match probe.
6	'PMPosY'	<i>y</i> -coordinate of the perfect match probe.
7	'PMIntensity'	Intensity value of the perfect match probe.
8	'PMStdDev'	Standard deviation of intensity value of the perfect match probe.
9	'PMPixels'	Number of pixels in the cell containing the perfect match probe.
10	'PMOutlier'	True/false flag indicating if the perfect match probe was marked as an outlier.
11	'PMMasked'	True/false flag indicating if the perfect match probe was masked.

probesetvalues

Column	Field	Description
12	'MMPosX'	<i>x</i> -coordinate of the mismatch probe.
13	'MMPosY'	<i>y</i> -coordinate of the mismatch probe.
14	'MMIntensity'	Intensity value of the mismatch probe.
15	'MMStdDev'	Standard deviation of intensity value of the mismatch probe.
16	'MMPixels'	Number of pixels in the cell containing the mismatch probe.
17	'MMOutlier'	True/false flag indicating if the mismatch probe was marked as an outlier.
18	'MMMasked'	True/false flag indicating if the mismatch probe was masked.
19	'GroupNumber'	Number identifying the group to which the probe pair belongs. For expression arrays, this is always 1. For genotyping arrays, this is typically 1 (allele A, sense), 2 (allele B, sense), 3 (allele A, antisense), or 4 (allele B, antisense).
20	'Direction'	Number identifying the direction of the probe pair. 1 = sense and 2 = antisense.

Note MATLAB software uses 1-based indexing for probe set numbers, while the Affymetrix CDF file uses 0-based indexing for probe set numbers. For example, `CDFStruct.ProbeSets(1)` has a `ProbeSetNumber` of 0 in the `ProbePairs` field.

PSValues = probesetvalues(*CELStruct*, *CDFStruct*, *PS*, 'Background', *BackgroundValue*) controls the background correction in the calculation. *BackgroundValue* can be:

- true (default) — Background values from the Background field in the *PSValues* matrix are used to calculate the probe intensity values.
- false — Background values are not calculated.
- A vector of precalculated background values (such as returned by the zonebackadj function) whose length is equal to the number of probes in *CELStruct*. These background values are used to calculate the probe intensity values.

Tip Including background correction in the calculation of the probe intensity values can be slow. Therefore, setting 'Background' to false can speed up the calculation. However, the values returned in the 'Background' field of the *PSValues* matrix will be zero.

ColumnNames = probesetvalues returns a cell array of strings containing the column names of the *PSValues* matrix. *ColumnNames* is returned only when you call probesetvalues without input arguments. The information contained in *ColumnNames* is common to all Affymetrix GeneChip arrays.

Examples

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

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

After you download the demo data, you will need the Affymetrix Data Transfer Tool to extract the CEL file from a DTT file. You can download the Affymetrix Data Transfer Tool from:

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

probesetvalues

The following example assumes that the `Ecoli-antisense-121502.CEL` file is stored on the MATLAB search path or in the current directory. It also assumes that the associated CDF library file, `Ecoli_ASv2.CDF`, is stored at `D:\Affymetrix\LibFiles\Ecoli`.

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

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

- 2 Read the contents of a CDF file into a MATLAB structure.

```
cdfStruct = affyread('D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

- 3 Use the `zonebackadj` function to return a matrix or cell array of vectors containing the estimated background values for each probe.

```
[baData,zones,background] = zonebackadj(celStruct,'cdf',cdfStruct);
```

- 4 Create a table of intensity values for the `argG_b3172_at` probe set.

```
psvals = probesetvalues(celStruct, cdfStruct, 'argG_b3172_at',...  
    'background',background);
```

See Also

Bioinformatics Toolbox functions: `affyread`, `celintensityread`, `probelibraryinfo`, `probesetlink`, `probesetlookup`, `probesetplot`, `rmabackadj`, `zonebackadj`

Purpose Align two profiles using Needleman-Wunsch global alignment

Syntax

```

Prof = profalign(Prof1, Prof2)
[Prof, H1, H2] = profalign(Prof1, Prof2)
profalign(..., 'ScoringMatrix', ScoringMatrixValue, ...)
profalign(..., 'GapOpen', {G1Value, G2Value}, ...)
profalign(..., 'ExtendGap', {E1Value, E2Value}, ...)
profalign(..., 'ExistingGapAdjust', ExistingGapAdjustValue,
...)
profalign(..., 'TerminalGapAdjust', TerminalGapAdjustValue,
...)
profalign(..., 'ShowScore', ShowScoreValue, ...)

```

Description *Prof* = profalign(*Prof1*, *Prof2*) returns a new profile (*Prof*) for the optimal global alignment of two profiles (*Prof1*, *Prof2*). The profiles (*Prof1*, *Prof2*) are numeric arrays of size [(4 or 5 or 20 or 21) x Profile Length] with counts or weighted profiles. Weighted profiles are used to down-weight similar sequences and up-weight divergent sequences. The output profile is a numeric matrix of size [(5 or 21) x New Profile Length] where the last row represents gaps. Original gaps in the input profiles are preserved. The output profile is the result of adding the aligned columns of the input profiles.

[*Prof*, *H1*, *H2*] = profalign(*Prof1*, *Prof2*) returns pointers that indicate how to rearrange the columns of the original profiles into the new profile.

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

profalign(..., 'ScoringMatrix', *ScoringMatrixValue*, ...) defines the scoring matrix (*ScoringMatrixValue*) to be used for the alignment. The default is 'BLOSUM50' for amino acids or 'NUC44' for nucleotide sequences.

`profalign(..., 'GapOpen', {G1Value, G2Value}, ...)` sets the penalties for opening a gap in the first and second profiles respectively. *G1Value* and *G2Value* can be either scalars or vectors. When using a vector, the number of elements is one more than the length of the input profile. Every element indicates the position specific penalty for opening a gap between two consecutive symbols in the sequence. The first and the last elements are the gap penalties used at the ends of the sequence. The default gap open penalties are {10,10}.

`profalign(..., 'ExtendGap', {E1Value, E2Value}, ...)` sets the penalties for extending a gap in the first and second profile respectively. *E1Value* and *E2Value* can be either scalars or vectors. When using a vector, the number of elements is one more than the length of the input profile. Every element indicates the position specific penalty for extending a gap between two consecutive symbols in the sequence. The first and the last elements are the gap penalties used at the ends of the sequence. If `ExtendGap` is not specified, then extensions to gaps are scored with the same value as `GapOpen`.

`profalign(..., 'ExistingGapAdjust', ExistingGapAdjustValue, ...)`, if *ExistingGapAdjustValue* is false, turns off the automatic adjustment based on existing gaps of the position-specific penalties for opening a gap. When *ExistingGapAdjustValue* is true (default), for every profile position, `profalign` proportionally lowers the penalty for opening a gap toward the penalty of extending a gap based on the proportion of gaps found in the contiguous symbols and on the weight of the input profile.

`profalign(..., 'TerminalGapAdjust', TerminalGapAdjustValue, ...)`, when *TerminalGapAdjustValue* is true, adjusts the penalty for opening a gap at the ends of the sequence to be equal to the penalty for extending a gap. Default is false.

`profalign(..., 'ShowScore', ShowScoreValue, ...)`, when *ShowScoreValue* is true, displays the scoring space and the winning path.

Examples

- 1 Read in sequences and create profiles.

```
ma1 = [ 'RGTANCDMQDA'; 'RGTAHCDMQDA'; 'RRRAPCDL-DA' ];
ma2 = [ 'RGTHCDLADAT'; 'RGTACDMADAA' ];
p1 = seqprofile(ma1,'gaps','all','counts',true);
p2 = seqprofile(ma2,'counts',true);
```

- 2 Merge two profiles into a single one by aligning them.

```
p = profalign(p1,p2);
seqlogo(p)
```

- 3 Use the output pointers to generate the multiple alignment.

```
[p, h1, h2] = profalign(p1,p2);
ma = repmat('-',5,12);
ma(1:3,h1) = ma1;
ma(4:5,h2) = ma2;
disp(ma)
```

- 4 Increase the gap penalty before cysteine in the second profile.

```
gapVec = 10 + [p2(aa2int('C'),:) 0] * 10
p3 = profalign(p1,p2,'gapopen',{10,gapVec});
seqlogo(p3)
```

- 5 Add a new sequence to a profile without inserting new gaps into the profile.

```
gapVec = [0 inf(1,11) 0];
p4 = profalign(p3,seqprofile('PLHFMSVLWDVQQWP'),...
    'gapopen',{gapVec,10});
seqlogo(p4)
```

See Also

Bioinformatics Toolbox functions: `hmmprofalign`, `multialign`, `nwalign`, `seqprofile`, `seqconsensus`

proteinplot

Purpose	Characteristics for amino acid sequences
Syntax	<code>proteinplot (SeqAA)</code>
Arguments	<code>SeqAA</code> Amino acid sequence or a structure with a field <code>Sequence</code> containing an amino acid sequence.
Description	<code>proteinplot (SeqAA)</code> loads an amino acid sequence into the protein plot GUI. <code>proteinplot</code> is a tool for analyzing a single amino acid sequence. You can use the results from <code>proteinplot</code> to compare the properties of several amino acid sequences. It displays smoothed line plots of various properties such as the hydrophobicity of the amino acids in the sequence.

Importing Sequences into proteinplot

1 In the MATLAB Command Window, type

```
proteinplot(Seq_AA)
```

The `proteinplot` interface opens and the sequence `Seq_AA` is shown in the **Sequence** text box.

2 Alternatively, type or paste an amino acid sequence into the **Sequence** text box.

You can import a sequence with the Import dialog box:

1 Click the **Import Sequence** button. The Import dialog box opens.

2 From the **Import From** list, select a variable in the MATLAB workspace, ASCII text file, FASTA formatted file, GenPept formatted file, or accession number in the GenPept database.

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.

proteinplot

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: `aaccount`, `atomiccomp`, `molviewer`, `molweight`, `pdbdistplot`, `seqtool`

MATLAB function: `plotyy`

Purpose

Plot properties of amino acid sequence

Syntax

```
proteinpropplot (SeqAA)
proteinpropplot(SeqAA, ...'PropertyTitle',
  PropertyTitleValue, ...)
proteinpropplot(SeqAA, ...'Startat', StartatValue, ...)
proteinpropplot(SeqAA, ...'Endat', EndatValue, ...)
proteinpropplot(SeqAA, ...'Smoothing', SmoothingValue, ...)
proteinpropplot(SeqAA, ...'EdgeWeight',
  EdgeWeightValue, ...)
proteinpropplot(SeqAA, ...'WindowLength',
  WindowLengthValue,
  ...)
```

proteinpropplot

Arguments

SeqAA Amino acid sequence. Enter any of the following:

- Character string of letters representing an amino acid
- Vector of integers representing an amino acid, such as returned by `aa2int`
- Structure containing a `Sequence` field that contains an amino acid sequence, such as returned by `getembl`, `getgenpept`, or `getpdb`

PropertyTitleValue String that specifies the property to plot. Default is `Hydrophobicity (Kyte & Doolittle)`. To display a list of properties to plot, enter a empty string for *PropertyTitleValue*. For example, type:

```
proteinpropplot(sequence, 'propertytitle', '')
```

Tip To access references for the properties, view the `proteinpropplot` m-file.

StartatValue Integer that specifies the starting point for the plot from the N-terminal end of the amino acid sequence *SeqAA*. Default is 1.

EndatValue Integer that specifies the ending point for the plot from the N-terminal end of the amino acid sequence *SeqAA*. Default is `length(SeqAA)`.

SmoothingValue String the specifies the smoothing method. Choices are:

- `linear` (default)
- `exponential`
- `lowess`

<i>EdgeWeightValue</i>	Value that specifies the edge weight used for linear and exponential smoothing methods. Decreasing this value emphasizes peaks in the plot. Choices are any value ≥ 0 and ≤ 1 . Default is 1.
<i>WindowLengthValue</i>	Integer that specifies the window length for the smoothing method. Increasing this value gives a smoother plot that shows less detail. Default is 11.

Description

proteinpropplot (*SeqAA*) displays a plot of the hydrophobicity (Kyte and Doolittle, 1982) of the residues in sequence *SeqAA*.

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

proteinpropplot(*SeqAA*, ... '*PropertyTitle*', *PropertyTitleValue*, ...) specifies a property to plot for the amino acid sequence *SeqAA*. Default is Hydrophobicity (Kyte & Doolittle). To display a list of possible properties to plot, enter an empty string for *PropertyTitleValue*. For example, type:

```
proteinpropplot(sequence, 'propertytitle', '')
```

Tip To access references for the properties, view the proteinpropplot M-file.

proteinpropplot(*SeqAA*, ... '*Startat*', *StartatValue*, ...) specifies the starting point for the plot from the N-terminal end of the amino acid sequence *SeqAA*. Default is 1.

proteinpropplot

`proteinpropplot(SeqAA, ...'Endat', EndatValue, ...)` specifies the ending point for the plot from the N-terminal end of the amino acid sequence `SeqAA`. Default is `length(SeqAA)`.

`proteinpropplot(SeqAA, ...'Smoothing', SmoothingValue, ...)` specifies the smoothing method. Choices are:

- linear (default)
- exponential
- lowess

`proteinpropplot(SeqAA, ...'EdgeWeight', EdgeWeightValue, ...)` specifies the edge weight used for linear and exponential smoothing methods. Decreasing this value emphasizes peaks in the plot. Choices are any value ≥ 0 and ≤ 1 . Default is 1.

`proteinpropplot(SeqAA, ...'WindowLength', WindowLengthValue, ...)` specifies the window length for the smoothing method. Increasing this value gives a smoother plot that shows less detail. Default is 11.

Examples

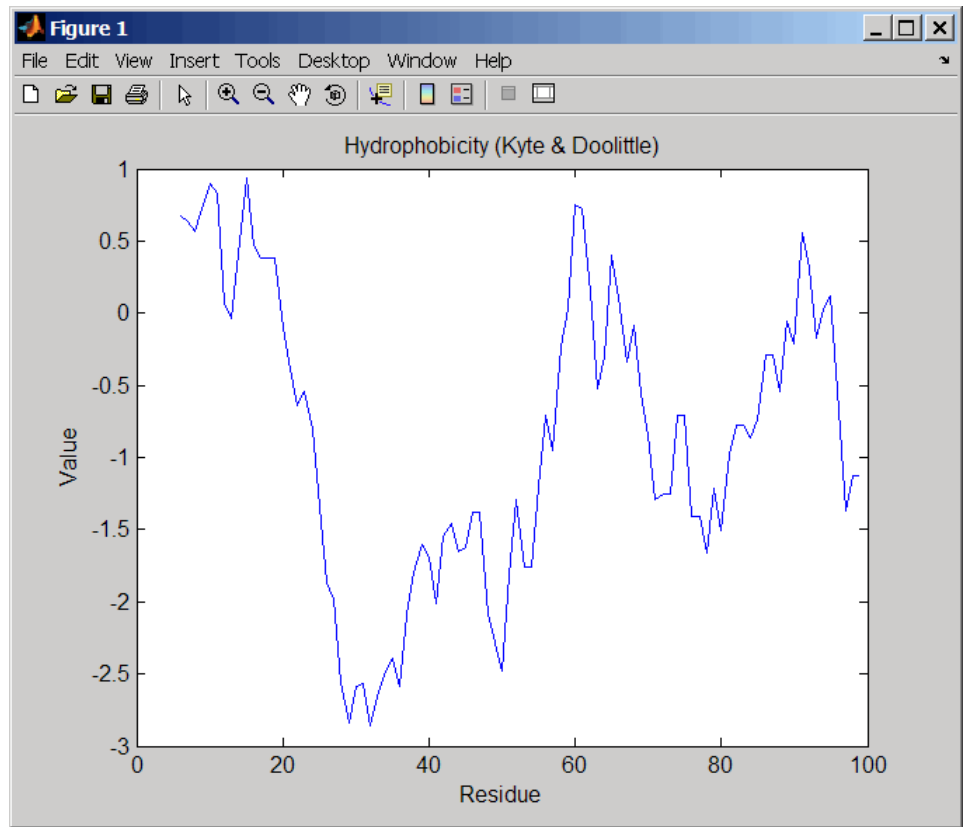
Plotting Hydrophobicity

- 1 Use the `getpdb` function to retrieve a protein sequence.

```
prion = getpdb('1HJM', 'SEQUENCEONLY', true);
```

- 2 Plot the hydrophobicity (Kyte and Doolittle, 1982) of the residues in the sequence.

```
proteinpropplot(prion)
```



Plotting Parallel Beta Strand

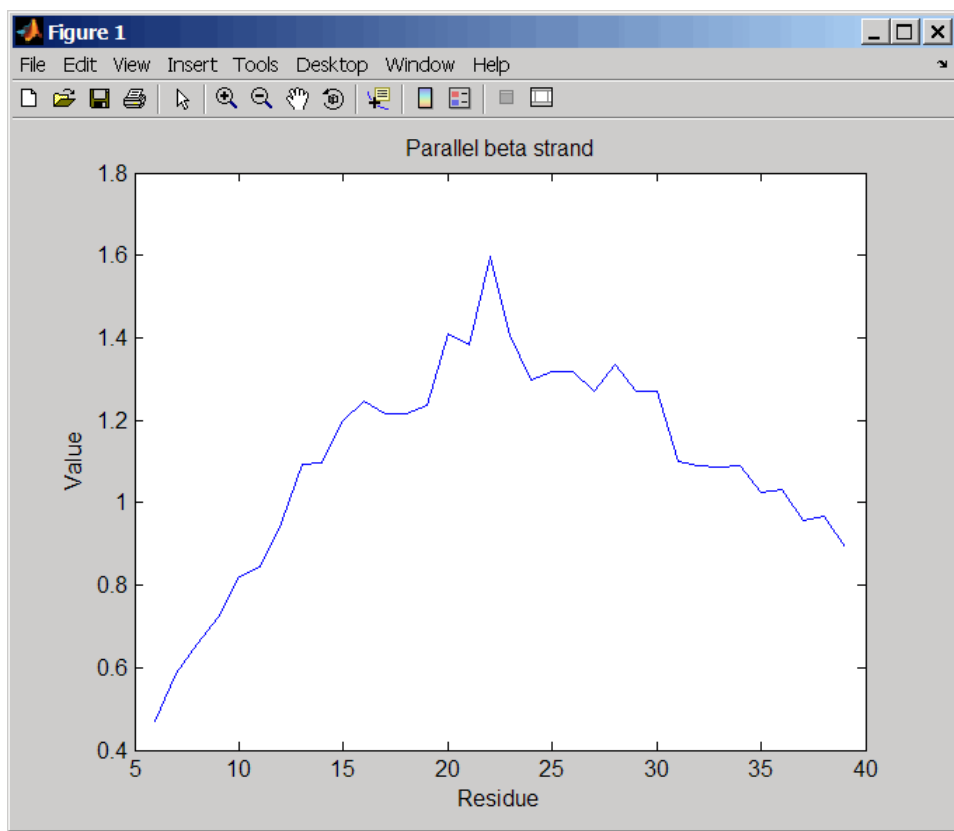
- 1 Use the `getgenpept` function to retrieve a protein sequence.

```
s = getgenpept('aad50640');
```

- 2 Plot the conformational preference for parallel beta strand for the residues in the sequence.

```
proteinpropplot(s,'propertytitle','Parallel beta strand')
```

proteinpropplot



References

[1] Kyte, J., and Doolittle, R.F. (1982). A simple method for displaying the hydrophatic character of a protein. *J Mol Biol* *157(1)*, 105–132.

See Also

Bioinformatics Toolbox functions: `account`, `atomiccomp`, `molviewer`, `molweight`, `pbdbdistplot`, `proteinplot`, `ramachandran`, `seqtool`

MATLAB function: `plotyy`

Purpose Quantile normalization over multiple arrays

Syntax
NormData = quantilenorm(*Data*)
NormData = quantilenorm(...,'MEDIAN', true)
NormData = quantilenorm(...,'DISPLAY', true)

Description *NormData* = quantilenorm(*Data*), where the columns of *Data* correspond to separate chips, normalizes the distributions of the values in each column.

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

NormData = quantilenorm(...,'MEDIAN', true) takes the median of the ranked values instead of the mean.

NormData = quantilenorm(...,'DISPLAY', true) plots the distributions of the columns and of the normalized data.

Examples

```
load yeastdata
normYeastValues = quantilenorm(yeastvalues,'display',1);
```

See Also affygcma, affyrma, malowess, manorm, rmabackadj, rmasummary

ramachandran

Purpose Draw Ramachandran plot for Protein Data Bank (PDB) data

Syntax

```
ramachandran(PDBid)  
ramachandran(File)  
ramachandran(PDBStruct)  
RamaStruct = ramachandran(...)  
ramachandran(..., 'Chain', ChainValue, ...)  
ramachandran(..., 'Plot', PlotValue, ...)  
ramachandran(..., 'Model', ModelValue, ...)  
ramachandran(..., 'Glycine', GlycineValue, ...)  
ramachandran(..., 'Regions', RegionsValue, ...)  
ramachandran(..., 'RegionDef', RegionDefValue, ...)
```

Arguments

<i>PDBid</i>	String specifying a unique identifier for a protein structure record in the PDB database.
--------------	---

Note Each structure in the PDB database is represented by a four-character alphanumeric identifier. For example, 4hbb is the identifier for hemoglobin.

<i>File</i>	String specifying a file name or a path and file name. The referenced file is a Protein Data Bank (PDB)-formatted file. If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Directory.
-------------	--

<i>PDBStruct</i>	MATLAB structure containing PDB-formatted data, such as returned by <code>getpdb</code> or <code>pdbread</code> .
------------------	---

<i>ChainValue</i>	<p>String or cell array of strings that specifies the chain(s) to compute the torsion angles for and plot.</p> <p>Choices are:</p> <ul style="list-style-type: none">• 'All' (default) — Torsion angles for all chains are computed and plotted.• A string specifying the chain ID, which is case sensitive.• A cell array of strings specifying chain IDs, which are case sensitive.
<i>PlotValue</i>	<p>String specifying how to plot chains. Choices are:</p> <ul style="list-style-type: none">• 'None' — Plots nothing.• 'Separate' — Plots torsion angles for all specified chains in separate plots.• 'Combined' (default) — Plots torsion angles for all specified chains in one combined plot.
<i>ModelValue</i>	<p>Integer that specifies the structure model to consider. Default is 1.</p>
<i>GlycineValue</i>	<p>Controls the highlighting of glycine residues with a circle in the plot. Choices are true or false (default).</p>

RegionsValue Controls the drawing of Ramachandran reference regions in the plot. Choices are true or false (default).

The default regions are core right-handed alpha, core beta, core left-handed alpha, and allowed, with the core regions corresponding to data points of preferred values of psi/phi angle pairs, and the allowed regions corresponding to possible, but disfavored values of psi/phi angle pairs, based on simple energy considerations. The boundaries of these default regions are based on the calculations by Morris et al., 1992.

Note If using the default colormap, red = right-handed core alpha, core beta, and core left-handed alpha, while yellow = allowed.

RegionDefValue MATLAB structure or array of structures (if specifying multiple regions) containing information (name, color, and boundaries) for custom reference regions in a Ramachandran plot. Each structure must contain the following fields:

- **Name** — String specifying a name for the region.
- **Color** — String or three-element numeric vector of RGB values specifying a color for the region in the plot.
- **Patch** — A 2-by-N matrix of values, the first row containing torsion angle phi (Φ) values, and the second row containing torsion angle psi (Ψ) values. When psi/phi angle pairs are plotted, the data points specify boundaries for the region. N is the number of data points needed to define the region.

Tip If you specify custom reference regions in which a smaller region is contained or covered by a larger region, list the structure for the smaller region first in the array so that it is plotted last and visible in the plot.

Return Values

RamaStruct

MATLAB structure or array of structures (if protein contains multiple chains). Each structure contains the following fields:

- Angles
- ResidueNum
- ResidueName
- Chain
- HPoints

For descriptions of the fields, see the following table.

Description

A Ramachandran plot is a plot of the torsion angle phi, Φ , (torsion angle between the C-N-CA-C atoms) versus the torsion angle psi, Ψ , (torsion angle between the N-CA-C-N atoms) for each residue of a protein sequence.

`ramachandran(PDBid)` generates the Ramachandran plot for the protein specified by the PDB database identifier *PDBid*.

`ramachandran(File)` generates the Ramachandran plot for the protein specified by *File*, a PDB-formatted file.

`ramachandran(PDBStruct)` generates the Ramachandran plot for the protein stored in *PDBStruct*, a MATLAB structure containing PDB-formatted data, such as returned by `getpdb` or `pdbread`.

RamaStruct = `ramachandran(...)` returns a MATLAB structure or array of structures (if protein contains multiple chains). Each structure contains the following fields.

ramachandran

Field	Description
Angles	<p data-bbox="609 317 1285 539">Three-column matrix containing the torsion angles phi (Φ), psi (Ψ), and omega (ω) for each residue in the sequence, ordered by residue sequence number. The number of rows in the matrix is equal to the number of rows in the <code>ResidueNum</code> column vector, which can be used to determine which residue corresponds to each row in the <code>Angles</code> matrix.</p> <hr data-bbox="609 591 1285 595"/> <p data-bbox="609 605 1285 795">Note The <code>Angles</code> matrix contains a row for each number in the range of residue sequence numbers, including residue sequence numbers missing from the PDB file. Rows corresponding to residue sequence numbers missing from the PDB file contain the value <code>NaN</code>.</p> <hr data-bbox="609 800 1285 803"/>

Field	Description
ResidueNum	<p>Column vector containing the residue sequence numbers from the PDB file.</p> <hr/> <p>Note The ResidueNum vector starts with one of the following:</p> <ul style="list-style-type: none"> • The lowest residue sequence number (if the lowest residue sequence number is negative or zero) • The number 1 (if the lowest residue sequence number is positive) <p>The ResidueNum vector ends with the highest residue sequence number and includes all numbers in the range, including residue sequence numbers missing from the PDB file.</p> <hr/> <p>The angles listed in the Angles matrix are in the same order as the residue sequence numbers in the ResidueNum vector. Therefore, you can use the ResidueNum vector to determine which residue corresponds to each row in the Angles matrix.</p>
ResidueName	Column vector containing the residue names for the protein.
Chain	A string specifying the chains in the protein.
HPoints	Handle to the data points in the plot.

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

ramachandran

`ramachandran(..., 'Chain', ChainValue, ...)` specifies the chain(s) to compute the torsion angles for and plot. Choices are:

- 'All' (default) — Torsion angles for all chains are computed and plotted.
- A string specifying the chain ID, which is case sensitive.
- A cell array of strings specifying chain IDs, which are case sensitive.

`ramachandran(..., 'Plot', PlotValue, ...)` specifies how to plot chains. Choices are:

- 'None' — Plots nothing.
- 'Separate' — Plots torsion angles for all specified chains in separate plots.
- 'Combined' (default) — Plots torsion angles for all specified chains in one combined plot.

`ramachandran(..., 'Model', ModelValue, ...)` specifies the structure model to consider. Default is 1.

`ramachandran(..., 'Glycine', GlycineValue, ...)` controls the highlighting of glycine residues with a circle in the plot. Choices are true or false (default).

`ramachandran(..., 'Regions', RegionsValue, ...)` controls the drawing of Ramachandran reference regions in the plot. Choices are true or false (default).

The default regions are core right-handed alpha, core beta, core left-handed alpha, and allowed, with the core regions corresponding to data points of preferred values of psi/phi angle pairs, and the allowed regions corresponding to possible, but disfavored values of psi/phi angle pairs, based on simple energy considerations. The boundaries of these default regions are based on the calculations by Morris et al., 1992.

Note If using the default colormap, then red = core right-handed alpha, core beta, and core left-handed alpha, while yellow = allowed.

`ramachandran(..., 'RegionDef', RegionDefValue, ...)` specifies information (name, color, and boundary) for custom reference regions in a Ramachandran plot. *RegionDefValue* is a MATLAB structure or array of structures containing the following fields:

- **Name** — String specifying a name for the region.
- **Color** — String or three-element numeric vector of RGB values specifying a color for the region in the plot.
- **Patch** — A 2-by-N matrix of values, the first row containing torsion angle phi (Φ) values, and the second row containing torsion angle psi (Ψ) values. When psi/phi angle pairs are plotted, the data points specify a boundary for the region. N is the number of data points needed to define the region.

Tip If you specify custom reference regions in which a smaller region is contained or covered by a larger region, list the structure for the smaller region first in the array so that it is plotted last and visible in the plot.

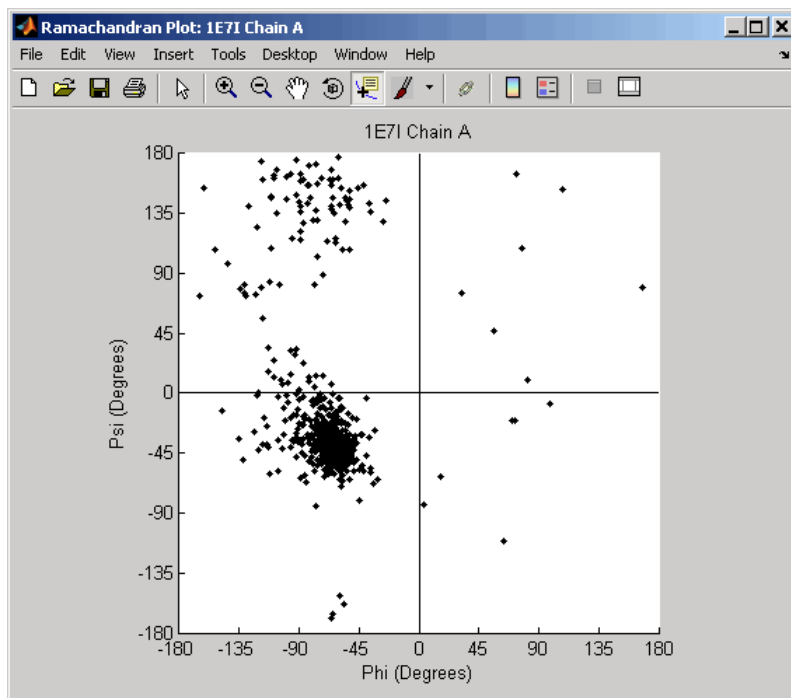
Examples

Drawing a Ramachandran Plot

Draw the Ramachandran plot for the human serum albumin complexed with octadecanoic acid, which has a PDB database identifier of 1E7I.

```
ramachandran('1E7I')
```

ramachandran



Drawing a Ramachandran Plot for a Specific Chain

- 1 Use the `getpdb` function to retrieve protein structure data for the human growth hormone from the PDB database, and save the information to a file.

```
getpdb('1a22', 'ToFile', '1a22.pdb');
```

- 2 Compute the torsion angles and draw the Ramachandran plot for chain A of the human growth hormone, represented in the pdb file, `1a22.pdb`.

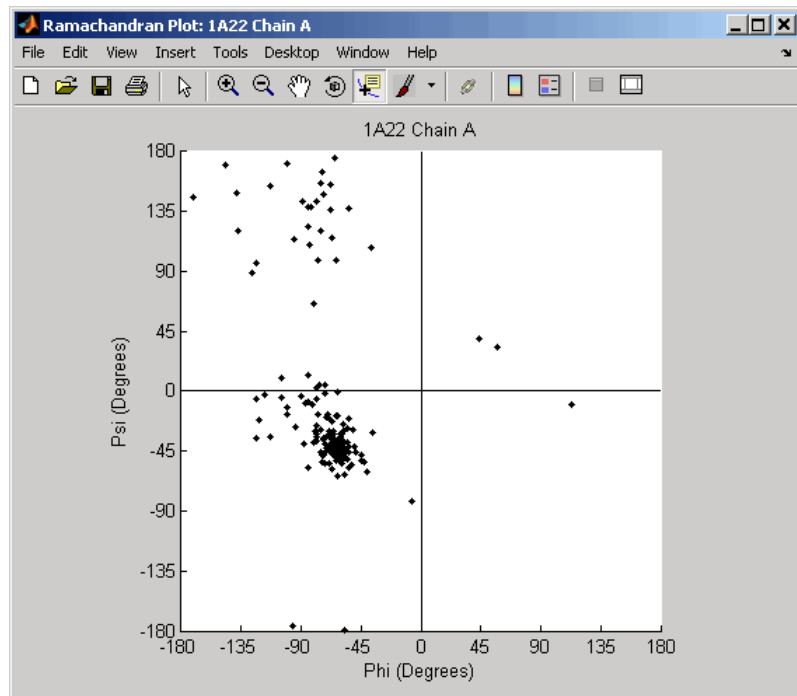
```
ChainA1a22Struct = ramachandran('1a22.pdb', 'chain', 'A')
```

```
ChainA1a22Struct =
```



```

Angles: [191x3 double]
ResidueNum: [191x1 double]
ResidueName: {191x1 cell}
Chain: 'A'
HPoints: 370.0012
    
```



Drawing Ramachandran Plots with Highlighted Glycine Residues and Ramachandran Regions

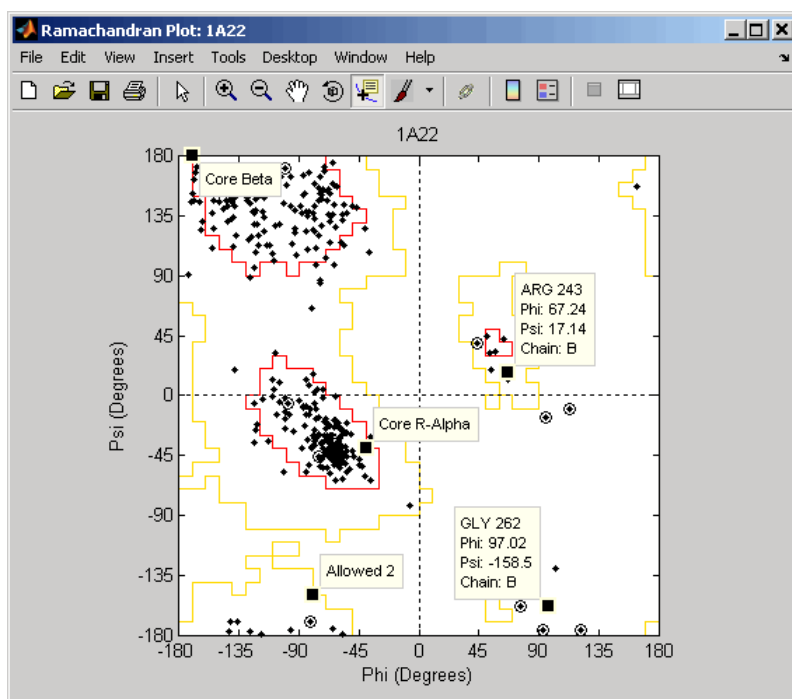
- 1 Use the `getpdb` function to retrieve protein structure data for the human growth hormone from the PDB database, and store the information in a structure.

ramachandran

```
Struct1a22 = getpdb('1a22');
```

- 2 Draw a combined Ramachandran plot for all chains of the human growth hormone, represented in the pdb structure, 1a22Struct. Highlight the glycine residues (with a circle), and draw the reference Ramachandran regions in the plot.

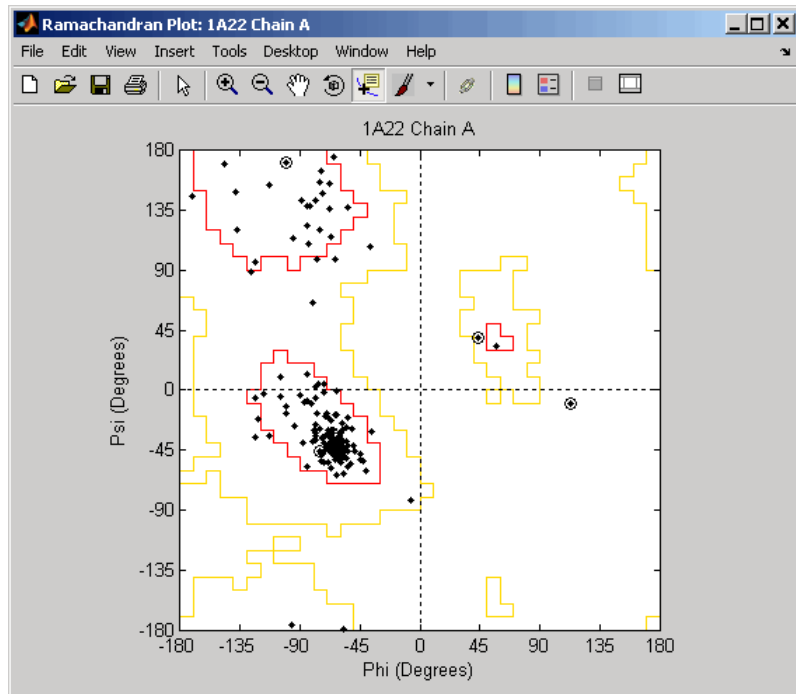
```
ramachandran(Struct1a22, 'glycine', true, 'regions', true);
```



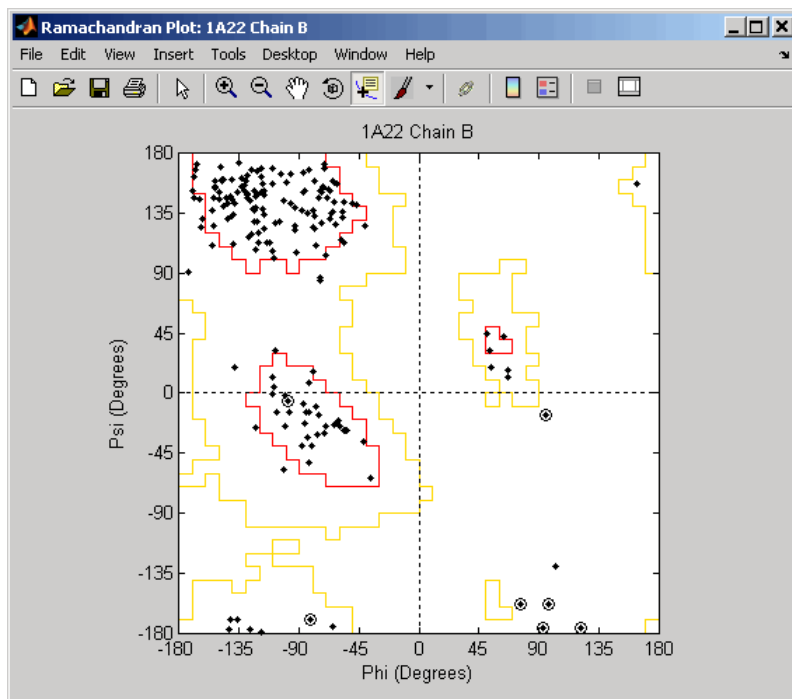
Tip Click a data point to display a data tip with information about the residue. Click a region to display a data tip defining the region. Press and hold the **Alt** key to display multiple data tips.

- 3** Draw a separate Ramachandran plot for each chain of the human growth hormone, represented in the pdb structure, 1a22Struct. Highlight the glycine residues (with a circle) and draw the reference Ramachandran regions in the plot.

```
ramachandran(Struct1a22,'plot','separate','chain','all',...
            'glycine',true,'regions',true)
```



ramachandran



Writing a Tab-Delimited Report File from a Ramachandran Structure

- 1 Create an array of two structures containing torsion angles for chains A and D in the Calcium/Calmodulin-dependent protein kinase, which has a PDB database identifier of 1hkx.

```
a = ramachandran('1hkx', 'chain', {'A', 'D'})
```

```
a =
```

```
1x2 struct array with fields:
```

```
Angles
```

```
ResidueNum
```

```
ResidueName
Chain
HPoints
```

- 2** Write a tab-delimited report file containing torsion angles phi (Φ) and psi (Ψ) for chains A and D in the Calcium/Calmodulin-dependent protein kinase.

```
fid = fopen('rama_1hkx_report.txt', 'wt');

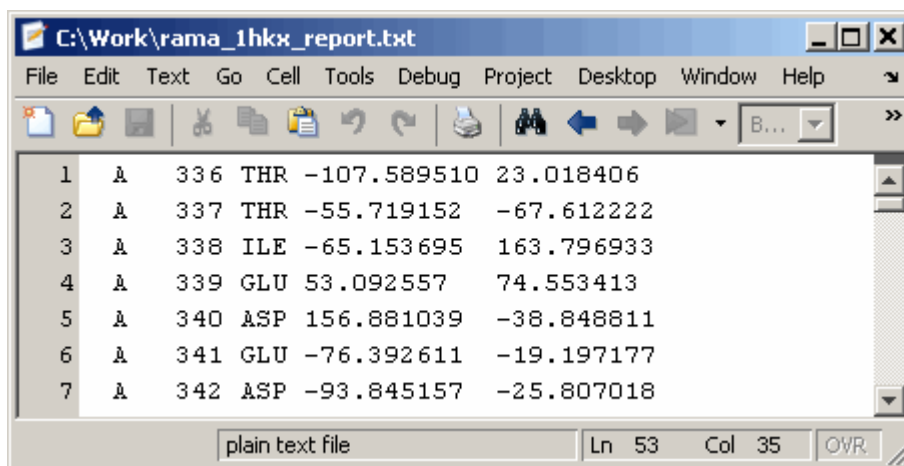
for c = 1:numel(a)
    for i = 1:length(a(c).Angles)
        if ~all(isnan(a(c).Angles(i,:)))
            fprintf(fid, '%s\t%d\t%s\t%f\t%f\n', a(c).Chain, ...
                a(c).ResidueNum(i), a(c).ResidueName{i}, ...
                a(c).Angles(i,1:2));
        end
    end
end

fclose(fid);
```

- 3** View the file you created in the MATLAB Editor.

```
edit rama_1hkx_report.txt
```

ramachandran



1	A	336	THR	-107.589510	23.018406
2	A	337	THR	-55.719152	-67.612222
3	A	338	ILE	-65.153695	163.796933
4	A	339	GLU	53.092557	74.553413
5	A	340	ASP	156.881039	-38.848811
6	A	341	GLU	-76.392611	-19.197177
7	A	342	ASP	-93.845157	-25.807018

References

[1] Morris, A.L., MacArthur, M.W., Hutchinson, E.G., and Thornton, J.M. (1992). Stereochemical Quality of Protein Structure Coordinates. *PROTEINS: Structure, Function, and Genetics* 12, 345–364.

See Also

Bioinformatics Toolbox functions: `getpdb`, `molviewer`, `pdbdistplot`, `pdbread`, `proteinpropplot`

Purpose Generate randomized subset of features

Syntax

```
[IDX, Z] = randfeatures(X, Group, 'PropertyName',
    PropertyValue...)
randfeatures(..., 'Classifier', C)
randfeatures(..., 'ClassOptions', CO)
randfeatures(..., 'PerformanceThreshold', PT)
randfeatures(..., 'ConfidenceThreshold', CT)
randfeatures(..., 'SubsetSize', SS)
randfeatures(..., 'PoolSize', PS)
randfeatures(..., 'NumberOfIndices', N)
randfeatures(..., 'CrossNorm', CN)
randfeatures(..., 'Verbose', VerboseValue)
```

Description [IDX, Z] = randfeatures(X, Group, 'PropertyName', PropertyValue...) performs a randomized subset feature search reinforced by classification. randfeatures randomly generates subsets of features used to classify the samples. Every subset is evaluated with the apparent error. Only the best subsets are kept, and they are joined into a single final pool. The cardinality for every feature in the pool gives the measurement of the significance.

X contains the training samples. Every column of X is an observed vector. Group contains the class labels. Group can be a numeric vector or a cell array of strings; numel(Group) must be the same as the number of columns in X, and numel(unique(Group)) must be greater than or equal to 2. Z is the classification significance for every feature. IDX contains the indices after sorting Z; i.e., the first one points to the most significant feature.

randfeatures(..., 'Classifier', C) sets the classifier. Options are

```
'da'    (default)  Discriminant analysis
'knn'   K nearest neighbors
```

randfeatures(..., 'ClassOptions', CO) is a cell with extra options for the selected classifier. Defaults are

randfeatures

{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 $0.95.^{(\text{number of classes})}$.

`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'	$x_{\text{new}} = (x - \text{mean}(x)) / \text{std}(x)$
'softmax'	$x_{\text{new}} = (1 + \exp((\text{mean}(x) - x) / \text{std}(x)))^{-1}$
'minmax'	$x_{\text{new}} = (x - \min(x)) / (\max(x) - \min(x))$

`randfeatures(..., 'Verbose', VerboseValue)`, when Verbose is true, turns off verbosity. Default is true.

Examples

Find a reduced set of genes that is sufficient for classification of all the cancer types in the t-matrix NCI60 data set. Load sample data.


```
load NCI60tmatrix
```

Select features.

```
I = randfeatures(X, GROUP, 'SubsetSize', 15, 'Classifier', 'da');
```

Test features with a linear discriminant classifier.

```
C = classify(X(I(1:25),:)', X(I(1:25),:)', GROUP);  
cp = classperf(GROUP, C);  
cp.CorrectRate
```

See Also

Bioinformatics Toolbox functions: `classperf`, `crossvalind`, `knnclassify`, `rankfeatures`, `svmclassify`

Statistics Toolbox functions: `classify`, `sequentialfs`

randseq

Purpose Generate random sequence from finite alphabet

Syntax

```
Seq = randseq(SeqLength)
Seq = randseq(SeqLength, ...'Alphabet', AlphabetValue, ...)
Seq = randseq(SeqLength, ...'Weights', WeightsValue, ...)
Seq = randseq(SeqLength, ...'FromStructure',
    FromStructureValue, ...)
Seq = randseq(SeqLength, ...'Case', CaseValue, ...)
Seq = randseq(SeqLength, ...'DataType', DataTypeValue, ...)
```

Arguments

<i>SeqLength</i>	Number of amino acids or nucleotides in random sequence .
<i>AlphabetValue</i>	Property to select the alphabet for the sequence. Enter 'dna'(default), 'rna', or 'amino'.
<i>WeightsValue</i>	Property to specify a weighted random sequence.
<i>FromStructureValue</i>	Property to specify a weighted random sequence using output structures from the functions from basecount, dimercount, codoncount, or aaccount.
<i>CaseValue</i>	Property to select the case of letters in a sequence when Alphabet is 'char'. Values are 'upper' (default) or 'lower'.
<i>DataTypeValue</i>	Property to select the data type for a sequence. Values are 'char'(default) for letter sequences, and 'uint8' or 'double' for numeric sequences.

Creates a sequence as an array of *DataType*.

Description *Seq = randseq(SeqLength)* creates a random sequence with a length specified by *SeqLength*.

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

`Seq = randseq(SeqLength, ...'Alphabet', AlphabetValue, ...)` generates a sequence from a specific alphabet.

`Seq = randseq(SeqLength, ...'Weights', WeightsValue, ...)` creates a weighted random sequence where the *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)`.

`Seq = randseq(SeqLength, ...'FromStructure', FromStructureValue, ...)` creates a weighted random sequence with weights given by the output structure from `basecount`, `dimercount`, `codoncount`, or `aacount`.

`Seq = randseq(SeqLength, ...'Case', CaseValue, ...)` specifies the case for a letter sequence.

`Seq = randseq(SeqLength, ...'DataType', DataTypeValue, ...)` specifies the data type for the sequence array.

Examples

Generate a random DNA sequence.

```
randseq(20)

ans =
TAGCTGGCCAAGCGAGCTTG
```

Generate a random RNA sequence.

```
randseq(20, 'alphabet', 'rna')

ans =
```

randseq

```
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)
[IDX, Z] = rankfeatures(X, Group, ...'Criterion',
CriterionValue, ...)
[IDX, Z] = rankfeatures(X, Group, ...'CCWeighting', ALPHA,
...)
[IDX, Z] = rankfeatures(X, Group, ...'NWeighting',
BETA, ...)
[IDX, Z] = rankfeatures(X, Group, ...'NumberOfIndices', N,
...)
[IDX, Z] = rankfeatures(X, Group, ...'CrossNorm', CN, ...)
```

Description

[*IDX*, *Z*] = rankfeatures(*X*, *Group*) ranks the features in *X* using an independent evaluation criterion for binary classification. *X* is a matrix where every column is an observed vector and the number of rows corresponds to the original number of features. *Group* contains the class labels.

IDX is the list of indices to the rows in *X* with the most significant features. *Z* is the absolute value of the criterion used (see below).

Group can be a numeric vector or a cell array of strings; numel(*Group*) is the same as the number of columns in *X*, and numel(unique(*Group*)) is equal to 2.

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

[*IDX*, *Z*] = rankfeatures(*X*, *Group*, ...'*Criterion*', *CriterionValue*, ...) sets the criterion used to assess the significance of every feature for separating two labeled groups. Choices are:

- '*ttest*' (default) — Absolute value two-sample t-test with pooled variance estimate.

rankfeatures

- 'entropy' — Relative entropy, also known as Kullback-Lieber distance or divergence.
- 'brattacharyya' — Minimum attainable classification error or Chernoff bound.
- 'roc' — Area between the empirical receiver operating characteristic (ROC) curve and the random classifier slope.
- 'wilcoxon' — Absolute value of the u-statistic of a two-sample unpaired Wilcoxon test, also known as Mann-Whitney.

Note 'ttest', 'entropy', and 'brattacharyya' assume normal distributed classes while 'roc' and 'wilcoxon' are nonparametric tests. All tests are feature independent.

[*IDX*, *Z*] = rankfeatures(*X*, *Group*, ...'CCWeighting', *ALPHA*, ...) uses correlation information to outweigh the *Z* value of potential features using $Z * (1 - ALPHA * (RHO))$, where *RHO* is the average of the absolute values of the cross-correlation coefficient between the candidate feature and all previously selected features. *ALPHA* sets the weighting factor. It is a scalar value between 0 and 1. When *ALPHA* is 0 (default) potential features are not weighted. A large value of *RHO* (close to 1) outweighs the significance statistic; this means that features that are highly correlated with the features already picked are less likely to be included in the output list.

[*IDX*, *Z*] = rankfeatures(*X*, *Group*, ...'NWeighting', *BETA*, ...) uses regional information to outweigh the *Z* value of potential features using $Z * (1 - \exp(- (DIST/BETA).^2))$, where *DIST* is the distance (in rows) between the candidate feature and previously selected features. *BETA* sets the weighting factor. It is greater than or equal to 0. When *BETA* is 0 (default) potential features are not weighted. A small *DIST* (close to 0) outweighs the significance statistics of only close features. This means that features that are close to already picked features are less likely to be included in the output list. This option is useful for extracting features from time series with temporal correlation.

BETA can also be a function of the feature location, specified using @ or an anonymous function. In both cases `rankfeatures` passes the row position of the feature to `BETA()` and expects back a value greater than or equal to 0.

Note You can use 'CCWeighting' and 'NWeighting' together.

`[IDX, Z] = rankfeatures(X, Group, ...'NumberOfIndices', N, ...)` sets the number of output indices in *IDX*. Default is the same as the number of features when *ALPHA* and *BETA* are 0, or 20 otherwise.

`[IDX, Z] = rankfeatures(X, Group, ...'CrossNorm', CN, ...)` applies independent normalization across the observations for every feature. Cross-normalization ensures comparability among different features, although it is not always necessary because the selected criterion might already account for this. Choices are:

- 'none' (default) — Intensities are not cross-normalized.
- 'meanvar' — $x_{\text{new}} = (x - \text{mean}(x)) / \text{std}(x)$
- 'softmax' — $x_{\text{new}} = (1 + \exp((\text{mean}(x) - x) / \text{std}(x)))^{-1}$
- 'minmax' — $x_{\text{new}} = (x - \min(x)) / (\max(x) - \min(x))$

Examples

- 1 Find a reduced set of genes that is sufficient for differentiating breast cancer cells from all other types of cancer in the t-matrix NCI60 data set. Load sample data.

```
load NCI60tmatrix
```

- 2 Get a logical index vector to the breast cancer cells.

```
BC = GROUP == 8;
```

- 3 Select features.

```
I = rankfeatures(X, BC, 'NumberOfIndices', 12);
```

- 4** Test features with a linear discriminant classifier.

```
C = classify(X(I,:)',X(I,:) ',double(BC));  
cp = classperf(BC,C);  
cp.CorrectRate
```

```
ans =
```

```
1
```

- 5** Use cross-correlation weighting to further reduce the required number of genes.

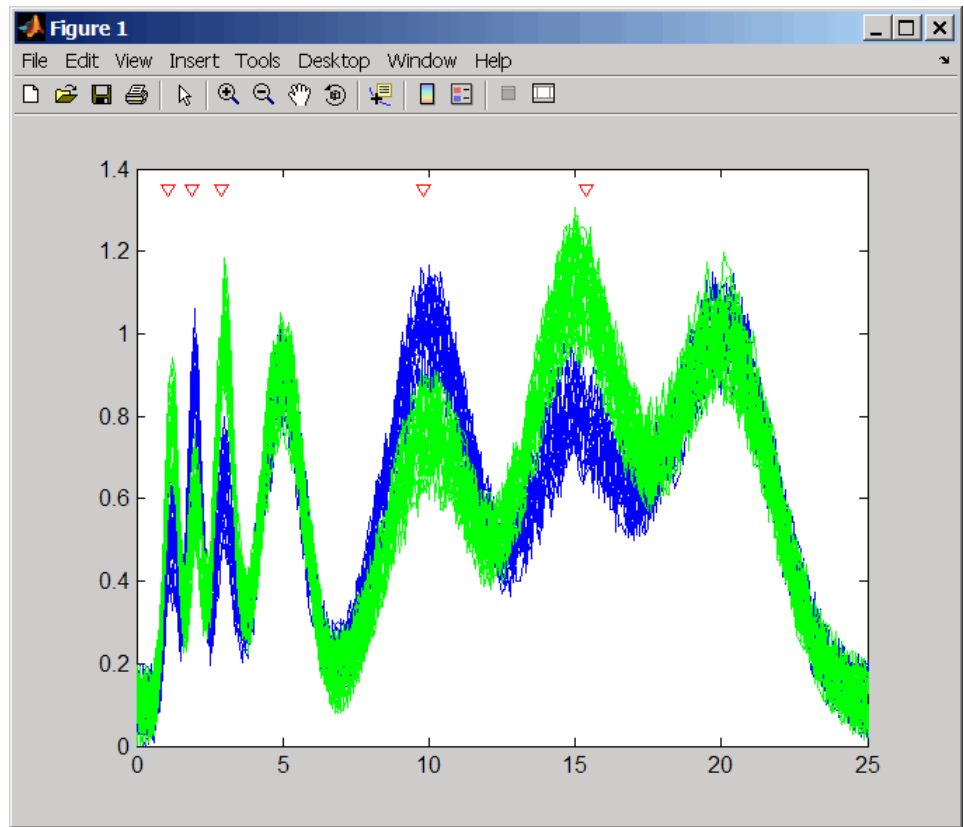
```
I = rankfeatures(X,BC,'CCWeighting',0.7,'NumberOfIndices',8);  
C = classify(X(I,:)',X(I,:) ',double(BC));  
cp = classperf(BC,C);  
cp.CorrectRate
```

```
ans =
```

```
1
```

- 6** Find the discriminant peaks of two groups of signals with Gaussian pulses modulated by two different sources.

```
load GaussianPulses  
f = rankfeatures(y',grp,'NWeighting',@(x) x/10+5,'NumberOfIndices',5);  
plot(t,y(grp==1,:), 'b',t,y(grp==2,:), 'g',t(f),1.35,'vr')
```

See Also

Bioinformatics Toolbox functions: `classperf`, `crossvalind`, `randfeatures`, `svmclassify`

Statistics Toolbox functions: `classify`, `sequentialfs`

rebasecuts

Purpose Find restriction enzymes that cut nucleotide sequence

Syntax
`[Enzymes, Sites] = rebasecuts(SeqNT)`
`rebasecuts(SeqNT, Group)`
`rebasecuts(SeqNT, [Q, R])`
`rebasecuts(SeqNT, S)`

Arguments

SeqNT Nucleotide sequence.

Group Cell array with the names of valid restriction enzymes.

Q, R, S Base positions that limit the search.

Return Values

Enzymes Cell array with the names of restriction enzymes from REBASE, the Restriction Enzyme Database.

Sites Vector of cut sites identified with the base position number before every cut.

Description

`[Enzymes, Sites] = rebasecuts(SeqNT)` finds all the restriction enzymes that cut *SeqNT*, a nucleotide sequence.

`rebasecuts(SeqNT, Group)` limits the search to *Group*, a list of enzymes.

`rebasecuts(SeqNT, [Q, R])` limits the search to those enzymes that cut after the base position specified by *Q* and before the base position specified by *R*.

`rebasecuts(SeqNT, S)` limits the search to those enzymes that cut just after the base position specified by *S*.

REBASE, the Restriction Enzyme Database, is a collection of information about restriction enzymes and related proteins. For more information about REBASE, see:

`http://rebase.neb.com/rebase/rebase.html`

Examples

- 1 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`, `cleavelookup`, `restrict`, `seq2regexp`, `seqshowwords`

MATLAB function: `regexp`

redbluecmap

Purpose Create red and blue colormap

Syntax `redbluecmap(Length)`

Arguments

Length Positive integer that specifies the length of (or the number of colors in) the colormap. Choices are positive integers ≥ 3 or ≤ 11 . Default is 11.

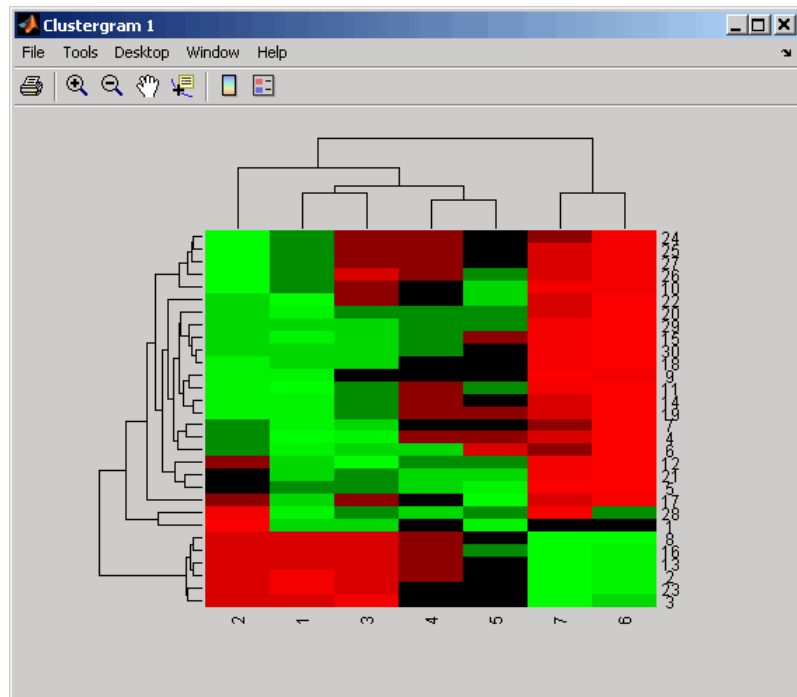
Description

`redbluecmap(Length)` returns a *Length*-by-3 matrix containing a red and blue diverging color palette. Low values are dark blue, values in the center of the map are white, and high values are dark red. *Length* is a positive integer ≥ 3 and ≤ 11 , which determines the number of colors in the colormap. Default is 11.

Examples

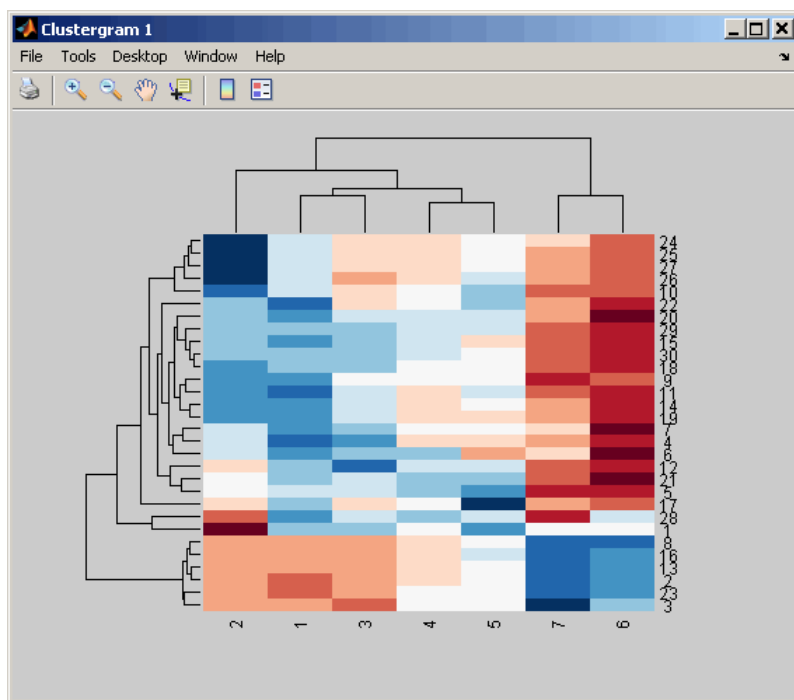
- 1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains `yeastvalues`, a matrix of gene expression data. Create a clustergram object and display the dendrograms and heat map from the gene expression data in the first 30 rows of the `yeastvalues` matrix.

```
load filteredyeastdata
cgo = clustergram(yeastvalues(1:30,:))
Clustergram object with 30 rows of nodes and 7 columns of nodes.
```



2 Reset the colormap of the heat map to redbluemap.

```
set(cgo, 'Colormap',redbluemap);
```



References

[1] <http://colorbrewer.org>

See Also

Bioinformatics Toolbox functions: `clustergram`, `redgreencmap`

MATLAB functions: `colormap`, `colormapeditor`

Purpose Create red and green colormap

Syntax `redgreencmap(Length)`
`redgreencmap(Length, 'Interpolation', InterpolationValue)`

Arguments

<i>Length</i>	Length of the colormap. Enter either 256 or 64. Default is the length of the colormap of the current figure.
<i>InterpolationValue</i>	Property that lets you set the algorithm for color interpolation. Choices are: <ul style="list-style-type: none">• 'linear'• 'quadratic'• 'cubic'• 'sigmoid' (default)

Note The sigmoid interpolation is tanh.

Description `redgreencmap(Length)` returns a *Length*-by-3 matrix containing a red and green colormap. Low values are bright green, values in the center of the map are black, and high values are red. Enter either 256 or 64 for *Length*. If *Length* is empty, the length of the map will be the same as the length of the colormap of the current figure.

`redgreencmap(Length, 'Interpolation', InterpolationValue)` lets you set the algorithm for color interpolation. Choices are:

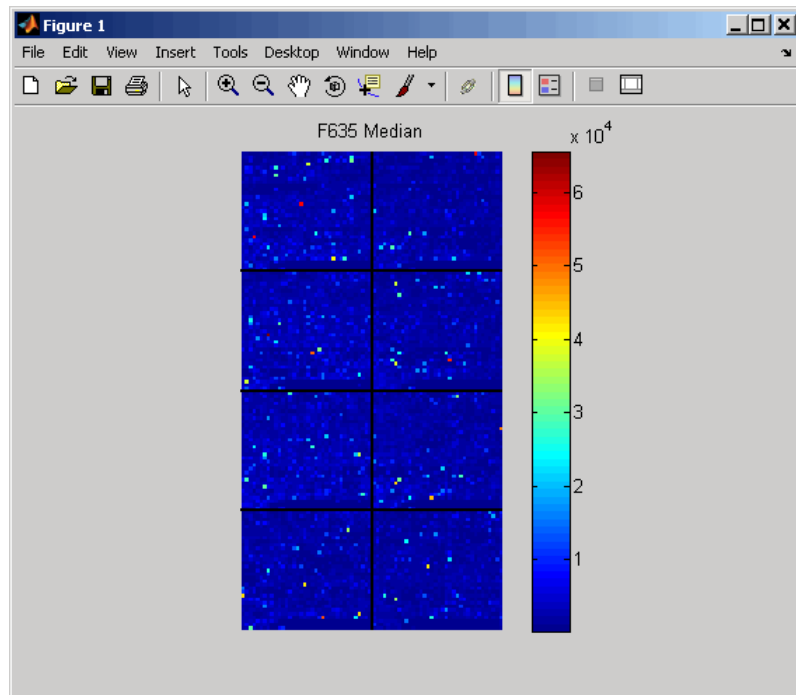
- 'linear'
- 'quadratic'
- 'cubic'
- 'sigmoid' (default)

Note The sigmoid interpolation is \tanh .

Examples

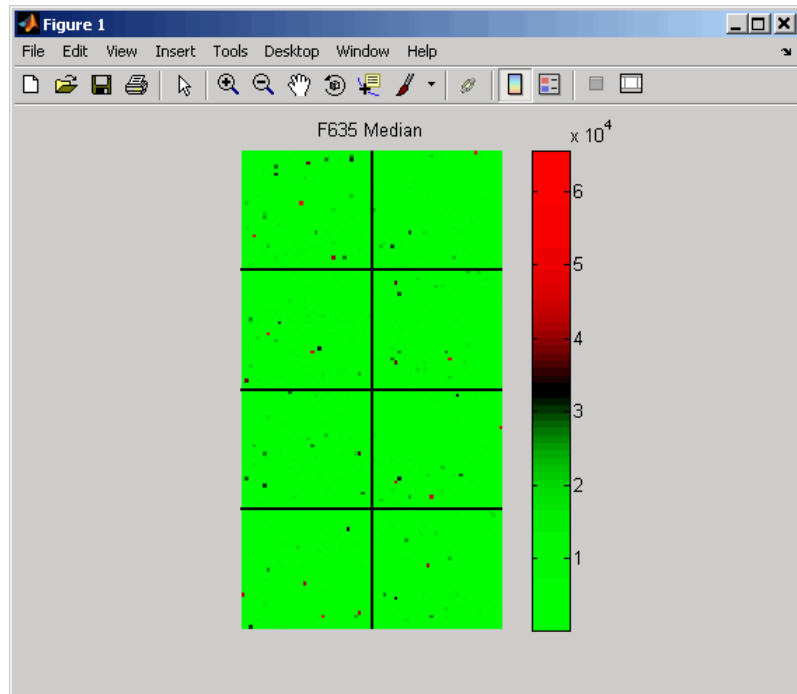
- 1 Create a MATLAB structure from the microarray data in a GenePix Results (GPR) file, then display an image of the 'F635 Median' field.

```
pd = gprread('mouse_a1pd.gpr');  
mimage(pd, 'F635 Median')
```



- 2 Reset the colormap of the current figure.

```
colormap(redgreenmap)
```


**See Also**

Bioinformatics Toolbox function: `clustergram`, `redbluecmap`

MATLAB functions: `colormap`, `colormapeditor`

restrict

Purpose Split nucleotide sequence at restriction site

Syntax

```
Fragments = restrict(SeqNT, Enzyme)  
Fragments = restrict(SeqNT, Pattern, Position)  
[Fragments, CuttingSites] = restrict(...)  
[Fragments, CuttingSites, Lengths] = restrict(...)  
... = restrict(..., 'PartialDigest', PartialDigestValue)
```

Arguments

<i>SeqNT</i>	Nucleotide sequence. Enter either a character string with the characters A, C, G, T, 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 <i>Sequence</i> .
<i>Enzyme</i>	Enter the name of a restriction enzyme from REBASE, the Restriction Enzyme Database.
<i>Pattern</i>	Enter a short nucleotide pattern. <i>Pattern</i> can be a regular expression.
<i>Position</i>	Defines the position on <i>Pattern</i> where the sequence is cut. <i>Position</i> =0 corresponds to the 5' end of <i>Pattern</i> .
<i>PartialDigestValue</i>	Property to specify a probability for partial digestion. Enter a value from 0 to 1.

Description

Fragments = restrict(*SeqNT*, *Enzyme*) cuts *SeqNT*, a nucleotide sequence, into fragments at the restriction sites of *Enzyme*, a restriction enzyme. The return values are stored in *Fragments*, a cell array of sequences.

Fragments = restrict(*SeqNT*, *Pattern*, *Position*) cuts *SeqNT*, a nucleotide sequence into fragments at restriction sites specified by *Pattern*, a nucleotide pattern.

[*Fragments*, *CuttingSites*] = restrict(...) returns a numeric vector with the indices representing the cutting sites. A 0 (zero) is

added to the list so `numel(Fragments)==numel(CuttingSites)`. You can use `CuttingSites+1` to point to the first base of every fragment respective to the original sequence.

`[Fragments, CuttingSites, Lengths] = restrict(...)` returns a numeric vector with the lengths of every fragment.

`... = restrict(..., 'PartialDigest', PartialDigestValue)` simulates a partial digest where each restriction site in the sequence has a *PartialDigestValue* or probability of being cut.

REBASE, the restriction enzyme database, is a collection of information about restriction enzymes and related proteins. For more information about REBASE or to search REBASE for the name of a restriction enzyme, see:

<http://rebase.neb.com/rebase/rebase.html>

Examples

- 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'
```

restrict

```
'CGCTCTGAAAAGCGGGAACCTCGTGG'  
'CGCTTTATTAA'
```

- 4 Use a regular expression for the enzyme pattern.

```
fragmentsRegExp = restrict(Seq, 'GCG[ ^C]', 3)
```

```
fragmentsRegExp =
```

```
'AGAGGGGTACGCGCTCTGAAAAGCG'  
'GGAACCTCGTGGCGCTTTATTAA'
```

- 5 Capture the cutting sites and fragment lengths with the fragments.

```
[fragments, cut_sites, lengths] = restrict(Seq, 'HspAI')
```

```
fragments =
```

```
'AGAGGGGTACG'  
'CGCTCTGAAAAGCGGGAACCTCGTGG'  
'CGCTTTATTAA'
```

```
cut_sites =
```

```
0  
11  
37
```

```
lengths =
```

```
11  
26  
11
```

See Also

Bioinformatics Toolbox functions: `cleave`, `cleavelookup`, `rebasecuts`, `seq2regexp`, `seqshowwords`

MATLAB function: `regexp`

Purpose Return reverse mapping (amino acid to nucleotide codon) for genetic code

Syntax

```
Map = revgeneticcode
Map = revgeneticcode(GeneticCode)
Map = revgeneticcode(..., 'Alphabet', AlphabetValue, ...)
Map = revgeneticcode(..., 'ThreeLetterCodes',
    ThreeLetterCodesValue, ...)
```

Arguments

<i>GeneticCode</i>	Integer or string specifying a genetic code number or code name from the table Genetic Code on page 2-935. Default is 1 or 'Standard'.
--------------------	--

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

<i>AlphabetValue</i>	String specifying the nucleotide alphabet to use in the map. Choices are:
----------------------	---

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

<i>ThreeLetterCodesValue</i>	Controls the use of three-letter amino acid codes as field names in the return structure <i>Map</i> . Choices are <code>true</code> for three-letter codes or <code>false</code> for one-letter codes. Default is <code>false</code> .
------------------------------	--

revgeneticcode

Return Values

Map

Structure containing the reverse mapping of amino acids to nucleotide codons for the standard genetic code. The *Map* structure contains a field for each amino acid.

Description

Map = revgeneticcode returns a structure containing the reverse mapping of amino acids to nucleotide codons for the standard genetic code. The *Map* structure contains a field for each amino acid.

Map = revgeneticcode(*GeneticCode*) returns a structure containing the reverse mapping of amino acids to nucleotide codons for the specified genetic code. *GeneticCode* is either:

- An integer or string specifying a code number or code name from the table Genetic Code on page 2-935
- The `transl_table` (code) number from the NCBI Web page describing genetic codes:

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

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

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

Map = revgeneticcode(..., '*Alphabet*', *AlphabetValue*, ...) specifies the nucleotide alphabet to use in the map. *AlphabetValue* can

be 'DNA', which uses the symbols A, C, G, and T, or 'RNA', which uses the symbols A, C, G, and U. Default is 'DNA'.

`Map = revgeneticcode(..., 'ThreeLetterCodes', ThreeLetterCodesValue, ...)` controls the use of three-letter amino acid codes as field names in the return structure *Map*. *ThreeLetterCodesValue* can be true for three-letter codes or false for one-letter codes. Default is false.

Genetic Code

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

revgeneticcode

Genetic Code (Continued)

Code Number	Code Name
22	Scenedesmus Obliquus Mitochondrial
23	Thraustochytrium Mitochondrial

Examples

- Return the reverse mapping of amino acids to nucleotide codons for the Standard genetic code.

```
map = revgeneticcode
```

```
map =
```

```
Name: 'Standard'  
A: { 'GCT' 'GCC' 'GCA' 'GCG' }  
R: { 'CGT' 'CGC' 'CGA' 'CGG' 'AGA' 'AGG' }  
N: { 'AAT' 'AAC' }  
D: { 'GAT' 'GAC' }  
C: { 'TGT' 'TGC' }  
Q: { 'CAA' 'CAG' }  
E: { 'GAA' 'GAG' }  
G: { 'GGT' 'GGC' 'GGA' 'GGG' }  
H: { 'CAT' 'CAC' }  
I: { 'ATT' 'ATC' 'ATA' }  
L: { 'TTA' 'TTG' 'CTT' 'CTC' 'CTA' 'CTG' }  
K: { 'AAA' 'AAG' }  
M: { 'ATG' }  
F: { 'TTT' 'TTC' }  
P: { 'CCT' 'CCC' 'CCA' 'CCG' }  
S: { 'TCT' 'TCC' 'TCA' 'TCG' 'AGT' 'AGC' }  
T: { 'ACT' 'ACC' 'ACA' 'ACG' }  
W: { 'TGG' }  
Y: { 'TAT' 'TAC' }  
V: { 'GTT' 'GTC' 'GTA' 'GTG' }
```



```
Stops: {'TAA' 'TAG' 'TGA'}  
Starts: {'TTG' 'CTG' 'ATG'}
```

- Return the reverse mapping of amino acids to nucleotide codons for the Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma genetic code, using the rna alphabet.

```
moldmap = revgeneticcode(4, 'Alphabet', 'rna');
```

- Return the reverse mapping of amino acids to nucleotide codons for the Flatworm Mitochondrial genetic code, using three-letter codes for the field names in the return structure.

```
wormmap = revgeneticcode('Flatworm Mitochondrial',...  
                          'ThreeLetterCodes', true);
```

References

- [1] NCBI Web page describing genetic codes:

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

See Also

Bioinformatics Toolbox functions: aa2nt, aminolookup, baselookup, geneticcode, nt2aa

rmabackadj

Purpose

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

Syntax

```
BackAdjustedMatrix = rmabackadj(PMData)  
BackAdjustedMatrix = rmabackadj(..., 'Method',  
MethodValue, ...)  
BackAdjustedMatrix = rmabackadj(..., 'Truncate',  
TruncateValue, ...)  
BackAdjustedMatrix = rmabackadj(..., 'Showplot',  
ShowplotValue, ...)
```

Arguments

<i>PMData</i>	Matrix of intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)
<i>MethodValue</i>	Specifies the estimation method for the background adjustment model parameters. Enter either 'RMA' (to use estimation method described by Bolstad, 2005) or 'MLE' (to estimate the parameters using maximum likelihood). Default is 'RMA'.

- TruncateValue* Specifies the background noise model. Enter either `true` (use a truncated Gaussian distribution) or `false` (use a nontruncated Gaussian distribution). Default is `true`.
- ShowplotValue* Controls the plotting of a histogram showing the distribution of PM probe intensity values (blue) and the convoluted probability distribution function (red), with estimated parameters `mu`, `sigma` and `alpha`. Enter either `'all'` (plot a histogram for each column or chip) or specify a subset of columns (chips) by entering the column number, list of numbers, or range of numbers.

For example:

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

Return Values

BackAdjustedMatrix Matrix of background-adjusted probe intensity values.

Description

BackAdjustedMatrix = `rmabackadj(PMData)` returns the background adjusted values of probe intensity values in the matrix, *PMData*. Note that each row in *PMData* corresponds to a perfect match (PM) probe and each column in *PMData* corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.) Details on the background adjustment are described by Bolstad, 2005.

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

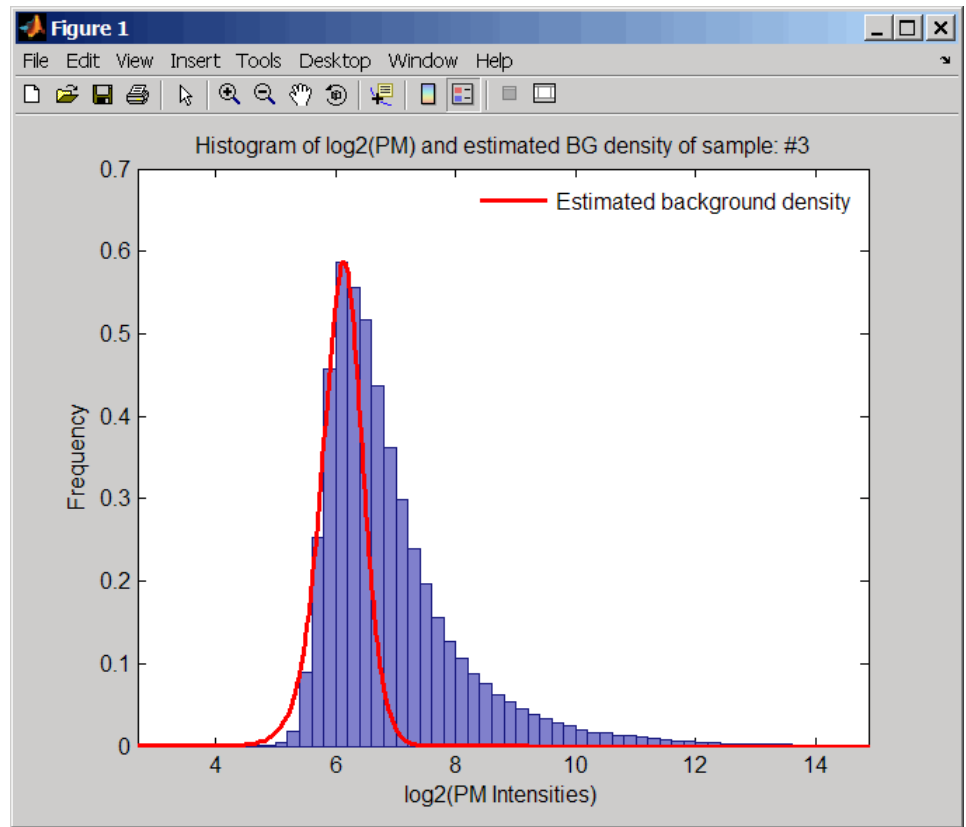
BackAdjustedMatrix = `rmabackadj(..., 'Method', MethodValue, ...)` specifies the estimation method for the background adjustment model parameters. When *MethodValue* is 'RMA', `rmabackadj` implements the estimation method described by Bolstad, 2005. When *MethodValue* is 'MLE', `rmabackadj` estimates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

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

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

For example:

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



Examples

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

```
load prostatecancerrawdata
```

- 2 Perform background adjustment on the PM probe intensity values in the matrix, `pmMatrix`, creating a new matrix, `BackgroundAdjustedMatrix`.

```
BackgroundAdjustedMatrix = rmabackadj(pmMatrix);
```

- 3** Perform background adjustment on the PM probe intensity values in only column 3 of the matrix, `pmMatrix`, creating a new matrix, `BackgroundAdjustedChip3`.

```
BackgroundAdjustedChip3 = rmabackadj(pmMatrix(:,3));
```

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

References

- [1] Irizarry, R.A., Hobbs, B., Collin, F., Beazer-Barclay, Y.D., Antonellis, K.J., Scherf, U., Speed, T.P. (2003). Exploration, Normalization, and Summaries of High Density Oligonucleotide Array Probe Level Data. *Biostatistics* 4, 249–264.
- [2] Bolstad, B. (2005). “affy: Built-in Processing Methods”
<http://www.bioconductor.org/packages/2.1/bioc/vignettes/affy/inst/doc/builtinMethods.pdf>
- [3] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L., Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. *Clinical Cancer Research* 11, 6823–6834.

See Also

Bioinformatics Toolbox functions: `affyinvarsetnorm`, `affyread`, `affyrma`, `celintensityread`, `probelibraryinfo`, `probesetlink`, `probesetlookup`, `probesetvalues`, `quantilenorm`, `rmasummary`

Purpose

Calculate gene expression values from Affymetrix microarray probe-level data using Robust Multi-array Average (RMA) procedure

Syntax

```
ExpressionMatrix = rmasummary(ProbeIndices, Data)  
ExpressionMatrix = rmasummary(ProbeIndices, Data,  
'Output', OutputValue)
```

Arguments

ProbeIndices

Column vector of probe indices. The convention for probe indices is, for each probe set, to label each probe 0 to $N - 1$, where N is the number of probes in the probe set.

Tip Use the `ProbeIndices` field in the structure returned by `celintensityread` as the *ProbeIndices* input.

Data

Matrix of natural-scale intensity values where each row corresponds to a perfect match (PM) probe and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.)

Tip Using a single-precision matrix for *Data* decreases memory usage.

Tip You can use the matrix from the `PMIntensities` field in the structure returned by `celintensityread` as the *Data* input. However, first ensure the matrix has been background adjusted, using the `rmabackadj` or `gcrmabackadjfunction`, and normalized, using the `quantilenorm` function.

OutputValue Specifies the scale of the returned gene expression values. *OutputValue* can be:

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

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

Description

ExpressionMatrix = `rmasummary(ProbeIndices, Data)` returns gene (probe set) expression values after calculating them from natural-scale probe intensities in the matrix *Data*, using the column vector of probe indices, *ProbeIndices*. Note that each row in *Data* corresponds to a perfect match (PM) probe, and each column corresponds to an Affymetrix CEL file. (Each CEL file is generated from a separate chip. All chips should be of the same type.) Note that the column vector *ProbeIndices* designates probes within each probe set by labeling each probe 0 to $N - 1$, where N is the number of probes in the probe set. Note that each row in *ExpressionMatrix* corresponds to a gene (probe set) and each column in *ExpressionMatrix* corresponds to an Affymetrix CEL file, which represents a single chip.

For a given probe set n , with J probe pairs, let Y_{ijn} denote the background-adjusted, base 2 log transformed and quantile-normalized PM probe intensity value of chip i and probe j . Y_{ijn} follows a linear additive model:

$$Y_{ijn} = U_{in} + A_{jn} + E_{ijn}; i = 1, \dots, I; j = 1, \dots, J; n = 1, \dots, N$$

where:

U_{in} = Gene expression of the probe set n on chip i

A_{jn} = Probe affinity effect for the j th probe in the probe set

E_{ijn} = Residual for the j th probe on the i th chip

The RMA method assumes $A_1 + A_2 + \dots + A_J = 0$ for all probe sets. A robust procedure, median polish, estimates U_i as the log scale measure of expression.

Note There is no column in *ExpressionMatrix* that contains probe set or gene information.

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

ExpressionMatrix = rmasummary(ProbeIndices, Data, 'Output', OutputValue) specifies the scale of the returned gene expression values. *OutputValue* can be:

- 'log'
- 'log2'
- 'log10'
- 'natural'
- @functionname

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

Examples

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

```
load prostatecancerrawdata
```

- 2 Perform background adjustment on the PM probe intensity values in the matrix, `pmMatrix`, using the `rmabackadj` function, thereby creating a new matrix, `BackgroundAdjustedMatrix`.

```
BackgroundAdjustedMatrix = rmabackadj(pmMatrix);
```

- 3 Normalize the data in `BackgroundAdjustedMatrix`, using the `quantilenorm` function.

```
NormMatrix = quantilenorm(BackgroundAdjustedMatrix);
```

- 4 Calculate gene expression values from the probe intensities in `NormMatrix`, creating a new matrix, `ExpressionMatrix`. (Use the `probeIndices` column vector provided to supply information on the probe indices.)

```
ExpressionMatrix = rmasummary(probeIndices, NormMatrix);
```

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

References

- [1] Irizarry, R.A., Hobbs, B., Collin, F., Beazer-Barclay, Y.D., Antonellis, K.J., Scherf, U., Speed, T.P. (2003). Exploration, Normalization, and Summaries of High Density Oligonucleotide Array Probe Level Data. *Biostatistics*. 4, 249–264.
- [2] Mosteller, F., and Tukey, J. (1977). *Data Analysis and Regression* (Reading, Massachusetts: Addison-Wesley Publishing Company), pp. 165–202.
- [3] Best, C.J.M., Gillespie, J.W., Yi, Y., Chandramouli, G.V.R., Perlmutter, M.A., Gathright, Y., Erickson, H.S., Georgevich, L.,

Tangrea, M.A., Duray, P.H., Gonzalez, S., Velasco, A., Linehan, W.M., Matusik, R.J., Price, D.K., Figg, W.D., Emmert-Buck, M.R., and Chuaqui, R.F. (2005). Molecular alterations in primary prostate cancer after androgen ablation therapy. *Clinical Cancer Research* 11, 6823–6834.

See Also

affygcma, affyinvaretnorm, affyrma, celintensityread, gcrmabackadj, mainvarsetnorm, malowess, manorm, quantilenorm, rmabackadj

Purpose

Convert RNA sequence of nucleotides to DNA sequence

Syntax

```
SeqDNA = rna2dna(SeqRNA)
```

Arguments

SeqRNA Nucleotide sequence for RNA. Enter a character string with the characters A, C, U, G, and the ambiguous nucleotide bases N, R, Y, K, M, S, W, B, D, H, and V.

Description

SeqDNA = `rna2dna(SeqRNA)` converts any uracil nucleotides in an RNA sequence into thymine (U→T), and returns in the same format as DNA. For example, if the RNA sequence is an integer sequence then so is *SeqRNA*.

Example

```
rna2dna('ACGAUGAGUCAUGCUU')  
  
ans =  
ACGATGAGTCATGCTT
```

See Also

Bioinformatics Toolbox function: `dna2rna`
MATLAB functions: `strrep`, `regexp`

rnaconvert

Purpose Convert secondary structure of RNA sequence between bracket and matrix notations

Syntax `RNAstruct2 = rnaconvert(RNAstruct)`

Arguments `RNAstruct` Secondary structure of an RNA sequence represented by either:

- Bracket notation
- Connectivity matrix

Tip Use the `rnafold` function to create `RNAstruct`.

Return Values `RNAstruct2` Secondary structure of an RNA sequence represented by either:

- **Bracket notation** — String of dots and brackets, where each dot represents an unpaired base, while a pair of equally nested, opening and closing brackets represents a base pair.
- **Connectivity matrix** — Binary, upper-triangular matrix, where $RNAmatrix(i, j) = 1$ if and only if the i th residue in the RNA sequence `Seq` is paired with the j th residue of `Seq`.

Description `RNAstruct2 = rnaconvert(RNAstruct)` returns `RNAstruct2`, the secondary structure of an RNA sequence, in matrix notation (if `RNAstruct` is in bracket notation), or in bracket notation (if `RNAstruct` is in matrix notation).

Examples

Converting from Bracket to Matrix Notation

- 1 Create a string representing a secondary structure of an RNA sequence in bracket notation.

```
Bracket = '((..((((.....))))).((.....)).)';
```

- 2 Convert the secondary structure to a connectivity matrix representation.

```
Matrix = rnaconvert(Bracket);
```

Converting from Matrix to Bracket Notation

- 1 Create a connectivity matrix representing a secondary structure of an RNA sequence.

```
Matrix2 = zeros(12);  
Matrix2(1,12) = 1;  
Matrix2(2,11) = 1;  
Matrix2(3,10) = 1;  
Matrix2(4,9) = 1;
```

- 2 Convert the secondary structure to bracket notation.

```
Bracket2 = rnaconvert(Matrix2)  
  
Bracket2 =  
  
((((.....))))
```

See Also

Bioinformatics Toolbox functions: rnafold, rnaplot

rnafold

Purpose Predict minimum free-energy secondary structure of RNA sequence

Syntax

```
rnafold(Seq)
RNAbracket = rnafold(Seq)
[RNAbracket, Energy] = rnafold(Seq)
[RNAbracket, Energy, RNAmatrix] = rnafold(Seq)
... = rnafold(Seq, ...'MinLoopSize', MinLoopSizeValue, ...)
... = rnafold(Seq, ...'NoGU', NoGUValue, ...)
... = rnafold(Seq, ...'Progress', ProgressValue, ...)
```

Arguments

<i>Seq</i>	Either of the following: <ul style="list-style-type: none">• String specifying an RNA sequence.• MATLAB structure containing a <code>Sequence</code> field that specifies an RNA sequence.
<i>MinLoopSizeValue</i>	Integer specifying the minimum size of the loops (in bases) to be considered when computing the free energy. Default is 3.
<i>NoGUValue</i>	Controls whether GU or UG pairs are forbidden to form. Choices are <code>true</code> or <code>false</code> (default).
<i>ProgressValue</i>	Controls the display of a progress bar during the computation of the minimum free-energy secondary structure. Choices are <code>true</code> or <code>false</code> (default).

Return Values

<i>RNAbracket</i>	String of dots and brackets indicating the bracket notation for the minimum-free energy secondary structure of an RNA sequence. In the bracket notation, each dot represents an unpaired base, while a pair of equally nested, opening and closing brackets represents a base pair.
<i>Energy</i>	Value specifying the energy (in kcal/mol) of the minimum free-energy secondary structure of an RNA sequence.
<i>RNAmatrix</i>	Connectivity matrix representing the minimum free-energy secondary structure of an RNA sequence. A binary, upper-triangular matrix where $RNAmatrix(i, j) = 1$ if and only if the i th residue in the RNA sequence <i>Seq</i> is paired with the j th residue of <i>Seq</i> .

Description

`rnafold(Seq)` predicts and displays the secondary structure (in bracket notation) associated with the minimum free energy for the RNA sequence, *Seq*, using the thermodynamic nearest-neighbor approach.

Note For long sequences, this prediction can be time consuming. For example, a 600-nucleotide sequence can take several minutes, and sequences greater than 1000 nucleotides can take over 1 hour, depending on your system.

RNAbracket = `rnafold(Seq)` predicts and returns the secondary structure associated with the minimum free energy for the RNA sequence, *Seq*, using the thermodynamic nearest-neighbor approach. The returned structure, *RNAbracket*, is in bracket notation, that is a vector of dots and brackets, where each dot represents an unpaired

base, while a pair of equally nested, opening and closing brackets represents a base pair.

`[RNAbracket, Energy] = rnafold(Seq)` also returns *Energy*, the energy value (in kcal/mol) of the minimum free-energy secondary structure of the RNA sequence.

`[RNAbracket, Energy, RNAmatrix] = rnafold(Seq)` also returns *RNAmatrix*, a connectivity matrix representing the secondary structure associated with the minimum free energy. *RNAmatrix* is an upper triangular matrix where *RNAmatrix*(i, j) = 1 if and only if the *i*th residue in the RNA sequence *Seq* is paired with the *j*th residue of *Seq*.

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

`... = rnafold(Seq, ...'MinLoopSize', MinLoopSizeValue, ...)` specifies the minimum size of the loops (in bases) to be considered when computing the free energy. Default is 3.

`... = rnafold(Seq, ...'NoGU', NoGUValue, ...)` controls whether GU or UG pairs are forbidden to form. Choices are `true` or `false` (default).

`... = rnafold(Seq, ...'Progress', ProgressValue, ...)` controls the display of a progress bar during the computation of the minimum free-energy secondary structure. Choices are `true` or `false` (default).

Examples

Determine the minimum free-energy secondary structure (in both bracket and matrix notation) and the energy value of the following RNA sequence:

```
seq = 'ACCCCUCCUCCUUGGAUCAAGGGGCUCAA';  
[bracket, energy, matrix] = rnafold(seq);  
bracket
```

bracket =

..((((((...((.....))...)))))).....

References

- [1] Wuchty, S., Fontana, W., Hofacker, I., and Schuster, P. (1999). Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers* 49, 145–165.
- [2] Matthews, D., Sabina, J., Zuker, M., and Turner, D. (1999). Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *J. Mol. Biol.* 288, 911–940.

See Also

Bioinformatics Toolbox functions: `rnaconvert`, `rnaplot`

rnaplot

Purpose Draw secondary structure of RNA sequence

Syntax

```
rnaplot(RNA2ndStruct)  
ha = rnaplot(RNA2ndStruct)  
[ha, H] = rnaplot(RNA2ndStruct)  
rnaplot(RNA2ndStruct, ... 'Sequence', SequenceValue, ...)  
rnaplot(RNA2ndStruct, ... 'Format', FormatValue, ...)  
rnaplot(RNA2ndStruct, ... 'Selection', SelectionValue, ...)  
rnaplot(RNA2ndStruct, ... 'ColorBy', ColorByValue, ...)
```

Arguments

<i>RNA2ndStruct</i>	Secondary structure of an RNA sequence represented by either:
---------------------	---

- String specifying bracket notation
- Connectivity matrix

Tip Use the `rnafold` function to create *RNA2ndStruct*.

<i>SequenceValue</i>	Sequence of the RNA secondary structure being plotted, specified by either of the following:
----------------------	--

- String of characters
- Structure containing a `Sequence` field that contains an RNA sequence

This information is used in the data tip displayed by clicking a base in the plot of the RNA secondary structure *RNA2ndStruct*. This information is required if you specify the 'Diagram' format or if you specify to highlight any of the following paired selections: 'AU', 'UA', 'GC', 'CG', 'GU' or 'UG'.

FormatValue String specifying the format of the plot. Choices are:

- 'Circle' (default)
- 'Diagram'
- 'Dotdiagram'
- 'Graph'
- 'Mountain'
- 'Tree'

Note If you specify 'Diagram', you must also use the 'Sequence' property to provide the RNA sequence.

SelectionValue Either of the following:

- Numeric array specifying the indices of residues to highlight in the plot.
- String specifying the subset of residues to highlight in the plot. Choices are:
 - 'Paired'
 - 'Unpaired'
 - 'AU' or 'UA'
 - 'GC' or 'CG'
 - 'GU' or 'UG'

Note If you specify 'AU', 'UA', 'GC', 'CG', 'GU', or 'UG', you must also use the 'Sequence' property to provide the RNA sequence.

ColorByValue String specifying a color scheme for the plot. Choices are:

- 'State' (default) — Color by pair state: paired bases and unpaired bases.
- 'Residue' — Color by residue type (A, C, G, and U).
- 'Pair' — Color by pair type (AU/UA, GC/CG, and GU/UG).

Note If you specify 'residue' or 'pair', you must also use the 'Sequence' property to provide the RNA sequence.

Note Because internal nodes of a tree correspond to paired residues, you cannot specify 'residue' if you specify 'Tree' for the 'Format' property.

Return Values

<i>ha</i>	Handle to the figure axis.
<i>H</i>	A structure of handles containing a subset of the following fields, based on what you specify for the 'Selection' and 'ColorBy' properties: <ul style="list-style-type: none">• Paired• Unpaired• A• C• G• U• AU• GC• GU• Selected

Description

`rnaplot(RNA2ndStruct)` draws the RNA secondary structure specified by *RNA2ndStruct*, the secondary structure of an RNA sequence represented by a string specifying bracket notation or a connectivity matrix.

`ha = rnaplot(RNA2ndStruct)` returns *ha*, a handle to the figure axis.

`[ha, H] = rnaplot(RNA2ndStruct)` also returns *H*, a structure of handles, which you can use to graph elements in a MATLAB Figure window.

Tip Use the handles returned in *H* to change properties of the graph elements, such as color, marker size, and marker type.

H contains a subset of the following fields, based on what you specify for the 'Selection' and 'ColorBy' properties.

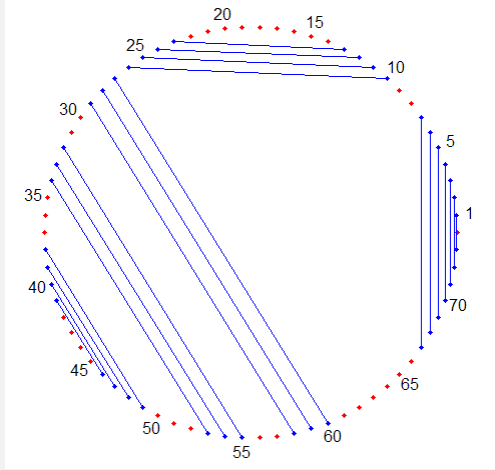
Field	Description
Paired	Handles to all paired residues
Unpaired	Handles to all unpaired residues
A	Handles to all A residues
C	Handles to all C residues
G	Handles to all G residues
U	Handles to all U residues
AU	Handles to all AU or UA pairs
GC	Handles to all GC or CG pairs
GU	Handles to all GU or UG pairs
Selected	Handles to all selected residues

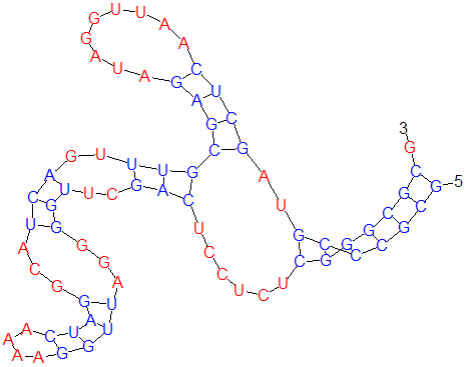
`rnaplot(RNA2ndStruct, ... 'PropertyName', PropertyValue, ...)` calls `rnaplot` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

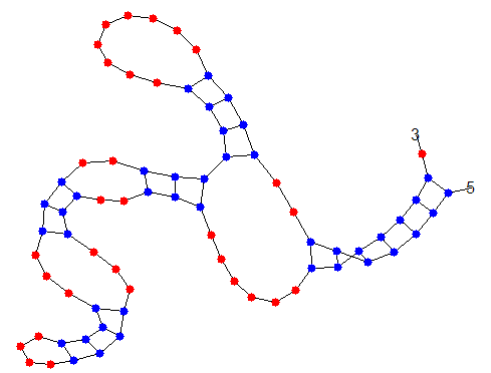
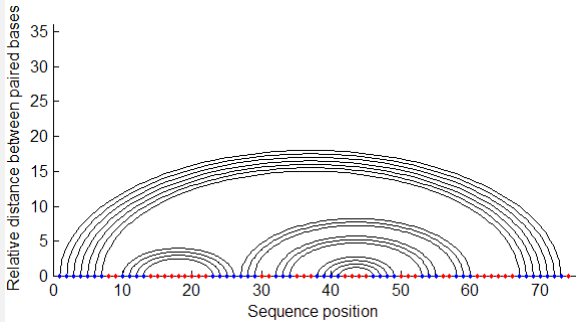
`rnaplot(RNA2ndStruct, ... 'Sequence', SequenceValue, ...)` draws the RNA secondary structure specified by *RNA2ndStruct*, and annotates it with the sequence positions supplied by *SequenceValue*, the RNA sequence specified by a string of characters or a structure containing a *Sequence* field.

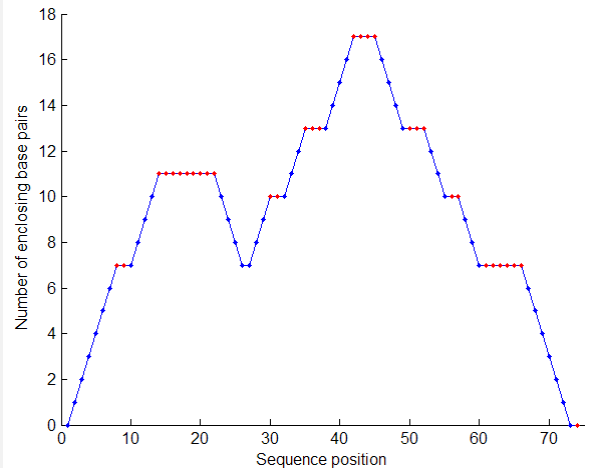
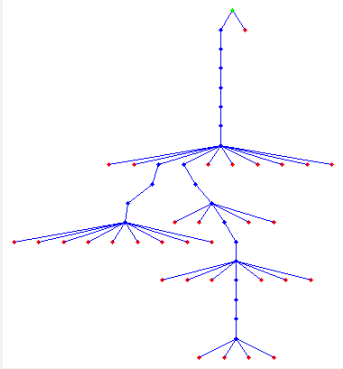
`rnaplot(RNA2ndStruct, ... 'Format', FormatValue, ...)` draws the RNA secondary structure specified by *RNA2ndStruct*, using the format specified by *FormatValue*.

FormatValue is a string specifying the format of the plot. Choices are as follows.

Format	Description
'Circle' (default)	<p data-bbox="716 317 1313 409">Each base is represented by a dot on the circumference of a circle of arbitrary size. Lines connect bases that pair with each other.</p> 

Format	Description
'Diagram'	<p data-bbox="669 319 1248 475">Two-dimensional representation of the RNA secondary structure. Each base is represented and identified by a letter. The backbone and hydrogen bonds between base pairs are represented by lines.</p>  <p data-bbox="669 951 1277 1045">Note If you specify 'Diagram', you must also use the 'Sequence' property to provide the RNA sequence.</p>

Format	Description
'Dotdiagram'	<p data-bbox="716 317 1297 473">Two-dimensional representation of the RNA secondary structure. Each base is represented and identified by a dot. The backbone and hydrogen bonds between base pairs are represented by lines.</p> 
'Graph'	<p data-bbox="716 907 1329 1064">Bases are displayed in their sequence position along the abscissa (x-axis) of a graph. Semi-elliptical lines connect bases that pair with each other. The height of the lines is proportional to the distance between paired bases.</p> 

Format	Description
'Mountain'	<p>Each base is represented by a dot in a two-dimensional plot, where the base position is in the abscissa (x-axis) and the number of base pairs enclosing a given base is in the ordinate (y-axis).</p> 
'Tree'	<p>Each base is represented by a node in a tree graph. Leaf nodes indicate unpaired bases, while each internal node indicates a base pair. The tree root is a fictitious node, not associated with any base in the secondary structure.</p> 
<p>Note To create a tree plot, you must have accepted a Graphviz software license (free). If you have not, you will be prompted to do so.</p>	

`rnaplot(RNA2ndStruct, ...'Selection', SelectionValue, ...)` draws the RNA secondary structure specified by *RNA2ndStruct*, highlighting a subset of residues specified by *SelectionValue*. *SelectionValue* can be either:

- Numeric array specifying the indices of residues to highlight in the plot.
- String specifying the subset of residues to highlight in the plot. Choices are:
 - 'Paired'
 - 'Unpaired'
 - 'AU' or 'UA'
 - 'GC' or 'CG'
 - 'GU' or 'UG'

Note If you specify 'AU', 'UA', 'GC', 'CG', 'GU', or 'UG', you must also use the 'Sequence' property to provide the RNA sequence.

`rnaplot(RNA2ndStruct, ...'ColorBy', ColorByValue, ...)` draws the RNA secondary structure specified by *RNA2ndStruct*, using a color scheme specified by *ColorByValue*, a string indicating a color scheme. Choices are:

- 'State' (default) — Color by pair state: paired bases and unpaired bases.
- 'Residue' — Color by residue type (A, C, G, and U).
- 'Pair' — Color by pair type (AU/UA, GC/CG, and GU/UG).

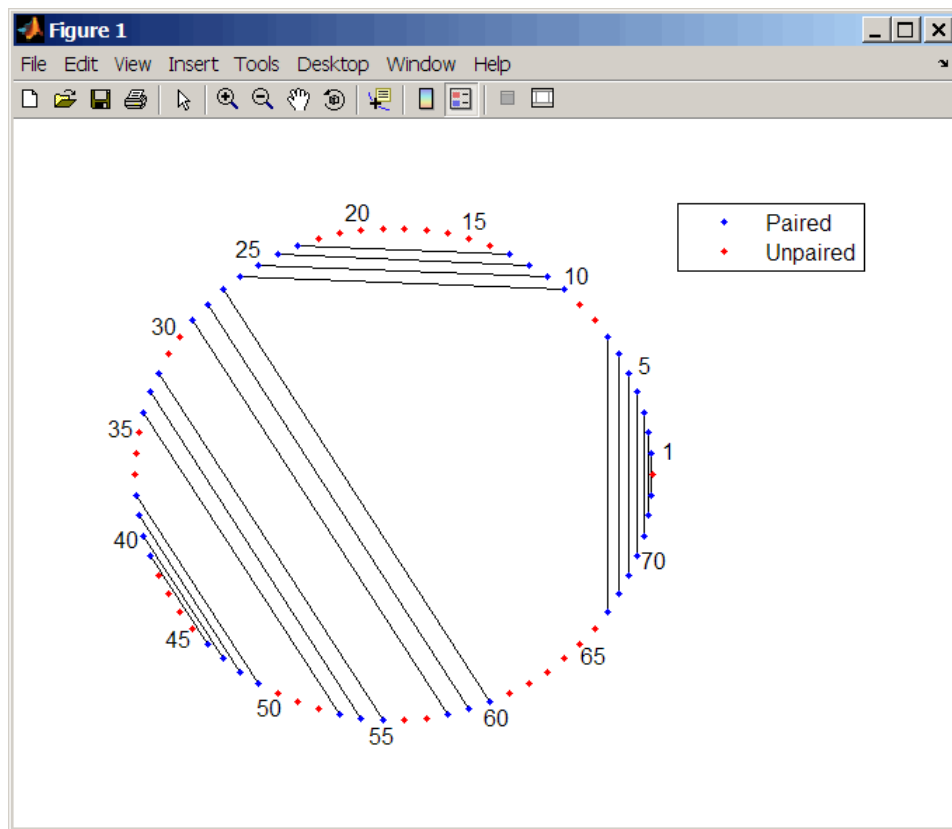
Note If you specify 'Residue' or 'Pair', you must also use the 'Sequence' property to provide the RNA sequence.

Note Because internal nodes of a tree correspond to paired residues, you cannot specify 'Residue' if you specify 'Tree' for the 'Format' property.

Examples

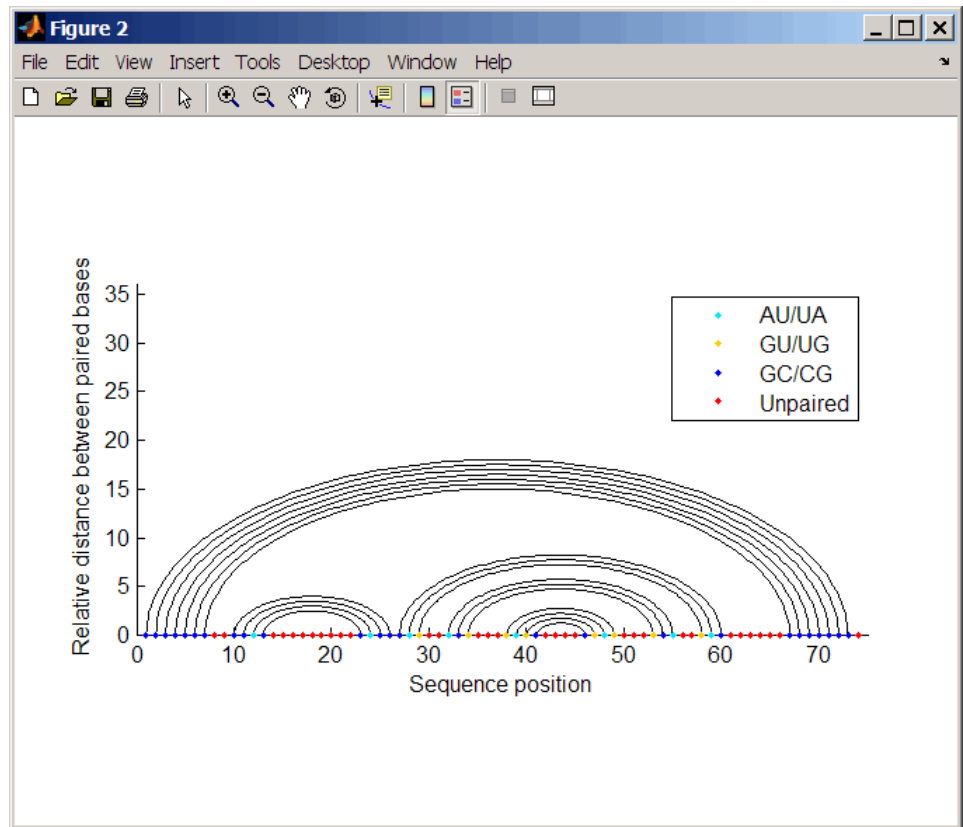
- 1 Determine the minimum free-energy secondary structure of an RNA sequence and plot it in circle format:

```
seq = 'GCGCCCGUAGCUCAAUUGGAUAGAGCGUUUGACUACGGAUCAAAGGUUAGGGGUUCGACUCCUCUCGGGCGCG';  
ss = rnafold(seq);  
rnaplot(ss)
```



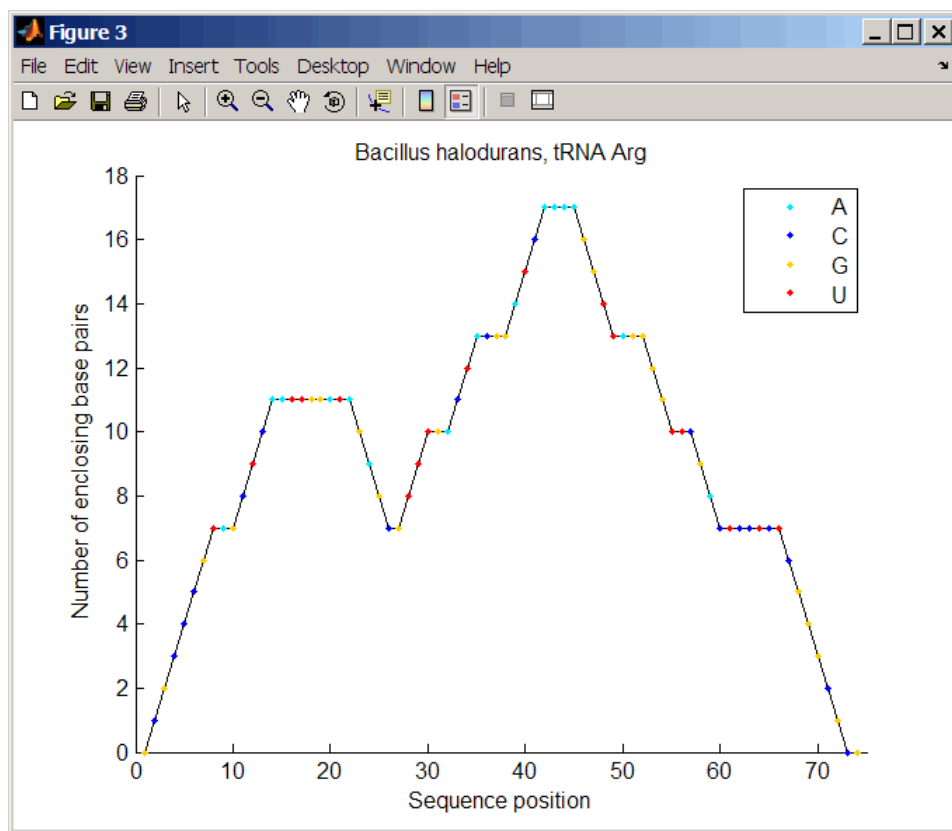
- 2** Plot the RNA sequence secondary structure in graph format and color it by pair type.

```
rnaplot(ss, 'sequence', seq, 'format', 'graph', 'colorby', 'pair')
```



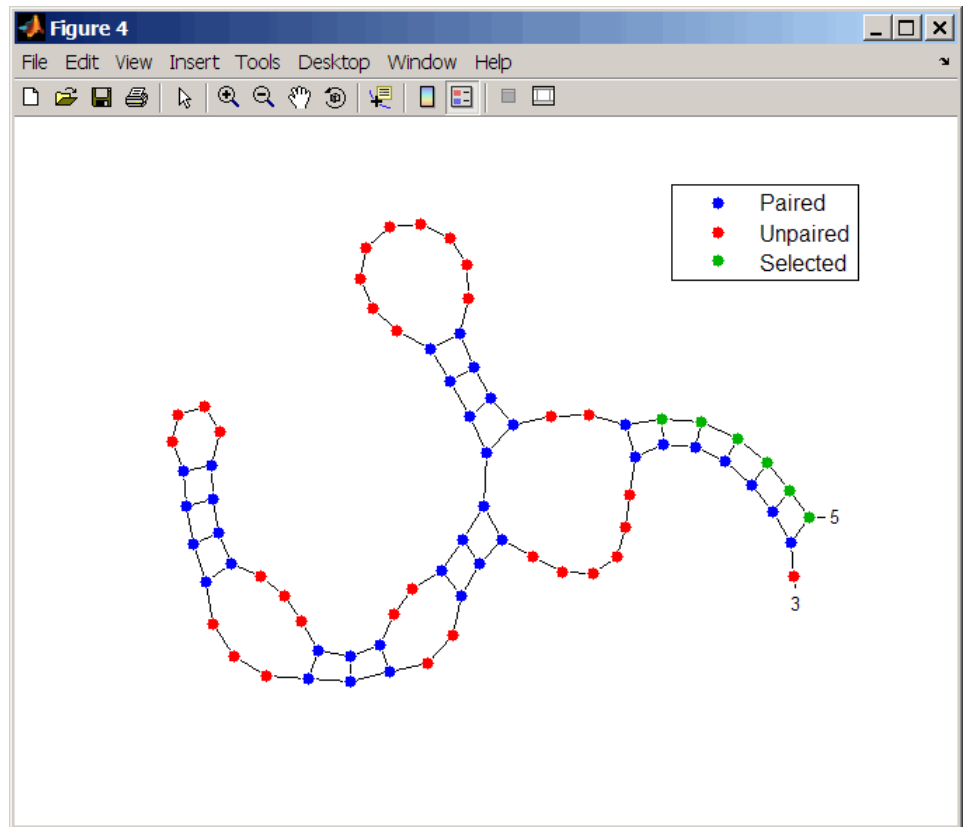
- 3** Plot the RNA sequence secondary structure in mountain format and color it by residue type. Use the handle to add a title to the plot.

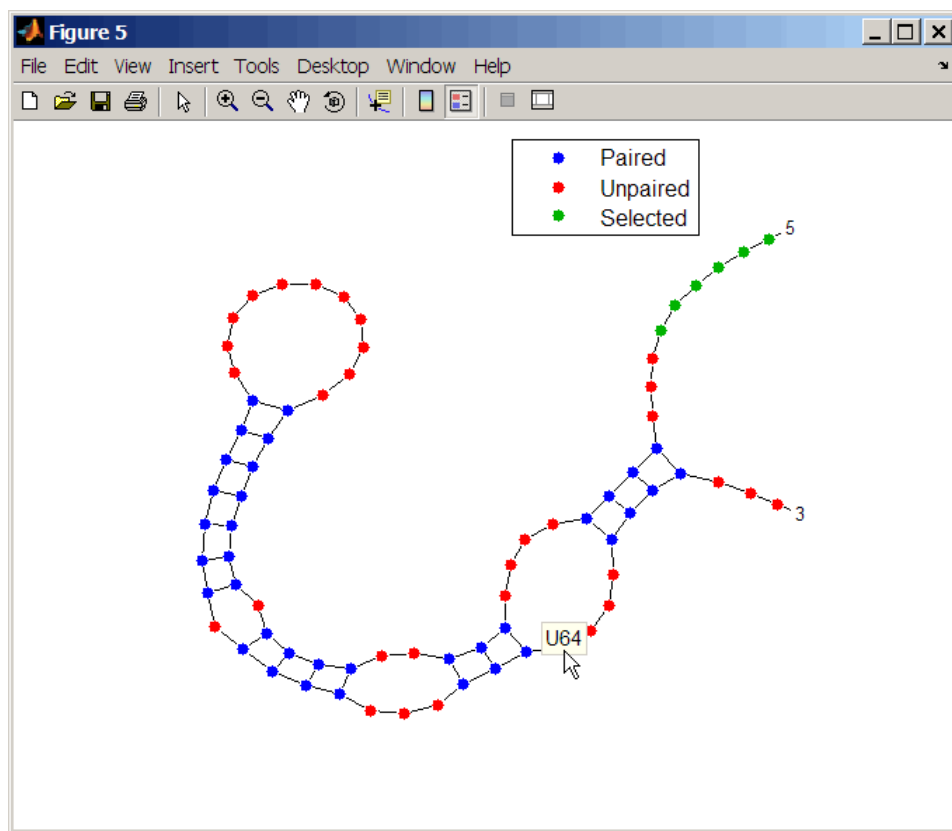
```
ha = rnaplot(ss, 'sequence', seq, 'format', 'mountain',...  
            'colorby', 'residue')  
title(ha, 'Bacillus halodurans, tRNA Arg')
```

- 4 Mutate the first six positions in the sequence and observe the effect the change has on the secondary structure by highlighting the first six residues.

```
seqMut = seq;
seqMut(1:6) = 'AAAAAA';
ssMut = rnafold(seqMut);
rnaplot(ss, 'sequence', seq, 'format', 'dotdiagram', 'selection', 1:6);
rnaplot(ssMut, 'sequence', seqMut, 'format', 'dotdiagram', 'selection', 1:6);
```





Tip If necessary, click-drag the legend to prevent it from covering the plot. Click a base in the plot to display a data tip with information on that base.

See Also

Bioinformatics Toolbox functions: `rnaconvert`, `rnafold`

samplealign

Purpose

Align two data sets containing sequential observations by introducing gaps

Syntax

```
[I, J] = samplealign(X, Y)
[I, J] = samplealign(X, Y, ...'Band', BandValue, ...)
[I, J] = samplealign(X, Y, ...'Width', WidthValue, ...)
[I, J] = samplealign(X, Y, ...'Gap', GapValue, ...)
[I, J] = samplealign(X, Y, ...'Quantile',
QuantileValue, ...)
[I, J] = samplealign(X, Y, ...'Distance',
DistanceValue, ...)
[I, J] = samplealign(X, Y, ...'Weights', WeightsValue, ...)
[I, J] = samplealign(X, Y, ...'ShowConstraints',
ShowConstraintsValue, ...)
[I, J] = samplealign(X, Y, ...'ShowNetwork',
ShowNetworkValue, ...)
[I, J] = samplealign(X, Y, ...'ShowAlignment',
ShowAlignmentValue, ...)
```

Arguments*X, Y*

Matrices of data where rows correspond to observations or samples, and columns correspond to features or dimensions. *X* and *Y* can have a different number of rows, but they must have the same number of columns. The first column is the reference dimension and must contain unique values in ascending order. The reference dimension could contain sample indices of the observations or a measurable value, such as time.

BandValue

Either of the following:

- Scalar.
- Function specified using $@(z)$, where z is the mid-point between a given observation in one data set and a given observation in the other data set.

BandValue specifies a maximum allowable distance between observations (along the reference dimension only) in the two data sets, thus limiting the number of potential matches between observations in two data sets. If S is the value in the reference dimension for a given observation (row) in one data set, then that observation is matched only with observations in the other data set whose values in the reference dimension fall within $S \pm \textit{BandValue}$. Then, only these potential matches are passed to the algorithm for further scoring. Default *BandValue* is Inf.

WidthValue

Either of the following:

- Two-element vector, $[U, V]$
- Scalar that is used for both U and V

WidthValue limits the number of potential matches between observations in two data sets; that is, each observation in X is scored to the closest U observations in Y , and each observation in Y is scored to the closest V observations in X . Then, only these potential matches are passed to the algorithm for further scoring. Closeness is measured using only the first column (reference dimension) in each data set. Default is `Inf` if 'Band' is specified; otherwise default is 10.

GapValue

Any of the following:

- Cell array, $\{G, H\}$, where G is either a scalar or a function handle specified using $@(X)$, and H is either a scalar or a function handle specified using $@(Y)$. The functions $@(X)$ and $@(Y)$ must calculate the penalty for each observation (row) when it is matched to a gap in the other data set. The functions $@(X)$ and $@(Y)$ must return a column vector with the same number of rows as X or Y , containing the gap penalty for each observation (row).
- Single function handle specified using $@(Z)$, which is used for both G and H . The function $@(Z)$ must calculate the penalty for each observation (row) in both X and Y when it is matched to a gap in the other data set. The function $@(Z)$ must take as arguments X and Y . The function $@(Z)$ must return a column vector with the same number of rows as X or Y , containing the gap penalty for each observation (row).
- Scalar that is used for both G and H .

GapValue specifies the position-dependent terms for assigning gap penalties. The calculated value, GPX , is the gap penalty for matching observations from the first data set X to gaps inserted in the second data set Y , and is the product of two terms: $GPX = G * QMS$. The term G takes its value as a function of the observations in X . Similarly, GPY is the gap penalty for matching observations from Y to gaps inserted in X , and is the product of two terms: $GPY = H * QMS$. The term H takes its value as a function of the observations in Y . By default, the term QMS is the 0.75 quantile of the score for the pairs of observations that are potential matches (that is, pairs that comply with the 'Band' and 'Width' constraints). Default *GapValue* is 1.

DistanceValue

Function handle specified using $@(R,S)$.
The function $@(R,S)$ must:

- Calculate the distance between pairs of observations that are potential matches.
- Take as arguments, R and S , matrices that have the same number of rows and columns, and whose paired rows represent all potential matches of observations in X and Y respectively.
- Return a column vector of positive values with the same number of elements as rows in R and S .

Default is the Euclidean distance between the pairs.

Caution All columns in X and Y , including the reference dimension, are considered when calculating distances. If you do not want to include the reference dimension in the distance calculations, use the 'Weight' property to exclude it.

WeightsValue

Either of the following:

- Logical row vector with the same number of elements as columns in *X* and *Y*, that specifies columns in *X* and *Y*.
- Numeric row vector with the same number of elements as columns in *X* and *Y*, that specifies the relative weights of the columns (features).

This property controls the inclusion/exclusion of columns (features) or the emphasis of columns (features) when calculating the distance score between observations that are potential matches, that is, when using the 'Distance' property. Default is a logical row vector with all elements set to true.

Tip Using a numeric row vector for *WeightsValue* and setting some values to 0 can simplify the distance calculation when the data sets have many columns (features).

Note The weight values are not considered when using the 'Band', 'Width', or 'Gap' property.

samplealign

- ShowConstraintsValue* Controls the display of the search space constrained by the specified 'Band' and 'Width' input parameters, thereby giving an indication of the memory required to run the algorithm with the specific 'Band' and 'Width' parameters on your data sets. Choices are true or false (default).
- ShowNetworkValue* Controls the display of the dynamic programming network, the match scores, the gap penalties, and the winning path. Choices are true or false (default).
- ShowAlignmentValue* Controls the display of the first and second columns of the X and Y data sets in the abscissa and the ordinate respectively, of a two-dimensional plot. Choices are true, false (default), or an integer specifying a column of the X and Y data sets to plot as the ordinate.

Return Values

- I Column vector containing indices of rows (observations) in X that match to a row (observation) in Y . Missing indices indicate that row (observation) is matched to a gap.
- J Column vector containing indices of rows (observations) in Y that match to a row (observation) in X . Missing indices indicate that row (observation) is matched to a gap.

Description

$[I, J] = \text{samplealign}(X, Y)$ aligns the observations in two matrices of data, X and Y , by introducing gaps. X and Y are matrices of data where rows correspond to observations or samples, and columns correspond to features or dimensions. X and Y can have different number of rows, but must have the same number of columns. The first column is the

reference dimension and must contain unique values in ascending order. The reference dimension could contain sample indices of the observations or a measurable value, such as time. The `samplealign` function uses a dynamic programming algorithm to minimize the sum of positive scores resulting from pairs of observations that are potential matches and the penalties resulting from the insertion of gaps. Return values I and J are column vectors containing indices that indicate the matches for each row (observation) in X and Y respectively.

Tip If you do not specify return values, `samplealign` does not run the dynamic programming algorithm. Running `samplealign` without return values, but setting the 'ShowConstraints', 'ShowNetwork', or 'ShowAlignment' property to true, lets you explore the constrained search space, the dynamic programming network, or the aligned observations, without running into potential memory problems.

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

`[I, J] = samplealign(X, Y, ...'Band', BandValue, ...)` specifies a maximum allowable distance between observations (along the reference dimension only) in the two data sets, thus limiting the number of potential matches between observations in the two data sets. If S is the value in the reference dimension for a given observation (row) in one data set, then that observation is matched only with observations in the other data set whose values in the reference dimension fall within $S \pm \text{BandValue}$. Then, only these potential matches are passed to the algorithm for further scoring. *BandValue* can be a scalar or a function specified using `@(z)`, where z is the mid-point between a given observation in one data set and a given observation in the other data set. Default *BandValue* is `Inf`.

samplealign

This constraint reduces the time and memory complexity of the algorithm from $O(MN)$ to $O(\sqrt{MN} * K)$, where M and N are the number of observations in X and Y respectively, and K is a small constant such that $K \ll M$ and $K \ll N$. Adjust *BandValue* to the maximum expected shift between the reference dimensions in the two data sets, that is, between $X(:,1)$ and $Y(:,1)$.

`[I, J] = samplealign(X, Y, ...'Width', WidthValue, ...)` limits the number of potential matches between observations in two data sets; that is, each observation in X is scored to the closest U observations in Y , and each observation in Y is scored to the closest V observations in X . Then, only these potential matches are passed to the algorithm for further scoring. *WidthValue* is either a two-element vector, $[U, V]$ or a scalar that is used for both U and V . Closeness is measured using only the first column (reference dimension) in each data set. Default is `Inf` if 'Band' is specified; otherwise default is 10.

This constraint reduces the time and memory complexity of the algorithm from $O(MN)$ to $O(\sqrt{MN} * \sqrt{UV})$, where M and N are the number of observations in X and Y respectively, and U and V are small such that $U \ll M$ and $V \ll N$.

Note If you specify both 'Band' and 'Width', only pairs of observations that meet both constraints are considered potential matches and passed to the algorithm for scoring.

Tip Specify 'Width' when you do not have a good estimate for the 'Band' property. To get an indication of the memory required to run the algorithm with specific 'Band' and 'Width' parameters on your data sets, run `samplealign`, but do not specify return values and set 'ShowConstraints' to true.

`[I, J] = samplealign(X, Y, ...'Gap', GapValue, ...)` specifies the position-dependent terms for assigning gap penalties.

GapValue is any of the following:

- Cell array, $\{G, H\}$, where G is either a scalar or a function handle specified using $@(X)$, and H is either a scalar or a function handle specified using $@(Y)$. The functions $@(X)$ and $@(Y)$ must calculate the penalty for each observation (row) when it is matched to a gap in the other data set. The functions $@(X)$ and $@(Y)$ must return a column vector with the same number of rows as X or Y , containing the gap penalty for each observation (row).
- Single function handle specified using $@(Z)$, that is used for both G and H . The function $@(Z)$ must calculate the penalty for each observation (row) in both X and Y when it is matched to a gap in the other data set. The function $@(Z)$ must take as arguments X and Y . The function $@(Z)$ must return a column vector with the same number of rows as X or Y , containing the gap penalty for each observation (row).
- Scalar that is used for both G and H .

The calculated value, GPX , is the gap penalty for matching observations from the first data set X to gaps inserted in the second data set Y , and is the product of two terms: $GPX = G * QMS$. The term G takes its value as a function of the observations in X . Similarly, GPY is the gap penalty for matching observations from Y to gaps inserted in X , and is the product of two terms: $GPY = H * QMS$. The term H takes its value as a function of the observations in Y . By default, the term QMS is the 0.75 quantile of the score for the pairs of observations that are potential matches (that is, pairs that comply with the 'Band' and 'Width' constraints).

If G and H are positive scalars, then GPX and GPY are independent of the observation where the gap is being inserted.

Default *GapValue* is 1, that is, both G and H are 1, which indicates that the default penalty for gap insertions in both sequences is equivalent to the quantile (set by the 'Quantile' property, default = 0.75) of the score for the pairs of observations that are potential matches.

samplealign

Note *GapValue* defaults to a relatively safe value. However, the success of the algorithm depends on the fine tuning of the gap penalties, which is application dependent. When the gap penalties are large relative to the score of the correct matches, `samplealign` returns alignments with fewer gaps, but with more incorrectly aligned regions. When the gap penalties are smaller, the output alignment contains longer regions with gaps and fewer matched observations. Set 'ShowNetwork' to true to compare the gap penalties to the score of matched observations in different regions of the alignment.

`[I, J] = samplealign(X, Y, ...'Quantile', QuantileValue, ...)` specifies the quantile value used to calculate the term *QMS*, which is used by the 'Gap' property to calculate gap penalties. *QuantileValue* is a scalar between 0 and 1. Default is 0.75.

Tip Set *QuantileValue* to an empty array ([]) to make the gap penalties independent of *QMS*, that is, *GPX* and *GPY* are functions of only the *G* and *H* input parameters respectively.

`[I, J] = samplealign(X, Y, ...'Distance', DistanceValue, ...)` specifies a function to calculate the distance between pairs of observations that are potential matches. *DistanceValue* is a function handle specified using `@(R,S)`. The function `@(R,S)` must take as arguments, *R* and *S*, matrices that have the same number of rows and columns, and whose paired rows represent all potential matches of observations in *X* and *Y* respectively. The function `@(R,S)` must return a column vector of positive values with the same number of elements as rows in *R* and *S*. Default is the Euclidean distance between the pairs.

Caution

All columns in X and Y , including the reference dimension, are considered when calculating distances. If you do not want to include the reference dimension in the distance calculations, use the 'Weight' property to exclude it.

`[I, J] = samplealign(X, Y, ...'Weights', WeightsValue, ...)` controls the inclusion/exclusion of columns (features) or the emphasis of columns (features) when calculating the distance score between observations that are potential matches, that is when using the 'Distance' property. *WeightsValue* can be a logical row vector that specifies columns in X and Y . *WeightsValue* can also be a numeric row vector with the same number of elements as columns in X and Y , that specifies the relative weights of the columns (features). Default is a logical row vector with all elements set to true.

Tip Using a numeric row vector for *WeightsValue* and setting some values to 0 can simplify the distance calculation when the data sets have many columns (features).

Note The weight values are not considered when computing the constrained alignment space, that is when using the 'Band' or 'Width' properties, or when calculating the gap penalties, that is when using the 'Gap' property.

`[I, J] = samplealign(X, Y, ...'ShowConstraints', ShowConstraintsValue, ...)` controls the display of the search space constrained by the input parameters 'Band' and 'Width', giving an indication of the memory required to run the algorithm with specific

'Band' and 'Width' on your data sets. Choices are true or false (default).

`[I, J] = samplealign(X, Y, ...'ShowNetwork', ShowNetworkValue, ...)` controls the display of the dynamic programming network, the match scores, the gap penalties, and the winning path. Choices are true or false (default).

`[I, J] = samplealign(X, Y, ...'ShowAlignment', ShowAlignmentValue, ...)` controls the display of the first and second columns of the X and Y data sets in the abscissa and the ordinate respectively, of a two-dimensional plot. Links between all the potential matches that meet the constraints are displayed, and the matches belonging to the output alignment are highlighted. Choices are true, false (default), or an integer specifying a column of the X and Y data sets to plot as the ordinate.

Examples

Warping a sine wave with a smooth function to more closely follow cyclical sunspot activity

- 1 Load `sunspot.dat`, a data file included with the MATLAB software, that contains the variable `sunspot`, which is a two-column matrix containing variations in sunspot activity over the last 300 years. The first column is the reference dimension (years), and the second column contains sunspot activity values. Sunspot activity is cyclical, reaching a maximum about every 11 years.

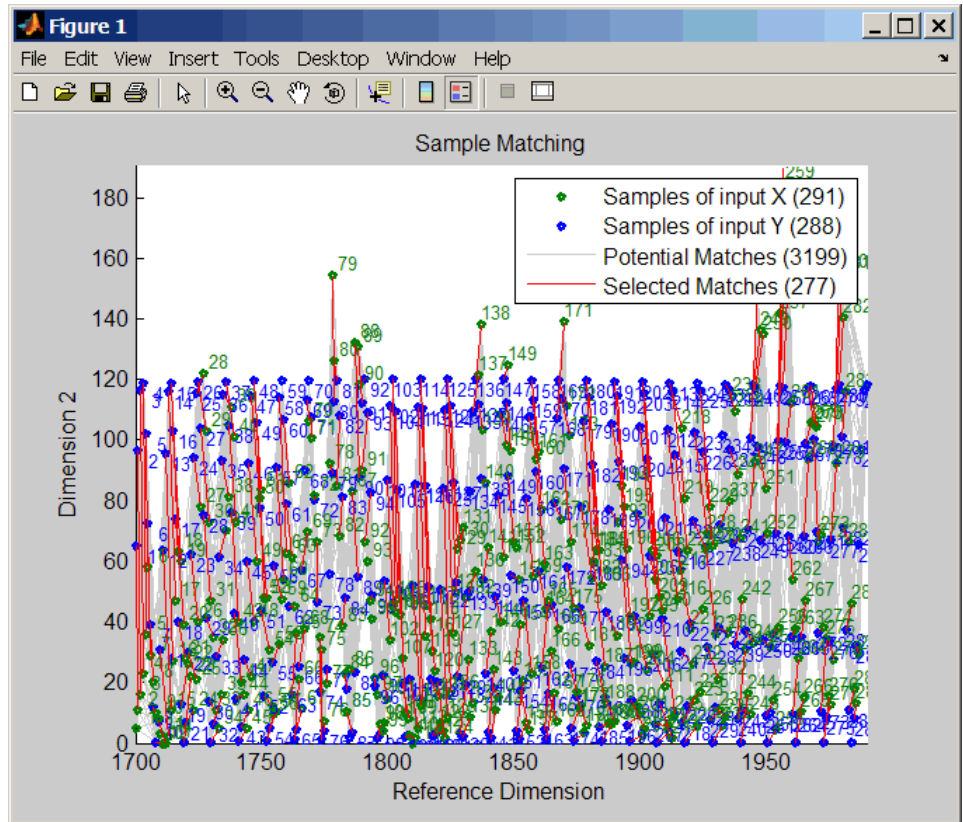
```
load sunspot.dat
```

- 2 Create a sine wave with a known period of sunspot activity.

```
years = (1700:1990)';  
T = 11.038;  
f = @(y) 60 + 60 * sin(y*(2*pi/T));
```

- 3 Align the observations between the sine wave and the sunspot activity by introducing gaps.


```
[i,j] = samplealign([years f(years)],sunspot,'weights',...
[0 1],'showalignment',true);
```



4 Estimate a smooth function to warp the sine wave.

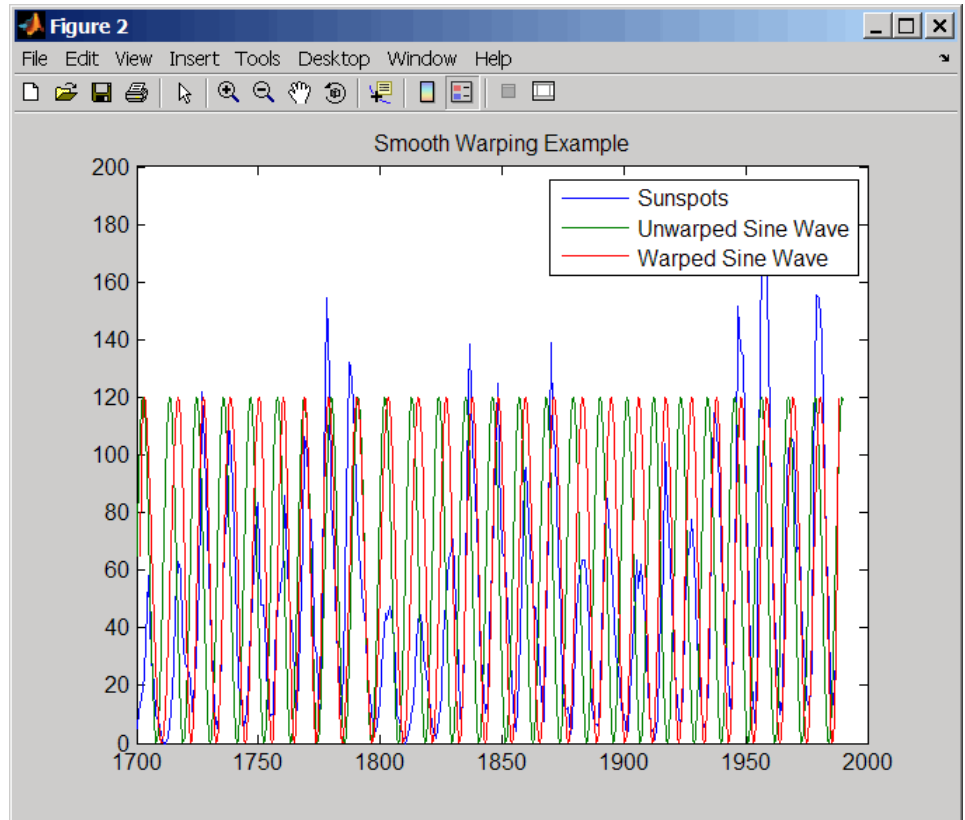
```
[p,s,mu] = polyfit(years(i),years(j),15);
wy = @(y) polyval(p,(y-mu(1))./mu(2));
```

5 Plot the sunspot cycles, unwarped sine wave, and warped sine wave.

```
years = (1700:1/12:1990)';
```

samplealign

```
figure
plot(sunspot(:,1),sunspot(:,2),years,f(years),wy(years),...
      f(years))
legend('Sunspots','Unwarped Sine Wave','Warped Sine Wave')
title('Smooth Warping Example')
```



Recovering a nonlinear warping between two signals containing noisy Gaussian peaks

- 1 Create two signals with noisy Gaussian peaks.

```

rand('twister',5489)
peakLoc = [30 60 90 130 150 200 230 300 380 430];
peakInt = [7 1 3 10 3 6 1 8 3 10];
time = 1:450;
comp = exp(-(bsxfun(@minus,time,peakLoc')./5).^2);
sig_1 = (peakInt + rand(1,10)) * comp + rand(1,450);
sig_2 = (peakInt + rand(1,10)) * comp + rand(1,450);

```

2 Define a nonlinear warping function.

```

wf = @(t) 1 + (t<=100).*0.01.*(t.^2) + (t>100).*...
(310+150*tanh(t./100-3));

```

3 Warp the second signal to distort it.

```

sig_2 = interp1(time,sig_2,wf(time),'pchip');

```

4 Align the observations between the two signals by introducing gaps.

```

[i,j] = samplealign([time;sig_1]',[time;sig_2]',...
'weights',[0,1],'band',35,'quantile',.5);

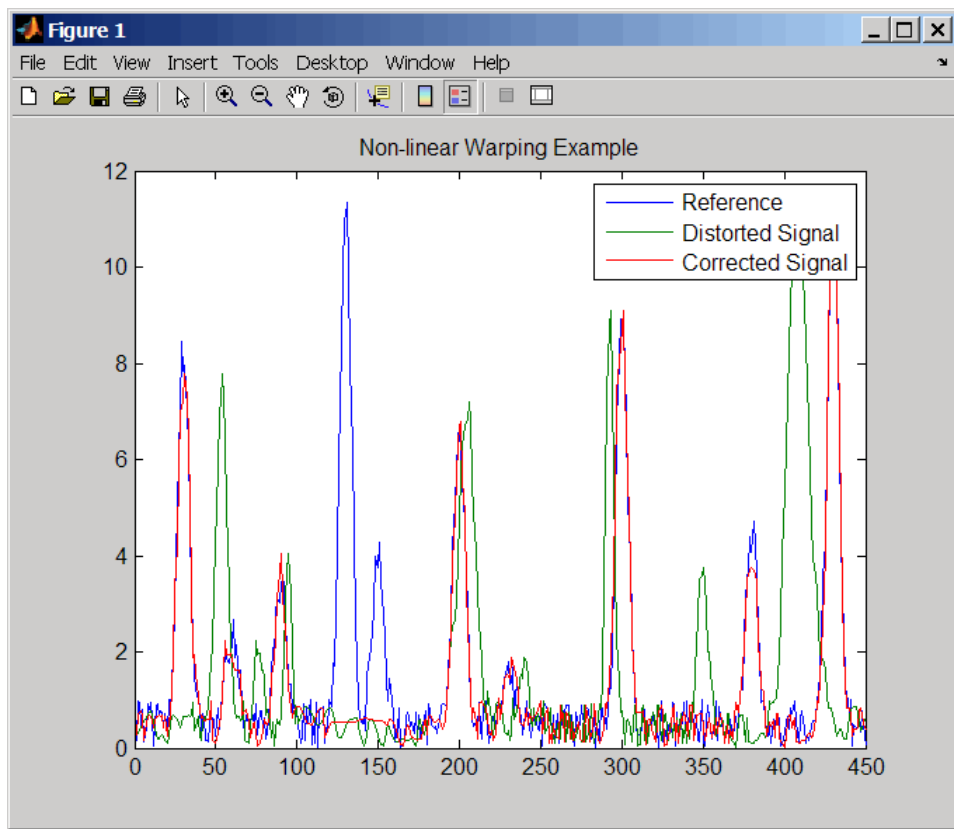
```

5 Plot the reference signal, distorted signal, and warped (corrected) signal.

```

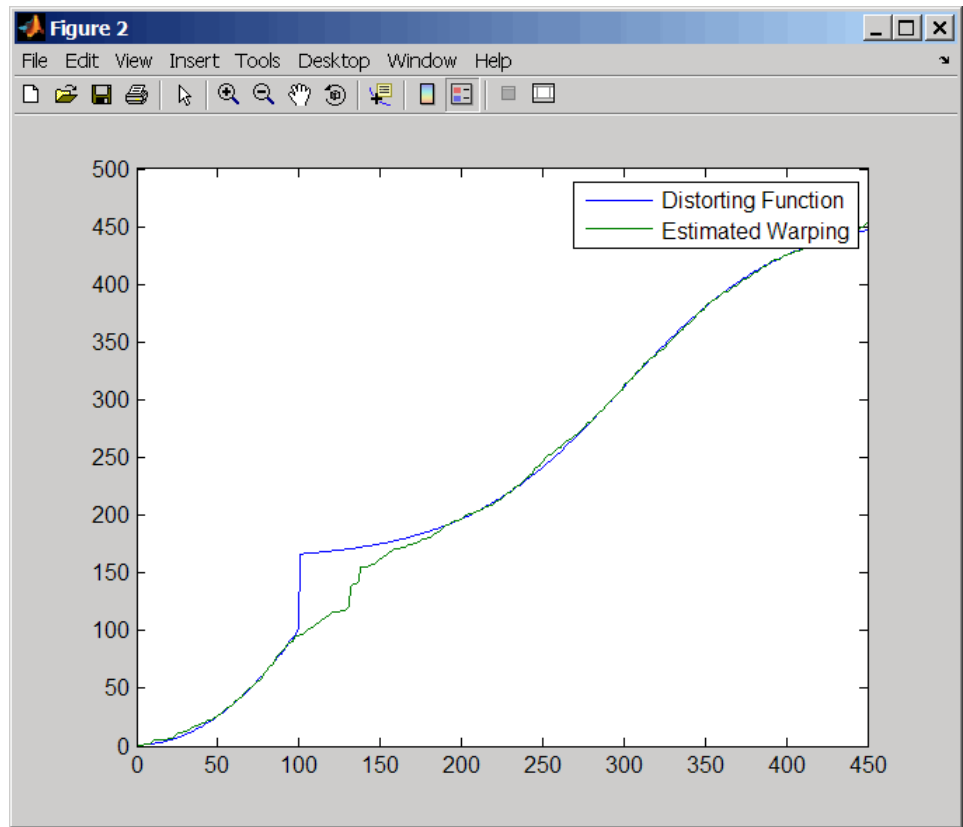
figure
sig_3 = interp1(time,sig_2,interp1(i,j,time,'pchip'),'pchip');
plot(time,sig_1,time,sig_2,time,sig_3)
legend('Reference','Distorted Signal','Corrected Signal')
title('Non-linear Warping Example')

```



6 Plot the real and the estimated warping functions.

```
figure
plot(time,wf(time),time,interp1(j,i,time,'pchip'))
legend('Distorting Function','Estimated Warping')
```



Note For examples of using function handles for the `Band`, `Gap`, and `Distance` properties, see the demo `Visualizing and Preprocessing Hyphenated Mass-Spectrometry Data Sets for Metabolite and Protein/Peptide Profiling`.

References

[1] Myers, C.S. and Rabiner, L.R. (1981). A comparative study of several dynamic time-warping algorithms for connected word recognition. *The Bell System Technical Journal* 60:7, 1389–1409.

samplealign

[2] Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. Acoustics, Speech and Signal Processing ASSP-26(1)*, 43–49.

See Also

Bioinformatics Toolbox functions: `msalign`, `msheatmap`, `mspalign`, `msppresample`, `msresample`

Purpose Read trace data from SCF file

Syntax

```

Sample = scfread('File')
[Sample, Probability] = scfread('File')
[Sample, Probability, Comments] = scfread('File')
[A, C, T, G] = scfread ('File')
[A, C, T, G, ProbA, ProbC, ProbG, ProbT] = scfread ('File')
[A, C, T, G, ProbA, ProbC, ProbG, ProbT, Comments, PkIndex,
Base] = scfread ('File')
    
```

Arguments

File SCF formatted file. Enter a file name or a path and file name.

Description scfread reads data from an SCF formatted file into MATLAB structures.

Sample = scfread('File') reads an SCF formatted file and returns the sample data in the structure *Sample*, which contains the following fields:

Field	Description
A	Column vector containing intensity of A fluorescence tag
C	Column vector containing intensity of C fluorescence tag
G	Column vector containing intensity of G fluorescence tag
T	Column vector containing intensity of T fluorescence tag

[*Sample*, *Probability*] = scfread('File') also returns the probability data in the structure *Probability*, which contains the following fields:

Field	Description
peak_index	Column vector containing the position in the SCF file for the start of the data for each peak
prob_A	Column vector containing the probability of each base in the sequence being an A
prob_C	Column vector containing the probability of each base in the sequence being a C
prob_G	Column vector containing the probability of each base in the sequence being a G
prob_T	Column vector containing the probability of each base in the sequence being a T
base	Column vector containing the called bases for the sequence

`[Sample, Probability, Comments] = scfread('File')` also returns the comment information from the SCF file in a character array `Comments`.

`[A, C, T, G] = scfread ('File')` returns the sample data for the four bases in separate variables.

`[A, C, T, G, ProbA, ProbC, ProbG, ProbT] = scfread ('File')` also returns the probabilities data for the four bases in separate variables.

`[A, C, T, G, ProbA, ProbC, ProbG, ProbT, Comments, PKIndex, Base] = scfread ('File')` also returns the peak indices and called bases in separate variables.

SCF files store data from DNA sequencing instruments. Each file includes sample data, sequence information, and the relative probabilities of each of the four bases. For more information on SCF files, see

http://www.mrc-lmb.cam.ac.uk/pubseq/manual/formats_unix_2.html

Examples

```
[sampleStruct, probStruct, Comments] = scfread('sample.scf')
sampleStruct =
```

```
A: [10827x1 double]
C: [10827x1 double]
G: [10827x1 double]
T: [10827x1 double]
```

```
probStruct =
```

```
peak_index: [742x1 double]
prob_A: [742x1 double]
prob_C: [742x1 double]
prob_G: [742x1 double]
prob_T: [742x1 double]
base: [742x1 char]
```

```
Comments =
```

```
SIGN=A=121,C=103,G=119,T=82
SPAC= 16.25
PRIM=0
MACH=Arkansas_SN312
DYEP=DT3700POP5{BD}v2.mob
NAME=HCIUP1D61207
LANE=6
GELN=
PROC=
RTRK=
CONV=phred version=0.990722.h
COMM=
SRCE=ABI 373A or 377
```

See Also

Bioinformatics Toolbox functions: `genbankread`, `traceplot`

seq2regex

Purpose	Convert sequence with ambiguous characters to regular expression	
Syntax	<pre>RegExp = seq2regex(Seq) RegExp = seq2regex(Seq, ...'Alphabet', AlphabetValue, ...) RegExp = seq2regex(Seq, ...'Ambiguous', AmbiguousValue, ...)</pre>	
Arguments	<i>Seq</i>	Either of the following: <ul style="list-style-type: none">• Character string of codes specifying an amino acid or nucleotide sequence.• Structure containing a <code>Sequence</code> field that contains an amino acid or nucleotide sequence, such as returned by <code>fastaread</code>, <code>getembl</code>, <code>getgenbank</code>, <code>getgenpept</code>, or <code>getpdb</code>.
	<i>AlphabetValue</i>	String specifying the sequence alphabet. Choices are: <ul style="list-style-type: none">• 'NT' (default) — Nucleotide• 'AA' — Amino acid
	<i>AmbiguousValue</i>	Controls whether ambiguous characters are included in <i>RegExp</i> , the regular expression return value. Choices are: <ul style="list-style-type: none">• <code>true</code> (default) — Include ambiguous characters in the return value• <code>false</code> — Return only unambiguous characters
Return Values	<i>RegExp</i>	Character string of codes specifying an amino acid or nucleotide sequence in regular expression format using IUB/IUPAC codes.

Description

RegExp = seq2regexp(*Seq*) converts ambiguous amino acid or nucleotide symbols in a sequence to a regular expression format using IUB/IUPAC codes.

RegExp = seq2regexp(*Seq*, ...'*PropertyName*', *PropertyValue*, ...) calls seq2regexp with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

RegExp = seq2regexp(*Seq*, ...'*Alphabet*', *AlphabetValue*, ...) specifies the sequence alphabet. *AlphabetValue* can be either 'NT' for nucleotide sequences or 'AA' for amino acid sequences. Default is 'NT'.

RegExp = seq2regexp(*Seq*, ...'*Ambiguous*', *AmbiguousValue*, ...) controls whether ambiguous characters are included in *RegExp*, the regular expression return value. Choices are true (default) or false. For example:

- If *Seq* = 'ACGTK', and *AmbiguousValue* is true, the MATLAB software returns ACGT[GTK] with the unambiguous characters G and T and the ambiguous character K.
- If *Seq* = 'ACGTK', and *AmbiguousValue* is false, the MATLAB software returns ACGT[GT] with only the unambiguous characters.

Nucleotide Conversions

Nucleotide Code	Nucleotide	Conversion
A	Adenosine	A
C	Cytosine	C
G	Guanine	G
T	Thymidine	T
U	Uridine	U

Nucleotide Conversions (Continued)

Nucleotide Code	Nucleotide	Conversion
R	Purine	[AG]
Y	Pyrimidine	[TC]
K	Keto	[GT]
M	Amino	[AC]
S	Strong interaction (3 H bonds)	[GC]
W	Weak interaction (2 H bonds)	[AT]
B	Not A	[CGT]
D	Not C	[AGT]
H	Not G	[ACT]
V	Not T or U	[ACG]
N	Any nucleotide	[ACGT]
-	Gap of indeterminate length	-
?	Unknown	?

Amino Acid Conversion

Amino Acid Code	Amino Acid	Conversion
B	Asparagine or Aspartic acid (Aspartate)	[DN]

Amino Acid Conversion (Continued)

Amino Acid Code	Amino Acid	Conversion
Z	Glutamine or Glutamic acid (Glutamate)	[EQ]
X	Any amino acid	[A R N D C Q E G H I L K M F P S T W Y V]

Example

- 1 Convert a nucleotide sequence into a regular expression.

```
seq2regexp('ACWTMAN')
```

```
ans =
```

```
AC[ATW]T[ACM]A[ACGTRYKMSWBDHVN]
```

- 2 Convert the same nucleotide sequence, but remove ambiguous characters from the regular expression.

```
seq2regexp('ACWTMAN', 'ambiguous', false)
```

```
ans =
```

```
AC[AT]T[AC]A[ACGT]
```

See Also

Bioinformatics Toolbox functions: `restrict`, `seqwordcount`

MATLAB functions: `regexp`, `regexp`

seqcomplement

Purpose Calculate complementary strand of nucleotide sequence

Syntax `SeqC = seqcomplement(SeqNT)`

Arguments `SeqNT` Enter either a character string with the characters A, T (U), G, C, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N, or a vector of integers. You can also enter a structure with the field `Sequence`.

Description `SeqC = seqcomplement(SeqNT)` calculates the complementary strand (A→T, C→G, G→C, T→A) of a DNA sequence and returns a sequence in the same format as `SeqNT`. For example, if `SeqNT` is an integer sequence then so is `SeqC`.

Example Return the complement of a DNA nucleotide sequence.

```
s = 'ATCG';
seqcomplement(s)

ans =
TAGC
```

See Also Bioinformatics Toolbox functions: `seqrcomplement`, `seqreverse`, `seqtool`

Purpose	Calculate consensus sequence	
Syntax	<pre> CSeq = seqconsensus(Seqs) [CSeq, Score] = seqconsensus(Seqs) CSeq = seqconsensus(Profile) seqconsensus(..., 'PropertyName', PropertyValue,...) seqconsensus(..., 'ScoringMatrix', ScoringMatrixValue) </pre>	
Arguments	<i>Seqs</i>	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 <code>Sequence</code> .
	<i>Profile</i>	Sequence profile. Enter a profile from the function <code>seqprofile</code> . <code>Profile</code> is a matrix of size <code>[20 (or 4) x Sequence Length]</code> with the frequency or count of amino acids (or nucleotides) for every position. <code>Profile</code> can also have 21 (or 5) rows if gaps are included in the consensus.
	<i>ScoringMatrixValue</i>	Scoring matrix. The default value is BLOSUM50 for amino acid sequences or NUC44 for nucleotide sequences. <code>ScoringMatrix</code> can also be a 21x21, 5x5, 20x20, or 4x4 numeric array. For the gap-included cases, gap scores (last row/column) are set to <code>mean(diag(ScoringMatrix))</code> for a gap matching with another gap, and set to <code>mean(nodiag(ScoringMatrix))</code> for a gap matching with another symbol
Description	<code>CSeq = seqconsensus(Seqs)</code> , for a multiply aligned set of sequences (<code>Seqs</code>), returns a string with the consensus sequence (<code>CSeq</code>). The frequency of symbols (20 amino acids, 4 nucleotides) in the set of sequences is determined with the function <code>seqprofile</code> . For ambiguous	

seqconsensus

nucleotide or amino acid symbols, the frequency or count is added to the standard set of symbols.

`[CSeq, Score] = seqconsensus(Seqs)` returns the conservation score of the consensus sequence. Scores are computed with the scoring matrix BLOSUM50 for amino acids or NUC44 for nucleotides. Scores are the average euclidean distance between the scored symbol and the M-dimensional consensus value. M is the size of the alphabet. The consensus value is the profile weighted by the scoring matrix.

`CSeq = seqconsensus(Profile)` returns a string with the consensus sequence (*CSeq*) from a sequence profile (*Profile*).

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

`seqconsensus(..., 'ScoringMatrix', ScoringMatrixValue)` specifies the scoring matrix.

The following input parameters are analogous to the function `seqprofile` when the alphabet is restricted to 'AA' or 'NT'.

`seqconsensus(..., 'Alphabet', AlphabetValue)`

`seqconsensus(..., 'Gaps', GapsValue)`

`seqconsensus(..., 'Ambiguous', AmbiguousValue)`

`seqconsensus(..., 'Limits', LimitsValue)`

Examples

```
seqs = fastaread('pf00002.fa');  
[C,S] = seqconsensus(seqs,'limits',[50 60],'gaps','all')
```

See Also

Bioinformatics Toolbox functions: `fastaread`, `multialignread`, `multialignwrite`, `profaalign`, `seqdisp`, `seqprofile`

Purpose

Format long sequence output for easy viewing

Syntax

```
seqdisp(Seq)
seqdisp(Seq, ...'Row', RowValue, ...)
seqdisp(Seq, ...'Column', ColumnValue, ...)
seqdisp(Seq, ...'ShowNumbers', ShowNumbersValue, ...)
```

Arguments

<i>Seq</i>	Nucleotide or amino acid sequence represented by any of the following: <ul style="list-style-type: none">• Character array• FASTA file name• MATLAB structure with the field <code>Sequence</code> Multiply aligned sequences are allowed. FASTA files can have the file extension <code>fa</code> , <code>fasta</code> , <code>fas</code> , <code>fsa</code> , or <code>fst</code> .
<i>RowValue</i>	Integer that specifies the length of each row. Default is 60.
<i>ColumnValue</i>	Integer that specifies the column width or number of symbols before displaying a space. Default is 10.
<i>ShowNumbersValue</i>	Controls the display of numbers at the start of each row. Choices are <code>true</code> (default) to show numbers, or <code>false</code> to hide numbers.

Description

`seqdisp(Seq)` displays a sequence in rows, with a default row length of 60 and a default column width of 10.

`seqdisp(Seq, ...'PropertyName', PropertyValue, ...)` calls `seqdisp` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each

seqdisp

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

`seqdisp(Seq, ... 'Row', RowValue, ...)` specifies the length of each row for the displayed sequence.

`seqdisp(Seq, ... 'Column', ColumnValue, ...)` specifies the number of letters to display before adding a space. *RowValue* must be larger than and evenly divisible by *ColumnValue*.

`seqdisp(Seq, ... 'ShowNumbers', ShowNumbersValue, ...)` controls the display of numbers at the start of each row. Choices are `true` (default) to show numbers, or `false` to hide numbers.

Examples

Read sequence information from the GenBank database. Display the sequence in rows with 50 letters, and within a row, separate every 10 letters with a space.

```
mouseHEXA = getgenbank('AK080777');
seqdisp(mouseHEXA, 'Row', 50, 'Column', 10)
```

Create and save a FASTA file with two sequences, and then display it.

```
hdr = ['Sequence A'; 'Sequence B'];
seq = ['TAGCTGRCCAAGGCCAAGCGAGCTTN'; 'ATCGACYGGTCCGGTTCGCTCGAAN']
fastawrite('local.fa', hdr, seq);
seqdisp('local.fa', 'ShowNumbers', false')
```

```
ans =
>Sequence A
  1 TAGCTGRCCA AGGCCAAGCG AGCTTN
>Sequence B
  1 ATCGACYGGT TCCGGTTCGC TCGAAN
```

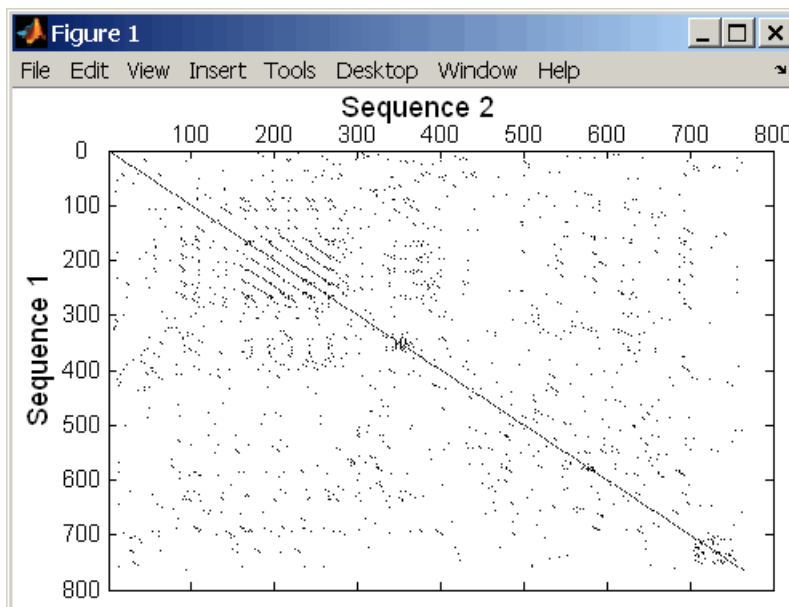
See Also

Bioinformatics Toolbox functions: `multialignread`, `multialignwrite`, `seqconsensus`, `seqlogo`, `seqprofile`, `seqshoworfs`, `seqshowwords`, `seqtool`, `getgenbank`

Purpose	Create dot plot of two sequences						
Syntax	<pre>seqdotplot (Seq1, Seq2) seqdotplot(Seq1,Seq2, Window, Number) Matches = seqdotplot(...) [Matches, Matrix] = seqdotplot(...)</pre>						
Arguments	<table><tr><td><i>Seq1, Seq2</i></td><td>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 <i>Sequence</i>.</td></tr><tr><td><i>Window</i></td><td>Enter an integer for the size of a window.</td></tr><tr><td><i>Number</i></td><td>Enter an integer for the number of characters within the window that match.</td></tr></table>	<i>Seq1, Seq2</i>	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 <i>Sequence</i> .	<i>Window</i>	Enter an integer for the size of a window.	<i>Number</i>	Enter an integer for the number of characters within the window that match.
<i>Seq1, Seq2</i>	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 <i>Sequence</i> .						
<i>Window</i>	Enter an integer for the size of a window.						
<i>Number</i>	Enter an integer for the number of characters within the window that match.						
Description	<p><code>seqdotplot (Seq1, Seq2)</code> plots a figure that visualizes the match between two sequences.</p> <p><code>seqdotplot(Seq1,Seq2, Window, Number)</code> plots sequence matches when there are at least <i>Number</i> matches in a window of size <i>Window</i>.</p> <p>When plotting nucleotide sequences, start with a <i>Window</i> of 11 and <i>Number</i> of 7.</p> <p><code>Matches = seqdotplot(...)</code> returns the number of dots in the dot plot matrix.</p> <p><code>[Matches, Matrix] = seqdotplot(...)</code> returns the dot plot as a sparse matrix.</p>						
Examples	<p>This example shows the similarities between the prion protein (PrP) nucleotide sequences of two ruminants, the moufflon and the golden takin.</p> <pre>moufflon = getgenbank('AB060288','Sequence',true); takin = getgenbank('AB060290','Sequence',true);</pre>						

seqdotplot

```
seqdotplot(moufflon,takin,11,7)
```



Note For the correct interpretation of a dot plot, your monitor's display resolution must be able to contain the sequence lengths. If the resolution is not adequate, `seqdotplot` resizes the image and returns a warning.

```
Matches = seqdotplot(moufflon,takin,11,7)
```

```
Matches =  
    5552
```

```
[Matches, Matrix] = seqdotplot(moufflon,takin,11,7)
```

See Also

Bioinformatics Toolbox functions: `nalign`, `swalign`

Purpose

Insert gaps into nucleotide or amino acid sequence

Syntax

NewSeq = seqinsertgaps(*Seq*, *Positions*)

NewSeq = seqinsertgaps(*Seq*, *GappedSeq*)

NewSeq = seqinsertgaps(*Seq*, *GappedSeq*, *Relationship*)

Arguments

Seq

Either of the following:

- String specifying a nucleotide or amino acid sequence
- MATLAB structure containing a `Sequence` field

Positions

Vector of integers to specify the positions in *Seq* before which to insert a gap.

GappedSeq

Either of the following:

- String specifying a nucleotide or amino acid sequence
- MATLAB structure containing a `Sequence` field

Relationship

Integer specifying the relationship between *Seq* and *GappedSeq*. Choices are:

- 1 — Both sequences use the same alphabet, that is both are nucleotide sequences or both are amino acid sequences.
- 3 — *Seq* contains nucleotides representing codons and *GappedSeq* contains amino acids (default).

Return Values

NewSeq

Sequence with gaps inserted, represented by a string specifying a nucleotide or amino acid sequence.

seqinsertgaps

Description

NewSeq = seqinsertgaps(*Seq*, *Positions*) inserts gaps in the sequence *Seq* before the positions specified by the integers in the vector *Positions*.

NewSeq = seqinsertgaps(*Seq*, *GappedSeq*) finds the gap positions in the sequence *GappedSeq*, then inserts gaps in the corresponding positions in the sequence *Seq*.

NewSeq = seqinsertgaps(*Seq*, *GappedSeq*, *Relationship*) specifies the relationship between *Seq* and *GappedSeq*. Enter 1 for *Relationship* when both sequences use the same alphabet, that is both are nucleotide sequences or both are amino acid sequences. Enter 3 for *Relationship* when *Seq* contains nucleotides representing codons and *GappedSeq* contains amino acids. Default is 3.

Examples

- 1 Retrieve two nucleotide sequences from the GenBank database for the neuraminidase (NA) protein of two strains of the Influenza A virus (H5N1).

```
hk01 = getgenbank('AF509094');  
vt04 = getgenbank('DQ094287');
```

- 2 Extract the coding region from the two nucleotide sequences.

```
hk01_cds = featuresparse(hk01,'feature','CDS','Sequence',true);  
vt04_cds = featuresparse(vt04,'feature','CDS','Sequence',true);
```

- 3 Align the amino acids sequences converted from the nucleotide sequences.

```
[sc,a1]=nwalign(nt2aa(hk01_cds),nt2aa(vt04_cds),'extendgap',1);
```

- 4 Use the seqinsertgaps function to copy the gaps from the aligned amino acid sequences to their corresponding nucleotide sequences, thus codon-aligning them.

```
hk01_aligned = seqinsertgaps(hk01_cds,a1(1,:))  
vt04_aligned = seqinsertgaps(vt04_cds,a1(3,:))
```

- 5 Once you have code aligned the two sequences, you can use them as input to other functions such as `dnds`, which calculates the synonymous and nonsynonymous substitutions rates of the codon-aligned nucleotide sequences. By setting `Verbose` to `true`, you can also display the codons considered in the computations and their amino acid translations.

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

See Also

Bioinformatics Toolbox functions: `dnds`, `dndsm1`, `int2aa`, `int2nt`

seqlinkage

Purpose Construct phylogenetic tree from pairwise distances

Syntax

```
Tree = seqlinkage(Dist)
Tree = seqlinkage(Dist, Method)
Tree = seqlinkage(Dist, Method, Names)
```

Arguments

<i>Dist</i>	Matrix or vector of pairwise distances, such as returned by the <code>seqpdist</code> function.
<i>Method</i>	String that specifies a distance method. Choices are: <ul style="list-style-type: none">• 'single'• 'complete'• 'average' (default)• 'weighted'• 'centroid'• 'median'
<i>Names</i>	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 <i>Dist</i> .

Description

`Tree = seqlinkage(Dist)` returns a phylogenetic tree object from the pairwise distances, *Dist*, between the species or products. *Dist* is a matrix or vector of pairwise distances, such as returned by the `seqpdist` function.

`Tree = seqlinkage(Dist, Method)` creates a phylogenetic tree object using a specified patristic distance method. The available methods are:

'single'	Nearest distance (single linkage method)
'complete'	Furthest distance (complete linkage method)
'average' (default)	Unweighted Pair Group Method Average (UPGMA, group average).
'weighted'	Weighted Pair Group Method Average (WPGMA)
'centroid'	Unweighted Pair Group Method Centroid (UPGMC)
'median'	Weighted Pair Group Method Centroid (WPGMC)

Tree = seqlinkage(*Dist*, *Method*, *Names*) passes a list of names to label the leaf nodes (for example, species or products) in a phylogenetic tree object.

Examples

```
% Load a multiple alignment of amino acids:
seqs = fastaread('pf00002.fa');
% Measure the 'Jukes-Cantor' pairwise distances:
dist = seqpdist(seqs,'method','jukes-cantor',...
               'indels','pair');
% Build the phylogenetic tree with the single linkage
% method and pass the names of the sequences:
tree = seqlinkage(dist,'single',seqs)
view(tree)
```

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreewrite`, `seqpdist`, `seqneighjoin`

Bioinformatics Toolbox methods of `phytree` object: `plot`, `view`

seqlogo

Purpose Display sequence logo for nucleotide or amino acid sequences

Syntax

```
seqlogo(Seqs)  
seqlogo(Profile)  
WgtMatrix = seqlogo(...)  
[WgtMatrix, Handle] = seqlogo(...)  
seqlogo(..., 'Displaylogo', DisplaylogoValue, ...)  
seqlogo(..., 'Alphabet', AlphabetValue, ...)  
seqlogo(..., 'Startat', StartatValue, ...)  
seqlogo(..., 'Endat', EndatValue, ...)  
seqlogo(..., 'SSCorrection', SSCorrectionValue, ...)
```

Arguments

<i>Seqs</i>	Set of pairwise or multiply aligned nucleotide or amino acid sequences, represented by any of the following: <ul style="list-style-type: none">• Character array• Cell array of strings• Array of structures containing a <code>Sequence</code> field
<i>Profile</i>	Sequence profile distribution matrix with the frequency of nucleotides or amino acids for every column in the multiple alignment, such as returned by the <code>seqprofile</code> function. The size of the frequency distribution matrix is: <ul style="list-style-type: none">• For nucleotides — [4 x sequence length]• For amino acids — [20 x sequence length] If gaps were included, <i>Profile</i> may have 5 rows (for nucleotides) or 21 rows (for amino acids), but <code>seqlogo</code> ignores gaps.
<i>DisplaylogoValue</i>	Controls the display of a sequence logo. Choices are <code>true</code> (default) or <code>false</code> .
<i>AlphabetValue</i>	String specifying the type of sequence (nucleotide or amino acid). Choices are <code>'NT'</code> (default) or <code>'AA'</code> .
<i>StartatValue</i>	Positive integer that specifies the starting position for the sequences in <i>Seqs</i> . Default starting position is 1.

EndatValue Positive integer that specifies the ending position for the sequences in *Seqs*. Default ending position is the maximum length of the sequences in *Seqs*.

SSCorrectionValue Controls the use of small sample correction in the estimation of the number of bits. Choices are true (default) or false.

Return Values

WgtMatrix Cell array containing the symbol list in *Seqs* or *Profile* and the weight matrix used to graphically display the sequence logo.

Handle Handle to the sequence logo figure.

Description

`seqlogo(Seqs)` displays a sequence logo for *Seqs*, a set of aligned sequences. The logo graphically displays the sequence conservation at a particular position in the alignment of sequences, measured in bits. The maximum sequence conservation per site is $\log_2(4)$ bits for nucleotide sequences and $\log_2(20)$ bits for amino acid sequences. If the sequence conservation value is zero or negative, no logo is displayed in that position.

`seqlogo(Profile)` displays a sequence logo for *Profile*, a sequence profile distribution matrix with the frequency of nucleotides or amino acids for every column in the multiple alignment, such as returned by the `seqprofile` function.

Color Code for Nucleotides

Nucleotide	Color
A	Green
C	Blue

Color Code for Nucleotides (Continued)

Nucleotide	Color
G	Yellow
T, U	Red
Other	Purple

Color Code for Amino Acids

Amino Acid	Chemical Property	Color
G S T Y C Q N	Polar	Green
A V L I P W F M	Hydrophobic	Orange
D E	Acidic	Red
K R H	Basic	Blue
Other	—	Tan

WgtMatrix = seqlogo(...) returns a cell array of unique symbols in the sequence *Seqs* or *Profile*, and the information weight matrix used to graphically display the logo.

[*WgtMatrix*, *Handle*] = seqlogo(...) returns a handle to the sequence logo figure.

seqlogo(*Seqs*, ...'*PropertyName*', *PropertyValue*, ...) calls seqpdist with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

seqlogo(..., 'Displaylogo', *DisplaylogoValue*, ...) controls the display of a sequence logo. Choices are true (default) or false.

seqlogo(..., 'Alphabet', *AlphabetValue*, ...) specifies the type of sequence (nucleotide or amino acid). Choices are 'NT' (default) or 'AA'.

Note If you provide amino acid sequences to `seqlogo`, you must set `Alphabet` to `'AA'`.

`seqlogo(..., 'Startat', StartatValue, ...)` specifies the starting position for the sequences in *Seqs*. Default starting position is 1.

`seqlogo(..., 'Endat', EndatValue, ...)` specifies the ending position for the sequences in *Seqs*. Default ending position is the maximum length of the sequences in *Seqs*.

`seqlogo(..., 'SSCorrection', SSCorrectionValue, ...)` controls the use of small sample correction in the estimation of the number of bits. Choices are `true` (default) or `false`.

Note A simple calculation of bits tends to overestimate the conservation at a particular location. To compensate for this overestimation, when `SSCorrection` is set to `true`, a rough estimate is applied as an approximate correction. This correction works better when the number of sequences is greater than 50.

Examples

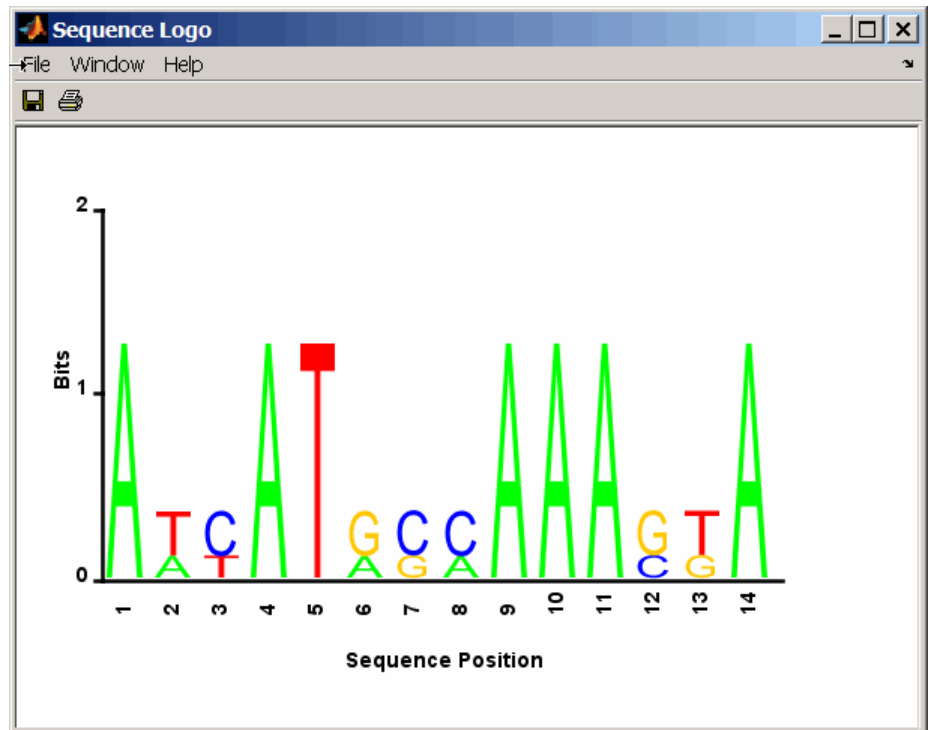
Displaying a Sequence Logo for a Nucleotide Sequence

1 Create a series of aligned nucleotide sequences.

```
S = {'ATTATAGCAAAC TA', ...  
     'AACATGCCAAAG TA', ...  
     'ATCATGCAAAAAG GA' }
```

2 Display the sequence logo.

```
seqlogo(S)
```



- 3** Notice that correction for small samples prevents you from seeing columns with information equal to $\log_2(4) = 2$ bits, but you can turn this adjustment off.

```
seqlogo(S, 'sscorrection', false)
```

Displaying a Sequence Logo for an Amino Acid Sequence

- 1** Create a series of aligned amino acid sequences.

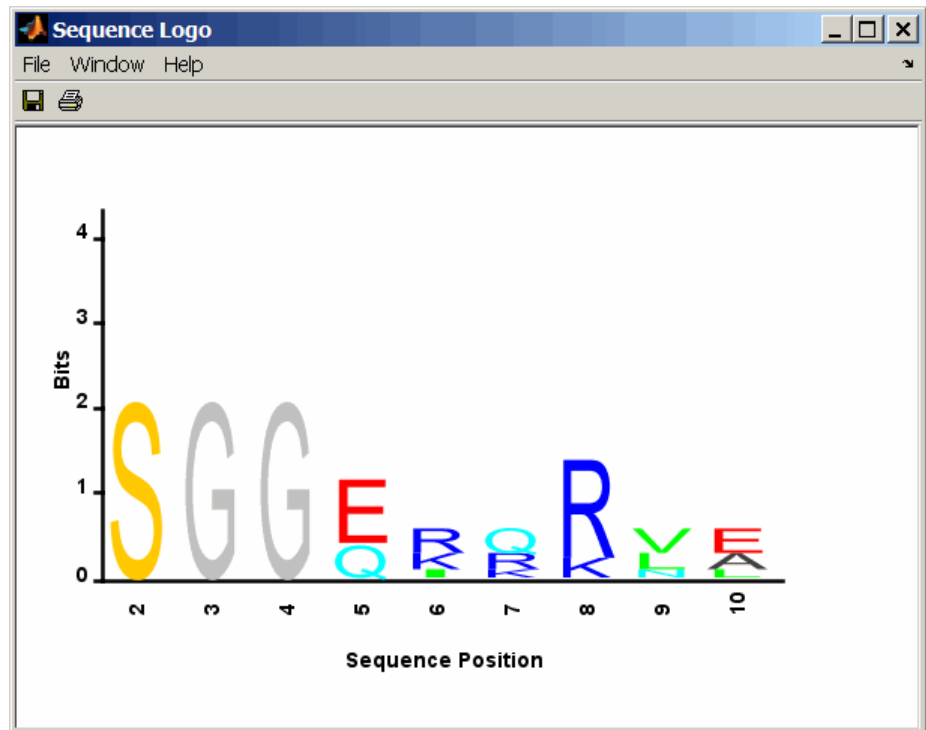
```
S2 = {'LSGGQRQRVAIARALAL', ...
      'LSGGEKQRVAIARALMN', ...
      'LSGGQIQRVLLARALAA', ...
      'LSGGERRRLEIACVLAL', ...}
```

seqlogo

```
'FSGGEKKKNELWQMLAL', ...  
'LSGGERRRLEIACVLAL'};
```

- 2 Display the sequence logo, specifying an amino acid sequence and limiting the logo to sequence positions 2 through 10.

```
seqlogo(S2, 'alphabet', 'aa', 'startAt', 2, 'endAt', 10)
```



References

[1] Schneider, T.D., and Stephens, R.M. (1990). Sequence Logos: A new way to display consensus sequences. *Nucleic Acids Research* 18, 6097–6100.

See Also

Bioinformatics Toolbox functions: `seqconsensus`, `seqdisp`, `seqprofile`

Purpose Find matches for every string in library

Syntax Index = seqmatch(Strings, Library)

Description Index = seqmatch(Strings, Library) looks through the elements of Library to find strings that begin with every string in Strings. Index contains the index to the first occurrence for every string in the query. Strings and Library must be cell arrays of strings.

Examples

```
lib = {'VIPS_HUMAN', 'SCCR_RABIT', 'CALR_PIG', 'VIPR_RAT', 'PACR_MOUSE'};
query = {'CALR', 'VIP'};
h = seqmatch(query, lib);
lib(h)
```

See Also MATLAB functions: regexp, strmatch

seqneighjoin

Purpose Neighbor-joining method for phylogenetic tree reconstruction

Syntax

```
Tree = seqneighjoin(Dist)
Tree = seqneighjoin(Dist, Method)
Tree = seqneighjoin(Dist, Method, Names)
seqneighjoin(..., 'Reroot', RerootValue)
```

Arguments

Dist Matrix or vector returned by the seqpdist function.

Method Method to compute the distances between nodes. Enter 'equivar' (default), 'firstorder', or 'average'.

Names Vector of structures with the fields 'Header', 'Name', or a cell array of strings. In all cases the number of elements must equal the number of samples used to generate the pairwise distances in *Dist*.

Description *Tree* = seqneighjoin(*Dist*) computes a phylogenetic tree object from *Dist*, pairwise distances between the species or products using the neighbor-joining method.

Tree = seqneighjoin(*Dist*, *Method*) specifies *Method*, a method to compute the distances of the new nodes to all other nodes at every iteration. The general expression to calculate the distances between the new node, *n*, after joining *i* and *j* and all other nodes (*k*), is given by

$$D(n,k) = a*D(i,k) + (1-a)*D(j,k) - a*D(n,i) - (1-a)*D(n,j)$$

This expression is guaranteed to find the correct tree with additive data (minimum variance reduction).

Choices for *Method* are:

Method	Description
'equivar' (default)	Assumes equal variance and independence of evolutionary distance estimates ($a = 1/2$). Such as in Studier and Keppeler, JMBE (1988).

Method	Description
'firstorder'	Assumes a first-order model of the variances and covariances of evolutionary distance estimates, 'a' is adjusted at every iteration to a value between 0 and 1. Such as in Gascuel, JMBE (1997).
'average'	New distances are the weighted average of previous distances while the branch distances are ignored. $D(n,k) = [D(i,k) + D(j,k)] / 2$ As in the original neighbor-joining algorithm by Saitou and Nei, JMBE (1987).

`Tree = seqneighjoin(Dist, Method, Names)` passes a list of names (*Names*) to label the leaf nodes (e.g., species or products) in the phylogenetic tree object.

`seqneighjoin(..., 'Reroot', RerootValue)`, when *RerootValue* is false, excludes rerooting the resulting tree. This is useful for observing the original linkage order followed by the algorithm. By default `seqneighjoin` reroots the resulting tree using the midpoint method.

Examples

- 1 Load a multiple alignment of amino acids.

```
seqs = fastaread('pf00002.fa');
```

- 2 Measure the Jukes-Cantor pairwise distances.

```
dist = seqpdist(seqs,'method','jukes-cantor','indels','pair');
```

- 3 Build the phylogenetic using the neighbor-joining algorithm.

```
tree = seqneighjoin(dist,'equivar',seqs)
view(tree)
```

References

- [1] Saitou, N., and Nei, M. (1987). The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution* 4(4), 406–425.
- [2] Gascuel, O. (1997). BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution* 14 685–695.
- [3] Studier, J.A., Keppler, K.J. (1988). A note on the neighbor-joining algorithm of Saitou and Nei. *Molecular Biology and Evolution* 5(6) 729–731.

See Also

Bioinformatics Toolbox functions: `multialign`, `phytree` (object constructor), `seqlinkage` (alternative method to create a phylogenetic tree), `seqpdist`

Methods of `phytree` object: `reroot`, `view`

Purpose

Calculate pairwise distance between sequences

Syntax

```

D = seqpdist(Seqs)
D = seqpdist(Seqs, ...'Method', MethodValue, ...)
D = seqpdist(Seqs, ...'Indels', IndelsValue, ...)
D = seqpdist(Seqs, ...'Optargs', OptargsValue, ...)
D = seqpdist(Seqs, ...'PairwiseAlignment',
    PairwiseAlignmentValue, ...)
D = seqpdist(Seqs, ...'JobManager', JobManagerValue, ...)
D = seqpdist(Seqs, ...'WaitInQueue', WaitInQueueValue, ...)
D = seqpdist(Seqs, ...'SquareForm', SquareFormValue, ...)
D = seqpdist(Seqs, ...'Alphabet', AlphabetValue, ...)
D = seqpdist(Seqs, ...'ScoringMatrix', ScoringMatrixValue,
    ...)
D = seqpdist(Seqs, ...'Scale', ScaleValue, ...)
D = seqpdist(Seqs, ...'GapOpen', GapOpenValue, ...)
D = seqpdist(Seqs, ...'ExtendGap', ExtendGapValue, ...)

```

Arguments

Seqs

Any of the following:

- Cell array containing nucleotide or amino acid sequences
- Vector of structures containing a Sequence field
- Matrix of characters, in which each row corresponds to a nucleotide or amino acid sequence

MethodValue

String that specifies the method for calculating pairwise distances. Default is Jukes-Cantor.

IndelsValue

String that specifies how to treat sites with gaps. Default is score.

- OptargsValue* String or cell array specifying one or more input arguments required or accepted by the distance method specified by the Method property.
- PairwiseAlignmentValue* Controls the global pairwise alignment of input sequences (using the `nwalign` function), while ignoring the multiple alignment of the input sequences (if any). Choices are `true` or `false`. Default is:
- `true` — When all input sequences do not have the same length.
 - `false` — When all input sequences have the same length.

Tip If your input sequences have the same length, `seqpdist` will assume they are aligned. If they are not aligned, do one of the following:

- Align the sequences before passing them to `seqpdist`, for example, using the `multialign` function.
 - Set `PairwiseAlignment` to `true` when using `seqpdist`.
-

<i>JobManagerValue</i>	A <code>jobmanager</code> object, such as returned by the Parallel Computing Toolbox function <code>findResource</code> , that represents an available distributed MATLAB resource. Specifying this property distributes pairwise alignments into a cluster of computers using the Parallel Computing Toolbox software. You must have the Parallel Computing Toolbox software to use this property.
<i>WaitInQueueValue</i>	Controls whether <code>seqpdist</code> waits for a distributed MATLAB resource to be available when you have set the <code>JobManager</code> property. Choices are <code>true</code> or <code>false</code> (default). You must have the Parallel Computing Toolbox software to use this property.
<i>SquareFormValue</i>	Controls the conversion of the output into a square matrix. Choices are <code>true</code> or <code>false</code> (default).
<i>AlphabetValue</i>	String specifying the type of sequence (nucleotide or amino acid). Choices are <code>'NT'</code> or <code>'AA'</code> (default).

<i>ScoringMatrixValue</i>	<p>String specifying the scoring matrix to use for the global pairwise alignment. Choices for amino acid sequences are:</p> <ul style="list-style-type: none">• 'PAM40'• 'PAM250'• 'DAYHOFF'• 'GONNET'• 'BLOSUM30' increasing by 5 up to 'BLOSUM90'• 'BLOSUM62'• 'BLOSUM100' <p>Default is:</p> <ul style="list-style-type: none">• 'NUC44' (when <i>AlphabetValue</i> equals 'NT')• 'BLOSUM50' (when <i>AlphabetValue</i> equals 'AA')
<i>ScaleValue</i>	<p>Positive value that specifies the scale factor used to return the score in arbitrary units. If the scoring matrix information also provides a scale factor, then both are used.</p>
<i>GapOpenValue</i>	<p>Positive integer specifying the penalty for opening a gap in the alignment. Default is 8.</p>
<i>ExtendedGapValue</i>	<p>Positive integer specifying the penalty for extending a gap. Default is equal to <i>GapOpenValue</i>.</p>

Return Values*D*

Vector containing biological distances between each pair of sequences stored in the *M* elements of *Seqs*.

Description

$D = \text{seqpdist}(\text{Seqs})$ returns *D*, a vector containing biological distances between each pair of sequences stored in the *M* sequences of *Seqs*, a cell array of sequences, a vector of structures, or a matrix or sequences.

D is a 1-by- $(M*(M-1)/2)$ row vector corresponding to the $M*(M-1)/2$ pairs of sequences in *Seqs*. The output *D* is arranged in the order $((2,1), (3,1), \dots, (M,1), (3,2), \dots, (M,2), \dots, (M,M-1))$. This is the lower-left triangle of the full *M*-by-*M* distance matrix. To get the distance between the *I*th and the *J*th sequences for $I > J$, use the formula $D((J-1)*(M-J/2)+I-J)$.

$D = \text{seqpdist}(\text{Seqs}, \dots, \text{'PropertyName'}, \text{PropertyValue}, \dots)$ calls *seqpdist* with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

$D = \text{seqpdist}(\text{Seqs}, \dots, \text{'Method'}, \text{MethodValue}, \dots)$ specifies a method to compute distances between every pair of sequences. Choices are shown in the following tables.

Methods for Nucleotides and Amino Acids

Method	Description
p-distance	Proportion of sites at which the two sequences are different. <i>p</i> is close to 1 for poorly related sequences, and <i>p</i> is close to 0 for similar sequences. $d = p$

Methods for Nucleotides and Amino Acids (Continued)

Method	Description
Jukes-Cantor (default)	<p>Maximum likelihood estimate of the number of substitutions between two sequences. p is described with the method p-distance.</p> <p>For nucleotides:</p> $d = -3/4 \log(1-p * 4/3)$ <p>For amino acids:</p> $d = -19/20 \log(1-p * 20/19)$
alignment-score	<p>Distance (d) between two sequences (1, 2) is computed from the pairwise alignment score between the two sequences ($score_{12}$), and the pairwise alignment score between each sequence and itself ($score_{11}$, $score_{22}$) as follows:</p> $d = (1 - score_{12}/score_{11}) * (1 - score_{12}/score_{22})$ <p>This option does not imply that prealigned input sequences will be realigned, it only scores them. Use with care; this distance method does not comply with the ultrametric condition. In the rare case where the score between sequences is greater than the score when aligning a sequence with itself, then $d = 0$.</p>

Methods with No Scoring of Gaps (Nucleotides Only)

Method	Description
Tajima-Nei	<p>Maximum likelihood estimate considering the background nucleotide frequencies. It can be computed from the input sequences or given by setting <code>Optargs</code> to <code>[gA gC gG gT]</code>. gA, gC, gG, gT are scalar values for the nucleotide frequencies.</p>

Methods with No Scoring of Gaps (Nucleotides Only) (Continued)

Method	Description
Kimura	Considers separately the transitional nucleotide substitution and the transversional nucleotide substitution.
Tamura	Considers separately the transitional nucleotide substitution, the transversional nucleotide substitution, and the GC content. GC content can be computed from the input sequences or given by setting <code>Optargs</code> to the proportion of GC content (scalar value from 0 to 1).
Hasegawa	Considers separately the transitional nucleotide substitution, the transversional nucleotide substitution, and the background nucleotide frequencies. Background frequencies can be computed from the input sequences or given by setting the <code>Optargs</code> property to <code>[gA gC gG gT]</code> .
Nei-Tamura	Considers separately the transitional nucleotide substitution between purines, the transitional nucleotide substitution between pyrimidines, the transversional nucleotide substitution, and the background nucleotide frequencies. Background frequencies can be computed from the input sequences or given by setting the <code>Optargs</code> property to <code>[gA gC gG gT]</code> .

Methods with No Scoring of Gaps (Amino Acids Only)

Method	Description
Poisson	Assumes that the number of amino acid substitutions at each site has a Poisson distribution.
Gamma	Assumes that the number of amino acid substitutions at each site has a Gamma distribution with parameter a . You can set a by using the <code>Optargs</code> property. Default is 2.

You can also specify a user-defined distance function using `@`, for example, `@distfun`. The distance function must be of the form:

```
function D = distfun(S1, S2, OptArgsValue)
```

The `distfun` function takes the following arguments:

- *S1*, *S2* — Two sequences of the same length (nucleotide or amino acid).
- *OptArgsValue* — Optional problem-dependent arguments.

The `distfun` function returns a scalar that represents the distance between *S1* and *S2*.

`D = seqpdist(Seqs, ...'Indels', IndelsValue, ...)` specifies how to treat sites with gaps. Choices are:

- `score` (default) — Scores these sites either as a point mutation or with the alignment parameters, depending on the method selected.
- `pairwise-del` — For every pairwise comparison, it ignores the sites with gaps.
- `complete-del` — Ignores all the columns in the multiple alignment that contain a gap. This option is available only if a multiple alignment was provided as the input *Seqs*.

`D = seqpdist(Seqs, ...'Optargs', OptargsValue, ...)` passes one or more arguments required or accepted by the distance method specified by the `Method` property. Use a string or cell array to pass one or multiple input arguments. For example, you can provide the nucleotide frequencies for the Tajima-Nei distance method, instead of computing them from the input sequences.

`D = seqpdist(Seqs, ...'PairwiseAlignment', PairwiseAlignmentValue, ...)` controls the global pairwise alignment of input sequences (using the `nwalgn` function), while ignoring the multiple alignment of the input sequences (if any). Default is:

- `true` — When all input sequences do not have the same length.
- `false` — When all input sequences have the same length.

Tip If your input sequences have the same length, `seqpdist` will assume they aligned. If they are not aligned, do one of the following:

- Align the sequences before passing them to `seqpdist`, for example, using the `multialign` function.
 - Set `PairwiseAlignment` to `true` when using `seqpdist`.
-

`D = seqpdist(Seqs, ...'JobManager', JobManagerValue, ...)` distributes pairwise alignments into a cluster of computers using the Parallel Computing Toolbox software. `JobManagerValue` is a `jobmanager` object such as returned by the Parallel Computing Toolbox function `findResource`, that represents an available distributed MATLAB resource. You must have the Parallel Computing Toolbox software to use this property.

`D = seqpdist(Seqs, ...'WaitInQueue', WaitInQueueValue, ...)` controls whether `seqpdist` waits for a distributed MATLAB resource to be available when you have set the `JobManager` property. When `WaitInQueueValue` is `true`, `seqpdist` waits in the job manager queue for an available worker. When `WaitInQueueValue` is `false` (default) and there are no workers immediately available, `seqpdist` stops and displays an error message. You must have the Parallel Computing Toolbox software and have also set the `JobManager` property to use this property.

`D = seqpdist(Seqs, ...'SquareForm', SquareFormValue, ...)`, controls the conversion of the output into a square matrix such that $D(I, J)$ denotes the distance between the I th and J th sequences. The square matrix is symmetric and has a zero diagonal. Choices are `true` or `false` (default). Setting `SquareForm` to `true` is the same as using the `squareform` function in the Statistics Toolbox software.

seqpdist

`D = seqpdist(Seqs, ...'Alphabet', AlphabetValue, ...)` specifies the type of sequence (nucleotide or amino acid). Choices are 'NT' or 'AA' (default).

The remaining input properties are available when the `Method` property equals 'alignment-score' or the `PairwiseAlignment` property equals true.

`D = seqpdist(Seqs, ...'ScoringMatrix', ScoringMatrixValue, ...)` specifies the scoring matrix to use for the global pairwise alignment. Default is:

- 'NUC44' (when *AlphabetValue* equals 'NT')
- 'BLOSUM50' (when *AlphabetValue* equals 'AA')

`D = seqpdist(Seqs, ...'Scale', ScaleValue, ...)` specifies the scale factor used to return the score in arbitrary units. Choices are any positive value. If the scoring matrix information also provides a scale factor, then both are used.

`D = seqpdist(Seqs, ...'GapOpen', GapOpenValue, ...)` specifies the penalty for opening a gap in the alignment. Choices are any positive integer. Default is 8.

`D = seqpdist(Seqs, ...'ExtendGap', ExtendGapValue, ...)` specifies the penalty for extending a gap in the alignment. Choices are any positive integer. Default is equal to *GapOpenValue*.

Examples

- 1 Read amino acids alignment data into a MATLAB structure.

```
seqs = fastaread('pf00002.fa');
```

- 2 For every possible pair of sequences in the multiple alignment, ignore sites with gaps and score with the scoring matrix PAM250.

```
dist = seqpdist(seqs,'Method','alignment-score',...  
               'Indels','pairwise-delete',...  
               'ScoringMatrix','pam250');
```

- 3** Force the realignment of every pair of sequences ignoring the provided multiple alignment.

```
dist = seqpdist(seqs,'Method','alignment-score',...
               'Indels','pairwise-delete',...
               'ScoringMatrix','pam250',...
               'PairwiseAlignment',true);
```

- 4** Measure the 'Jukes-Cantor' pairwise distances after realigning every pair of sequences, counting the gaps as point mutations.

```
dist = seqpdist(seqs,'Method','jukes-cantor',...
               'Indels','score',...
               'Scoringmatrix','pam250',...
               'PairwiseAlignment',true);
```

See Also

Bioinformatics Toolbox functions: `fastaread`, `dnds`, `dndsml`, `multialign`, `nwalign`, `phytree` (object constructor), `seqlinkage`

Bioinformatics Toolbox object: `phytree` object

Bioinformatics Toolbox method of a `phytree` object: `pdist`

seqprofile

Purpose Calculate sequence profile from set of multiply aligned sequences

Syntax

```
Profile = seqprofile(Seqs)  
[Profile, Symbols] = seqprofile(Seqs)  
seqprofile(Seqs, ...'Alphabet', AlphabetValue, ...)  
seqprofile(Seqs, ...'Counts', CountsValue, ...)  
seqprofile(Seqs, ...'Gaps', GapsValue, ...)  
seqprofile(Seqs, ...'Ambiguous', AmbiguousValue, ...)  
seqprofile(Seqs, ...'Limits', LimitsValue, ...)
```

Arguments

Seqs Set of multiply aligned sequences represented by any of the following:

- Array of strings
- Cell array of strings
- Array of structures containing the field `Sequence`

AlphabetValue String specifying the sequence alphabet. Choices are:

- 'NT' — Nucleotides
- 'AA' — Amino acids (default)
- 'none' — No alphabet

When `Alphabet` is 'none', the symbol list is based on the observed symbols. Each character can be any symbol, except for a hyphen (-) and a period (.), which are reserved for gaps.

CountsValue Controls returning frequency (ratio of counts/total counts) or counts. Choices are `true` (counts) or `false` (frequency). Default is `false`.

<i>GapsValue</i>	String that controls the counting of gaps in a sequence. Choices are: <ul style="list-style-type: none"> • 'all' — Counts all gaps • 'noflanks' — Counts all gaps except those at the flanks of every sequence • 'none' — Default. Counts no gaps.
<i>AmbiguousValue</i>	Controls counting ambiguous symbols. Enter 'Count' to add partial counts to the standard symbols.
<i>LimitsValue</i>	Specifies whether to use part of the sequence. Enter a [1x2] vector with the first position and the last position to include in the profile. Default is [1,SeqLength].

Description

Profile = seqprofile(*Seqs*) returns *Profile*, a matrix of size [20 (or 4) x SequenceLength] with the frequency of amino acids (or nucleotides) for every column in the multiple alignment. The order of the rows is given by

- 4 nucleotides — A C G T/U
- 20 amino acids — A R N D C Q E G H I L K M F P S T W Y V

[*Profile*, *Symbols*] = seqprofile(*Seqs*) returns *Symbols*, a unique symbol list where every symbol in the list corresponds to a row in *Profile*, the profile.

seqprofile(*Seqs*, ...'*PropertyName*', *PropertyValue*, ...) calls seqprofile with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

seqprofile(*Seqs*, ...'*Alphabet*', *AlphabetValue*, ...) selects a nucleotide alphabet, amino acid alphabet, or no alphabet.

seqprofile

`seqprofile(Seqs, ...'Counts', CountsValue, ...)` when `Counts` is true, returns the counts instead of the frequency.

`seqprofile(Seqs, ...'Gaps', GapsValue, ...)` appends a row to the bottom of a profile (`Profile`) with the count for gaps.

`seqprofile(Seqs, ...'Ambiguous', AmbiguousValue, ...)` when `Ambiguous` is 'count', counts the ambiguous amino acid symbols (B Z X) and nucleotide symbols (R Y K M S W B D H V N) with the standard symbols. For example, the amino acid X adds a 1/20 count to every row while the amino acid B counts as 1/2 at the D and N rows.

`seqprofile(Seqs, ...'Limits', LimitsValue, ...)` specifies the start and end positions for the profile relative to the indices of the multiple alignment.

Examples

```
seqs = fastaread('pf00002.fa');  
[P,S] = seqprofile(seqs,'limits',[50 60],'gaps','all')
```

See Also

Bioinformatics Toolbox functions: `fastaread`, `multialignread`, `multialignwrite`, `seqconsensus`, `seqdisp`, `seqlogo`

Purpose Calculate reverse complement of nucleotide sequence

Syntax `SeqRC = seqrcomplement(SeqNT)`

Arguments

<code>SeqNT</code>	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 <code>Sequence</code> .
--------------------	---

Description `seqrcomplement` calculates the reverse complementary strand of a DNA sequence.

`SeqRC = seqrcomplement(SeqNT)` calculates the reverse complementary strand 3' → 5' (A→T, C→G, G→C, T→A) for a DNA sequence and returns a sequence in the same format as `SeqNT`. For example, if `SeqNT` is an integer sequence then so is `SeqRC`.

Examples Reverse a DNA nucleotide sequence and then return its complement.

```
s = 'ATCG'  
seqrcomplement(s)  
  
ans =  
CGAT
```

See Also Bioinformatics Toolbox functions: `codoncount`, `palindromes`, `seqcomplement`, `seqreverse`, `seqtool`

seqreverse

Purpose Reverse letters or numbers in nucleotide sequence

Syntax `SeqR = seqreverse(SeqNT)`

Arguments

SeqNT Enter a nucleotide sequence. Enter either a character string with the characters A, T (U), G, C, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N, or a vector of integers. You can also enter a structure with the field `Sequence`.

SeqR Returns a sequence in the same format as the nucleotide sequence. For example, if *SeqNT* is an integer sequence, then so is *SeqR*.

Description `seqreverse` calculates the reverse strand of a DNA or RNA sequence.

`SeqR = seqreverse(SeqNT)` calculates the reverse strand 3' → 5' of the nucleotide sequence.

Examples

Reverse a nucleotide sequence.

```
s = 'ATCG'  
seqreverse(s)
```

```
ans =  
GCTA
```

See Also

Bioinformatics Toolbox functions: `seqcomplement`, `seqrcomplement`, `seqtool`

MATLAB function: `flip1r`

Purpose

Display open reading frames in sequence

Syntax

```
seqshoworfs(SeqNT)
seqshoworfs(SeqNT, ...'Frames', FramesValue, ...)
seqshoworfs(SeqNT, ...'GeneticCode', GeneticCodeValue, ...)
seqshoworfs(SeqNT, ...'MinimumLength', MinimumLengthValue,
             ...)
seqshoworfs(SeqNT, ...'AlternativeStartCodons',
             AlternativeStartCodonsValue, ...)
seqshoworfs(SeqNT, ...'Color', ColorValue, ...)
seqshoworfs(SeqNT, ...'Columns', ColumnsValue, ...)
```

Arguments

<i>SeqNT</i>	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 <code>Sequence</code> .
<i>FramesValue</i>	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.
<i>GeneticCodeValue</i>	Genetic code name. Enter a code number or a code name from the table Genetic Code on page 2-1038.
<i>MinimumLengthValue</i>	Property to set the minimum number of codons in an ORF.

AlternativeStartCodonsValue Property to control using alternative start codons. Enter either true or false. The default value is false.

ColorValue 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.

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

Genetic Code (Continued)

Code Number	Code Name
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

seqshoworfs identifies and highlights all open reading frames using the standard or an alternative genetic code.

seqshoworfs(*SeqNT*) displays the sequence with all open reading frames highlighted, and it returns a structure of start and stop positions for each ORF in each reading frame. The standard genetic code is used with start codon 'AUG' and stop codons 'UAA', 'UAG', and 'UGA'.

seqshoworfs(*SeqNT*, ... '*PropertyName*', *PropertyValue*, ...) calls seqshoworfs with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

seqshoworfs

`seqshoworfs(SeqNT, ... 'Frames', FramesValue, ...)` specifies the reading frames to display. The default is to display the first, second, and third reading frames with ORFs highlighted in each frame.

`seqshoworfs(SeqNT, ... 'GeneticCode', GeneticCodeValue, ...)` specifies the genetic code to use for finding open reading frames.

`seqshoworfs(SeqNT, ... 'MinimumLength', MinimumLengthValue, ...)` sets the minimum number of codons for an ORF to be considered valid. The default value is 10.

`seqshoworfs(SeqNT, ... 'AlternativeStartCodons', AlternativeStartCodonsValue, ...)` uses alternative start codons if `AlternativeStartCodons` is set to true. For example, in the human mitochondrial genetic code, AUA and AUU are known to be alternative start codons. For more details on alternative start codons, see

<http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=t#SG1>

`seqshoworfs(SeqNT, ... 'Color', ColorValue, ...)` specifies the color used to highlight the open reading frames in the output display. The default color scheme is blue for the first reading frame, red for the second, and green for the third frame.

`seqshoworfs(SeqNT, ... 'Columns', ColumnsValue, ...)` specifies how many columns per line to use in the output. The default value is 64.

Examples

Look for the open reading frames in a random nucleotide sequence.

```
s = randseq(200, 'alphabet', 'dna');
seqshoworfs(s);
```




```

Open Reading Frames
Frame 1
000001
TAGCTTCATCGTTGACTTCTACTAAAAGCAAGCTCCTGAGTAGCTGGCCAAGCGAGCTTGCTTG
000065
TGCCCGGCTGCGGCGGTTGTATCCTGAATACGCCATGCGCCAGTGGACTGCGTAGACCTATTTT
000129
CCAGCTGCGCCTGATGAAGGCGCAACACGAAGGAAAGACGGGACCCAGGGCGACGTCTATTAA
000193  AAGATAAT

Frame 2
000001
TAGCTTCATCGTTGACTTCTACTAAAAGCAAGCTCCTGAGTAGCTGGCCAAGCGAGCTTGCTTG
000065
TGCCCGGCTGCGGCGGTTGTATCCTGAATACGCCATGCGCCAGTGGACTGCGTAGACCTATTTT
000129
CCAGCTGCGCCTGATGAAGGCGCAACACGAAGGAAAGACGGGACCCAGGGCGACGTCTATTAA
000193  AAGATAAT

Frame 3
000001
TAGCTTCATCGTTGACTTCTACTAAAAGCAAGCTCCTGAGTAGCTGGCCAAGCGAGCTTGCTTG
000065
TGCCCGGCTGCGGCGGTTGTATCCTGAATACGCCATGCGCCAGTGGACTGCGTAGACCTATTTT
000129
CCAGCTGCGCCTGATGAAGGCGCAACACGAAGGAAAGACGGGACCCAGGGCGACGTCTATTAA
000193  AAGATAAT

```

Identify the open reading frames in a GenBank sequence.

seqshoworfs

```
HLA_DQB1 = getgenbank('NM_002123');  
seqshoworfs(HLA_DQB1.Sequence);
```

See Also

Bioinformatics Toolbox functions: `codoncount`, `cpgisland`, `geneticcode`, `seqdisp`, `seqshowwords`, `seqtool`, `seqwordcount`

MATLAB function: `regexp`

Purpose

Graphically display words in sequence

Syntax

```
seqshowwords(Seq, Word)
seqshowwords(Seq, Word, ...'Color', ColorValue, ...)
seqshowwords(Seq, Word, ...'Columns', ColumnsValue, ...)
seqshowwords(Seq, Word, ...'Alphabet', AlphabetValue, ...)
```

Arguments

<i>Seq</i>	Enter either a nucleotide or amino acid sequence. You can also enter a structure with the field <code>Sequence</code> .
<i>Word</i>	Enter a short character sequence.
<i>ColorValue</i>	Property to select the color for highlighted characters. Enter a 1-by-3 RGB vector specifying the intensity (0 255) of the red, green, and blue components, or enter a character from the following list: 'b'— blue, 'g'— green, 'r'— red, 'c'— cyan, 'm'— magenta, or 'y'— yellow. The default color is red 'r'.
<i>ColumnsValue</i>	Property to specify the number of characters in a line. Default value is 64.
<i>AlphabetValue</i>	Property to select the alphabet. Enter 'AA' for amino acid sequences or 'NT' for nucleotide sequences. The default is 'NT'.

Description

`seqshowwords(Seq, Word)` displays the sequence with all occurrences of a word highlighted, and returns a structure with the start and stop positions for all occurrences of the word in the sequence.

`seqshowwords(Seq, Word, ...'PropertyName', PropertyValue, ...)` calls `seqshowwords` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must

seqshowwords

be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

`seqshowwords(Seq, Word, ... 'Color', ColorValue, ...)` selects the color used to highlight the words in the output display.

`seqshowwords(Seq, Word, ... 'Columns', ColumnsValue, ...)` specifies how many columns per line to use in the output.

`seqshowwords(Seq, Word, ... 'Alphabet', AlphabetValue, ...)` selects the alphabet for the sequence (*Seq*) and the word (*Word*).

If the search work (*Word*) contains nucleotide or amino acid symbols that represent multiple possible symbols, then `seqshowwords` shows all matches. For example, the symbol R represents either G or A (purines). If *Word* is 'ART', then `seqshowwords` shows occurrences of both 'AAT' and 'AGT'.

Examples

This example shows two matches, 'TAGT' and 'TAAT', for the word 'BART'.

```
seqshowwords('GCTAGTAACGTATATATAAT', 'BART')
```

```
ans =  
  Start: [3 17]  
  Stop: [6 20]
```

```
000001 GCTAGTAACGTATATATAAT
```

`seqshowwords` does not highlight overlapping patterns multiple times. This example highlights two places, the first occurrence of 'TATA' and the 'TATATATA' immediately after 'CG'. The final 'TA' is not highlighted because the preceding 'TA' is part of an already matched pattern.

```
seqshowwords('GCTATAACGTATATATATA', 'TATA')
```

```
ans =  
  Start: [3 10 14]
```

```
Stop: [6 13 17]
```

```
000001 GCTATAACGTATATATATA
```

To highlight all multiple repeats of TA, use the regular expression 'TA(TA)*TA'.

```
seqshowwords('GCTATAACGTATATATATA','TA(TA)*TA')
```

```
ans =
```

```
Start: [3 10]
```

```
Stop: [6 19]
```

```
000001 GCTATAACGTATATATATA
```

See Also

Bioinformatics Toolbox functions: `palindromes`, `cleave`, `restrict`, `seqdisp`, `seqtool`, `seqwordcount`

MATLAB functions: `strfind`, `regexp`

seqtool

Purpose Open tool to interactively explore biological sequences

Syntax
`seqtool(Seq)`
`seqtool(..., 'PropertyName', PropertyValue,...)`
`seqtool(..., 'Alphabet', AlphabetValue)`

Arguments

Seq Struct with a field *Sequence*, a character array, or a file name with an extension of *.gbk*, *.gpt*, *.fasta*, *.fa*, or *.ebi*

Description

`seqtool(Seq)` loads a sequence (*Seq*) into the seqtool GUI.

`seqtool(..., 'PropertyName', PropertyValue,...)` defines optional properties using property name/value pairs.

`seqtool(..., 'Alphabet', AlphabetValue)` specifies an alphabet (*AlphabetValue*) for the sequence (*Seq*). Default is 'AA', except when all of the symbols in the sequence are A, C, G, T, and -, then *AlphabetValue* is set to 'NT'. Use 'AA' when you want to force an amino acid sequence alphabet.

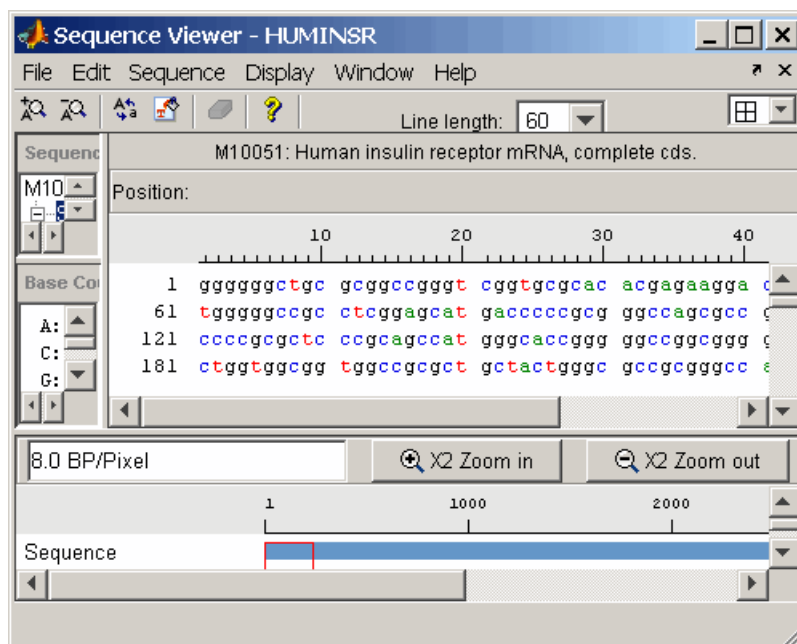
Example

1 Get a sequence from Genbank.

```
S = getgenbank('M10051')
```

2 Open the sequence tool window with the sequence.

```
seqtool(S)
```



See Also

Bioinformatics Toolbox functions: aa2nt, aaccount, aminolookup, basecount, baselookup, dimercount, emblread, fastaread, fastawrite, genbankread, geneticcode, genpeptread, getembl, getgenbank, getgenpept, nt2aa, proteinplot, seqcomplement, seqdisp, seqrcomplement, seqreverse, seqshoworfs, seqshowwords, seqwordcount

seqwordcount

Purpose Count number of occurrences of word in sequence

Syntax seqwordcount(Seq, Word)

Arguments

Seq	Enter a nucleotide or amino acid sequence of characters. You can also enter a structure with the field Sequence.
Word	Enter a short sequence of characters.

Description seqwordcount(Seq, Word) counts the number of times that a word appears in a sequence, and then returns the number of occurrences of that word.

If Word contains nucleotide or amino acid symbols that represent multiple possible symbols (ambiguous characters), then seqwordcount counts all matches. For example, the symbol R represents either G or A (purines). For another example, if word equals 'ART', then seqwordcount counts occurrences of both 'AAT' and 'AGT'.

Examples seqwordcount does not count overlapping patterns multiple times. In the following example, seqwordcount reports three matches. TATATATA is counted as two distinct matches, not three overlapping occurrences.

```
seqwordcount('GCTATAACGTATATATAT', 'TATA')  
  
ans =  
    3
```

The following example reports two matches ('TAGT' and 'TAAT'). B is the ambiguous code for G, T, or C, while R is an ambiguous code for G and A.

```
seqwordcount('GCTAGTAACGTATATATAAT', 'BART')  
  
ans =  
    2
```


See Also

Bioinformatics Toolbox functions: `codoncount`, `seqshoworfs`, `seqshowwords`, `seqtool`, `seq2regexp`

MATLAB function: `strfind`

showalignment

Purpose Display color-coded sequence alignment

Syntax `showalignment(Alignment)`
`showalignment(Alignment, ...'MatchColor',`
`MatchColorValue, ...)`
`showalignment(Alignment,`
`... 'SimilarColor' SimilarColorValue,`
`...)`
`showalignment(Alignment, ...'StartPointers',`
`StartPointersValue, ...)`
`showalignment(Alignment, ...'Columns', ColumnsValue, ...)`

Arguments

Alignment

Either of the following:

- 3-by-N character array showing the pairwise alignment of two sequences, such as returned by the `nwalign` or `swalign` function.
- Vector or character array containing a multiple alignment, such as returned by the `multialign` function.

MatchColorValue

Color to highlight matching characters. Can be either of the following:

- 1-by-3 RGB vector
- String specifying one of the following colors:
 - 'r' — red
 - 'g' — green
 - 'b' — blue
 - 'c' — cyan
 - 'm' — magenta
 - 'y' — yellow

Default is red, [255 0 0] or 'r'.

showalignment

SimilarColorValue Color to highlight similar characters. Can be either of the following:

- 1-by-3 RGB vector
- String specifying one of the following colors:
 - 'r' — red
 - 'g' — green
 - 'b' — blue
 - 'c' — cyan
 - 'm' — magenta
 - 'y' — yellow

Default is magenta, [255 0 255] or 'm'.

StarterPointersValue Two-element vector that specifies the starting indices in the original sequences of a local pairwise alignment, such as the third output returned by the `swalign` function.

ColumnsValue Scalar that specifies the number of characters to display in one row when displaying a pairwise alignment. Default is 64.

Description

`showalignment(Alignment)` displays a color-coded alignment in a MATLAB Figure window. For pairwise sequence alignments, matching and similar characters are highlighted in red and magenta, respectively. For multiple sequence alignments, highly conserved positions are highlighted in red and conserved positions are highlighted in magenta.

`showalignment(Alignment, ...'PropertyName', PropertyValue, ...)` calls `showalignment` with optional properties that use property name/property value pairs. You can specify one or more properties in

any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`showalignment(Alignment, ...'MatchColor', MatchColorValue, ...)` specifies the color to highlight matching characters in the output display. Default is red, [255 0 0] or 'r'. For example, to use cyan, enter [0 255 255] or 'c'.

`showalignment(Alignment, ...'SimilarColor' SimilarColorValue, ...)` specifies the color to highlight similar characters in the output display. Default is magenta, [255 0 255] or 'm'.

Note The 'StartPointers' and 'Columns' properties are available only when displaying pairwise alignments.

`showalignment(Alignment, ...'StartPointers', StartPointersValue, ...)` specifies the starting indices in the original sequences of a local pairwise alignment.

Tip You can use the third output returned by the `swalign` function as the *StartPointersValue*.

`showalignment(Alignment, ...'Columns', ColumnsValue, ...)` specifies the number of characters to display in one row when displaying a pairwise alignment, and labels the start of each row with the sequence positions. Default is 64.

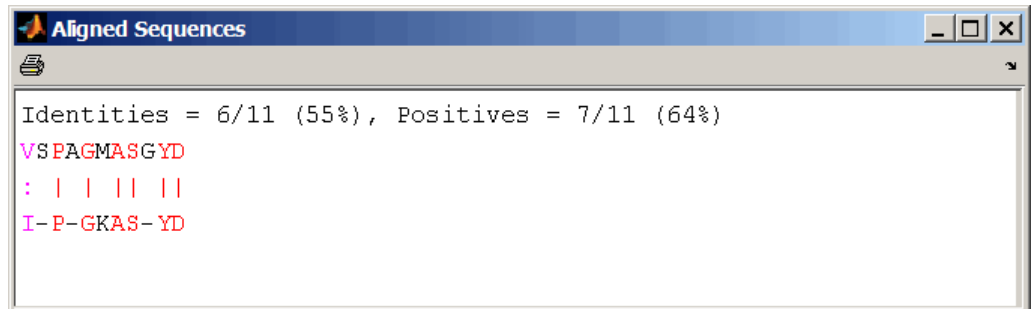
Tip To view a multiple-sequence alignment and interact with it, use the `multialignviewer` function.

showalignment

Examples

Globally align two amino acid sequences and display their color-coded alignment.

```
[Score, Alignment] = nwalignment('VSPAGMASGYD','IPGKASYD');  
showalignment(Alignment);
```



```
Aligned Sequences  
Identities = 6/11 (55%), Positives = 7/11 (64%)  
VSPAGMASGYD  
: | | | |  
I-P-GKAS-YD
```

Read a multiple-sequence alignment file and display the color-coded alignment.

```
gag = multialignmentread('aagag.aln');  
showalignment(gag)
```

```

Aligned Sequences
HIV-2      MGAR-NSVLRGKKADELERIRLRPGGKKKYRLKHIVWAAANKLDRFGLAESLLES
HIV2-MCN13 MGAR-NSVLKGGKKADELETIRLRPGGKKKYRLKHIVWAAANELDRFGLAESLLES
SIVMM251   MGAR-NSVLSGKKADELEKIRLRPGGKKKYMLKHVVWAAANELDRFGLAESLLEI
SIVMM239   MGVR-NSVLSGKKADELEKIRLRPNGKKKYMLKHVVWAAANELDRFGLAESLLEI
HIV-2 UC1  MGAR-SSVLSGKKTDELEKVRRLRPGGKKRYCLKHIWAVNELDRFGLAESLLES
SIVsmSL92b MGAR-GSVLSGKKADELEKVRRLRPGGRKKYMLKHIWAARELDRFGSAESLLES
SIVAGM677A MGGG-HSALSGRSLDTFEKIRLRPNGKKKYQIKHLIWAGKEMERFGLHEKLLLE
SIVAGM3    MGAA-TSALNRRQLDKFEHIRLRPTGKKKYQIKHLIWAGKEMERFGLHERLLES
SIVmnd5440 MGAS-ASGLRGEKLDLEKIRLRPSGKKKYQLKHVIWVSKELDRFGLHEKLLLES
HIV-1      MGAR-ASVLSGGKLDKWEKIRLRPGGKKKYRLKHIVWASRELERYALNPGLLET
HIV1-NDK   MGAR-ASVLSGGKLDTWERIRLRPGGKKKYALKHLIWASRELERFTLNPGLLET
SIVcpz     MGAR-ASVLTGGRLDAWEKIRLRPGGKKKYMMKHLVWASRELDRFACNPGLMET
CIVcpzUS   MGAR-ASVLTGGRLDAWEKIRLRPGGKKKYMMKHLVWASRELERFACNPGLMET
SIVcpzTAN1 MGAR-ASVLRGDKLDTWESIRLKSRRGKKYLIKHLVWAGSELQRFAMNPGLMET
SIVmon     MGARHSAMLSGTKLDKYEKVRLRPRGKKKYLIKHLVWAAKELDRFGLSDSLLLE
SIVlhoest  MGSQ-NSVLSRQIEKDFCSVRLRPGSKKTYQKRHVEWATKELDRFGLGSQLLLE

```

Tip To view a multiple-sequence alignment and interact with it, use the `multialignviewer` function.

See Also

Bioinformatics Toolbox functions: `multialign`, `multialignviewer`, `nwalgn`, `swalign`

showhmmprof

Purpose Plot hidden Markov model (HMM) profile

Syntax

```
showhmmprof(Model)  
showhmmprof(Model, ... 'Scale', ScaleValue, ...)  
showhmmprof(Model, ... 'Order', OrderValue, ...)
```

Arguments

<i>Model</i>	Hidden Markov model created by the function gethmmprof or pfamhmmread.
<i>ScaleValue</i>	Property to select a probability scale. Enter one of the following values: <ul style="list-style-type: none">• 'logprob' — Log probabilities• 'prob' — Probabilities• 'logodds' — Log-odd ratios
<i>OrderValue</i>	Property to specify the order of the amino acid alphabet. Enter a character string with the 20 standard amino acids characters A R N D C Q E G H I L K M F P S T W Y V. The ambiguous characters B Z X are not allowed.

Description showhmmprof(*Model*) plots a profile hidden Markov model described by the structure *Model*.

showhmmprof(..., '*PropertyName*', *PropertyValue*, ...) calls showhmmprof with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

showhmmprof(*Model*, ... 'Scale', *ScaleValue*, ...) specifies the scale to use. If log probabilities (*ScaleValue*= 'logprob'), probabilities (*ScaleValue*= 'prob'), or log-odd ratios (*ScaleValue*= 'logodds'). To compute the log-odd ratios, the null model probabilities are used for symbol emission and equally distributed transitions are used for the null transition probabilities. The default *ScaleValue* is 'logprob'.

`showhmmprof(Model, ...'Order', OrderValue, ...)` specifies the order in which the symbols are arranged along the vertical axis. This option allows you to reorder the alphabet and group the symbols according to their properties.

Examples

- 1 Load a model example.

```
model = pfamhmmread('pf00002.ls')
```

- 2 Plot the profile.

```
showhmmprof(model, 'Scale', 'logodds')
```

- 3 Order the alphabet by hydrophobicity.

```
hydrophobic = 'IVLFCMAGTSWYPHNDQEKR'
```

- 4 Plot the profile.

```
showhmmprof(model, 'Order', hydrophobic)
```

See Also

Bioinformatics Toolbox functions: `gethmmprof`, `hmmprofalign`, `hmmprofestimate`, `hmmprofgenerate`, `hmmprofstruct`, `pfamhmmread`

sptread

Purpose Read data from SPOT file

Syntax `SPOTData = sptread(File)`
`SPOTData = sptread(File, 'CleanColNames', CleanColNamesValue)`

Arguments

<i>File</i>	Either of the following: <ul style="list-style-type: none">• String specifying a file name, a path and file name, or a URL pointing to a file. The referenced file is a SPOT-formatted file (ASCII text file). If you specify only a file name, that file must be on the MATLAB search path or in the MATLAB Current Directory.• MATLAB character array that contains the text of a SPOT-formatted file.
<i>CleanColNamesValue</i>	Controls the use of valid MATLAB variable names.

Description `SPOTData = sptread(File)` reads *File*, a SPOT-formatted file, and creates *SPOTData*, a MATLAB structure containing the following fields:

Header
Data
Blocks
Columns
Rows
IDs
ColumnNames
Indices
Shape

`SPOTData = sptread(File, 'CleanColNames', CleanColNamesValue)` controls the use of valid MATLAB variable names. The column names in the SPOT-formatted file contain periods and some characters that cannot be used in MATLAB variable names. If you plan to use the column names as variable names in a function, use this option with `CleanColNames` set to `true` and the function will return the field `ColumnNames` with valid variable names.

The `Indices` field of the structure includes the indices that you can use for plotting heat maps of the data.

Examples

- 1 Read in a sample SPOT file and plot the median foreground intensity for the 635 nm channel. Note that the example file `spotdata.txt` is not provided with the Bioinformatics Toolbox software.

```
spotStruct = sptread('spotdata.txt')
mimage(spotStruct, 'Rmedian');
```

- 2 Alternately, create a similar plot using more basic graphics commands.

```
Rmedian = magetfield(spotStruct, 'Rmedian');
imagesc(Rmedian(spotStruct.Indices));
colormap bone
colorbar
```

See Also

Bioinformatics Toolbox functions: `affyread`, `agferead`, `celintensityread`, `geoseriesread`, `geosoftread`, `gprread`, `ilmnbsread`, `imageneread`, `maboxplot`, `magetfield`

svmclassify

Purpose Classify data using support vector machine

Syntax `Group = svmclassify(SVMStruct, Sample)`
`Group = svmclassify(SVMStruct, Sample, 'Showplot', ShowplotValue)`

Description `Group = svmclassify(SVMStruct, Sample)` classifies each row of the data in `Sample` using the information in a support vector machine classifier structure `SVMStruct`, created using the `svmtrain` function. `Sample` must have the same number of columns as the data used to train the classifier in `svmtrain`. `Group` indicates the group to which each row of `Sample` has been assigned.

`Group = svmclassify(SVMStruct, Sample, 'Showplot', ShowplotValue)` controls the plotting of the sample data in the figure created using the `Showplot` property with the `svmtrain` function.

Examples

- 1 Load the sample data, which includes Fisher's iris data of 5 measurements on a sample of 150 irises.

```
load fisheriris
```

- 2 Create data, a two-column matrix containing sepal length and sepal width measurements for 150 irises.

```
data = [meas(:,1), meas(:,2)];
```

- 3 From the species vector, create a new column vector, `groups`, to classify data into two groups: Setosa and non-Setosa.

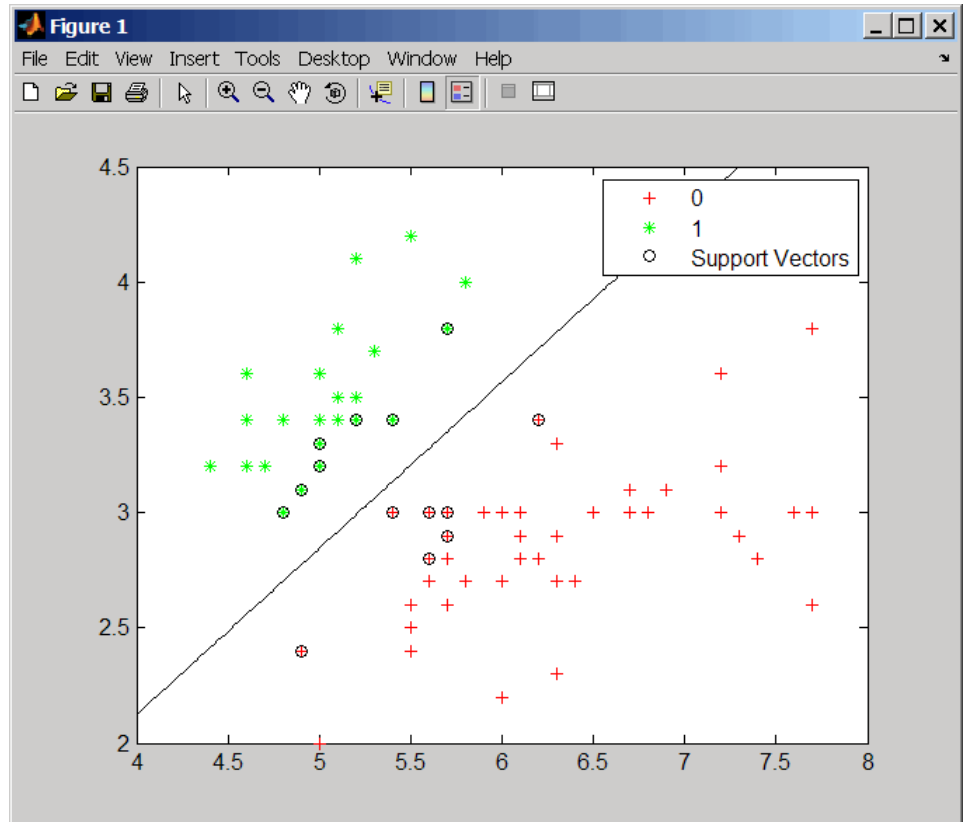
```
groups = ismember(species, 'setosa');
```

- 4 Randomly select training and test sets.

```
[train, test] = crossvalind('holdOut', groups);  
cp = classperf(groups);
```

- 5 Use the `svmtrain` function to train an SVM classifier using a linear kernel function and plot the grouped data.

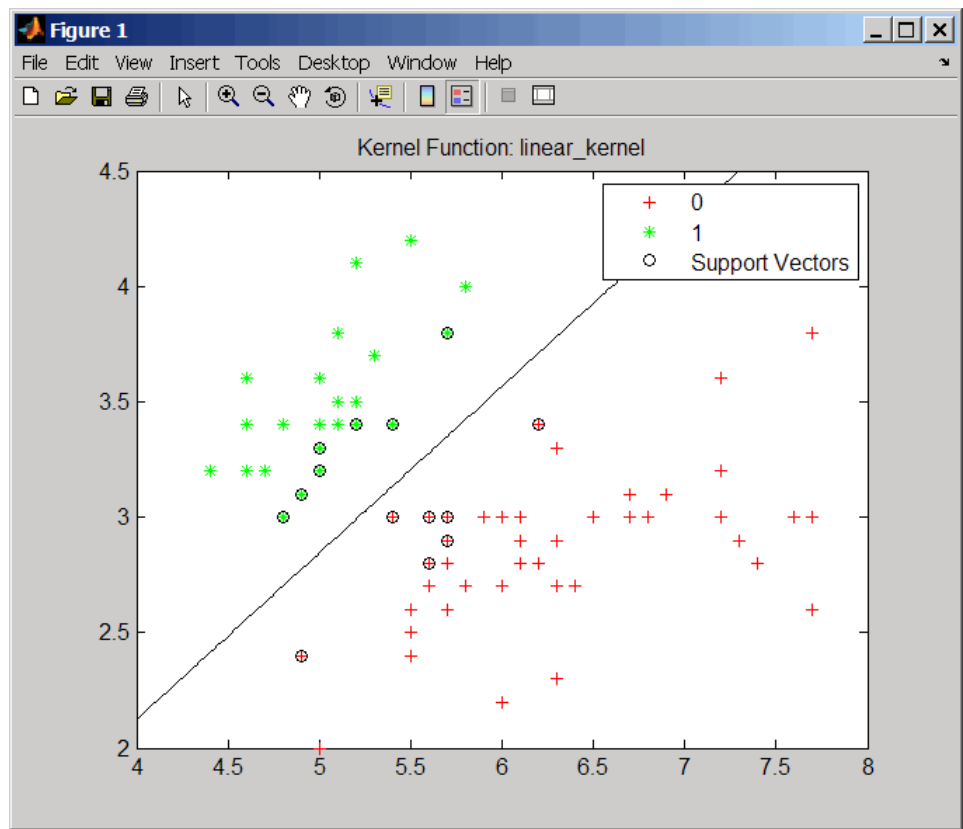
```
svmStruct = svmtrain(data(train,:),groups(train),'showplot',true);
```



- 6 Add a title to the plot, using the `KernelFunction` field from the `svmStruct` structure as the title.

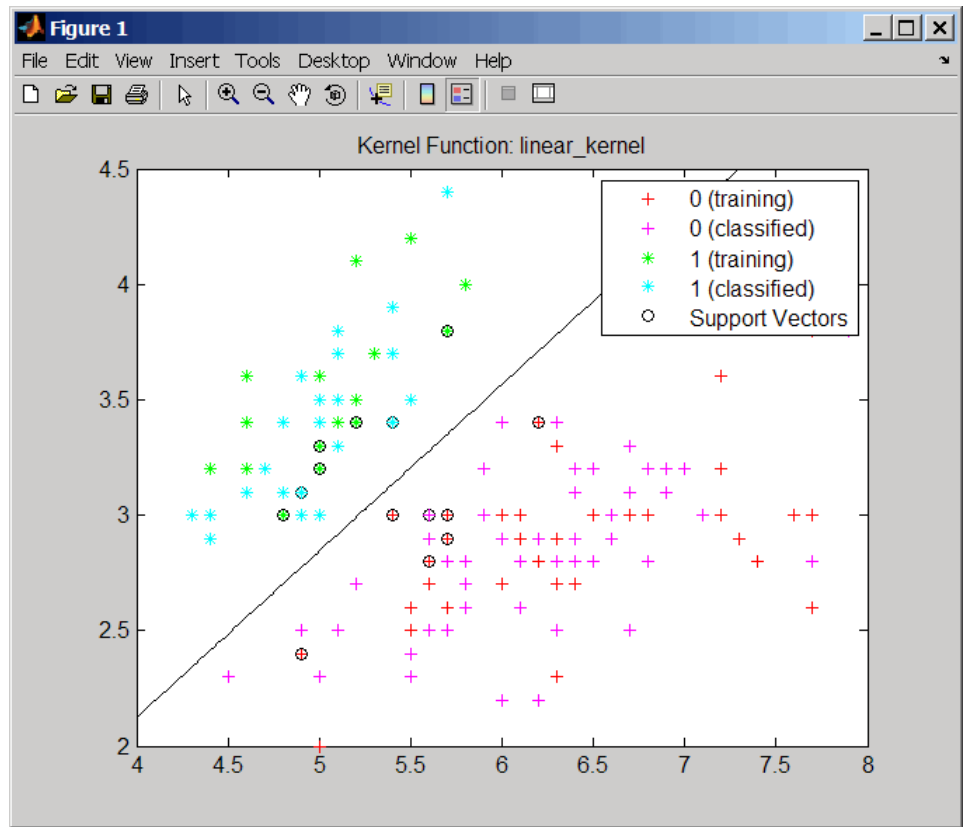
```
title(sprintf('Kernel Function: %s',...
             func2str(svmStruct.KernelFunction)),...
       'interpreter','none');
```

svmclassify



7 Classify the test set using a support vector machine.

```
classes = svmclassify(svmStruct,data(test,:), 'showplot', true);
```



8 Evaluate the performance of the classifier.

```
classperf(cp, classes, test);
cp.CorrectRate
```

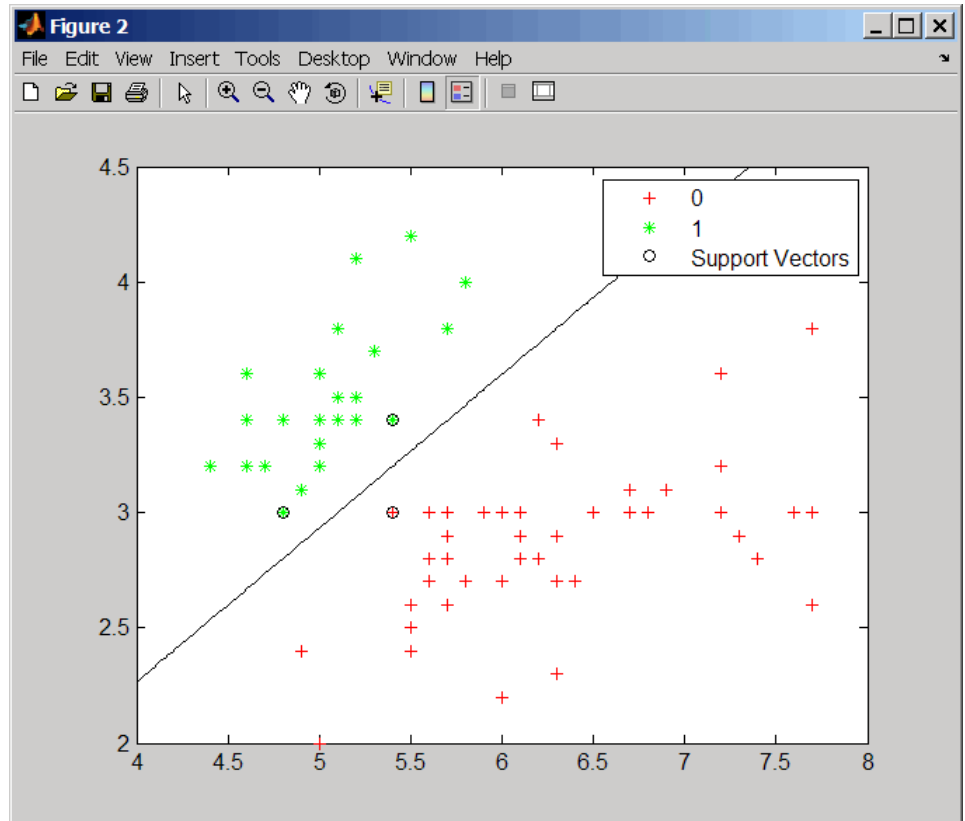
```
ans =
```

```
0.9867
```

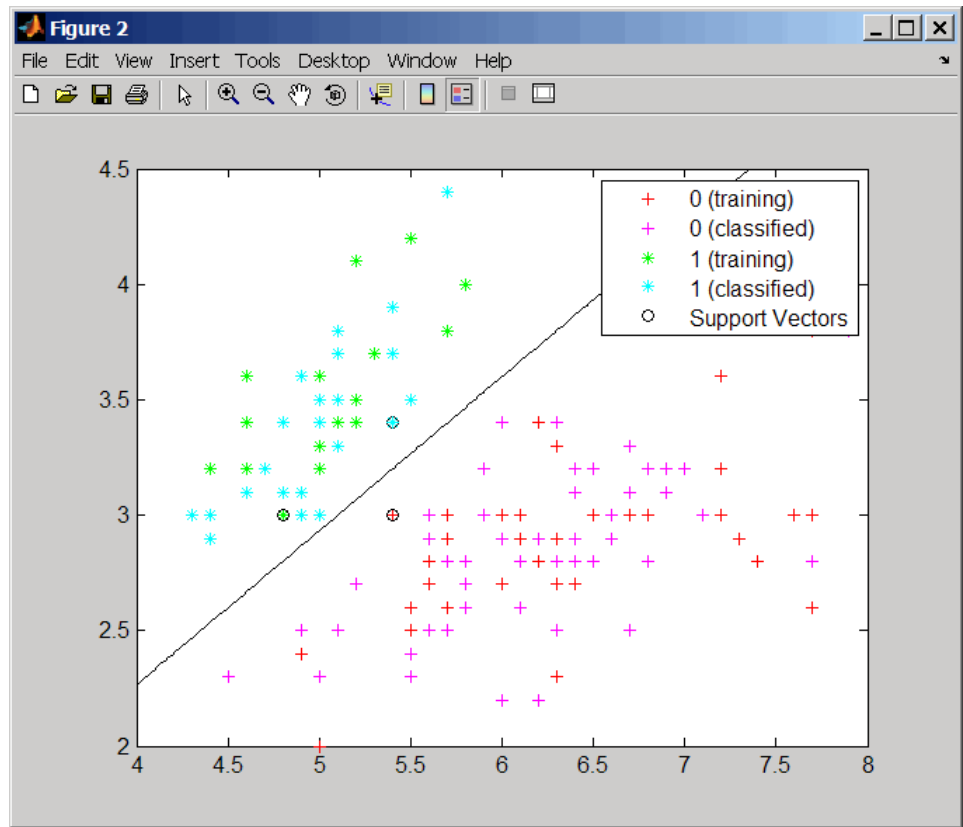
9 Use a one-norm, hard margin support vector machine classifier by changing the boxconstraint property.

svmclassify

```
figure  
svmStruct = svmtrain(data(train,:),groups(train),...  
                    'showplot',true,'boxconstraint',1e6);
```



```
classes = svmclassify(svmStruct,data(test,:), 'showplot',true);
```

10 Evaluate the performance of the classifier.

```
classperf(cp,classes,test);  
cp.CorrectRate
```

```
ans =
```

```
0.9867
```

References

[1] Kecman, V., Learning and Soft Computing, MIT Press, Cambridge, MA. 2001.

[2] Suykens, J.A.K., Van Gestel, T., De Brabanter, J., De Moor, B., and Vandewalle, J., Least Squares Support Vector Machines, World Scientific, Singapore, 2002.

[3] Scholkopf, B., and Smola, A.J., Learning with Kernels, MIT Press, Cambridge, MA. 2002.

[4] Cristianini, N., and Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, First Edition (Cambridge: Cambridge University Press). <http://www.support-vector.net/>

See Also

Bioinformatics Toolbox functions: `classperf`, `crossvalind`, `knnclassify`, `svmtrain`

Statistics Toolbox function: `classify`

Optimization Toolbox™ function: `quadprog`

Purpose Create or edit Sequential Minimal Optimization (SMO) options structure

Syntax

```
SMO_OptsStruct = svmsmoset('Property1Name', Property1Value,
    'Property2Name', Property2Value, ...)
SMO_OptsStruct = svmsmoset(OldOpts, 'Property1Name',
    Property1Value, 'Property2Name', Property2Value, ...)
SMO_OptsStruct = svmsmoset(OldOpts, NewOpts)
```

Arguments

<i>OldOpts</i>	Structure that specifies options used by the SMO method of the svmtrain function.
<i>NewOpts</i>	Structure that specifies options used by the SMO method of the svmtrain function.

<i>PropertyName</i>	Description of PropertyValue
TolKKT	Value that specifies the tolerance with which the KKT conditions are checked. KKT conditions are Karush-Kuhn-Tucker conditions. Default is 1.0000e-003.
MaxIter	Integer that specifies the maximum number of iterations of the main loop. If this limit is exceeded before the algorithm converges, then the algorithm stops and returns an error. Default is 15000.

<i>PropertyName</i>	<i>Description of PropertyValue</i>
Display	<p>String that specifies the level of information about the optimization iterations that is displayed as the algorithm runs. Choices are:</p> <ul style="list-style-type: none"> • <code>off</code> — Default. Reports nothing. • <code>iter</code> — Reports every 500 iterations. • <code>final</code> — Reports only when the algorithm finishes.
KKTViolationLevel	<p>Value that specifies the fraction of variables allowed to violate the KKT conditions. Choices are any value ≥ 0 and < 1. Default is 0. For example, if you set <code>KKTViolationLevel</code> to 0.05, then 5% of the variables are allowed to violate the KKT conditions.</p> <hr/> <p>Tip Set this option to a positive value to help the algorithm converge if it is fluctuating near a good solution.</p> <hr/> <p>For more information on KKT conditions, see Cristianini, et al. 2000.</p>
KernelCacheLimit	<p>Value that specifies the size of the kernel matrix cache. The algorithm keeps a matrix with up to <code>KernelCacheLimit</code> \times <code>KernelCacheLimit</code> double-precision, floating-point numbers in memory. Default is 5000.</p>

Return Values

SMO_OptsStruct Structure that specifies options used by the `SMO` method used by the `svmtrain` function.

Description

SMO_OptsStruct = svmsmoset('Property1Name', Property1Value, 'Property2Name', Property2Value, ...) creates *SMO_OptsStruct*, an SMO options structure from the specified inputs. This structure can be used as input for the svmtrain function.

SMO_OptsStruct = svmsmoset(OldOpts, 'Property1Name', Property1Value, 'Property2Name', Property2Value, ...) alters the options in *OldOpts*, an existing SMO options structure, with the specified inputs, creating a new output options structure.

SMO_OptsStruct = svmsmoset(OldOpts, NewOpts) alters the options in *OldOpts*, an existing SMO options structure, with the options specified in *NewOpts*, another SMO options structure, creating a new output options structure.

Examples

- 1 Create an SMO options structure and specify the Display, MaxIter, and KernelCacheLimit properties.

```
opts = svmsmoset('Display','final','MaxIter',20000,...
                'KernelCacheLimit',1000)
```

```
opts =
```

```

                Display: 'final'
                TolKKT: 1.0000e-003
                MaxIter: 20000
KKTViolationLevel: 0
                KernelCacheLimit: 1000
```

- 2 Create an alternate SMO options structure from the previous structure. Specify different Display and KKTViolationLevel properties.

```
alt_opts = svmsmoset(opts,'Display','iter','KKTViolationLevel',.05)
```

```
alt_opts =
```

```
                Display: 'iter'
```

TolKKT: 1.0000e-003
MaxIter: 20000
KKTViolationLevel: 0.0500
KernelCacheLimit: 1000

References

- [1] Cristianini, N., and Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, First Edition (Cambridge: Cambridge University Press). <http://www.support-vector.net/>
- [2] Platt, J.C. (1999). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. In Advances in Kernel Methods - Support Vector Learning, B. Scholkopf, J.C. Burges, and A.J. Smola, eds. (Cambridge MA: MIT Press), pp. 185–208.
- [3] Fan, R.E., Chen, P.H., and Lin, C.J. (2005). Working Set Selection Using Second Order Information for Training SVM. *Journal of Machine Learning Research* 6, 1889–1918.
- [4] Bottou, L. and Lin, C.J. (2006). Support Vector Machine Solvers. <http://www.csie.ntu.edu.tw/~cjlin/papers.html>

See Also

Bioinformatics Toolbox functions: `svmclassify`, `svmtrain`
Optimization Toolbox function: `optimset`

Purpose

Train support vector machine classifier

Syntax

```
SVMStruct = svmtrain(Training, Group)
SVMStruct = svmtrain(..., 'Kernel_Function',
Kernel_FunctionValue, ...)
SVMStruct = svmtrain(..., 'RBF_Sigma', RBFSigmaValue, ...)
SVMStruct = svmtrain(..., 'Polyorder', PolyorderValue, ...)
SVMStruct = svmtrain(..., 'Mlp_Params',
Mlp_ParamsValue, ...)
SVMStruct = svmtrain(..., 'Method', MethodValue, ...)
SVMStruct = svmtrain(..., 'QuadProg_Opts',
QuadProg_OptsValue, ...)
SVMStruct = svmtrain(..., 'SMO_Opts', SMO_OptsValue, ...)
SVMStruct = svmtrain(..., 'BoxConstraint',
BoxConstraintValue, ...)
SVMStruct = svmtrain(..., 'Autoscale', AutoscaleValue, ...)
SVMStruct = svmtrain(..., 'Showplot', ShowplotValue, ...)
```

Arguments

Training

Matrix of training data, where each row corresponds to an observation or replicate, and each column corresponds to a feature or variable.

Group

Column vector, character array, or cell array of strings for classifying data in *Training* into two groups. It has the same number of elements as there are rows in *Training*. Each element specifies the group to which the corresponding row in *Training* belongs.

<i>Kernel_FunctionValue</i>	String or function handle specifying the kernel function that maps the training data into kernel space. Choices are: <ul style="list-style-type: none">• <code>linear</code> — Default. Linear kernel or dot product.• <code>quadratic</code> — Quadratic kernel.• <code>rbf</code> — Gaussian Radial Basis Function kernel with a default scaling factor, <code>sigma</code>, of 1.• <code>polynomial</code> — Polynomial kernel with a default order of 3.• <code>mlp</code> — Multilayer Perceptron kernel with default scale and bias parameters of [1, -1].• <code>@functionname</code> — Handle to a kernel function specified using <code>@</code> and the <code>functionname</code>. For example, <code>@kfun</code>, or an anonymous function.
<i>RBFSigmaValue</i>	Positive number that specifies the scaling factor, <code>sigma</code> , in the radial basis function kernel. Default is 1.
<i>PolyorderValue</i>	Positive number that specifies the order of a polynomial kernel. Default is 3.
<i>Mlp_ParamsValue</i>	Two-element vector, [<code>p1</code> , <code>p2</code>], that specifies the scale and bias parameters of the multilayer perceptron (<code>mlp</code>) kernel. $K = \tanh(p1*U*V' + p2)$. <code>p1</code> must be > 0 , and <code>p2</code> must be < 0 . Default is [1, -1].

MethodValue

String specifying the method to find the separating hyperplane. Choices are:

- **QP** — Quadratic Programming (requires the Optimization Toolbox software). The classifier is a two-norm, soft-margin support vector machine.
- **SMO** — Sequential Minimal Optimization. The classifier is a one-norm, soft-margin support vector machine.
- **LS** — Least-Squares.

If you installed the Optimization Toolbox software, the QP method is the default. Otherwise, the SMO method is the default.

QuadProg_OptsValue

An options structure created by the `optimset` function (Optimization Toolbox software). This structure specifies options used by the **QP** method. For more information on creating this structure, see the `optimset` and `quadprog` reference pages.

SMO_OptsValue

An options structure created by the `svmsmoset` function. This structure specifies options used by the **SMO** method. For more information on creating this structure, see the `svmsmoset` function.

<i>BoxConstraintValue</i>	<p>Box constraints for the soft margin. Choices are:</p> <ul style="list-style-type: none">• Strictly positive numeric scalar.• Array of strictly positive values with the number of elements equal to the number of rows in the <i>Training</i> matrix. <p>If <i>BoxConstraintValue</i> is a scalar, it is automatically rescaled by $N/(2*N1)$ for the data points of group one and by $N/(2*N2)$ for the data points of group two. $N1$ is the number of elements in group one, $N2$ is the number of elements in group two, and $N = N1 + N2$. This rescaling is done to take into account unbalanced groups, that is cases where $N1$ and $N2$ have very different values.</p> <p>If <i>BoxConstraintValue</i> is an array, then each array element is taken as a box constraint for the data point with the same index.</p> <p>Default is a scalar value of 1.</p>
<i>AutoscaleValue</i>	<p>Controls the shifting and scaling of data points before training. When <i>AutoscaleValue</i> is true, the columns of the input data matrix <i>Training</i> are shifted to zero mean and scaled to unit variance.</p> <p>Default is true.</p>
<i>ShowplotValue</i>	<p>Controls the display of a plot of the grouped data, including the separating line for the classifier, when using two-dimensional data. Choices are true or false (default).</p>

Return Values*SVMStruct*

Structure containing information about the trained SVM classifier, including the following fields:

- SupportVectors
- Alpha
- Bias
- KernelFunction
- KernelFunctionArgs
- GroupNames
- SupportVectorIndices
- ScaleData
- FigureHandles

Tip You can use *SVMStruct* as input to the `svmclassify` function, to use for classification.

Description

SVMStruct = `svmtrain(Training, Group)` trains a support vector machine (SVM) classifier using *Training*, a matrix of training data taken from two groups, specified by *Group*. `svmtrain` treats NaNs or empty strings in *Group* as missing values and ignores the corresponding rows of *Training*. Information about the trained SVM classifier is returned in *SVMStruct*, a structure with the following fields.

- SupportVectors
- Alpha
- Bias
- KernelFunction

- KernelFunctionArgs
- GroupNames
- SupportVectorIndices
- ScaleData
- FigureHandles

SVMStruct = svmtrain(*Training*, *Group*, ...'*PropertyName*', *PropertyValue*, ...) calls svmtrain with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

SVMStruct = svmtrain(..., 'Kernel_Function', *Kernel_FunctionValue*, ...) specifies the kernel function (*Kernel_FunctionValue*) that maps the training data into kernel space. *Kernel_FunctionValue* can be one of the following strings or a function handle:

- linear — Default. Linear kernel or dot product.
- quadratic — Quadratic kernel.
- rbf — Gaussian Radial Basis Function kernel with a default scaling factor, sigma, of 1.
- polynomial — Polynomial kernel with a default order of 3.
- mlp — Multilayer Perceptron kernel with default scale and bias parameters of [1, -1].
- @functionname — Handle to a kernel function specified using @and the functionname. For example, @kfun, or an anonymous function.

A kernel function must be of the following form:

```
function K = kfun(U, V)
```

Input arguments U and V are matrices with m and n rows respectively. Return value K is an m -by- n matrix. If $kfun$ is parameterized, you can use anonymous functions to capture the problem-dependent parameters. For example, suppose that your kernel function is:

```
function K = kfun(U,V,P1,P2)
K = tanh(P1*(U*V')+P2);
```

You can set values for $P1$ and $P2$ and then use an anonymous function as follows:

```
@(U,V) kfun(U,V,P1,P2)
```

For more information on the types of functions that can be used as kernel functions, see Cristianini and Shawe-Taylor, 2000.

`SVMStruct = svmtrain(..., 'RBF_Sigma', RBFSigmaValue, ...)` specifies the scaling factor, σ , in the radial basis function kernel. *RBFSigmaValue* must be a positive number. Default is 1.

`SVMStruct = svmtrain(..., 'Polyorder', PolyorderValue, ...)` specifies the order of a polynomial kernel. *PolyorderValue* must be a positive number. Default is 3.

`SVMStruct = svmtrain(..., 'Mlp_Params', Mlp_ParamsValue, ...)` specifies the scale and bias parameters of the multilayer perceptron (mlp) kernel as a two-element vector, $[p1, p2]$. $K = \tanh(p1*U*V' + p2)$, $p1 > 0$, and $p2 < 0$. $p1$ must be > 0 , and $p2$ must be < 0 . Default is $[1, -1]$.

`SVMStruct = svmtrain(..., 'Method', MethodValue, ...)` specifies the method to find the separating hyperplane. Choices are:

- QP — Quadratic Programming (requires the Optimization Toolbox software). The classifier is a two-norm, soft-margin support vector machine.
- SMO — Sequential Minimal Optimization. The classifier is a one-norm, soft-margin support vector machine.
- LS — Least-Squares.

If you installed the Optimization Toolbox software, the QP method is the default. Otherwise, the SMO method is the default.

Note If you specify the QP method, the classifier is a two-norm, soft-margin support vector machine.

SVMStruct = svmtrain(..., 'QuadProg_Opts', *QuadProg_OptsValue*, ...) specifies an options structure created by the `optimset` function (Optimization Toolbox software). This structure specifies options used by the QP method. For more information on creating this structure, see the `optimset` and `quadprog` functions.

SVMStruct = svmtrain(..., 'SMO_Opts', *SMO_OptsValue*, ...) specifies an options structure created by `svmsmoset` function. This structure specifies options used by the SMO method. For more information on creating this structure, see the `svmsmoset` function.

SVMStruct = svmtrain(..., 'BoxConstraint', *BoxConstraintValue*, ...) specifies box constraints for the soft margin. *BoxConstraintValue* can be either of the following:

- Strictly positive numeric scalar
- Array of strictly positive values with the number of elements equal to the number of rows in the *Training* matrix

If *BoxConstraintValue* is a scalar, it is automatically rescaled by $N/(2*N1)$ for the data points of group one and by $N/(2*N2)$ for the data points of group two. $N1$ is the number of elements in group one, $N2$ is the number of elements in group two, and $N = N1 + N2$. This rescaling is done to take into account unbalanced groups, that is cases where $N1$ and $N2$ have very different values.

If *BoxConstraintValue* is an array, then each array element is taken as a box constraint for the data point with the same index.

Default is a scalar value of 1.

`SVMStruct = svmtrain(..., 'Autoscale', AutoscaleValue, ...)` controls the shifting and scaling of data points before training. When `AutoscaleValue` is true, the columns of the input data matrix `Training` are shifted to zero mean and scaled to unit variance. Default is true.

`SVMStruct = svmtrain(..., 'Showplot', ShowplotValue, ...)`, controls the display of a plot of the grouped data, including the separating line for the classifier, when using two-dimensional data. Choices are true or false (default).

Memory Usage and Out of Memory Error

When you set 'Method' to 'QP', the `svmtrain` function operates on a data set containing N elements, it creates an $(N+1)$ -by- $(N+1)$ matrix to find the separating hyperplane. This matrix needs at least $8 * (n+1)^2$ bytes of contiguous memory. If this size of contiguous memory is not available, the software displays an “out of memory” message.

When you set 'Method' to 'SMO', memory consumption is controlled by the SMO option `KernelCacheLimit`. For more information on the `KernelCacheLimit` option, see the `svmsmoset` function. The SMO algorithm stores only a submatrix of the kernel matrix, limited by the size specified by the `KernelCacheLimit` option. However, if the number of data points exceeds the size specified by the `KernelCacheLimit` option, the SMO algorithm slows down because it has to recalculate the kernel matrix elements.

When using `svmtrain` on large data sets, and you run out of memory or the optimization step is very time consuming, try either of the following:

- Use a smaller number of samples and use cross validation to test the performance of the classifier.
- Set 'Method' to 'SMO', and set the `KernelCacheLimit` option as large as your system permits. For information on setting the `KernelCacheLimit` option, see the `svmsmoset` function.

Tip If you set 'Method' to 'SMO', setting the 'BoxConstraint' property as small as possible will help the SMO algorithm run faster.

Examples

- 1 Load the sample data, which includes Fisher's iris data of 5 measurements on a sample of 150 irises.

```
load fisheriris
```

- 2 Create `data`, a two-column matrix containing sepal length and sepal width measurements for 150 irises.

```
data = [meas(:,1), meas(:,2)];
```

- 3 From the `species` vector, create a new column vector, `groups`, to classify data into two groups: Setosa and non-Setosa.

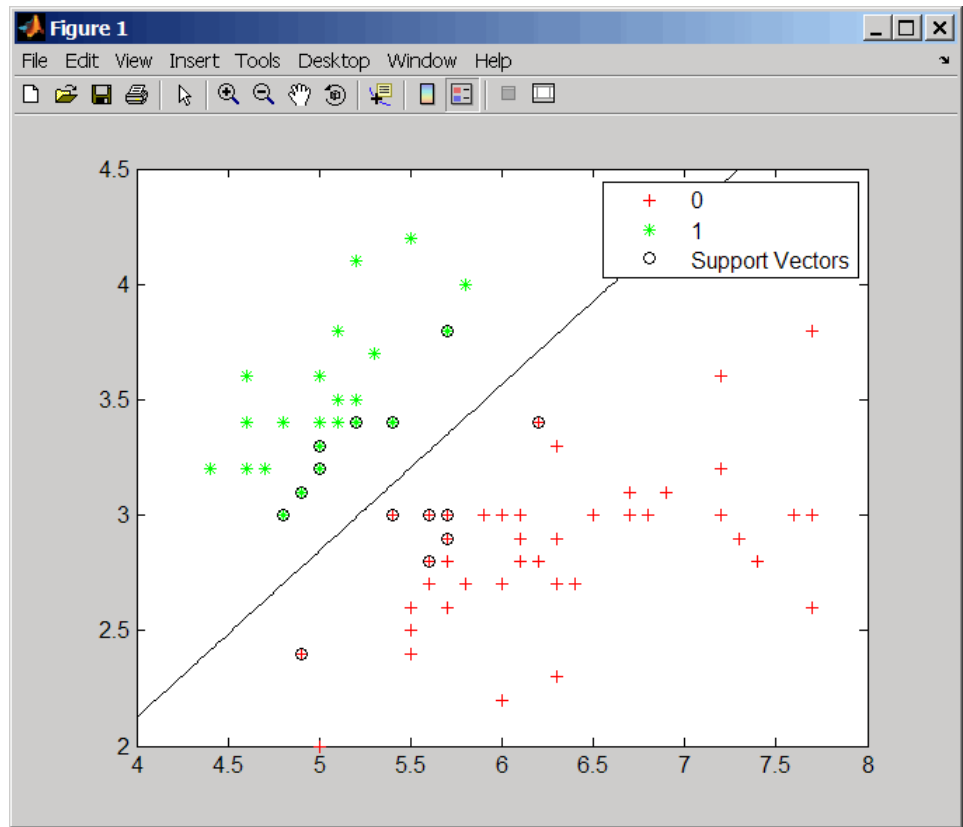
```
groups = ismember(species, 'setosa');
```

- 4 Randomly select training and test sets.

```
[train, test] = crossvalind('holdOut', groups);  
cp = classperf(groups);
```

- 5 Train an SVM classifier using a linear kernel function and plot the grouped data.

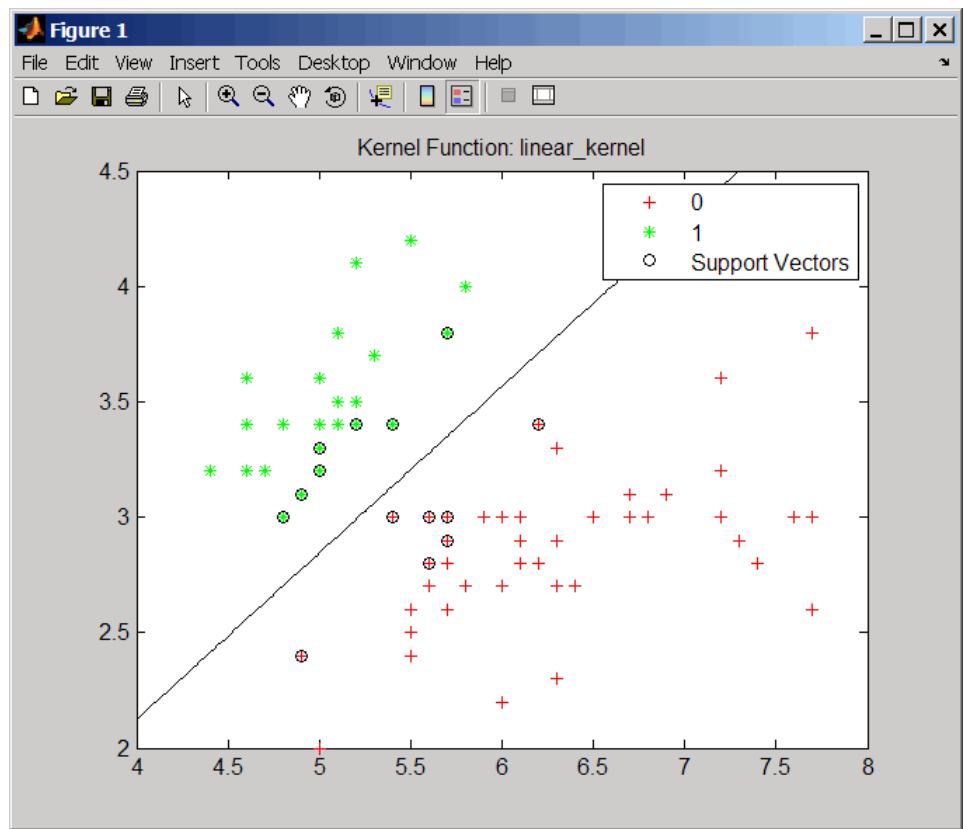
```
svmStruct = svmtrain(data(train,:), groups(train), 'showplot', true);
```

- 6 Add a title to the plot, using the KernelFunction field from the svmStruct structure as the title.

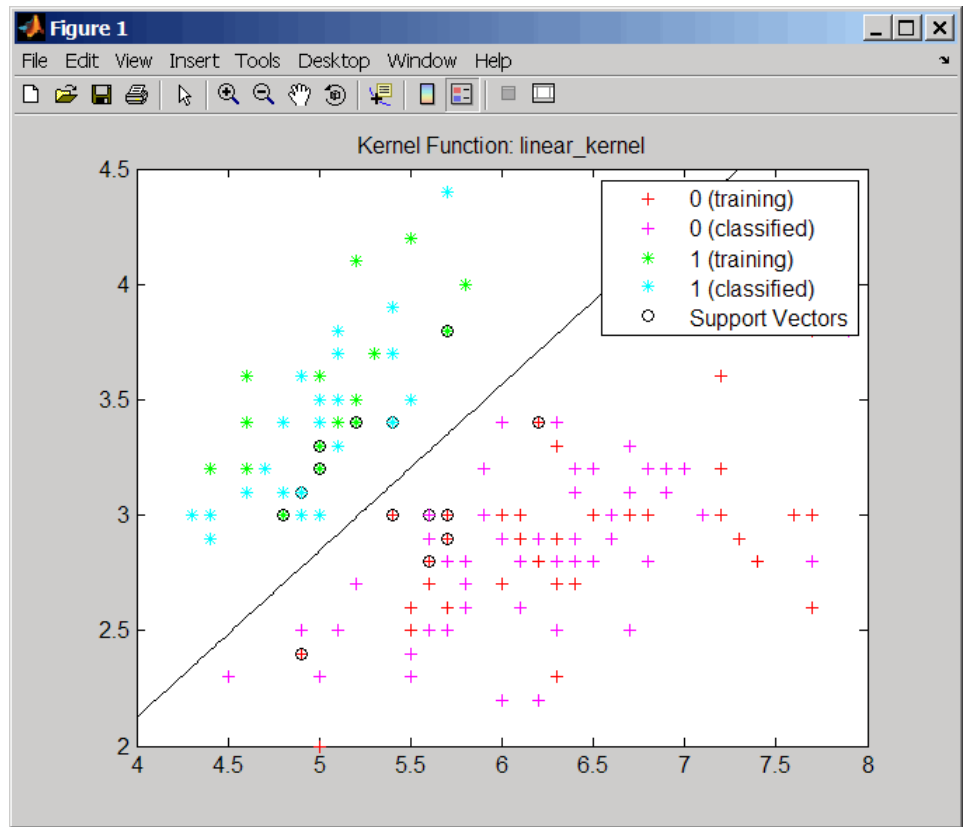
```
title(sprintf('Kernel Function: %s',...
             func2str(svmStruct.KernelFunction)),...
       'interpreter','none');
```

svmtrain



7 Use the `svmclassify` function to classify the test set.

```
classes = svmclassify(svmStruct,data(test,:), 'showplot', true);
```



8 Evaluate the performance of the classifier.

```
classperf(cp, classes, test);
cp.CorrectRate
```

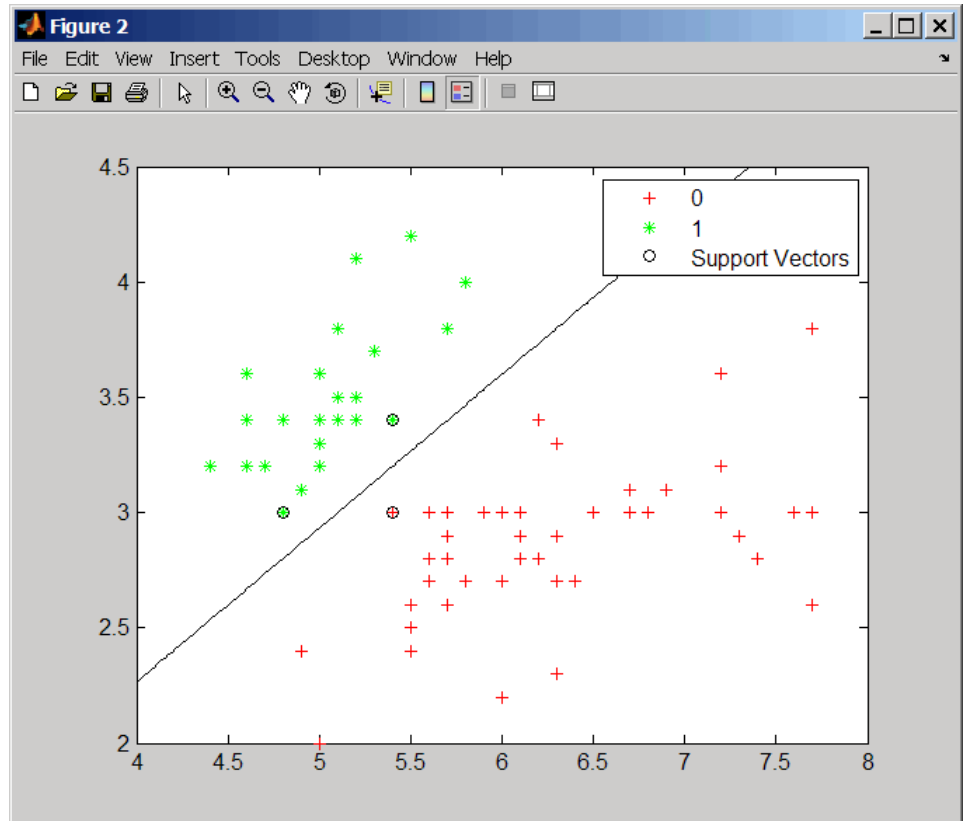
```
ans =
```

```
0.9867
```

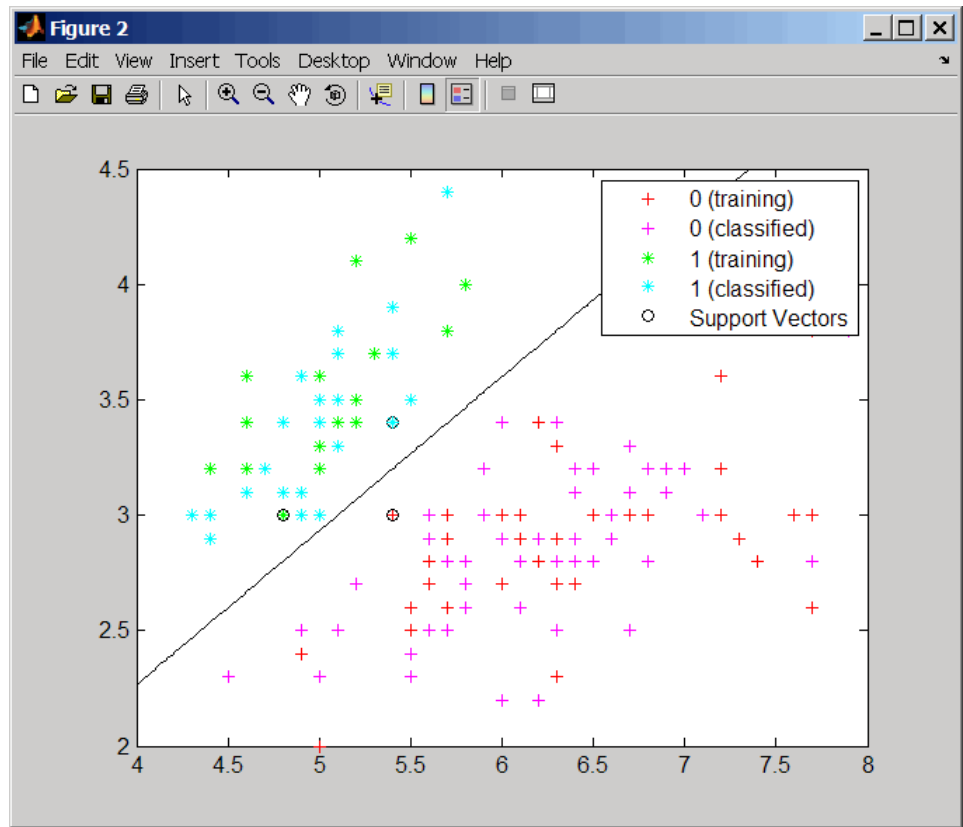
9 Use a one-norm, hard margin support vector machine classifier by changing the boxconstraint property.

svmtrain

```
figure
svmStruct = svmtrain(data(train,:),groups(train),...
                    'showplot',true,'boxconstraint',1e6);
```



```
classes = svmclassify(svmStruct,data(test,:), 'showplot',true);
```



10 Evaluate the performance of the classifier.

```
classperf(cp,classes,test);  
cp.CorrectRate
```

```
ans =
```

```
0.9867
```

References

- [1] Kecman, V. (2001). Learning and Soft Computing (Cambridge, MA: MIT Press).
- [2] Suykens, J.A.K., Van Gestel, T., De Brabanter, J., De Moor, B., and Vandewalle, J. (2002). Least Squares Support Vector Machines (Singapore: World Scientific).
- [3] Scholkopf, B., and Smola, A.J. (2002). Learning with Kernels (Cambridge, MA: MIT Press).
- [4] Cristianini, N. and Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, First Edition (Cambridge: Cambridge University Press).
<http://www.support-vector.net/>

See Also

Bioinformatics Toolbox functions: `knnclassify`, `svmclassify`, `svmsmset`

Statistics Toolbox function: `classify`

Optimization Toolbox function: `quadprog`

MATLAB function: `optimset`

Purpose

Locally align two sequences using Smith-Waterman algorithm

Syntax

```
Score = swalign(Seq1, Seq2)
[Score, Alignment] = swalign(Seq1, Seq2)
[Score, Alignment, Start] = swalign(Seq1, Seq2)
... = swalign(Seq1,Seq2, ...'Alphabet', AlphabetValue)
... = swalign(Seq1,Seq2, ...'ScoringMatrix',
    ScoringMatrixValue, ...)
... = swalign(Seq1,Seq2, ...'Scale', ScaleValue, ...)
... = swalign(Seq1,Seq2, ...'GapOpen', GapOpenValue, ...)
... = swalign(Seq1,Seq2, ...'ExtendGap',
    ExtendGapValue, ...)
... = swalign(Seq1,Seq2, ...'Showscore',
    ShowscoreValue, ...)
```

Arguments

Seq1, Seq2

Amino acid or nucleotide sequences. Enter any of the following:

- Character string of letters representing amino acids or nucleotides, such as returned by `int2aa` or `int2nt`
- Vector of integers representing amino acids or nucleotides, such as returned by `aa2int` or `nt2int`
- Structure containing a `Sequence` field

Tip For help with letter and integer representations of amino acids and nucleotides, see [Amino Acid Lookup](#) on page 2-91 or [Nucleotide Lookup](#) on page 2-102.

AlphabetValue

String specifying the type of sequence. Choices are 'AA' (default) or 'NT'.

ScoringMatrixValue String specifying the scoring matrix to use for the local alignment. Choices for amino acid sequences are:

- 'PAM40'
- 'PAM250'
- 'DAYHOFF'
- 'GONNET'
- 'BLOSUM30' increasing by 5 up to 'BLOSUM90'
- 'BLOSUM62'
- 'BLOSUM100'

Default is:

- 'BLOSUM50' (when *AlphabetValue* equals 'AA')
- 'NUC44' (when *AlphabetValue* equals 'NT')

Note All of the above scoring matrices have a built-in scale factor that returns *Score* in bits.

ScaleValue Scale factor used to return *Score* in arbitrary units other than bits. Choices are any positive value. For example, if you enter $\log(2)$ for *ScaleValue*, then *swalign* returns *Score* in nats.

GapOpenValue Penalty for opening a gap in the alignment. Choices are any positive integer. Default is 8.

<i>ExtendGapValue</i>	Penalty for extending a gap. Choices are any positive integer. Default is equal to <i>GapOpenValue</i> .
<i>ShowscoreValue</i>	Controls the display of the scoring space and the winning path of the alignment. Choices are true or false (default).

Return Values

<i>Score</i>	Optimal local alignment score in bits.
<i>Alignment</i>	3-by-N character array showing the two sequences, <i>Seq1</i> and <i>Seq2</i> , in the first and third rows, and symbols representing the optimal local alignment between them in the second row.
<i>Start</i>	2-by-1 vector of indices indicating the starting point in each sequence for the alignment.

Description

Score = `swalign(Seq1, Seq2)` returns the optimal local alignment score in bits. The scale factor used to calculate the score is provided by the scoring matrix.

`[Score, Alignment]` = `swalign(Seq1, Seq2)` returns a 3-by-N character array showing the two sequences, *Seq1* and *Seq2*, in the first and third rows, and symbols representing the optimal local alignment between them in the second row. The symbol | indicates amino acids or nucleotides that match exactly. The symbol : indicates amino acids or nucleotides that are related as defined by the scoring matrix (nonmatches with a zero or positive scoring matrix value).

`[Score, Alignment, Start]` = `swalign(Seq1, Seq2)` returns a 2-by-1 vector of indices indicating the starting point in each sequence for the alignment.

... = swalign(Seq1,Seq2, ...'*PropertyName*', *PropertyValue*, ...) calls swalign with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = swalign(Seq1,Seq2, ...'*Alphabet*', *AlphabetValue*) specifies the type of sequences. Choices are 'AA' (default) or 'NT'.

... = swalign(Seq1,Seq2, ...'*ScoringMatrix*', *ScoringMatrixValue*, ...) specifies the scoring matrix to use for the local alignment. Default is:

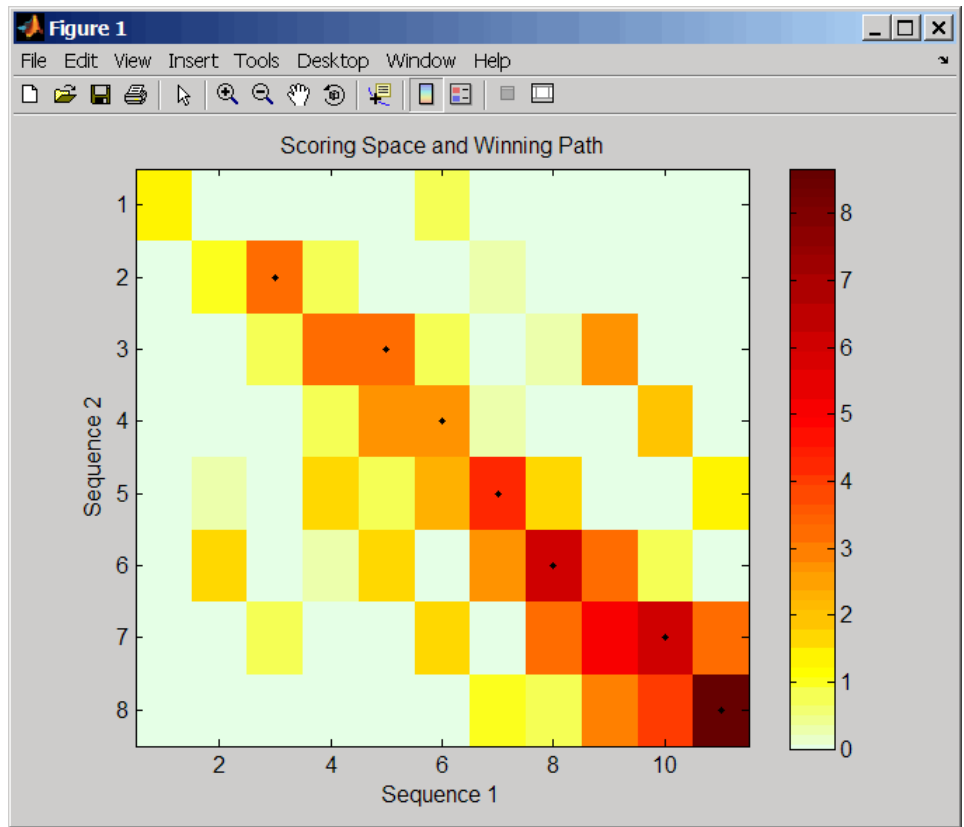
- 'BLOSUM50' (when *AlphabetValue* equals 'AA')
- 'NUC44' (when *AlphabetValue* equals 'NT')

... = swalign(Seq1,Seq2, ...'*Scale*', *ScaleValue*, ...) specifies the scale factor used to return *Score* in arbitrary units other than bits. Choices are any positive value.

... = swalign(Seq1,Seq2, ...'*GapOpen*', *GapOpenValue*, ...) specifies the penalty for opening a gap in the alignment. Choices are any positive integer. Default is 8.

... = swalign(Seq1,Seq2, ...'*ExtendGap*', *ExtendGapValue*, ...) specifies the penalty for extending a gap in the alignment. Choices are any positive integer. Default is equal to *GapOpenValue*.

... = swalign(Seq1,Seq2, ...'*Showscore*', *ShowscoreValue*, ...) controls the display of the scoring space and winning path of the alignment. Choices are true or false (default)



The scoring space is a heat map displaying the best scores for all the partial alignments of two sequences. The color of each $(n1, n2)$ coordinate in the scoring space represents the best score for the pairing of subsequences $Seq1(s1:n1)$ and $Seq2(s2:n2)$, where $n1$ is a position in $Seq1$, $n2$ is a position in $Seq2$, $s1$ is any position in $Seq1$ between $1:n1$, and $s2$ is any position in $Seq2$ between $1:n2$. The best score for a pairing of specific subsequences is determined by scoring all possible alignments of the subsequences by summing matches and gap penalties.

The winning path is represented by black dots in the scoring space and represents the pairing of positions in the optimal local alignment. The

color of the last point (lower right) of the winning path represents the optimal local alignment score for the two sequences and is the *Score* output returned by *swalign*.

Tip The scoring space visually shows tandem repeats, small segments that potentially align, and partial alignments of domains from rearranged sequences.

Examples

- 1 Locally align two amino acid sequences using the BLOSUM50 (default) scoring matrix and the default values for the *GapOpen* and *ExtendGap* properties. Return the optimal local alignment score in bits and the alignment character array. Return the optimal global alignment score in bits and the alignment character array.

```
[Score, Alignment] = swalign('VSPAGMASGYD','IPGKASYD')
```

```
Score =
```

```
8.6667
```

```
Alignment =
```

```
PAGMASGYD
| | || ||
P-GKAS-YD
```

- 2 Locally align two amino acid sequences specifying the PAM250 scoring matrix and a gap open penalty of 5.

```
[Score, Alignment] = swalign('HEAGAWGHEE','PAWHEAE',...
                             'ScoringMatrix','pam250',...
                             'GapOpen',5)
```

```
Score =
```

```

      8
Alignment =

```

```

GAWGHE
: || ||
PAW-HE

```

- 3** Locally align two amino acid sequences returning the *Score* in nat units (nats) by specifying a scale factor of $\log(2)$.

```
[Score, Alignment] = swalign('HEAGAWGHEE', 'PAWHEAE', 'Scale', log(2))
```

```
Score =
```

```
6.4694
```

```
Alignment =
```

```

AWGHE
|| ||
AW-HE

```

References

- [1] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis* (Cambridge University Press).
- [2] Smith, T., and Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology* *147*, 195–197.

See Also

Bioinformatics Toolbox functions: `blosum`, `nt2aa`, `nwalign`, `pam`, `pdbsuperpose`, `seqdotplot`, `showalignment`

traceplot

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

Perform background adjustment on Affymetrix microarray probe-level data using zone-based method

Syntax

```
BackAdjustedData = zonebackadj(Data)
[BackAdjustedData, ZoneStruct] = zonebackadj(Data)
[BackAdjustedData, ZoneStruct,
 Background] = zonebackadj(Data)
... = zonebackadj(Data, ...'NumZones', NumZonesValue, ...)
... = zonebackadj(Data, ...'Percent', PercentValue, ...)
... = zonebackadj(Data, ...'SmoothFactor',
 SmoothFactorValue,
 ...)
... = zonebackadj(Data, ...'NoiseFrac',
 NoiseFracValue, ...)
... = zonebackadj(Data, ...'CDF', CDFValue, ...)
... = zonebackadj(Data, ...'Mask', MaskValue, ...)
... = zonebackadj(Data, ...'Showplot', ShowplotValue, ...)
```

Arguments

Data

Either of the following:

- MATLAB structure containing probe intensities from an Affymetrix CEL file, such as returned by `affyread` when used to read a CEL file.
- Array of MATLAB structures containing probe intensities from multiple Affymetrix CEL files.

NumZonesValue

Scalar or two-element vector that specifies the number of zones to use in the background adjustment. If a scalar, it must be a square number. If a two-element vector, the first element specifies the number of rows and the second element specifies the number of columns in a nonsquare grid. Default is 16.

<i>PercentValue</i>	Value that specifies a percentage, P , such that the lowest P percent of ranked intensity values from each zone is used to estimate the background for that zone. Default is 2.
<i>SmoothFactorValue</i>	Value that specifies the smoothing factor used in the calculation of the weighted average of the contributions of each zone to the background of a point. Default is 100.
<i>NoiseFracValue</i>	Value that specifies the noise fraction, NF , such that the background-adjusted value is given by $\max((Intensity - WeightedBackground), NF * LocalNoiseEstimate)$. Default is 0.5.
<i>CDFValue</i>	<p>Either of the following:</p> <ul style="list-style-type: none">• String specifying a file name or path and file name of an Affymetrix CDF library file. If you specify only a file name, the file must be on the MATLAB search path or in the current directory.• MATLAB structure containing information from an Affymetrix CDF library file, such as returned by <code>affyread</code> when used to read a CDF file. <p>The CDF library file or structure specifies control cells, which are not used in the background estimates.</p>
<i>MaskValue</i>	Logical vector that specifies which cells to mask and not use in the background estimates. In the vector, 0 = not masked and 1 = masked. Defaults are the values in the Masked column of the Probes field of the CEL file.
<i>ShowplotValue</i>	Controls the plotting of an image of the background estimates. Choices are true or false (default).

Return Values

<i>BackAdjustedData</i>	Matrix or cell array of vectors containing background-adjusted probe intensity values.
<i>ZoneStruct</i>	MATLAB structure containing the centers of the zones used to perform the background adjustment and the estimates of the background values at the center of each zone.
<i>Background</i>	Matrix or cell array of vectors containing the estimated background values for each probe.

Description

BackAdjustedData = zonebackadj(*Data*) returns the background-adjusted probe intensities from *Data*, which contains probe intensities from Affymetrix CEL files. Details of the background adjustment are described in Statistical Algorithms Description Document.

[*BackAdjustedData*, *ZoneStruct*] = zonebackadj(*Data*) also returns a structure containing the centers of the zones used to perform the background adjustment and the estimates of the background values at the center of each zone.

[*BackAdjustedData*, *ZoneStruct*, *Background*] = zonebackadj(*Data*) also returns a matrix or cell array of vectors containing the estimated background values for each probe.

... = zonebackadj(*Data*, ...'*PropertyName*', *PropertyValue*, ...) calls zonebackadj with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

... = zonebackadj(*Data*, ...'*NumZones*', *NumZonesValue*, ...) specifies the number of zones to use in the background adjustment.

NumZonesValue can be either a scalar that is a square number or a two-element array in which the first element specifies the number of rows and the second element specifies the number of columns in a nonsquare grid. Default is 16.

... = zonebackadj(*Data*, ... 'Percent', *PercentValue*, ...) specifies a percentage, *P*, such that the lowest *P* percent of ranked intensity values from each zone is used to estimate the background for that zone. Default is 2.

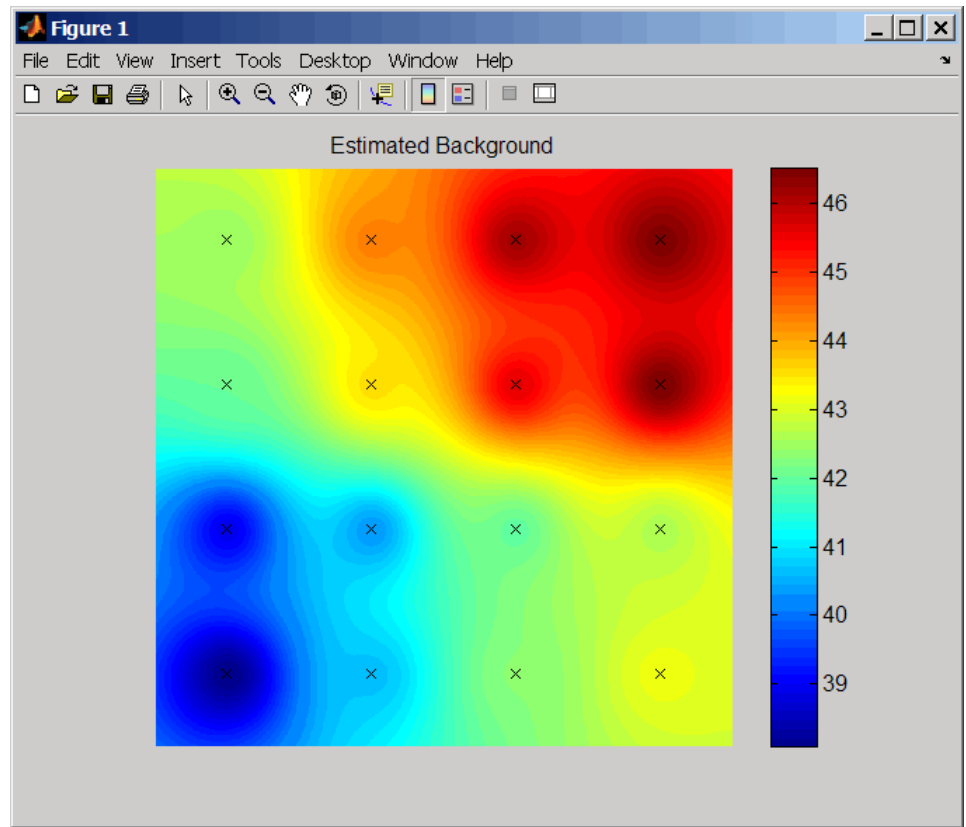
... = zonebackadj(*Data*, ... 'SmoothFactor', *SmoothFactorValue*, ...) specifies the smoothing factor used in the calculation of the weighted average of the contributions of each zone to the background of a point, thus providing a smooth transition between zones. Default is 100.

... = zonebackadj(*Data*, ... 'NoiseFrac', *NoiseFracValue*, ...) specifies the noise fraction, such that the background-adjusted value is given by $\max((\text{Intensity} - \text{WeightedBackground}), \text{NF} * \text{LocalNoiseEstimate})$, where NF is *NoiseFracValue*. Default is 0.5.

... = zonebackadj(*Data*, ... 'CDF', *CDFValue*, ...) specifies an Affymetrix CDF library file or structure, which specifies control cells, which are not used in the background estimates.

... = zonebackadj(*Data*, ... 'Mask', *MaskValue*, ...) specifies a logical vector of that specifies which cells to mask and not use in the background estimates. In the vector, 0 = not masked and 1 = masked. Defaults are the values in the Masked column of the Probes field of the CEL file.

... = zonebackadj(*Data*, ... 'Showplot', *ShowplotValue*, ...) plots an image of the background estimates. Choices are true or false (default).



Examples

The following example uses a sample CEL file and CDF library file from the *E. coli* Antisense Genome array, which you can download from:

http://www.affymetrix.com/support/technical/sample_data/demo_data.affx

After you download the demo data, you will need the Affymetrix Data Transfer Tool to extract the CEL file from a DTT file. You can download the Affymetrix Data Transfer Tool from:

<http://www.affymetrix.com/products/software/specific/dtt.affx>

The following example assumes that the `Ecoli-antisense-121502.CEL` file is stored on the MATLAB search path or in the current directory. It also assumes that the associated CDF library file, `Ecoli_ASv2.CDF`, is stored at `D:\Affymetrix\LibFiles\Ecoli`.

- 1 Use the `affyread` function to read an Affymetrix CEL file and create `celStruct`, a MATLAB structure containing probe intensities for a single Affymetrix GeneChip.

```
celStruct = affyread('Ecoli-antisense-121502.CEL');
```

- 2 Perform background adjustment on the probe intensities in the structure, excluding the probe intensities from the control cells on the chip.

```
BackAdjMatrix = zonebackadj(celStruct, 'cdf',...  
                             'D:\Affymetrix\LibFiles\Ecoli\Ecoli_ASv2.CDF');
```

References

[1] Statistical Algorithms Description Document,
http://www.affymetrix.com/support/technical/whitepapers/sadd_whitepaper.pdf

See Also

Bioinformatics Toolbox functions: `affyinvarsetnorm`, `affyread`, `celintensityread`, `gcrma`, `gcrmabackadj`, `probelibraryinfo`, `probesetlink`, `probesetlookup`, `probesetvalues`, `quantilenorm`, `rbackadj`, `rmasummary`

Method Reference

Phylogenetic Tree (p. 3-1)	Select, modify, and plot phylogenetic trees using <code>phytree</code> object methods
Graph Visualization (p. 3-2)	View relationships between data visually with interactive maps, hierarchy plots, and pathways using <code>biograph</code> object methods
Gene Ontology (p. 3-3)	Explore and analyze Gene Ontology data using <code>geneont</code> object methods
Clustergram (p. 3-4)	Visualize and explore hierarchical clustering analysis data using <code>clustergram</code> object methods
DataMatrix (p. 3-4)	Explore and analyze microarray data using <code>DataMatrix</code> object methods

Phylogenetic Tree

Following are methods for use with a `phytree` object.

<code>get (phytree)</code>	Retrieve information about phylogenetic tree object
<code>getbyname (phytree)</code>	Branches and leaves from <code>phytree</code> object
<code>getcanonical (phytree)</code>	Calculate canonical form of phylogenetic tree

<code>getmatrix (phytree)</code>	Convert phytree object into relationship matrix
<code>getnewickstr (phytree)</code>	Create Newick-formatted string
<code>pdist (phytree)</code>	Calculate pairwise patristic distances in phytree object
<code>plot (phytree)</code>	Draw phylogenetic tree
<code>prune (phytree)</code>	Remove branch nodes from phylogenetic tree
<code>reorder (phytree)</code>	Reorder leaves of phylogenetic tree
<code>reroot (phytree)</code>	Change root of phylogenetic tree
<code>select (phytree)</code>	Select tree branches and leaves in phytree object
<code>subtree (phytree)</code>	Extract phylogenetic subtree
<code>view (phytree)</code>	View phylogenetic tree
<code>weights (phytree)</code>	Calculate weights for phylogenetic tree

Graph Visualization

Following are methods for use with a `biograph` object.

<code>allshortestpaths (biograph)</code>	Find all shortest paths in biograph object
<code>conncomp (biograph)</code>	Find strongly or weakly connected components in biograph object
<code>dolayout (biograph)</code>	Calculate node positions and edge trajectories
<code>get (biograph)</code>	Retrieve information about biograph object
<code>getancestors (biograph)</code>	Find ancestors in biograph object

<code>getdescendants (biograph)</code>	Find descendants in biograph object
<code>getedgesbynodeid (biograph)</code>	Get handles to edges in biograph object
<code>getmatrix (biograph)</code>	Get connection matrix from biograph object
<code>getnodesbyid (biograph)</code>	Get handles to nodes
<code>getrelatives (biograph)</code>	Find relatives in biograph object
<code>isdag (biograph)</code>	Test for cycles in biograph object
<code>isomorphism (biograph)</code>	Find isomorphism between two biograph objects
<code>isspantree (biograph)</code>	Determine if tree created from biograph object is spanning tree
<code>maxflow (biograph)</code>	Calculate maximum flow in biograph object
<code>minspantree (biograph)</code>	Find minimal spanning tree in biograph object
<code>set (biograph)</code>	Set property of biograph object
<code>shortestpath (biograph)</code>	Solve shortest path problem in biograph object
<code>topoorder (biograph)</code>	Perform topological sort of directed acyclic graph extracted from biograph object
<code>traverse (biograph)</code>	Traverse biograph object by following adjacent nodes
<code>view (biograph)</code>	Draw figure from biograph object

Gene Ontology

Following are methods for use with a `geneont` object.

<code>getancestors (geneont)</code>	Find terms that are ancestors of specified Gene Ontology (GO) term
<code>getdescendants (geneont)</code>	Find terms that are descendants of specified Gene Ontology (GO) term
<code>getmatrix (geneont)</code>	Convert geneont object into relationship matrix
<code>getrelatives (geneont)</code>	Find terms that are relatives of specified Gene Ontology (GO) term

Clustergram

Following are methods for use with a `clustergram` object.

<code>get (clustergram)</code>	Retrieve information about clustergram object
<code>plot (clustergram)</code>	Render clustergram heat map and dendrograms for clustergram object
<code>set (clustergram)</code>	Set property of clustergram object
<code>view (clustergram)</code>	View clustergram heat map and dendrograms for clustergram object

DataMatrix

Following are methods for use with a `DataMatrix` object.

<code>colnames (DataMatrix)</code>	Retrieve or set column names of DataMatrix object
<code>disp (DataMatrix)</code>	Display DataMatrix object
<code>dmarrayfun (DataMatrix)</code>	Apply function to each element in DataMatrix object

<code>dmbsxfun (DataMatrix)</code>	Apply element-by-element binary operation to two DataMatrix objects with singleton expansion enabled
<code>double (DataMatrix)</code>	Convert DataMatrix object to double-precision array
<code>eq (DataMatrix)</code>	Test DataMatrix objects for equality
<code>ge (DataMatrix)</code>	Test DataMatrix objects for greater than or equal to
<code>get (DataMatrix)</code>	Retrieve information about DataMatrix object
<code>gt (DataMatrix)</code>	Test DataMatrix objects for greater than
<code>horzcat (DataMatrix)</code>	Concatenate DataMatrix objects horizontally
<code>isequal (DataMatrix)</code>	Test DataMatrix objects for equality
<code>isequalwithequalnans (DataMatrix)</code>	Test DataMatrix objects for equality, treating NaNs as equal
<code>ldivide (DataMatrix)</code>	Left array divide DataMatrix objects
<code>le (DataMatrix)</code>	Test DataMatrix objects for less than or equal to
<code>lt (DataMatrix)</code>	Test DataMatrix objects for less than
<code>max (DataMatrix)</code>	Return maximum values in DataMatrix object
<code>mean (DataMatrix)</code>	Return average or mean values in DataMatrix object
<code>median (DataMatrix)</code>	Return median values in DataMatrix object
<code>min (DataMatrix)</code>	Return minimum values in DataMatrix object
<code>minus (DataMatrix)</code>	Subtract DataMatrix objects
<code>ndims (DataMatrix)</code>	Return number of dimensions in DataMatrix object

<code>ne (DataMatrix)</code>	Test DataMatrix objects for inequality
<code>numel (DataMatrix)</code>	Return number of elements in DataMatrix object
<code>plot (DataMatrix)</code>	Draw 2-D line plot of DataMatrix object
<code>plus (DataMatrix)</code>	Add DataMatrix objects
<code>power (DataMatrix)</code>	Array power DataMatrix objects
<code>rdivide (DataMatrix)</code>	Right array divide DataMatrix objects
<code>rownames (DataMatrix)</code>	Retrieve or set row names of DataMatrix object
<code>set (DataMatrix)</code>	Set property of DataMatrix object
<code>single (DataMatrix)</code>	Convert DataMatrix object to single-precision array
<code>sortcols (DataMatrix)</code>	Sort columns of DataMatrix object in ascending or descending order
<code>sortrows (DataMatrix)</code>	Sort rows of DataMatrix object in ascending or descending order
<code>std (DataMatrix)</code>	Return standard deviation values in DataMatrix object
<code>sum (DataMatrix)</code>	Return sum of elements in DataMatrix object
<code>times (DataMatrix)</code>	Multiply DataMatrix objects
<code>var (DataMatrix)</code>	Return variance values in DataMatrix object
<code>vertcat (DataMatrix)</code>	Concatenate DataMatrix objects vertically

Methods — Alphabetical List

allshortestpaths (biograph)

Purpose Find all shortest paths in biograph object

Syntax

```
[dist] = allshortestpaths(BGObj)
[dist] = allshortestpaths(BGObj, ...'Directed',
DirectedValue, ...)
[dist] = allshortestpaths(BGObj, ...'Weights', WeightsValue,
...)
```

Arguments

BGObj Biograph object created by biograph (object constructor).

DirectedValue Property that indicates whether the graph is directed or undirected. Enter `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`.

WeightsValue Column vector that specifies custom weights for the edges in the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. It must have one entry for every nonzero value (edge) in the matrix. The order of the custom weights in the vector must match the order of the nonzero values in the matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, `allshortestpaths` gets weight information from the nonzero entries in the matrix.

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions” in the *Bioinformatics Toolbox User’s Guide*.

`[dist] = allshortestpaths(BGObj)` finds the shortest paths between every pair of nodes in a graph represented by an N-by-N adjacency matrix extracted from a biograph object, *BGObj*, using Johnson’s

algorithm. Nonzero entries in the matrix represent the weights of the edges.

Output *dist* is an N-by-N matrix where *dist*(S,T) is the distance of the shortest path from node S to node T. A 0 in this matrix indicates the source node; an Inf is an unreachable node.

Johnson's algorithm has a time complexity of $O(N \cdot \log(N) + N \cdot E)$, where N and E are the number of nodes and edges respectively.

[...] = allshortestpaths(*BGObj*, '*PropertyName*', *PropertyValue*, ...) calls allshortestpaths with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

[*dist*] = allshortestpaths(*BGObj*, ...'*Directed*', *DirectedValue*, ...) indicates whether the graph is directed or undirected. Set *DirectedValue* to false for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is true.

[*dist*] = allshortestpaths(*BGObj*, ...'*Weights*', *WeightsValue*, ...) lets you specify custom weights for the edges. *WeightsValue* is a column vector having one entry for every nonzero value (edge) in the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. The order of the custom weights in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, allshortestpaths gets weight information from the nonzero entries in the N-by-N adjacency matrix.

References

- [1] Johnson, D.B. (1977). Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM* 24(1), 1-13.
- [2] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). *The Boost Graph Library User Guide and Reference Manual*, (Upper Saddle River, NJ:Pearson Education).

allshortestpaths (biograph)

See Also

Bioinformatics Toolbox functions: `biograph` (object constructor), `graphallshortestpaths`

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object: `conncomp`, `isdag`, `isomorphism`, `isspantree`, `maxflow`, `minspantree`, `shortestpath`, `topoorder`, `traverse`

Purpose

Retrieve or set column names of DataMatrix object

Syntax

ReturnColNames = colnames(*DMObj*)

ReturnColNames = colnames(*DMObj*, *ColIndices*)

DMObjNew = colnames(*DMObj*, *ColIndices*, *ColNames*)

colnames (DataMatrix)

Arguments

<i>DMObj</i>	DataMatrix object, such as created by <code>DataMatrix</code> (object constructor).
<i>ColIndices</i>	One or more columns in <i>DMObj</i> , specified by any of the following: <ul style="list-style-type: none">• Positive integer• Vector of positive integers• String specifying a column name• Cell array of strings• Logical vector
<i>ColNames</i>	Column names specified by any of the following: <ul style="list-style-type: none">• Numeric vector• Cell array of strings• Character array• Single string, which is used as a prefix for column names, with column numbers appended to the prefix• Logical <code>true</code> or <code>false</code> (default). If <code>true</code>, unique column names are assigned using the format <code>col1</code>, <code>col2</code>, <code>col3</code>, etc. If <code>false</code>, no column names are assigned.

Note The number of elements in *ColNames* must equal the number of elements in *ColIndices*.

Return Values

ReturnColNames String or cell array of strings containing column names in *DMatrix*.

DMatrixNew DataMatrix object created with names specified by *ColIndices* and *ColNames*.

Description

ReturnColNames = `colnames(DMatrix)` returns *ReturnColNames*, a cell array of strings specifying the column names in *DMatrix*, a DataMatrix object.

ReturnColNames = `colnames(DMatrix, ColIndices)` returns the column names specified by *ColIndices*. *ColIndices* can be a positive integer, vector of positive integers, string specifying a column name, cell array of strings, or a logical vector.

DMatrixNew = `colnames(DMatrix, ColIndices, ColNames)` returns *DMatrixNew*, a DataMatrix object with columns specified by *ColIndices* set to the names specified by *ColNames*. The number of elements in *ColIndices* must equal the number of elements in *ColNames*.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a DataMatrix object: `rownames`

conncomp (biograph)

Purpose Find strongly or weakly connected components in biograph object

Syntax

```
[S, C] = conncomp(BGObj)
[S, C] = conncomp(BGObj, ...'Directed', DirectedValue, ...)
[S, C] = conncomp(BGObj, ...'Weak', WeakValue, ...)
```

Arguments

<i>BGObj</i>	Biograph object created by biograph (object constructor).
<i>DirectedValue</i>	Property that indicates whether the graph is directed or undirected. Enter <code>false</code> for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is <code>true</code> . A DFS-based algorithm computes the connected components. Time complexity is $O(N+E)$, where N and E are number of nodes and edges respectively.
<i>WeakValue</i>	Property that indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set <i>WeakValue</i> to <code>true</code> to find weakly connected components. Default is <code>false</code> , which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is $O(N+E)$, where N and E are number of nodes and edges respectively.

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions” in the *Bioinformatics Toolbox User’s Guide*.

`[S, C] = conncomp(BGObj)` finds the strongly connected components of an N-by-N adjacency matrix extracted from a biograph object, *BGObj* using Tarjan's algorithm. A strongly connected component is a maximal group of nodes that are mutually reachable without violating the edge directions. The N-by-N sparse matrix represents a directed graph; all nonzero entries in the matrix indicate the presence of an edge.

The number of components found is returned in *S*, and *C* is a vector indicating to which component each node belongs.

Tarjan's algorithm has a time complexity of $O(N+E)$, where *N* and *E* are the number of nodes and edges respectively.

`[S, C] = conncomp(BGObj, ...'PropertyName', PropertyValue, ...)` calls `conncomp` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

`[S, C] = conncomp(BGObj, ...'Directed', DirectedValue, ...)` indicates whether the graph is directed or undirected. Set *DirectedValue* to `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`. A DFS-based algorithm computes the connected components. Time complexity is $O(N+E)$, where *N* and *E* are number of nodes and edges respectively.

`[S, C] = conncomp(BGObj, ...'Weak', WeakValue, ...)` indicates whether to find weakly connected components or strongly connected components. A weakly connected component is a maximal group of nodes that are mutually reachable by violating the edge directions. Set *WeakValue* to `true` to find weakly connected components. Default is `false`, which finds strongly connected components. The state of this parameter has no effect on undirected graphs because weakly and strongly connected components are the same in undirected graphs. Time complexity is $O(N+E)$, where *N* and *E* are number of nodes and edges respectively.

conncomp (biograph)

Note By definition, a single node can be a strongly connected component.

Note A directed acyclic graph (DAG) cannot have any strongly connected components larger than one.

References

- [1] Tarjan, R.E., (1972). Depth first search and linear graph algorithms. *SIAM Journal on Computing* 1(2), 146–160.
- [2] Sedgewick, R., (2002). *Algorithms in C++, Part 5 Graph Algorithms* (Addison-Wesley).
- [3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). *The Boost Graph Library User Guide and Reference Manual*, (Upper Saddle River, NJ:Pearson Education).

See Also

Bioinformatics Toolbox functions: `biograph` (object constructor), `graphconncomp`

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object:
`allshortestpaths`, `isdag`, `isomorphism`, `isspantree`, `maxflow`,
`minspantree`, `shortestpath`, `topoorder`, `traverse`

Purpose Display DataMatrix object

Syntax `disp(DMObj)`

Arguments *DMObj* DataMatrix object, such as created by `DataMatrix` (object constructor).

Description `disp(DMObj)` displays the DataMatrix object *DMObj*, including row names and column names, without printing the DataMatrix object name.

See Also Bioinformatics Toolbox function: `DataMatrix` (object constructor)
Bioinformatics Toolbox object: `DataMatrix` object

dmarrayfun (DataMatrix)

Purpose

Apply function to each element in DataMatrix object

Syntax

```
DMObjNew1 = dmarrayfun(Func, DMObj1)  
DMObjNew1 = dmarrayfun(Func, DMObj1, DMObj2, ...)   
[DMObjNew1, DMObjNew2, ...] = dmarrayfun(Func, DMObj1, ...)   
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1,  
... 'UniformOutput', UniformOutputValue, ...)   
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1,  
... 'DataMatrixOutput', DataMatrixOutputValue, ...)   
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ... 'Rows',  
... RowsValue, ...)   
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ... 'Columns',  
... ColumnsValue, ...)   
[DMObjNew1, ...] = dmarrayfun(Func, DMObj1,  
... 'ErrorHandler',  
... ErrorHandlerValue, ...)
```

Arguments

Func

Function handle for a function that returns one or more scalars, and returns values of the same class each time it is called.

DMObj1

DataMatrix object, such as created by `DataMatrix` (object constructor).

DMObj2

Either of the following:

- DataMatrix object, such as created by `DataMatrix` (object constructor)
- MATLAB numeric array

Note *DMObj2* and subsequent input objects or arrays must be the same size (number of rows and columns) as *DMObj1*.

<i>UniformOutputValue</i>	Specifies whether <i>Func</i> must return output values without encapsulation in a cell array. Choices are <code>true</code> (default) or <code>false</code> . If <code>true</code> , <code>dmarrayfun</code> must return scalar values that can be concatenated into an array. These values can also be a cell array. If <code>false</code> , <code>dmarrayfun</code> returns a cell array (or multiple cell arrays), where the I,Jth cell contains the value equal to <i>Func</i> (<i>DMObj1</i> (I,J),...).
<i>DataMatrixOutputValue</i>	Specifies whether return values must be <code>DataMatrix</code> objects. Choices are <code>true</code> (default) or <code>false</code> . If you set the 'UniformOutput' property to <code>false</code> , this property is ignored.
<i>RowsValue</i> , <i>ColumnsValue</i>	Specifies the rows or columns to which to apply the function. Choices are: <ul style="list-style-type: none">• Positive integer• Vector of positive integers• String specifying a row or column name• Cell array of strings• Logical vector
<i>ErrorHandlerValue</i>	Specifies a function handle to a function that <code>dmarrayfun</code> calls if the call to <i>Func</i> fails.

dmarrayfun (DataMatrix)

Return Values

DMObjNew1,
DMObjNew2

DataMatrix objects created from applying the function to each element in one or more DataMatrix objects. The size (number of rows and columns), row names, and column names will be the same as *DMObj1*.

Description

DMObjNew1 = dmarrayfun(*Func*, *DMObj1*) applies the function specified by *Func* to each element in *DMObj1*, a DataMatrix object, and returns the results in *DMObjNew1*, a new DataMatrix object. *DMObjNew1* has the same size (number of rows and columns), row names, and column names as *DMObj1*. The I,Jth element of *DMObjNew1* is equal to *Func*(*DMObj1*(I,J)), where *Func* is a function handle for a function that takes one input argument, returns one scalar value, and returns values of the same class each time it is called.

DMObjNew1 = dmarrayfun(*Func*, *DMObj1*, *DMObj2*, ...) evaluates the function specified by *Func* using elements in *DMObj1*, *DMObj2*, etc. as input arguments. The I,Jth element of *DMObjNew1* is equal to *Func*(*DMObj1*(I,J), *DMObj2*(I,J), ...), where *Func* is a function handle for a function that takes multiple input arguments, returns one scalar, and returns values of the same class each time it is called.

[*DMObjNew1*, *DMObjNew2*, ...] = dmarrayfun(*Func*, *DMObj1*, ...) evaluates the function specified by *Func* using elements in *DMObj1*, and possibly other input arguments. *Func* is a function handle for a function that takes one or more input arguments, returns multiple scalars, and returns values of the same class each time it is called. It returns DataMatrix objects *DMObjNew1*, *DMObjNew2*, etc. with each one corresponding to one of the outputs of *Func*. The outputs of *Func* may be of different classes, however, but each output must be the same each time it is called.

[*DMObjNew1*, ...] = dmarrayfun(*Func*, *DMObj1*, ...'*PropertyName*', *PropertyValue*, ...) calls dmarrayfun with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName*

must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'UniformOutput', UniformOutputValue, ...)` specifies whether *Func* must return output values without encapsulation in a cell array. Choices are `true` (default) or `false`. If `true`, `dmarrayfun` must return scalar values that can be concatenated into an array. These values can also be a cell array. If `false`, `dmarrayfun` returns a cell array (or multiple cell arrays), where the *I,J*th cell contains the value equal to *Func*(*DMObj1*(*I,J*),...).

`[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'DataMatrixOutput', DataMatrixOutputValue, ...)` specifies whether return values must be `DataMatrix` objects. Choices are `true` (default) or `false`. If you set the `'UniformOutput'` property to `false`, this property is ignored.

`[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'Rows', RowsValue, ...)` applies the function only to the rows in the `DataMatrix` object specified by *RowsValue*, which can be a positive integer, vector of positive integers, string specifying a row name, cell array of strings, or a logical vector.

`[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'Columns', ColumnsValue, ...)` applies the function only to the columns in the `DataMatrix` object specified by *ColsValue*, which can be a positive integer, vector of positive integers, string specifying a column name, cell array of strings, or a logical vector.

`[DMObjNew1, ...] = dmarrayfun(Func, DMObj1, ...'ErrorHandler', ErrorHandlerValue, ...)` specifies a function handle to a function that `dmarrayfun` calls if the call to *Func* fails. The error handling function will be called with these input arguments:

- Structure with the following fields:
 - `identifier` — Identifier of the error

dmarrayfun (DataMatrix)

- `message` — Error message text
- `index` — Linear index into the input array(s) at which the error occurred
- Set of input arguments at which the call to the function failed

If you do not specify *ErrorHandlerValue*, `dmarrayfun` rethrows the error from the call to *Func*.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a `DataMatrix` object: `dmbsxfun`

MATLAB function: `arrayfun`

Purpose	Apply element-by-element binary operation to two DataMatrix objects with singleton expansion enabled
Syntax	$DMObjNew = dmbsxfun(Func, DMObj1, DMObj2)$
Arguments	<p><i>Func</i> Function handle for an M-file function or a built-in function. For more information on built-in functions, see <code>bsxfun</code>.</p> <p><i>DMObj1, DMObj2</i> Either of the following:</p> <ul style="list-style-type: none">• DataMatrix object, such as created by <code>DataMatrix</code> (object constructor)• MATLAB numeric array <p>At least one of these input arguments must be a DataMatrix object.</p>
Return Values	<p><i>DMObjNew</i> DataMatrix object or MATLAB numeric array created from element-by-element binary operation of two DataMatrix objects with singleton expansion enabled.</p>
Description	<p>$DMObjNew = dmbsxfun(Func, DMObj1, DMObj2)$ applies an element-by-element binary operation to the DataMatrix objects <i>DMObj1</i> and <i>DMObj2</i>, with singleton expansion enabled. <i>Func</i> is a function handle, and can be for an M-file function or a built-in function. For more information on built-in functions, see <code>bsxfun</code>.</p> <p><i>DMObj1</i> and <i>DMObj2</i> can be DataMatrix objects or MATLAB numeric arrays; however, at least one of these input arguments must be a DataMatrix object. <i>DMObj1</i> and <i>DMObj2</i> must have the same number of rows or the same number or columns. If they don't have the same number of rows, then one must be a row vector and its rows are expanded down to be equal to the larger matrix. If they don't have the</p>

dmbsxfun (DataMatrix)

same number of columns, then one must be a column vector and its columns are expanded across to be equal to the larger matrix.

DMObjNew is a DataMatrix object, unless the larger input argument is a MATLAB numeric array; then *DMObjNew* is also a numeric array. The size (number of rows and columns) of *DMObjNew* is equal to the larger of the two input arguments. The row names and column names of *DMObjNew* come from the larger input argument, or, if both inputs are the same size, from the first input argument.

Examples

- 1 Use the DataMatrix constructor function to create a DataMatrix object.

```
A = bioma.data.DataMatrix(magic(3), 'RowNames', true, 'ColNames', true)
```

- 2 Use the built-in function @minus to subtract the column means from this DataMatrix object.

```
A = dmbsxfun(@minus, A, mean(A))
```

See Also

Bioinformatics Toolbox function: DataMatrix (object constructor)

Bioinformatics Toolbox object: DataMatrix object

MATLAB function: bsxfun

Purpose Calculate node positions and edge trajectories

Syntax
`dolayout(BGobj)`
`dolayout(BGobj, 'Paths', PathsOnlyValue)`

Arguments

<i>BGobj</i>	Biograph object created by the biograph function (object constructor).
<i>PathsOnlyValue</i>	Controls the calculation of only the edge paths, leaving the nodes at their current positions. Choices are true or false (default).

Description

Note To use the `dolayout` method, you must have accepted a Graphviz software license (free). If you have not, you will be prompted to do so.

`dolayout(BGobj)` calls the layout engine to calculate the optimal position for each node so that its 2-D rendering is clean and uncluttered, and then calculates the best curves to represent the edges. The layout engine uses the following properties of the biograph object:

- **LayoutType** — Specifies the layout engine as 'hierarchical', 'equilibrium', or 'radial'.
- **LayoutScale** — Rescales the sizes of the node before calling the layout engine. This gives more space to the layout and reduces the overlapping of nodes.
- **NodeAutoSize** — Controls precalculating the node size before calling the layout engine. When **NodeAutoSize** is set to 'on', the layout engine uses the node properties **FontSize** and **Shape**, and the biograph object property **LayoutScale** to precalculate the actual size of each node. When **NodeAutoSize** is set to 'off', the layout engine uses the node property **Size**.

dolayout (biograph)

For more information on the above properties, see Properties of a Biograph Object on page 5-4. For information on accessing and specifying the above properties of a biograph object, see Determining Properties and Property Values of a Biograph Object on page 5-10 and Specifying Properties of a Biograph Object on page 5-11.

`dolayout(BGObj, 'Paths', PathsOnlyValue)` controls the calculation of only the edge paths, leaving the nodes at their current positions. Choices are `true` or `false` (default).

Examples

- 1 Create a biograph object.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];
bg = biograph(cm)
Biograph object with 5 nodes and 9 edges.
bg.nodes(1).Position

ans =

    []
```

Nodes do not have a position yet.

- 2 Call the layout engine and render the graph.

```
dolayout(bg)
bg.nodes(1).Position

ans =

    112    224

view(bg)
```

- 3 Manually modify a node position and recalculate the paths only.

```
bg.nodes(1).Position = [150 150];
dolayout(bg, 'Pathsonly', true)
```

`view(bg)`

See Also

Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object: `dolayout`, `get`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `set`, `view`

double (DataMatrix)

Purpose Convert DataMatrix object to double-precision array

Syntax

```
B = double(DMObj)  
B = double(DMObj, Rows)  
B = double(DMObj, Rows, Cols)
```

Arguments

<i>DMObj</i>	DataMatrix object, such as created by <code>DataMatrix</code> (object constructor).
<i>Rows</i> , <i>Cols</i>	Row(s) or column(s) in <i>DMObj</i> , specified by one of the following: <ul style="list-style-type: none">• Scalar• Vector of positive integers• String specifying a row or column name• Cell array of row or column names• Logical vector

Return Values

<i>B</i>	MATLAB numeric array.
----------	-----------------------

Description

B = `double(DMObj)` converts *DMObj*, a DataMatrix object, to a double-precision array, which it returns in *B*.

B = `double(DMObj, Rows)` converts a subset of *DMObj*, a DataMatrix object, specified by *Rows*, to a double-precision array, which it returns in *B*. *Rows* can be a positive integer, vector of positive integers, string specifying a row name, cell array of row names, or a logical vector.

B = `double(DMObj, Rows, Cols)` converts a subset of *DMObj*, a DataMatrix object, specified by *Rows* and *Cols*, to a double-precision array, which it returns in *B*. *Cols* can be a positive integer, vector of

positive integers, string specifying a column name, cell array of column names, or a logical vector.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a `DataMatrix` object: `single`

eq (DataMatrix)

Purpose Test DataMatrix objects for equality

Syntax

```
T = eq(DMObj1, DMObj2)
T = DMObj1 == DMObj2
T = eq(DMObj1, B)
T = DMObj1 == B
T = eq(B, DMObj1)
T = B == DMObj1
```

Arguments

DMObj1, *DMObj2* DataMatrix objects, such as created by DataMatrix (object constructor).

B MATLAB numeric or logical array.

Return Values

T Logical matrix of the same size as *DMObj1* and *DMObj2* or *DMObj1* and *B*. It contains logical 1 (true) where elements in the first input are equal to the corresponding element in the second input, and logical 0 (false) when they are not equal.

Description

$T = \text{eq}(DMObj1, DMObj2)$ or the equivalent $T = DMObj1 == DMObj2$ compares each element in DataMatrix object *DMObj1* to the corresponding element in DataMatrix object *DMObj2*, and returns *T*, a logical matrix of the same size as *DMObj1* and *DMObj2*, containing logical 1 (true) where elements in *DMObj1* are equal to the corresponding element in *DMObj2*, and logical 0 (false) when they are not equal. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). *DMObj1* and *DMObj2* can have different Name properties.

$T = \text{eq}(DMObj1, B)$ or the equivalent $T = DMObj1 == B$ compares each element in DataMatrix object *DMObj1* to the corresponding element in *B*, a numeric or logical array, and returns *T*, a logical matrix of the same size as *DMObj1* and *B*, containing logical 1 (true) where elements

in *DMObj1* are equal to the corresponding element in *B*, and logical 0 (false) when they are not equal. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

$T = \text{eq}(B, \text{DMObj1})$ or the equivalent $T = B == \text{DMObj1}$ compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are equal to the corresponding element in *DMObj1*, and logical 0 (false) when they are not equal. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls $T = \text{eq}(X, Y)$ for the syntax $T = X == Y$ when *X* or *Y* is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a DataMatrix object: `ne`

ge (DataMatrix)

Purpose Test DataMatrix objects for greater than or equal to

Syntax

```
T = ge(DMObj1, DMObj2)
T = DMObj1 >= DMObj2
T = ge(DMObj1, B)
T = DMObj1 >= B
T = ge(B, DMObj1)
T = B >= DMObj1
```

Arguments

DMObj1, DMObj2 DataMatrix objects, such as created by DataMatrix (object constructor).

B MATLAB numeric or logical array.

Return Values

T Logical matrix of the same size as *DMObj1* and *DMObj2* or *DMObj1* and *B*. It contains logical 1 (true) where elements in the first input are greater than or equal to the corresponding element in the second input, and logical 0 (false) otherwise.

Description

$T = \text{ge}(DMObj1, DMObj2)$ or the equivalent $T = DMObj1 \geq DMObj2$ compares each element in DataMatrix object *DMObj1* to the corresponding element in DataMatrix object *DMObj2*, and returns *T*, a logical matrix of the same size as *DMObj1* and *DMObj2*, containing logical 1 (true) where elements in *DMObj1* are greater than or equal to the corresponding element in *DMObj2*, and logical 0 (false) otherwise. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). *DMObj1* and *DMObj2* can have different Name properties.

$T = \text{ge}(DMObj1, B)$ or the equivalent $T = DMObj1 \geq B$ compares each element in DataMatrix object *DMObj1* to the corresponding element in *B*, a numeric or logical array, and returns *T*, a logical matrix of the same size as *DMObj1* and *B*, containing logical 1 (true) where elements

in *DMObj1* are greater than or equal to the corresponding element in *B*, and logical 0 (false) otherwise. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

$T = \text{ge}(B, \text{DMObj1})$ or the equivalent $T = B \geq \text{DMObj1}$ compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are greater than or equal to the corresponding element in *DMObj1*, and logical 0 (false) otherwise. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls $T = \text{ge}(X, Y)$ for the syntax $T = X \geq Y$ when *X* or *Y* is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a DataMatrix object: `le`

get (biograph)

Purpose

Retrieve information about biograph object

Syntax

```
get(BGobj)  
BGStruct = get(BGobj)  
PropertyValue = get(BGobj, 'PropertyName')  
[Property1Value, Property2Value, ...] = get(BGobj,  
    'Property1Name', 'Property2Name', ...)
```

Arguments

BGobj Biograph object created with the function `biograph`.
PropertyName Property name for a biograph object.

Return Values

BGStruct Scalar structure, in which each field name is a property of a biograph object, and each field contains the value of that property.
PropertyValue Value of the property specified by *PropertyName*.

Description

`get(BGobj)` displays all properties and their current values of *BGobj*, a biograph object.

`BGStruct = get(BGobj)` returns all properties of *BGobj*, a biograph object, to *BGStruct*, a scalar structure, in which each field name is a property of a biograph object, and each field contains the value of that property.

`PropertyValue = get(BGobj, 'PropertyName')` returns the value of the specified property of *BGobj*, a biograph object.

`[Property1Value, Property2Value, ...] = get(BGobj, 'Property1Name', 'Property2Name', ...)` returns the values of the specified properties of *BGobj*, a biograph object.

Properties of a Biograph Object

Property	Description
ID	String to identify the biograph object. Default is ''.
Label	String to label the biograph object. Default is ''.
Description	String that describes the biograph object. Default is ''.
LayoutType	String that specifies the algorithm for the layout engine. Choices are: <ul style="list-style-type: none">• 'hierarchical' (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges.• 'radial' — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges.• 'equilibrium' — Calculates layout by minimizing the energy in a dynamic spring system.

get (biograph)

Properties of a Biograph Object (Continued)

Property	Description
EdgeType	<p>String that specifies how edges display. Choices are:</p> <ul style="list-style-type: none">• 'straight'• 'curved' (default)• 'segmented' <hr/> <p>Note Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with LayoutType equal to 'equilibrium' or 'radial' cannot produce curved or segmented edges.</p> <hr/>
Scale	Positive number that post-scales the node coordinates. Default is 1.
LayoutScale	Positive number that scales the size of the nodes before calling the layout engine. Default is 1.
EdgeTextColor	Three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.
EdgeFontSize	Positive number that sets the size of the edge font in points. Default is 8.
ShowArrows	Controls the display of arrows with the edges. Choices are 'on' (default) or 'off'.
ArrowSize	Positive number that sets the size of the arrows in points. Default is 8.
ShowWeights	Controls the display of text indicating the weight of the edges. Choices are 'on' (default) or 'off'.

Properties of a Biograph Object (Continued)

Property	Description
ShowTextInNodes	<p>String that specifies the node property used to label nodes when you display a biograph object using the view method. Choices are:</p> <ul style="list-style-type: none">• 'Label' — Uses the Label property of the node object (default).• 'ID' — Uses the ID property of the node object.• 'None'
NodeAutoSize	<p>Controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.</p>
NodeCallback	<p>User-defined callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph object in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, @(node) inspect(node), which displays the Property Inspector dialog box.</p>

get (biograph)

Properties of a Biograph Object (Continued)

Property	Description
EdgeCallback	User-defined callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph object in the Biograph Viewer, you can double-click an edge to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, @(edge) inspect(edge), which displays the Property Inspector dialog box.
CustomNodeDrawFcn	Function handle to a customized function to draw nodes. Default is [].
Nodes	Read-only column vector with handles to node objects of a biograph object. The size of the vector is the number of nodes. For properties of node objects, see Properties of a Node Object on page 5-7.
Edges	Read-only column vector with handles to edge objects of a biograph object. The size of the vector is the number of edges. For properties of edge objects, see Properties of an Edge Object on page 5-9.

Examples

- 1 Create a biograph object and assign the node IDs.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];  
ids = {'M30931','L07625','K03454','M27323','M15390'};  
bg = biograph(cm,ids);
```

- 2 Use the get function to display the node IDs.

```
get(bg.nodes,'ID')
```

```
ans =  
    'M30931'  
    'L07625'  
    'K03454'  
    'M27323'  
    'M15390'
```

See Also

Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox method of a `biograph` object: `set`

get (clustergram)

Purpose Retrieve information about clustergram object

Syntax

```
get(CGobj)
CGStruct = get(CGobj)
PropertyValue = get(CGobj, 'PropertyName')
[Property1Value, Property2Value, ...] = get(CGobj,
    'Property1Name', 'Property2Name', ...)
```

Arguments

<i>CGobj</i>	Clustergram object created with the function clustergram.
--------------	---

PropertyName Property name for a clustergram object.

Description `get(CGobj)` displays all properties and their current values of *CGobj*, a clustergram object.

`CGStruct = get(CGobj)` returns all properties of *CGobj*, a clustergram object, to *CGStruct*, a scalar structure, in which each field name is a property of a clustergram object, and each field contains the value of that property.

`PropertyValue = get(CGobj, 'PropertyName')` returns the value of the specified property of *CGobj*, a clustergram object.

`[Property1Value, Property2Value, ...] = get(CGobj, 'Property1Name', 'Property2Name', ...)` returns the values of the specified properties of *CGobj*, a clustergram object.

Properties of a Clustergram Object

Property	Description
RowLabels	Vector of numbers or cell array of text strings to label the rows in the dendrogram and heat map. Default is a vector of values 1 through M , where M is the number of rows in <i>Data</i> , the matrix of data used by the <code>clustergram</code> function to create the clustergram object.
ColumnLabels	Vector of numbers or cell array of text strings to label the columns in the dendrogram and heat map. Default is a vector of values 1 through N , where N is the number of columns in <i>Data</i> , the matrix of data used by the <code>clustergram</code> function to create the clustergram object.
RowGroupNames	A cell array of text strings containing the names of the row groups exported to a clustergram object created using the Export Group to Workspace command in the Clustergram window.
RowNodeNames	A cell array of text strings containing the names of the row nodes exported to a clustergram object created using the Export Group to Workspace command in the Clustergram window.
ColumnGroupNames	A cell array of text strings containing the names of the column groups exported to a clustergram object created using the Export Group to Workspace command in the Clustergram window.

get (clustergram)

Properties of a Clustergram Object (Continued)

Property	Description
ColumnNodeNames	A cell array of text strings containing the names of the column nodes exported to a clustergram object created using the Export Group to Workspace command in the Clustergram window.
ExprValues	An M -by- N matrix of data, where M and N are the number of row nodes and column nodes respectively, exported to a clustergram object created using the Export Group to Workspace command in the Clustergram window. If the matrix contains gene expression data, typically each row corresponds to a gene and each column corresponds to a sample.
Standardize	<p>Text string that specifies the dimension for standardizing the values in the data. The standardized values are transformed so that the mean is 0 and the standard deviation is 1 in the specified dimension. Possibilities are:</p> <ul style="list-style-type: none">• 'Column (1)' — Standardized along the columns of data.• 'Row (2)' — Standardized along the rows of data.• 'None (3)' — Did not perform standardization.

Properties of a Clustergram Object (Continued)

Property	Description
Cluster	Text string that specifies the dimension for clustering the values in the data. Possibilities are: <ul style="list-style-type: none"> • 'Row (1)' — Clustered rows of data only. • 'Column (2)' — Clustered columns of data only. • 'All (3)' — Clustered rows of data, then cluster columns of row-clustered data.
RowPdist	String or cell array that specifies the distance metric and optional arguments passed to the <code>pdist</code> function (Statistics Toolbox software) used to calculate the pairwise distances between rows. For information on possibilities, see the <code>pdist</code> function.
ColumnPdist	String or cell array that specifies the distance metric and optional arguments passed to the <code>pdist</code> function (Statistics Toolbox software) used to calculate the pairwise distances between columns. For information on possibilities, see the <code>pdist</code> function.
Linkage	String or two-element cell array of strings that specifies the linkage method passed to the <code>linkage</code> function (Statistics Toolbox software) used to create the hierarchical cluster tree for rows and columns. If a two-element cell array of strings, the first element is for linkage between rows, and the second element is for linkage between columns. For information on possibilities, see the <code>linkage</code> function.

get (clustergram)

Properties of a Clustergram Object (Continued)

Property	Description
Dendrogram	Scalar or two-element numeric vector or cell array that specifies the 'colorthreshold' property passed to the dendrogram function (Statistics Toolbox software) used to create the dendrogram plot. If a two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns. For more information, see the dendrogram function.
OptimalLeafOrder	Property that enabled or disabled the optimal leaf ordering calculation, which determines the leaf order that maximizes the similarity between neighboring leaves. Possibilities are 1 (enabled) or 0 (disabled).
LogTrans	Controlled the \log_2 transform of the data from natural scale. Possibilities are 1 (true) or 0 (false).
ColorMap	Either of the following: <ul style="list-style-type: none">• M-by-3 matrix of RGB values• Name or function handle of a function that returns a colormap, such as redgreencmap or redbluecmap
DisplayRange	Positive scalar that specifies the display range of standardized values. For example, if you specify redgreencmap for the 'ColorMap' property, pure red represents values \geq DisplayRange, and pure green represents values $\leq -$ DisplayRange.

Properties of a Clustergram Object (Continued)

Property	Description
SymmetricRange	Property to force the color scale of the heat map to be symmetric around zero. Possibilities are 1 (true) or 0 (false).
Ratio	<p>Either of the following:</p> <ul style="list-style-type: none"> • Scalar • Two-element vector <p>It specifies the ratio of space that the row and column dendrograms occupy relative to the heat map. If <code>Ratio</code> is a scalar, it is the ratio for both dendrograms. If <code>Ratio</code> is a two-element vector, the first element is for the ratio of the row dendrogram width to the heat map width, and the second element is for the ratio of the column dendrogram height to the heat map height. The second element is ignored for one-dimensional clustergrams.</p>
Impute	<p>Any of the following:</p> <ul style="list-style-type: none"> • Name of a function that imputes missing data. • Handle to a function that imputes missing data. • Cell array where the first element is the name of or handle to a function that imputes missing data and the remaining elements are property name/property value pairs used as inputs to the function.

get (clustergram)

Properties of a Clustergram Object (Continued)

Property	Description
RowMarkers	<p>Optional structure array for annotating the groups (clusters) of rows determined by the <code>clustergram</code> function. Each structure in the array represents a group of rows and contains the following fields:</p> <ul style="list-style-type: none">• GroupNumber — Number to annotate the row group.• Annotation — String specifying text to annotate the row group.• Color — String or three-element vector of RGB values specifying a color, which is used to label the row group. For more information on specifying colors, see <code>colorspec</code>. If this field is empty, default is 'blue'.
ColumnMarkers	<p>Optional structure array for annotating groups (clusters) of columns determined by the <code>clustergram</code> function. Each structure in the array represents a group of rows and contains the following fields:</p> <ul style="list-style-type: none">• GroupNumber — Number to annotate the column group.• Annotation — String specifying text to annotate the column group.• Color — String or three-element vector of RGB values specifying a color, which is used to label the column group. For more information on specifying colors, see <code>colorspec</code>. If this field is empty, default is 'blue'.

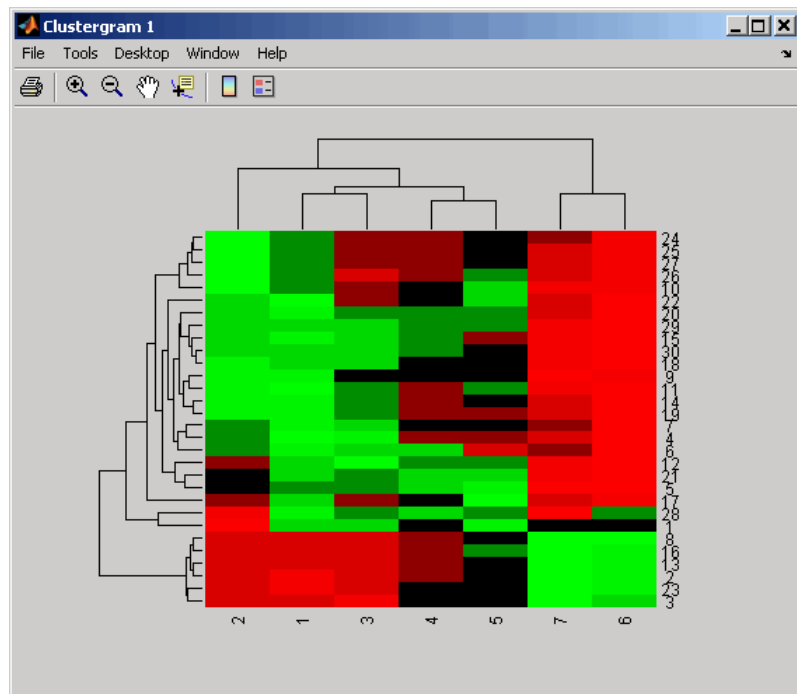
Examples

- 1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeastvalues, a matrix of gene expression data.

```
load filteredyeastdata
```

- 2 Create a clustergram object and display the dendrograms and heat map from the gene expression data in the first 30 rows of the yeastvalues matrix.

```
cgo = clustergram(yeastvalues(1:30,:))  
Clustergram object with 30 rows of nodes and 7 columns of nodes.
```



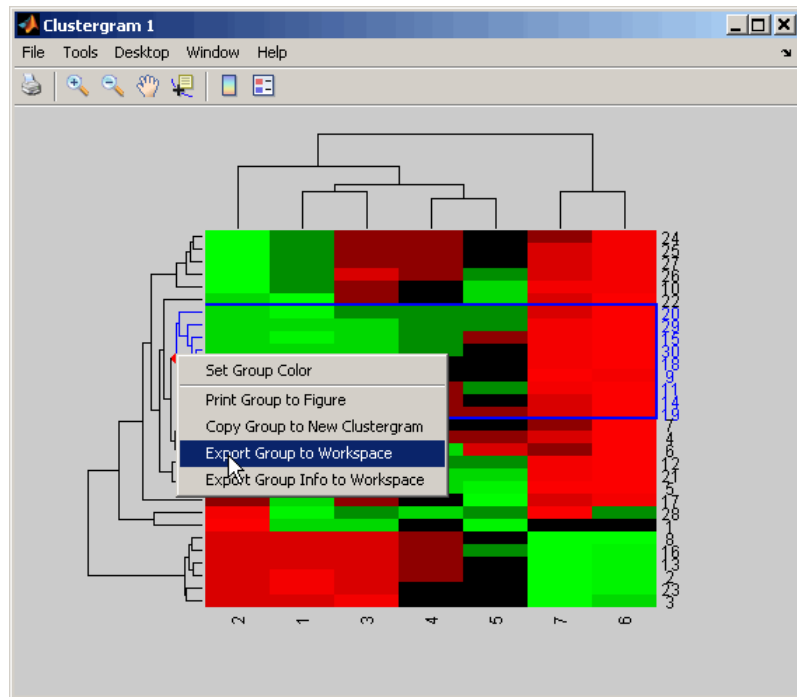
- 3 Use the get method to display the properties of the clustergram object, cgo.

get (clustergram)

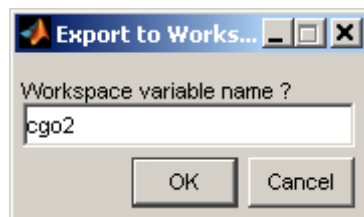
```
get(cgo)

    RowLabels: {30x1 cell}
    ColumnLabels: {7x1 cell}
    Standardize: {'ROW (2)'}
    Cluster: {'ALL (3)'}
    RowPDist: {'Euclidean'}
    ColumnPDist: {'Euclidean'}
    Linkage: {'Average'}
    Dendrogram: {[0]}
    OptimalLeafOrder: 1
    LogTrans: 0
    Colormap: [11x3 double]
    DisplayRange: 3
    SymmetricRange: 1
    Ratio: [0.2000 0.2000]
    Impute: []
    RowMarkers: []
    ColumnMarkers: []
```

- 4 Export a clustergram object of a group (cluster) of rows to the MATLAB Workspace by right-clicking a node in the row dendrogram, and then selecting **Export Group to Workspace**.



- 5 In the Export to Workspace dialog box, type **cgo2** for the Workspace variable name for the clustergram object, and then click **OK**.



- 6 Use the get method to display the properties of cgo2, the clustergram object of the exported group.

```
get(cgo2)
```

get (clustergram)

```
RowGroupNames: {8x1 cell}
RowNodeNames: {9x1 cell}
ColumnGroupNames: {6x1 cell}
ColumnNodeNames: {7x1 cell}
ExprValues: [9x7 double]
Standardize: {'ROW (2)'}
Cluster: {'ALL (3)'}
RowPDist: {'Euclidean'}
ColumnPDist: {'Euclidean'}
Linkage: {'Average'}
Dendrogram: {[0]}
OptimalLeafOrder: 1
LogTrans: 0
Colormap: [11x3 double]
DisplayRange: 3
SymmetricRange: 1
Ratio: [0.2000 0.2000]
Impute: []
RowMarkers: []
ColumnMarkers: []
```

See Also

Bioinformatics Toolbox function: `clustergram` (object constructor)

Bioinformatics Toolbox object: `clustergram` object

Bioinformatics Toolbox methods of a `clustergram` object: `plot`, `set`, `view`

Purpose	Retrieve information about DataMatrix object
Syntax	<pre>get(DMObj) DMStruct = get(DMObj) PropertyValue = get(DMObj, 'PropertyName') [Property1Value, Property2Value, ...] = get(DMObj, 'Property1Name', 'Property2Name', ...)</pre>
Arguments	<p><i>DMObj</i> DataMatrix object, such as created by DataMatrix (object constructor).</p> <p><i>PropertyName</i> Property name of a DataMatrix object.</p>
Return Values	<p><i>DMStruct</i> Scalar structure, in which each field name is a property of a DataMatrix object, and each field contains the value of that property.</p> <p><i>PropertyValue</i> Value of the property specified by <i>PropertyName</i>.</p>
Description	<p><code>get(DMObj)</code> displays all properties and their current values of <i>DMObj</i>, a DataMatrix object.</p> <p><code>DMStruct = get(DMObj)</code> returns all properties of <i>DMObj</i>, a DataMatrix object, to <i>DMStruct</i>, a scalar structure, in which each field name is a property of a DataMatrix object, and each field contains the value of that property.</p> <p><code>PropertyValue = get(DMObj, 'PropertyName')</code> returns the value of the specified property of <i>DMObj</i>, a DataMatrix object.</p> <p><code>[Property1Value, Property2Value, ...] = get(DMObj, 'Property1Name', 'Property2Name', ...)</code> returns the values of the specified properties of <i>DMObj</i>, a DataMatrix object.</p>

get (DataMatrix)

Properties of a DataMatrix Object

Property	Description
Name	String that describes the DataMatrix object. Default is ''.
RowNames	Empty array or cell array of strings that specifies the names for the rows, typically gene names or probe identifiers. The number of elements in the cell array must equal the number of rows in the matrix. Default is an empty array.
ColNames	Empty array or cell array of strings that specifies the names for the columns, typically sample identifiers. The number of elements in the cell array must equal the number of columns in the matrix.
NRows	Positive number that specifies the number of rows in the matrix.
NCols	Positive number that specifies the number of columns in the matrix.
NDims	Positive number that specifies the number of dimensions in the matrix.
ElementClass	String that specifies the class type, such as single or double.

Examples

- 1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: `yeastvalues`, a matrix of gene expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`.

```
load filteredyeastdata
```


- 2 Import the microarray object package so that the `DataMatrix` constructor function will be available.

```
import bioma.data.*
```

- 3 Create a `DataMatrix` object from the gene expression data in the first 30 rows of the `yeastvalues` matrix. Use the `genes` column vector and `times` row vector to specify the row names and column names.

```
dmo = DataMatrix(yeastvalues(1:30,:),genes(1:30,:),times);
```

- 4 Use the `get` method to display the properties of the `DataMatrix` object, `dmo`.

```
get(dmo)
```

```
      Name: ''  
      RowNames: {30x1 cell}  
      ColNames: {' 0' ' 9.5' '11.5' '13.5' '15.5' '18.5'}  
      NRows: 30  
      NCols: 7  
      NDims: 2  
      ElementClass: 'double'
```

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a `DataMatrix` object: `set`

get (phytree)

Purpose Retrieve information about phylogenetic tree object

Syntax `[Value1, Value2, ...] = get(Tree, 'Property1', 'Property2', ...)`
`get(Tree)`
`V = get(Tree)`

Arguments

<i>Tree</i>	Phytree object created with the function <code>phytree</code> .
<i>Name</i>	Property name for a <code>phytree</code> object.

Description `[Value1, Value2, ...] = get(Tree, 'Property1', 'Property2', ...)` returns the specified properties from a `phytree` object (`Tree`).

Properties for a `phytree` object are listed in the following table.

Property	Description
NumLeaves	Number of leaves
NumBranches	Number of branches
NumNodes	Number of nodes (NumLeaves + NumBranches)
Pointers	Branch to leaf/branch connectivity list
Distances	Edge length for every leaf/branch
LeafNames	Names of the leaves
BranchNames	Names of the branches
NodeNames	Names of all the nodes

`get(Tree)` displays all property names and their current values for a `phytree` object (`Tree`).

`V = get(Tree)` returns a structure where each field name is the name of a property of a phytree object (*Tree*) and each field contains the value of that property.

Examples

- 1 Read in a phylogenetic tree from a file.

```
tr = phytread('pf00002.tree')
```

Phylogenetic tree object with 33 leaves (32 branches)

- 2 Get the names of the leaves.

```
protein_names = get(tr, 'LeafNames')
```

```
protein_names =
```

```
    'Q9YHC6_RANRI/126-382'
```

```
    'VIPR1_RAT/140-397'
```

```
    'VIPR_CARAU/100-359'
```

```
    ...
```

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreeread`

Bioinformatics Toolbox methods of `phytree` object: `getbyname`, `select`

getancestors (biograph)

Purpose Find ancestors in biograph object

Syntax `Nodes = getancestors(BiographNode)`
`Nodes = getancestors(BiographNode, NumGenerations)`

Arguments

<i>BiographNode</i>	Node in a biograph object.
<i>NumGenerations</i>	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*).

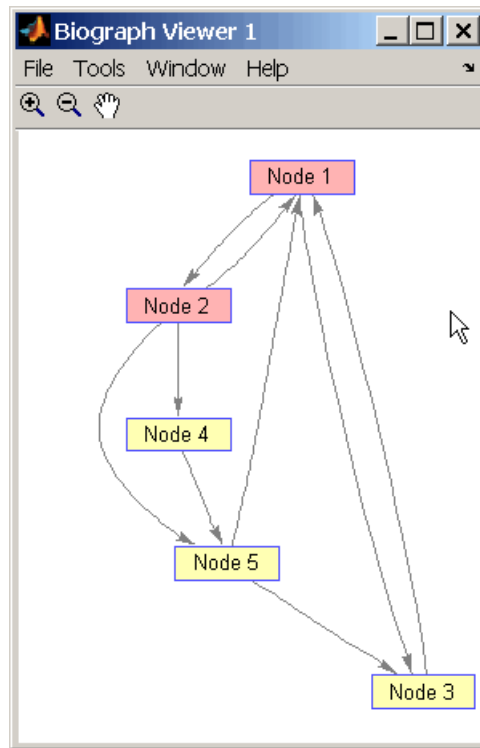
Examples

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.

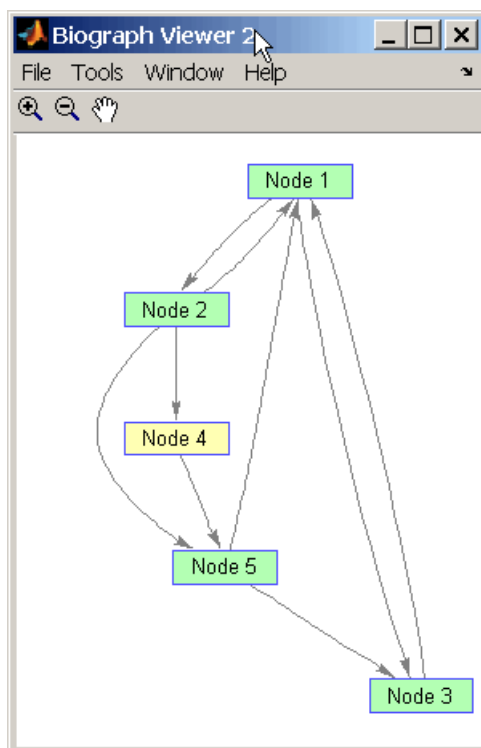
```
ancNodes = getancestors(bg.nodes(2));  
set(ancNodes, 'Color', [1 .7 .7]);  
bg.view;
```



3 Find two generations of ancestors for node 2.

```
ancNodes = getancestors(bg.nodes(2),2);  
set(ancNodes,'Color',[.7 1 .7]);  
bg.view;
```

getancestors (biograph)



See Also

Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object: `dolayout`, `get`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `set`, `view`

Purpose Find terms that are ancestors of specified Gene Ontology (GO) term

Syntax

```
AncestorIDs = getancestors(GeneontObj, ID)
[AncestorIDs, Counts] = getancestors(GeneontObj, ID)
... = getancestors(..., 'Height', HeightValue, ...)
... = getancestors(..., 'Relationtype', RelationtypeValue,
    ...)
... = getancestors(..., 'Exclude', ExcludeValue, ...)
```

Arguments

<i>GeneontObj</i>	A geneont object, such as created by the geneont function.
<i>ID</i>	GO term identifier or vector of identifiers.
<i>HeightValue</i>	Positive integer specifying the number of levels to search upward in the gene ontology.
<i>RelationtypeValue</i>	String specifying the relationship types to search for in the gene ontology. Choices are: <ul style="list-style-type: none">• 'is_a'• 'part_of'• 'both' (default)
<i>ExcludeValue</i>	Controls excluding <i>ID</i> , the original queried term(s), from the output <i>AncestorIDs</i> , unless the term was reached while searching the gene ontology. Choices are true or false (default).

getancestors (geneont)

Return Values

<i>AncestorIDs</i>	Vector of GO term identifiers including <i>ID</i> .
<i>Counts</i>	Column vector with the same number of elements as terms in <i>GeneontObj</i> , indicating the number of times each ancestor is found.

Description

AncestorIDs = `getancestors(GeneontObj, ID)` searches *GeneontObj*, a geneont object, for GO terms that are ancestors of the GO term(s) specified by *ID*, which is a GO term identifier or vector of identifiers. It returns *AncestorIDs*, a vector of GO term identifiers including *ID*. *ID* is a nonnegative integer or a vector containing nonnegative integers.

`[AncestorIDs, Counts]` = `getancestors(GeneontObj, ID)` also returns the number of times each ancestor is found. *Counts* is a column vector with the same number of elements as terms in *GeneontObj*.

Tip The *Counts* return value is useful when tallying counts in gene enrichment studies. For more information, see the Gene Ontology Enrichment in Microarray Data demo.

`... = getancestors(..., 'PropertyName', PropertyValue, ...)` calls `getancestors` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`... = getancestors(..., 'Height', HeightValue, ...)` searches up through a specified number of levels, *HeightValue*, in the gene ontology. *HeightValue* is a positive integer. Default is `Inf`.


```
... = getancestors(..., 'Relationship', RelationshipValue,  
...) searches for specified relationship types, RelationshipValue, in  
the gene ontology. RelationshipValue is a string. Choices are 'is_a',  
'part_of', or 'both' (default).
```

```
... = getancestors(..., 'Exclude', ExcludeValue, ...)  
controls excluding ID, the original queried term(s), from the output  
AncestorIDs, unless the term was reached while searching the gene  
ontology. Choices are true or false (default).
```

Examples

- 1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GO = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of terms in the database.

```
Gene Ontology object with 24316 Terms.
```

- 2 Retrieve the ancestors of the Gene Ontology term with an identifier of 46680.

```
ancestors = getancestors(GO,46680)
```

```
ancestors =  
    8150  
    9636  
   17085  
   42221  
   46680  
   50896
```

- 3 Create a subordinate Gene Ontology.

```
subontology = GO(ancestors)
```

```
Gene Ontology object with 6 Terms.
```

getancestors (geneont)

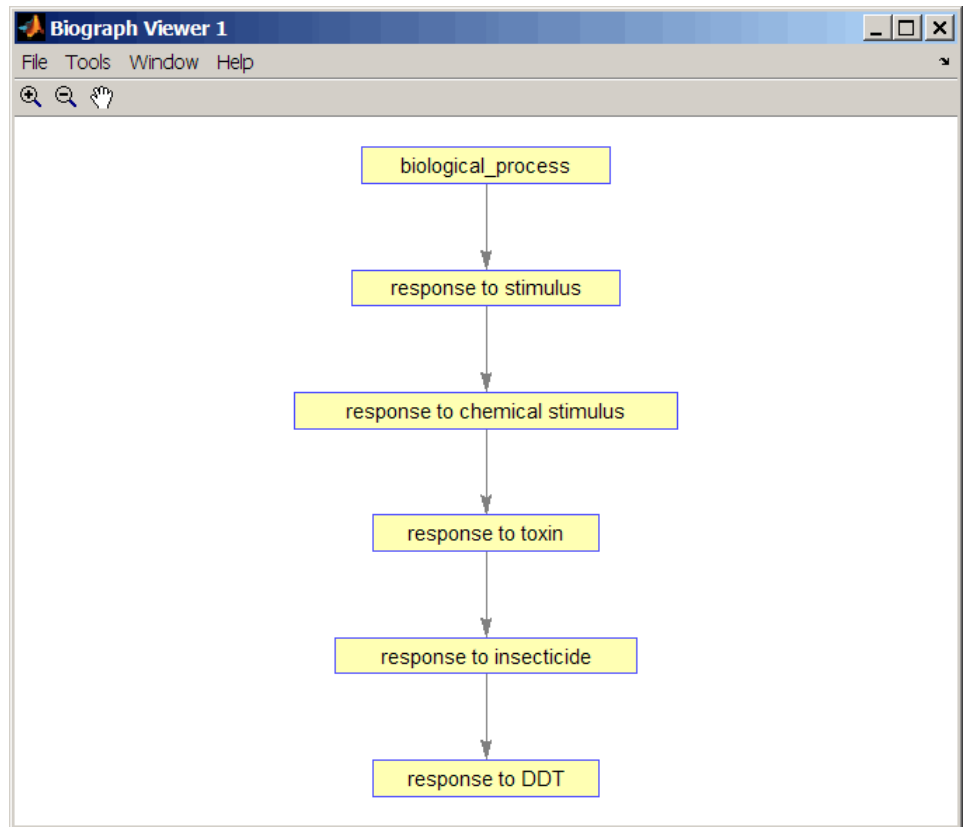
- 4** Create a report of the subordinate Gene Ontology terms.

```
[cm acc rels] = getmatrix(subontology);  
rpt = [num2goid(acc) get(subontology.Terms, 'name')];  
disp(sprintf('%s --> %s\n',rpt{:}))
```

```
G0:0008150 --> biological_process  
G0:0009636 --> response to toxin  
G0:0017085 --> response to insecticide  
G0:0042221 --> response to chemical stimulus  
G0:0046680 --> response to DDT  
G0:0050896 --> response to stimulus
```

- 5** View relationships of the subordinate Gene Ontology using the biograph function and methods.

```
BG = biograph(cm, get(subontology.Terms, 'name'));  
view(BG)
```



See Also

Bioinformatics Toolbox functions: `geneont` (object constructor), `goannotread`, `num2goid`

Bioinformatics Toolbox methods of `geneont` object: `getdescendants`, `getmatrix`, `getrelatives`

getbyname (phytree)

Purpose Branches and leaves from phytree object

Syntax
`S = getbyname(Tree, Expression)`
`S = getbyname(Tree, String, 'Exact', true)`

Arguments

Tree phytree object created by phytree function (object constructor).

Expression Regular expression. When *Expression* is a cell array of strings, getbyname returns a matrix where every column corresponds to every query in *Expression*.

For information about the symbols that you can use in a matching regular expression, see the MATLAB function regexp.

String String or cell array of strings.

Description

`S = getbyname(Tree, Expression)` returns a logical vector (*S*) of size NumNodes-by-1 with the node names of a phylogenetic tree (*Tree*) that match the regular expression (*Expression*) regardless of letter case.

`S = getbyname(Tree, String, 'Exact', true)` looks for exact string matches and ignores case. When *String* is a cell array of char strings, getbyname returns a vector with indices.

Examples

1 Load a phylogenetic tree created from a protein family.

```
tr = phytreeread('pf00002.tree');
```

2 Select all the 'mouse' and 'human' proteins.

```
sel = getbyname(tr,{'mouse','human'});  
view(tr,any(sel,2));
```

See Also

Bioinformatics Toolbox function: phytree (object constructor)

Bioinformatics Toolbox methods of phytree object: get, prune, select

getcanonical (phytree)

Purpose Calculate canonical form of phylogenetic tree

Syntax `Pointers = getcanonical(Tree)`
`[Pointers, Distances, Names] = getcanonical(Tree)`

Arguments

<i>Tree</i>	phytree object created by phytree function (object constructor).
-------------	--

Description `Pointers = getcanonical(Tree)` returns the pointers for the canonical form of a phylogenetic tree (*Tree*). In a canonical tree the leaves are ordered alphabetically and the branches are ordered first by their width and then alphabetically by their first element. A canonical tree is isomorphic to all the trees with the same skeleton independently of the order of their leaves and branches.

`[Pointers, Distances, Names] = getcanonical(Tree)` returns, in addition to the pointers described above, the reordered distances (*Distances*) and node names (*Names*).

Examples

- 1 Create two phylogenetic trees with the same skeleton but slightly different distances.

```
b = [1 2; 3 4; 5 6; 7 8;9 10];  
tr_1 = phytree(b,[.1 .2 .3 .3 .4 ]');  
tr_2 = phytree(b,[.2 .1 .2 .3 .4 ]');
```

- 2 Plot the trees.

```
plot(tr_1)  
plot(tr_2)
```

- 3 Check whether the trees have an isomorphic construction.

```
isequal(getcanonical(tr_1),getcanonical(tr_2))
```

```
ans =  
    1
```

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreeread`

Bioinformatics Toolbox methods of `phytree` object: `getbyname`, `select`, `subtree`

getdescendants (biograph)

Purpose Find descendants in biograph object

Syntax `Nodes = getdescendants(BiographNode)`
`Nodes = getdescendants(BiographNode, NumGenerations)`

Arguments

BiographNode Node in a biograph object.

NumGenerations Number of generations. Enter a positive integer.

Description `Nodes = getdescendants(BiographNode)` finds a given node (*BiographNode*) all of its direct descendants.

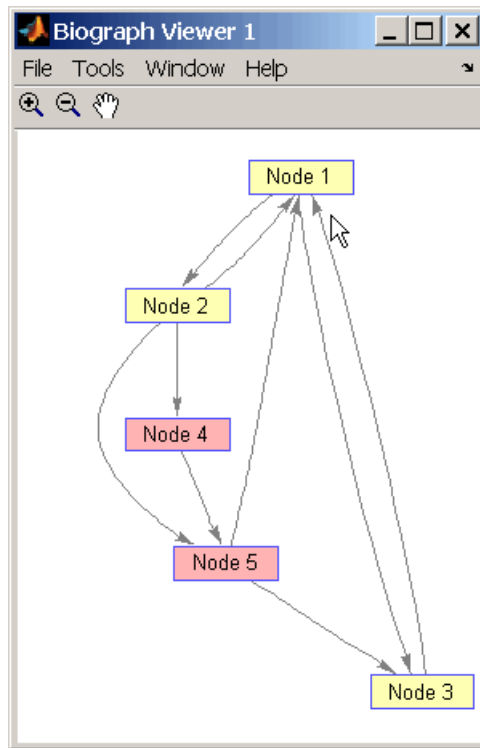
`Nodes = getdescendants(BiographNode, NumGenerations)` finds the node (*BiographNode*) and all of its direct descendants up to a specified number of generations (*NumGenerations*).

Examples **1** Create a biograph object.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];  
bg = biograph(cm)
```

2 Find one generation of descendants for node 4.

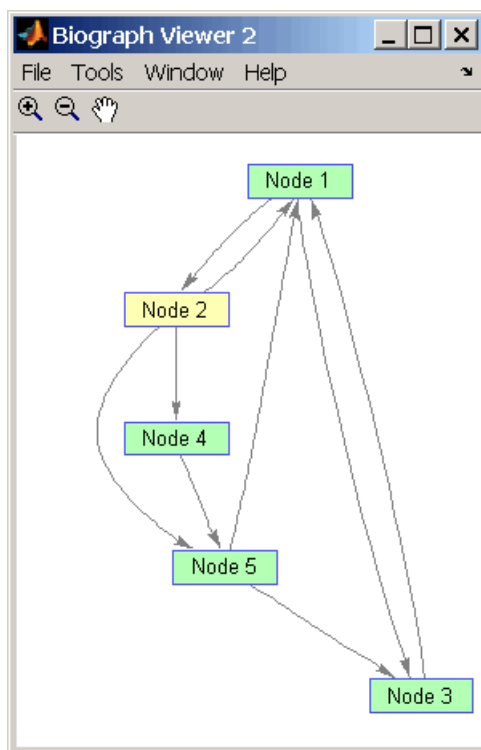
```
desNodes = getdescendants(bg.nodes(4));  
set(desNodes, 'Color', [1 .7 .7]);  
bg.view;
```

3 Find two generations of descendants for node 4.

```
desNodes = getdescendants(bg.nodes(4),2);  
set(desNodes,'Color',[.7 1 .7]);  
bg.view;
```

getdescendants (biograph)



See Also

Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object: `dolayout`, `get`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `set`, `view`

Purpose Find terms that are descendants of specified Gene Ontology (GO) term

Syntax

```
DescendantIDs = getdescendants(GeneontObj, ID)  
[DescendantIDs, Counts] = getdescendants(GeneontObj, ID)  
... = getdescendants(..., 'Depth', DepthValue, ...)  
... = getdescendants(..., 'Relationtype',  
RelationtypeValue,  
...)  
... = getdescendants(..., 'Exclude', ExcludeValue, ...)
```

Arguments

<i>GeneontObj</i>	A geneont object, such as created by the geneont function.
<i>ID</i>	GO term identifier or vector of identifiers.
<i>DepthValue</i>	Positive integer specifying the number of levels to search downward in the gene ontology.
<i>RelationtypeValue</i>	String specifying the relationship types to search for in the gene ontology. Choices are: <ul style="list-style-type: none">• 'is_a'• 'part_of'• 'both' (default)
<i>ExcludeValue</i>	Controls excluding <i>ID</i> , the original queried term(s), from the output <i>DescendantIDs</i> , unless the term was reached while searching the gene ontology. Choices are true or false (default).

Return Values

<i>DescendantIDs</i>	Vector of GO term identifiers including <i>ID</i> .
<i>Counts</i>	Column vector with the same number of elements as terms in <i>GeneontObj</i> , indicating the number of times each descendant is found.

getdescendants (geneont)

Description

DescendantIDs = `getdescendants(GeneontObj, ID)` searches *GeneontObj*, a geneont object, for GO terms that are descendants of the GO term(s) specified by *ID*, which is a GO term identifier or vector of identifiers. It returns *DescendantIDs*, a vector of GO term identifiers including *ID*. *ID* is a nonnegative integer or a vector containing nonnegative integers.

`[DescendantIDs, Counts]` = `getdescendants(GeneontObj, ID)` also returns the number of times each descendant is found. *Counts* is a column vector with the same number of elements as terms in *GeneontObj*.

Tip The *Counts* return value is useful when tallying counts in gene enrichment studies. For more information, see the Gene Ontology Enrichment in Microarray Data demo.

`... = getdescendants(..., 'PropertyName', PropertyValue, ...)` calls `getdescendants` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`... = getdescendants(..., 'Depth', DepthValue, ...)` searches down through a specified number of levels, *DepthValue*, in the gene ontology. *DepthValue* is a positive integer. Default is Inf.

`... = getdescendants(..., 'Relationtype', RelationtypeValue, ...)` searches for specified relationship types, *RelationtypeValue*, in the gene ontology. *RelationtypeValue* is a string. Choices are 'is_a', 'part_of', or 'both' (default).

`... = getdescendants(..., 'Exclude', ExcludeValue, ...)` controls excluding *ID*, the original queried term(s), from the output *DescendantIDs*, unless the term was found while searching the gene ontology. Choices are true or false (default).

Examples

- 1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GO = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of terms in the database.

```
Gene Ontology object with 24316 Terms.
```

- 2 Retrieve the descendants of the mitochondrial respiratory chain GO term with an identifier of 5746.

```
descendants = getdescendants(GO,5746)
```

```
descendants =
```

```
5746  
5747  
5749  
5750  
5751  
42652  
42653
```

- 3 Create a subordinate Gene Ontology.

```
subontology = GO(descendants)
```

```
Gene Ontology object with 7 Terms.
```

- 4 Create a report of the subordinate Gene Ontology terms.

```
[cm acc rels] = getmatrix(subontology);  
rpt = [num2goid(acc) get(subontology.Terms, 'name')];  
disp(sprintf('%s --> %s\n',rpt{:}))
```

```
GO:0005746 --> mitochondrial respiratory chain  
GO:0005747 --> mitochondrial respiratory chain complex I
```

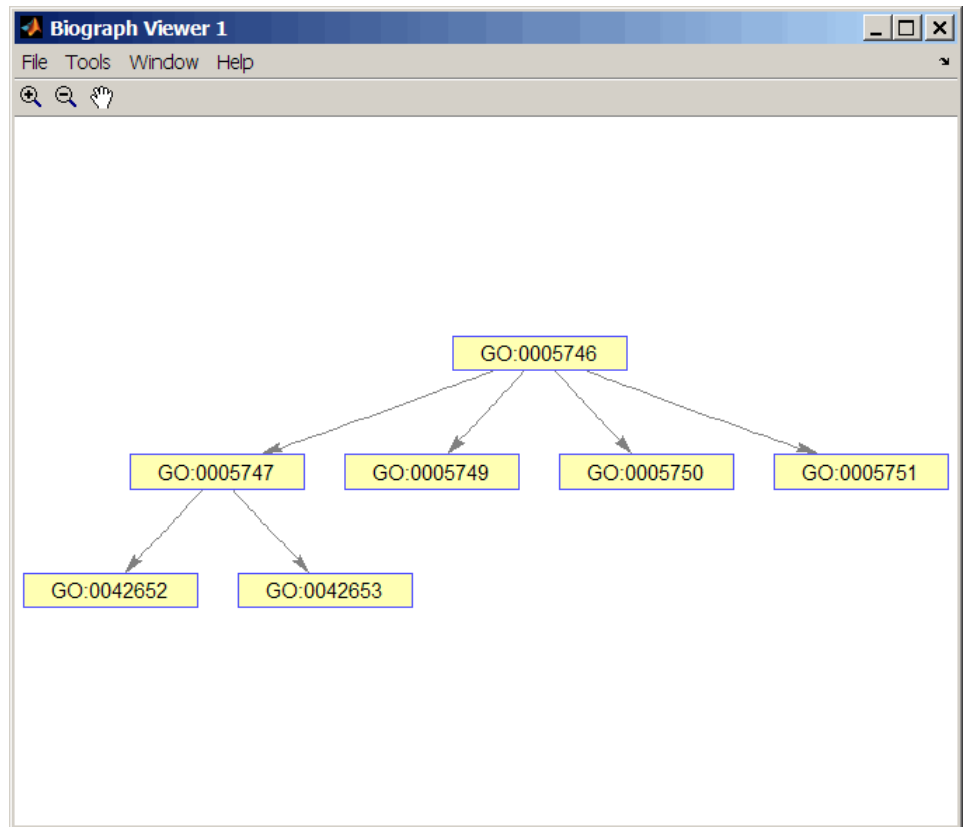
getdescendants (geneont)

```
GO:0005749 --> mitochondrial respiratory chain complex II
GO:0005750 --> mitochondrial respiratory chain complex III
GO:0005751 --> mitochondrial respiratory chain complex IV
GO:0042652 --> mitochondrial respiratory chain complex I, peripheral segment
GO:0042653 --> mitochondrial respiratory chain complex I, membrane segment
```

- 5** View relationships of the subordinate Gene Ontology using the `biograph` function and methods.

```
BG = biograph(cm, num2goid(acc));
view(BG)
```

getdescendants (geneont)



See Also

Bioinformatics Toolbox functions: `geneont` (object constructor), `goannotread`, `num2go`

Bioinformatics Toolbox methods of `geneont` object: `getancestors`, `getmatrix`, `getrelatives`

getedgesbynodeid (biograph)

Purpose Get handles to edges in biograph object

Syntax `Edges = getedgesbynodeid(BGobj, SourceIDs, SinkIDs)`

Arguments

<i>BGobj</i>	Biograph object.
<i>SourceIDs</i> , <i>SinkIDs</i>	Enter a cell string, or an empty cell array (gets all edges).

Description `Edges = getedgesbynodeid(BGobj, SourceIDs, SinkIDs)` gets the handles to the edges that connect the specified source nodes (*SourceIDs*) to the specified sink nodes (*SinkIDs*) in a biograph object.

Example

1 Create a biograph object for the Hominidae family.

```
species = {'Homo', 'Pan', 'Gorilla', 'Pongo', 'Baboon', ...  
          'Macaca', 'Gibbon'};  
cm = magic(7)>25 & 1-eye(7);  
bg = biograph(cm, species);
```

2 Find all the edges that connect to the Homo node.

```
EdgesIn = getedgesbynodeid(bg, [], 'Homo');  
EdgesOut = getedgesbynodeid(bg, 'Homo', []);  
set(EdgesIn, 'LineColor', [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 function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object: `dolayout`, `get`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `set`, `view`

getmatrix (biograph)

Purpose Get connection matrix from biograph object

Syntax `[Matrix, ID, Distances] = getmatrix(BGObj)`

Arguments *BGObj* Biograph object created by biograph (object constructor).

Description `[Matrix, ID, Distances] = getmatrix(BGObj)` converts the biograph object, *BiographObj*, into a logical sparse matrix, *Matrix*, in which 1 indicates that a node (row index) is connected to another node (column index). *ID* is a cell array of strings listing the ID properties for each node, and corresponds to the rows and columns of *Matrix*. *Distances* is a column vector with one entry for every nonzero entry in *Matrix* traversed column-wise and representing the respective Weight property for each edge.

Examples

```
cm = [0 1 1 0 0;2 0 0 4 4;4 0 0 0 0;0 0 0 0 2;4 0 5 0 0];
bg = biograph(cm);
[cm, IDs, dist] = getmatrix(bg)
```

See Also Bioinformatics Toolbox function: biograph (object constructor)
Bioinformatics Toolbox object: biograph object
Bioinformatics Toolbox methods of a biograph object: dolayout, getancestors, getdescendants, getedgesbynodeid, getnodesbyid, getrelatives, view

Purpose

Convert geneont object into relationship matrix

Syntax

```
[Matrix, ID, Relationship] = getmatrix(GeneontObj)
```

Arguments

GeneontObj geneont object created by geneont (object constructor)

Description

[*Matrix*, *ID*, *Relationship*] = getmatrix(*GeneontObj*) converts a geneont object, *GeneontObj*, into *Matrix*, a matrix of relationship values between nodes (row and column indices), in which 0 indicates no relationship, 1 indicates an “is_a” relationship, and 2 indicates a “part_of” relationship. *ID* is a column vector listing Gene Ontology IDs that correspond to the rows and columns of *Matrix*. *Relationship* is a cell array of strings defining the types of relationships.

Examples

```
GO = geneont('LIVE',true);  
[MATRIX, ID, REL] = getmatrix(GO);
```

See Also

Bioinformatics Toolbox functions: geneont (object constructor), goannotread, num2goid

Bioinformatics Toolbox object: geneont object

Bioinformatics Toolbox methods of geneont object: getancestors, getdescendants, getmatrix, getrelatives

getmatrix (phytree)

Purpose	Convert phytree object into relationship matrix
Syntax	<code>[Matrix, ID, Distances] = getmatrix(PhytreeObj)</code>
Arguments	<i>PhytreeObj</i> phytree object created by phytree (object constructor).
Description	<code>[Matrix, ID, Distances] = getmatrix(PhytreeObj)</code> converts a phytree object, <i>PhytreeObj</i> , into a logical sparse matrix, <i>Matrix</i> , in which 1 indicates that a branch node (row index) is connected to its child (column index). The child can be either another branch node or a leaf node. <i>ID</i> is a column vector of strings listing the labels that correspond to the rows and columns of <i>Matrix</i> , with the labels from 1 to <i>Number of Leaves</i> being the leaf nodes, then the labels from <i>Number of Leaves</i> + 1 to <i>Number of Leaves</i> + <i>Number of Branches</i> being the branch nodes, and the label for the last branch node also being the root node. <i>Distances</i> is a column vector with one entry for every nonzero entry in <i>Matrix</i> traversed column-wise and representing the distance between the branch node and the child.
Examples	<pre>T = phytreeread('pf00002.tree') [MATRIX, ID, DIST] = getmatrix(T);</pre>
See Also	Bioinformatics Toolbox functions: phytree (object constructor), phytreetool Bioinformatics Toolbox object: phytree object Bioinformatics Toolbox methods of phytree object: get, pdist, prune

Purpose Create Newick-formatted string

Syntax

```
String = getnewickstr(Tree)
getnewickstr(..., 'PropertyName', PropertyValue,...)
getnewickstr(..., 'Distances', DistancesValue)
getnewickstr(..., 'BranchNames', BranchNamesValue)
```

Arguments

<i>Tree</i>	Phytree object created with the function <code>phytree</code> .
<i>DistancesValue</i>	Property to control including or excluding distances in the output. Enter either <code>true</code> (include distances) or <code>false</code> (exclude distances). Default is <code>true</code> .
<i>BranchNamesValue</i>	Property to control including or excluding branch names in the output. Enter either <code>true</code> (include branch names) or <code>false</code> (exclude branch names). Default is <code>false</code> .

Description

`String = getnewickstr(Tree)` returns the Newick formatted string of a phylogenetic tree object (*Tree*).

`getnewickstr(..., 'PropertyName', PropertyValue,...)` defines optional properties using property name/value pairs.

`getnewickstr(..., 'Distances', DistancesValue)`, when *DistancesValue* is `false`, excludes the distances from the output.

`getnewickstr(..., 'BranchNames', BranchNamesValue)`, when *BranchNamesValue* is `true`, includes the branch names in the output.

References

Information about the Newick tree format.

<http://evolution.genetics.washington.edu/phylip/newicktree.html>

getnewickstr (phytree)

Examples

1 Create some random sequences.

```
seqs = int2nt(ceil(rand(10)*4));
```

2 Calculate pairwise distances.

```
dist = seqpdist(seqs, 'alpha', 'nt');
```

3 Construct a phylogenetic tree.

```
tree = seqlinkage(dist);
```

4 Get the Newick string.

```
str = getnewickstr(tree)
```

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreeread`, `phytreetool`, `phytreewrite`, `seqlinkage`

Bioinformatics Toolbox methods of `phytree` object: `get`, `getbyname`, `getcanonical`

Purpose

Get handles to nodes

Syntax

```
NodesHandles = getnodesbyid(BGobj,NodeIDs)
```

Arguments

BGobj Biograph object.

NodeIDs Enter a cell string of node identifications.

Description

NodesHandles = getnodesbyid(*BGobj*,*NodeIDs*) gets the handles for the specified nodes (*NodeIDs*) in a biograph object.

Example

- 1 Create a biograph object.

```
species = {'Homosapiens', 'Pan', 'Gorilla', 'Pongo', 'Baboon', ...  
          'Macaca', 'Gibbon'};  
cm = magic(7)>25 & 1-eye(7);  
bg = biograph(cm, species)
```

- 2 Find the handles to members of the Cercopithecidae family and members of the Hominidae family.

```
Cercopithecidae = {'Macaca', 'Baboon'};  
Hominidae = {'Homosapiens', 'Pan', 'Gorilla', 'Pongo'};  
CercopithecidaeNodes = getnodesbyid(bg, Cercopithecidae);  
HominidaeNodes = getnodesbyid(bg, Hominidae);
```

- 3 Color the families differently and draw a graph.

See Also

Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a biograph object: `dolayout`, `get`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `set`, `view`

getrelatives (biograph)

Purpose Find relatives in biograph object

Syntax `Nodes = getrelatives(BiographNode)`
`Nodes = getrelatives(BiographNode, NumGenerations)`

Arguments

<i>BiographNode</i>	Node in a biograph object.
<i>NumGenerations</i>	Number of generations. Enter a positive integer.

Description `Nodes = getrelatives(BiographNode)` finds all the direct relatives for a given node (*BiographNode*).

`Nodes = getrelatives(BiographNode, NumGenerations)` finds the direct relatives for a given node (*BiographNode*) up to a specified number of generations (*NumGenerations*).

Examples

1 Create a biograph object.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];  
bg = biograph(cm)
```

2 Find all nodes interacting with node 1.

```
intNodes = getrelatives(bg.nodes(1));  
set(intNodes, 'Color', [.7 .7 1]);  
bg.view;
```

See Also

Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a biograph object: `dolayout`, `get`, `getancestors`, `getdescendants`, `getedgesbynodeid`, `getnodesbyid`, `getrelatives`, `set`, `view`

Purpose

Find terms that are relatives of specified Gene Ontology (GO) term

Syntax

```
RelativeIDs = getrelatives(GeneontObj, ID)  
[RelativeIDs, Counts] = getrelatives(GeneontObj, ID)  
... = getrelatives(..., 'Height', HeightValue, ...)  
... = getrelatives(..., 'Depth', DepthValue, ...)  
... = getrelatives(..., 'Levels', LevelsValue, ...)  
... = getrelatives(..., 'Relationtype', RelationtypeValue,  
    ...)  
... = getrelatives(..., 'Exclude', ExcludeValue, ...)
```

Arguments

<i>GeneontObj</i>	A geneont object, such as created by the geneont function.
<i>ID</i>	GO term identifier or vector of identifiers.
<i>HeightValue</i>	Positive integer specifying the number of levels to search upward in the gene ontology.
<i>DepthValue</i>	Positive integer specifying the number of levels to search downward in the gene ontology.
<i>LevelsValue</i>	Positive integer specifying the number of levels up and down to search in the gene ontology. When specified, it overrides <i>HeightValue</i> and <i>DepthValue</i> .
<i>RelationtypeValue</i>	String specifying the relationship types to search for in the gene ontology. Choices are: <ul style="list-style-type: none">• 'is_a'• 'part_of'• 'both' (default)
<i>ExcludeValue</i>	Controls excluding <i>ID</i> , the original queried term(s), from the output <i>RelativeIDs</i> , unless the term was reached while searching the gene ontology. Choices are true or false (default).

getrelatives (geneont)

Return Values

<i>RelativeIDs</i>	Vector of GO term identifiers including <i>ID</i> .
<i>Counts</i>	Column vector with the same number of elements as terms in <i>GeneontObj</i> , indicating the number of times each relative is found.

Description

RelativeIDs = `getrelatives(GeneontObj, ID)` searches *GeneontObj*, a geneont object, for GO terms that are relatives of the GO term(s) specified by *ID*, which is a GO term identifier or vector of identifiers. It returns *RelativeIDs*, a vector of GO term identifiers including *ID*. *ID* is a nonnegative integer or a vector containing nonnegative integers.

`[RelativeIDs, Counts]` = `getrelatives(GeneontObj, ID)` also returns the number of times each relative is found. *Counts* is a column vector with the same number of elements as terms in *GeneontObj*.

Tip The *Counts* return value is useful when tallying counts in gene enrichment studies. For more information, see the Gene Ontology Enrichment in Microarray Data demo.

`... = getrelatives(..., 'PropertyName', PropertyValue, ...)` calls `getrelatives` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`... = getrelatives(..., 'Height', HeightValue, ...)` searches up through a specified number of levels, *HeightValue*, in the gene ontology. *HeightValue* is a positive integer. Default is 1.

... = getrelatives(..., 'Depth', *DepthValue*, ...) searches down through a specified number of levels, *DepthValue*, in the gene ontology. *DepthValue* is a positive integer. Default is 1.

... = getrelatives(..., 'Levels', *LevelsValue*, ...) searches up and down through a specified number of levels, *LevelsValue*, in the gene ontology. *LevelsValue* is a positive integer. When specified, it overrides *HeightValue* and *DepthValue*.

... = getrelatives(..., 'Relationship', *RelationshipValue*, ...) searches for specified relationship types, *RelationshipValue*, in the gene ontology. *RelationshipValue* is a string. Choices are 'is_a', 'part_of', or 'both' (default).

... = getrelatives(..., 'Exclude', *ExcludeValue*, ...) controls excluding *ID*, the original queried term(s), from the output *RelativeIDs*, unless a term was found while searching the gene ontology. Choices are true or false (default).

Examples

- 1 Download the current version of the Gene Ontology database from the Web into a geneont object in the MATLAB software.

```
GO = geneont('LIVE', true)
```

The MATLAB software creates a geneont object and displays the number of terms in the database.

```
Gene Ontology object with 24316 Terms.
```

- 2 Retrieve the immediate relatives for the mitochondrial membrane GO term with an identifier of 31966.

```
relatives = getrelatives(GO,31966,'levels',1)
```

```
relatives =
```

```
5740  
5741  
5743
```

getrelatives (geneont)

```
31090
31966
44429
44455
```

- 3** Create a subordinate Gene Ontology.

```
subontology = GO(relatives)
```

Gene Ontology object with 7 Terms.

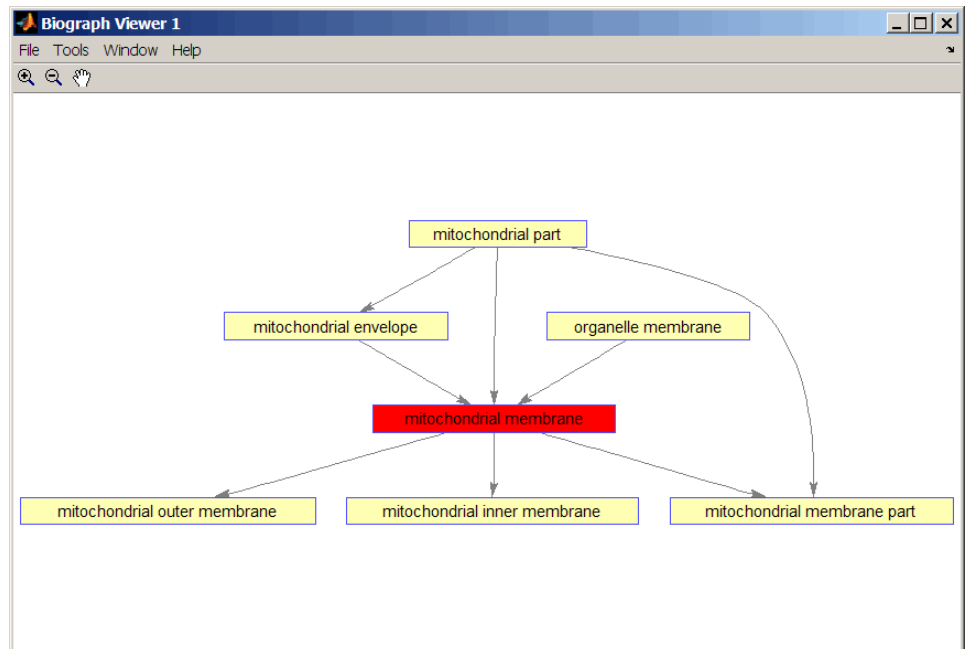
- 4** Create a report of the subordinate Gene Ontology terms.

```
[cm acc rels] = getmatrix(subontology);
rpt = [num2goid(acc) get(subontology.Terms, 'name')];
disp(sprintf('%s --> %s\n',rpt{:}))
```

```
GO:0005740 --> mitochondrial envelope
GO:0005741 --> mitochondrial outer membrane
GO:0005743 --> mitochondrial inner membrane
GO:0031090 --> organelle membrane
GO:0031966 --> mitochondrial membrane
GO:0044429 --> mitochondrial part
GO:0044455 --> mitochondrial membrane part
```

- 5** View relationships of the subordinate Gene Ontology using the `biograph` function and methods and color the mitochondrial membrane GO terms red.

```
BG = biograph(cm, get(subontology.Terms, 'name'));
BG.nodes(acc==31966).Color = [1 0 0];
view(BG)
```



- 6 Retrieve all relatives for the mitochondrial outer membrane GO term with an identifier of 5741.

```
relatives = getrelatives(GO,5741,'levels',inf);
```

- 7 Create a subordinate Gene Ontology.

```
subontology = GO(relatives)
```

Gene Ontology object with 28 Terms.

- 8 View relationships using the biograph functions.

```
[cm acc rels] = getmatrix(subontology);  
BG = biograph(cm, get(subontology.Terms, 'name'));  
BG.nodes(acc==5741).Color = [1 0 0];
```

getrelatives (geneont)

`view(BG)`

See Also

Bioinformatics Toolbox functions: `geneont` (object constructor),
`goannotread`, `num2goid`

Bioinformatics Toolbox methods of `geneont` object: `getancestors`,
`getdescendants`, `getmatrix`

Purpose	Test DataMatrix objects for greater than				
Syntax	$T = \text{gt}(DMObj1, DMObj2)$ $T = DMObj1 > DMObj2$ $T = \text{gt}(DMObj1, B)$ $T = DMObj1 > B$ $T = \text{gt}(B, DMObj1)$ $T = B > DMObj1$				
Arguments	<table><tr><td>$DMObj1, DMObj2$</td><td>DataMatrix objects, such as created by DataMatrix (object constructor).</td></tr><tr><td>B</td><td>MATLAB numeric or logical array.</td></tr></table>	$DMObj1, DMObj2$	DataMatrix objects, such as created by DataMatrix (object constructor).	B	MATLAB numeric or logical array.
$DMObj1, DMObj2$	DataMatrix objects, such as created by DataMatrix (object constructor).				
B	MATLAB numeric or logical array.				
Return Values	T Logical matrix of the same size as $DMObj1$ and $DMObj2$ or $DMObj1$ and B . It contains logical 1 (true) where elements in the first input are greater than the corresponding element in the second input, and logical 0 (false) otherwise.				
Description	<p>$T = \text{gt}(DMObj1, DMObj2)$ or the equivalent $T = DMObj1 > DMObj2$ compares each element in DataMatrix object $DMObj1$ to the corresponding element in DataMatrix object $DMObj2$, and returns T, a logical matrix of the same size as $DMObj1$ and $DMObj2$, containing logical 1 (true) where elements in $DMObj1$ are greater than the corresponding element in $DMObj2$, and logical 0 (false) otherwise. $DMObj1$ and $DMObj2$ must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). $DMObj1$ and $DMObj2$ can have different Name properties.</p> <p>$T = \text{gt}(DMObj1, B)$ or the equivalent $T = DMObj1 > B$ compares each element in DataMatrix object $DMObj1$ to the corresponding element in B, a numeric or logical array, and returns T, a logical matrix of the same size as $DMObj1$ and B, containing logical 1 (true) where elements</p>				

gt (DataMatrix)

in *DMObj1* are greater than the corresponding element in *B*, and logical 0 (false) otherwise. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

$T = \text{gt}(B, \text{DMObj1})$ or the equivalent $T = B > \text{DMObj1}$ compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are greater than the corresponding element in *DMObj1*, and logical 0 (false) otherwise. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls $T = \text{gt}(X, Y)$ for the syntax $T = X > Y$ when *X* or *Y* is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a DataMatrix object: `lt`

Purpose

Concatenate DataMatrix objects horizontally

Syntax

```
DMObjNew = horzcat(DMObj1, DMObj2, ...)  
DMObjNew = (DMObj1, DMObj2, ...)  
DMObjNew = horzcat(DMObj1, B, ...)  
DMObjNew = (DMObj1, B, ...)
```

Arguments

DMObj1, *DMObj2* DataMatrix objects, such as created by DataMatrix (object constructor).

B MATLAB numeric or logical array.

Return Values

DMObjNew DataMatrix object created by horizontal concatenation.

Description

DMObjNew = horzcat(*DMObj1*, *DMObj2*, ...) or the equivalent *DMObjNew* = (*DMObj1*, *DMObj2*, ...) horizontally concatenates the DataMatrix objects *DMObj1* and *DMObj2* into *DMObjNew*, another DataMatrix object. *DMObj1* and *DMObj2* must have the same number of rows. The row names and the order of rows for *DMObjNew* are the same as *DMObj1*. The row names of *DMObj2* and any other DataMatrix object input arguments are not preserved. The column names for *DMObjNew* are the column names of *DMObj1*, *DMObj2*, and other DataMatrix object input arguments.

DMObjNew = horzcat(*DMObj1*, *B*, ...) or the equivalent *DMObjNew* = (*DMObj1*, *B*, ...) horizontally concatenates the DataMatrix object *DMObj1* and a numeric or logical array *B* into *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same number of rows. The row names for *DMObjNew* are the same as *DMObj1*. The row names of *DMObj2* and any other DataMatrix object input arguments are not preserved. The column names for *DMObjNew* are the column names of *DMObj1* and empty for the columns from *B*.

horzcat (DataMatrix)

MATLAB calls $DMObjNew = \text{horzcat}(X1, X2, X3, \dots)$ for the syntax $DMObjNew = [X1, X2, X3, \dots]$ when any one of $X1, X2, X3$, etc. is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a DataMatrix object: `vertcat`

Purpose Test for cycles in biograph object

Syntax `isdag(BGObj)`

Arguments `BGObj` Biograph object created by `biograph` (object constructor).

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions” in the *Bioinformatics Toolbox User’s Guide*.

`isdag(BGObj)` returns logical 1 (`true`) if an N-by-N adjacency matrix extracted from a biograph object, `BGObj`, is a directed acyclic graph (DAG) and logical 0 (`false`) otherwise. In the N-by-N sparse matrix, all nonzero entries indicate the presence of an edge.

References

[1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

Bioinformatics Toolbox functions: `biograph` (object constructor), `graphisdag`

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a biograph object:
`allshortestpaths`, `conncomp`, `isomorphism`, `isspantree`, `maxflow`,
`minspantree`, `shortestpath`, `topoorder`, `traverse`

isequal (DataMatrix)

Purpose	Test DataMatrix objects for equality
Syntax	$TF = \text{isequal}(DMObj1, DMObj2)$ $TF = \text{isequal}(DMObj1, DMObj2, DMObj3, \dots)$
Arguments	$DMObj1, DMObj2, DMObj3$ DataMatrix objects, such as created by DataMatrix (object constructor).
Return Values	TF Logical value indicating if inputs are numerically equal (have the same contents), have the same size (same NRows and NCols properties), and have the same RowNames and ColNames properties. NaNs are not considered equal to each other.
Description	$TF = \text{isequal}(DMObj1, DMObj2)$ returns logical 1 (true) if the input DataMatrix objects, $DMObj1$ and $DMObj2$, meet the following: <ul style="list-style-type: none">• Are numerically equal (have the same contents)• Have the same size (same NRows and NCols properties)• Have the same RowNames and ColNames properties Otherwise, it returns logical 0 (false). $DMObj1$ and $DMObj2$ do not have to have the same Name property. NaNs are not considered equal to each other. $TF = \text{isequal}(DMObj1, DMObj2, DMObj3, \dots)$ returns logical 1 (true) if all input DataMatrix objects, $DMObj1, DMObj2, DMObj3$, etc. meet the following: <ul style="list-style-type: none">• Are numerically equal (have the same contents)• Have the same size (same NRows and NCols properties)• Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). The input DataMatrix objects do not have to have the same Name property. NaNs are not considered equal to each other.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a `DataMatrix` object:

`isequalwithequalnans`

isequalwithequalnans (DataMatrix)

Purpose	Test DataMatrix objects for equality, treating NaNs as equal
Syntax	$TF = \text{isequalwithequalnans}(DMObj1, DMObj2)$ $TF = \text{isequalwithequalnans}(DMObj1, DMObj2, DMObj3, \dots)$
Arguments	$DMObj1, DMObj2, DMObj3$ DataMatrix objects, such as created by DataMatrix (object constructor).
Return Values	TF Logical value indicating if inputs are numerically equal (have the same contents), have the same size (same NRows and NCols properties), and have the same RowNames and ColNames properties. NaNs are considered equal to each other.
Description	<p>$TF = \text{isequalwithequalnans}(DMObj1, DMObj2)$ returns logical 1 (true) if the input DataMatrix objects, $DMObj1$ and $DMObj2$, meet the following:</p> <ul style="list-style-type: none">• Are numerically equal (have the same contents)• Have the same size (same NRows and NCols properties)• Have the same RowNames and ColNames properties <p>Otherwise, it returns logical 0 (false). $DMObj1$ and $DMObj2$ do not have to have the same Name property. NaNs are considered equal to each other.</p> <p>$TF = \text{isequalwithequalnans}(DMObj1, DMObj2, DMObj3, \dots)$ returns logical 1 (true) if all input DataMatrix objects, $DMObj1, DMObj2, DMObj3$, etc. meet the following:</p> <ul style="list-style-type: none">• Are numerically equal (have the same contents)• Have the same size (same NRows and NCols properties)

isequalwithhequalnans (DataMatrix)

- Have the same RowNames and ColNames properties

Otherwise, it returns logical 0 (false). The input DataMatrix objects do not have to have the same Name property. NaNs are considered equal to each other.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a `DataMatrix` object: `isequal`

isomorphism (biograph)

Purpose Find isomorphism between two biograph objects

Syntax

```
[Isomorphic, Map] = isomorphism(BGObj1, BGObj2)
[Isomorphic, Map] = isomorphism(BGObj1, BGObj2, 'Directed',
    DirectedValue)
```

Arguments

<i>BGObj1</i>	Biograph object created by biograph (object constructor).
<i>BGObj2</i>	Biograph object created by biograph (object constructor).
<i>DirectedValue</i>	Property that indicates whether the graphs are directed or undirected. Enter <code>false</code> when both <i>BGObj1</i> and <i>BGObj2</i> produce undirected graphs. In this case, the upper triangles of the sparse matrices extracted from <i>BGObj1</i> and <i>BGObj2</i> are ignored. Default is <code>true</code> , meaning that both graphs are directed.

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions” in the *Bioinformatics Toolbox User’s Guide*.

`[Isomorphic, Map] = isomorphism(BGObj1, BGObj2)` returns logical 1 (`true`) in *Isomorphic* if two N-by-N adjacency matrices extracted from biograph objects *BGObj1* and *BGObj2* are isomorphic graphs, and logical 0 (`false`) otherwise. A graph isomorphism is a 1-to-1 mapping of the nodes in the graph from *BGObj1* and the nodes in the graph from *BGObj2* such that adjacencies are preserved. Return value *Isomorphic* is Boolean. When *Isomorphic* is `true`, *Map* is a row vector containing the node indices that map from *BGObj2* to *BGObj1*. When *Isomorphic* is `false`, the worst-case time complexity is $O(N!)$, where N is the number of nodes.

[*Isomorphic*, *Map*] = `isomorphism(BGObj1, BGObj2, 'Directed', DirectedValue)` indicates whether the graphs are directed or undirected. Set *DirectedValue* to `false` when both *BGObj1* and *BGObj2* produce undirected graphs. In this case, the upper triangles of the sparse matrices extracted from *BGObj1* and *BGObj2* are ignored. The default is `true`, meaning that both graphs are directed.

References

- [1] Fortin, S. (1996). The Graph Isomorphism Problem. Technical Report, 96-20, Dept. of Computer Science, University of Alberta, Edmonton, Alberta, Canada.
- [2] McKay, B.D. (1981). Practical Graph Isomorphism. *Congressus Numerantium* 30, 45-87.
- [3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

Bioinformatics Toolbox functions: `biograph` (object constructor), `graphisomorphism`

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object:
`allshortestpaths`, `conncomp`, `isdag`, `isspantree`, `maxflow`,
`minspantree`, `shortestpath`, `topoorder`, `traverse`

isspantree (biograph)

Purpose Determine if tree created from biograph object is spanning tree

Syntax `TF = isspantree(BGObj)`

Arguments

`BGObj` Biograph object created by biograph (object constructor).

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions” in the *Bioinformatics Toolbox User’s Guide*.

`TF = isspantree(BGObj)` returns logical 1 (true) if the N-by-N adjacency matrix extracted from a biograph object, `BGObj`, is a spanning tree, and logical 0 (false) otherwise. A spanning tree must touch all the nodes and must be acyclic. The lower triangle of the N-by-N adjacency matrix represents an undirected graph, and all nonzero entries indicate the presence of an edge.

References

[1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also Bioinformatics Toolbox functions: `biograph` (object constructor), `graphisspantree`

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a biograph object:
`allshortestpaths`, `conncomp`, `isdag`, `isomorphism`, `maxflow`,
`minspantree`, `shortestpath`, `topoorder`, `traverse`

Purpose	Left array divide DataMatrix objects
Syntax	<pre><i>DMObjNew</i> = ldivide(<i>DMObj1</i>, <i>DMObj2</i>) <i>DMObjNew</i> = <i>DMObj1</i> .\ <i>DMObj2</i> <i>DMObjNew</i> = ldivide(<i>DMObj1</i>, <i>B</i>) <i>DMObjNew</i> = <i>DMObj1</i> .\ <i>B</i> <i>DMObjNew</i> = ldivide(<i>B</i>, <i>DMObj1</i>) <i>DMObjNew</i> = <i>B</i> .\ <i>DMObj1</i></pre>
Arguments	<p><i>DMObj1</i>, <i>DMObj2</i> DataMatrix objects, such as created by DataMatrix (object constructor).</p> <p><i>B</i> MATLAB numeric or logical array.</p>
Return Values	<p><i>DMObjNew</i> DataMatrix object created by left array division.</p>
Description	<p><i>DMObjNew</i> = ldivide(<i>DMObj1</i>, <i>DMObj2</i>) or the equivalent <i>DMObjNew</i> = <i>DMObj1</i> .\ <i>DMObj2</i> performs an element-by-element left array division of the DataMatrix objects <i>DMObj1</i> and <i>DMObj2</i> and places the results in <i>DMObjNew</i>, another DataMatrix object. In other words, ldivide divides each element in <i>DMObj2</i> by the corresponding element in <i>DMObj1</i>. <i>DMObj1</i> and <i>DMObj2</i> must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for <i>DMObjNew</i> are the same as <i>DMObj1</i>, unless <i>DMObj1</i> is a scalar; then they are the same as <i>DMObj2</i>.</p> <p><i>DMObjNew</i> = ldivide(<i>DMObj1</i>, <i>B</i>) or the equivalent <i>DMObjNew</i> = <i>DMObj1</i> .\ <i>B</i> performs an element-by-element left array division of the DataMatrix object <i>DMObj1</i> and <i>B</i>, a numeric or logical array, and places the results in <i>DMObjNew</i>, another DataMatrix object. In other words, ldivide divides each element in <i>B</i> by the corresponding element in <i>DMObj1</i>. <i>DMObj1</i> and <i>B</i> must have the same size (number of rows and</p>

ldivide (DataMatrix)

columns), unless B is a scalar. The size (number of rows and columns), row names, and column names for $DMObjNew$ are the same as $DMObj1$.

$DMObjNew = \text{ldivide}(B, DMObj1)$ or the equivalent $DMObjNew = B \cdot \backslash DMObj1$ performs an element-by-element left array division of B , a numeric or logical array, and the DataMatrix object $DMObj1$, and places the results in $DMObjNew$, another DataMatrix object. In other words, ldivide divides each element in $DMObj1$ by the corresponding element in $B.DMObj1$ and B must have the same size (number of rows and columns), unless B is a scalar. The size (number of rows and columns), row names, and column names for $DMObjNew$ are the same as $DMObj1$.

Note Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls $DMObjNew = \text{ldivide}(X, Y)$ for the syntax $DMObjNew = X \cdot \backslash Y$ when X or Y is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a DataMatrix object: `rdivide`, `times`

Purpose	Test DataMatrix objects for less than or equal to
Syntax	$T = \text{le}(DMObj1, DMObj2)$ $T = DMObj1 \leq DMObj2$ $T = \text{le}(DMObj1, B)$ $T = DMObj1 \leq B$ $T = \text{le}(B, DMObj1)$ $T = B \leq DMObj1$
Arguments	$DMObj1, DMObj2$ DataMatrix objects, such as created by DataMatrix (object constructor). B MATLAB numeric or logical array.
Return Values	T Logical matrix of the same size as $DMObj1$ and $DMObj2$ or $DMObj1$ and B . It contains logical 1 (true) where elements in the first input are less than or equal to the corresponding element in the second input, and logical 0 (false) otherwise.
Description	$T = \text{le}(DMObj1, DMObj2)$ or the equivalent $T = DMObj1 \leq DMObj2$ compares each element in DataMatrix object $DMObj1$ to the corresponding element in DataMatrix object $DMObj2$, and returns T , a logical matrix of the same size as $DMObj1$ and $DMObj2$, containing logical 1 (true) where elements in $DMObj1$ are less than or equal to the corresponding element in $DMObj2$, and logical 0 (false) otherwise. $DMObj1$ and $DMObj2$ must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). $DMObj1$ and $DMObj2$ can have different Name properties. $T = \text{le}(DMObj1, B)$ or the equivalent $T = DMObj1 \leq B$ compares each element in DataMatrix object $DMObj1$ to the corresponding element in B , a numeric or logical array, and returns T , a logical matrix of the same size as $DMObj1$ and B , containing logical 1 (true) where elements

le (DataMatrix)

in *DMObj1* are less than or equal to the corresponding element in *B*, and logical 0 (false) otherwise. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

$T = \text{le}(B, \text{DMObj1})$ or the equivalent $T = B \leq \text{DMObj1}$ compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are less than or equal to the corresponding element in *DMObj1*, and logical 0 (false) otherwise. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls $T = \text{le}(X, Y)$ for the syntax $T = X \leq Y$ when *X* or *Y* is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a DataMatrix object: `ge`

Purpose	Test DataMatrix objects for less than
Syntax	$T = \text{lt}(DMObj1, DMObj2)$ $T = DMObj1 < DMObj2$ $T = \text{lt}(DMObj1, B)$ $T = DMObj1 < B$ $T = \text{lt}(B, DMObj1)$ $T = B < DMObj1$
Arguments	<p><i>DMObj1</i>, <i>DMObj2</i> DataMatrix objects, such as created by DataMatrix (object constructor).</p> <p><i>B</i> MATLAB numeric or logical array.</p>
Return Values	<p><i>T</i> Logical matrix of the same size as <i>DMObj1</i> and <i>DMObj2</i> or <i>DMObj1</i> and <i>B</i>. It contains logical 1 (true) where elements in the first input are less than the corresponding element in the second input, and logical 0 (false) otherwise.</p>
Description	<p>$T = \text{lt}(DMObj1, DMObj2)$ or the equivalent $T = DMObj1 < DMObj2$ compares each element in DataMatrix object <i>DMObj1</i> to the corresponding element in DataMatrix object <i>DMObj2</i>, and returns <i>T</i>, a logical matrix of the same size as <i>DMObj1</i> and <i>DMObj2</i>, containing logical 1 (true) where elements in <i>DMObj1</i> are less than the corresponding element in <i>DMObj2</i>, and logical 0 (false) otherwise. <i>DMObj1</i> and <i>DMObj2</i> must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). <i>DMObj1</i> and <i>DMObj2</i> can have different Name properties.</p> <p>$T = \text{lt}(DMObj1, B)$ or the equivalent $T = DMObj1 < B$ compares each element in DataMatrix object <i>DMObj1</i> to the corresponding element in <i>B</i>, a numeric or logical array, and returns <i>T</i>, a logical matrix of the same size as <i>DMObj1</i> and <i>B</i>, containing logical 1 (true) where elements</p>

lt (DataMatrix)

in *DMObj1* are less than the corresponding element in *B*, and logical 0 (false) otherwise. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

$T = \text{lt}(B, \text{DMObj1})$ or the equivalent $T = B < \text{DMObj1}$ compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are less than the corresponding element in *DMObj1*, and logical 0 (false) otherwise. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls $T = \text{lt}(X, Y)$ for the syntax $T = X < Y$ when *X* or *Y* is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a DataMatrix object: `gt`

Purpose

Return maximum values in DataMatrix object

Syntax

```
M = max(DMObj1)  
[M, Indices] = max(DMObj1)  
[M, Indices, Names] = max(DMObj1)  
... = max(DMObj1, [], Dim)  
MA = max(DMObj1, DMObj2)
```

Arguments

DMObj1, *DMObj2* DataMatrix objects, such as created by DataMatrix (object constructor).

Note *DMObj1* and *DMObj2* must be the same size, unless one is a scalar.

Dim

Scalar specifying the dimension of *DMObj* to return the maximum values. Choices are:

- 1 — Default. Returns a row vector containing a maximum value for each column.
- 2 — Returns a column vector containing a maximum value for each row.

max (DataMatrix)

Return Values

<i>M</i>	One of the following: <ul style="list-style-type: none">• Scalar specifying the maximum value in <i>DMObj</i> when it contains vector of data• Row vector containing the maximum value for each column in <i>DMObj</i> (when <i>Dim = 1</i>)• Column vector containing the maximum value for each row in <i>DMObj</i> (when <i>Dim = 2</i>)
<i>Indices</i>	Either of the following: <ul style="list-style-type: none">• Positive integer specifying the index of the maximum value in a DataMatrix object containing a vector of data• Vector containing the indices for the maximum value in each column (if <i>Dim = 1</i>) or row (if <i>Dim = 2</i>) in a DataMatrix object containing a matrix of data
<i>Names</i>	Vector of the row names (if <i>Dim = 1</i>) or column names (if <i>Dim = 2</i>) corresponding to the maximum value in each column or each row of a DataMatrix object.
<i>MA</i>	Numeric array created from the maximum elements in either of the following: <ul style="list-style-type: none">• Two DataMatrix objects• A DataMatrix object and a numeric array

Description

$M = \max(DMObj1)$ returns the maximum value(s) in *DMObj1*, a DataMatrix object. If *DMObj1* contains a vector of data, *M* is a scalar. If *DMObj1* contains a matrix of data, *M* is a row vector containing a maximum value in each column.

$[M, Indices] = \max(DMObj1)$ returns *Indices*, the indices of the maximum value(s) in *DMObj1*, a DataMatrix object. If *DMObj1* contains

a vector of data, *Indices* is a positive integer. If *DObj1* contains a matrix of data, *Indices* is a vector containing the indices for the maximum value in each column (if *Dim* = 1) or row (if *Dim* = 2). If there are multiple maximum values in a column or row, the index for the first value is returned.

$[M, Indices, Names] = \max(DObj1)$ returns *Names*, a vector of the row names (if *Dim* = 1) or column names (if *Dim* = 2) corresponding to the maximum value in each column or each row of *DObj1*, a DataMatrix object. If there are multiple maximum values in a column or row, the row or column name for the first value is returned.

$\dots = \max(DObj1, [], Dim)$ specifies which dimension to return the maximum values for, that is each column or each row in a DataMatrix object. If *Dim* = 1, returns *M*, a row vector containing the maximum value in each column. If *Dim* = 2, returns *M*, a column vector containing the maximum value in each row. Default *Dim* = 1.

$MA = \max(DObj1, DObj2)$ returns *MA*, a numeric array containing the larger of the two values from each position of *DObj1* and *DObj2*. *DObj1* and *DObj2* can both be DataMatrix objects, or one can be a DataMatrix object and the other a numeric array. They must be the same size, unless one is a scalar. *MA* has the same size (number of rows and columns) as the first nonscalar input.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a DataMatrix object: `min`, `sum`

maxflow (biograph)

Purpose Calculate maximum flow in biograph object

Syntax

```
[MaxFlow, FlowMatrix, Cut] = maxflow(BGObj, SNode, TNode)
[...] = maxflow(BGObj, SNode, TNode, ...'Capacity',
CapacityValue, ...)
[...] = maxflow(BGObj, SNode, TNode, ...'Method', MethodValue,
...)
```

Arguments

<i>BGObj</i>	Biograph object created by biograph (object constructor).
<i>SNode</i>	Node in a directed graph represented by an N-by-N adjacency matrix extracted from biograph object, <i>BGObj</i> .
<i>TNode</i>	Node in a directed graph represented by an N-by-N adjacency matrix extracted from biograph object, <i>BGObj</i> .

<i>CapacityValue</i>	Column vector that specifies custom capacities for the edges in the N-by-N adjacency matrix. It must have one entry for every nonzero value (edge) in the N-by-N adjacency matrix. The order of the custom capacities in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. By default, <code>maxflow</code> gets capacity information from the nonzero entries in the N-by-N adjacency matrix.
<i>MethodValue</i>	String that specifies the algorithm used to find the minimal spanning tree (MST). Choices are: <ul style="list-style-type: none">• 'Edmonds' — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the <i>labeling algorithm</i>. Time complexity is $O(N \cdot E^2)$, where N and E are the number of nodes and edges respectively.• 'Goldberg' — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as <i>preflow-push</i>. Time complexity is $O(N^2 \cdot \sqrt{E})$, where N and E are the number of nodes and edges respectively.

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions” in the *Bioinformatics Toolbox User’s Guide*.

`[MaxFlow, FlowMatrix, Cut] = maxflow(BGObj, SNode, TNode)` calculates the maximum flow of a directed graph represented by an N-by-N adjacency matrix extracted from a biograph object, `BGObj`, from node `SNode` to node `TNode`. Nonzero entries in the matrix determine the capacity of the edges. Output `MaxFlow` is the maximum flow, and `FlowMatrix` is a sparse matrix with all the flow values for every edge. `FlowMatrix(X,Y)` is the flow from node `X` to node `Y`. Output `Cut`

maxflow (biograph)

is a logical row vector indicating the nodes connected to *SNode* after calculating the minimum cut between *SNode* and *TNode*. If several solutions to the minimum cut problem exist, then *Cut* is a matrix.

Tip The algorithm that determines *Cut*, all minimum cuts, has a time complexity of $O(2^N)$, where N is the number of nodes. If this information is not needed, use the `maxflow` method without the third output.

`[...] = maxflow(BGObj, SNode, TNode, ...'PropertyName', PropertyValue, ...)` calls `maxflow` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

`[...] = maxflow(BGObj, SNode, TNode, ...'Capacity', CapacityValue, ...)` lets you specify custom capacities for the edges. *CapacityValue* is a column vector having one entry for every nonzero value (edge) in the N -by- N adjacency matrix. The order of the custom capacities in the vector must match the order of the nonzero values in the matrix when it is traversed column-wise. By default, `graphmaxflow` gets capacity information from the nonzero entries in the matrix.

`[...] = maxflow(BGObj, SNode, TNode, ...'Method', MethodValue, ...)` lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- 'Edmonds' — Uses the Edmonds and Karp algorithm, the implementation of which is based on a variation called the *labeling algorithm*. Time complexity is $O(N \cdot E^2)$, where N and E are the number of nodes and edges respectively.
- 'Goldberg' — Default algorithm. Uses the Goldberg algorithm, which uses the generic method known as *preflow-push*. Time

complexity is $O(N^2 \sqrt{E})$, where N and E are the number of nodes and edges respectively.

References

- [1] Edmonds, J. and Karp, R.M. (1972). Theoretical improvements in the algorithmic efficiency for network flow problems. *Journal of the ACM* 19, 248-264.
- [2] Goldberg, A.V. (1985). A New Max-Flow Algorithm. MIT Technical Report MIT/LCS/TM-291, Laboratory for Computer Science, MIT.
- [3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). *The Boost Graph Library User Guide and Reference Manual*, (Upper Saddle River, NJ:Pearson Education).

See Also

Bioinformatics Toolbox functions: `biograph` (object constructor), `graphmaxflow`

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object:
`allshortestpaths`, `conncomp`, `isdag`, `isomorphism`, `isspantree`,
`minspantree`, `shortestpath`, `topoorder`, `traverse`

mean (DataMatrix)

Purpose Return average or mean values in DataMatrix object

Syntax

```
M = mean(DMatrix)  
M = mean(DMatrix, Dim)  
M = mean(DMatrix, Dim, IgnoreNaN)
```

Arguments

<i>DMatrix</i>	DataMatrix object, such as created by <code>DataMatrix</code> (object constructor).
<i>Dim</i>	Scalar specifying the dimension of <i>DMatrix</i> to calculate the means. Choices are: <ul style="list-style-type: none">• 1 — Default. Returns mean values for elements in each column.• 2 — Returns mean values for elements in each row.
<i>IgnoreNaN</i>	Specifies if NaNs should be ignored. Choices are <code>true</code> (default) or <code>false</code> .

Return Values

M

Either of the following:

- Row vector containing the mean values from elements in each column in *DMatrix* (when *Dim* = 1)
- Column vector containing the mean values from elements in each row in *DMatrix* (when *Dim* = 2)

Description

M = `mean(DMatrix)` returns the mean values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. *M* is a row vector containing the mean values for elements in each column in *DMatrix*.

M = `mean(DMatrix, Dim)` returns the mean values of the elements in the columns or rows of a DataMatrix object, as specified by *Dim*. If *Dim* = 1,

returns M , a row vector containing the mean values for elements in each column in $DMObj$. If $Dim = 2$, returns M , a column vector containing the mean values for elements in each row in $DMObj$. Default $Dim = 1$.

$M = \text{mean}(DMObj, Dim, IgnoreNaN)$ specifies if NaNs should be ignored. *IgnoreNaN* can be true (default) or false.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a `DataMatrix` object: `max`, `median`, `min`, `sum`

median (DataMatrix)

Purpose Return median values in DataMatrix object

Syntax

```
Med = median(DMObj)  
Med = median(DMObj, Dim)  
Med = median(DMObj, Dim, IgnoreNaN)
```

Arguments

<i>DMObj</i>	DataMatrix object, such as created by <code>DataMatrix</code> (object constructor).
<i>Dim</i>	Scalar specifying the dimension of <i>DMObj</i> to calculate the medians. Choices are: <ul style="list-style-type: none">• 1 — Default. Returns median values for elements in each column.• 2 — Returns median values for elements in each row.
<i>IgnoreNaN</i>	Specifies if NaNs should be ignored. Choices are true (default) or false.

Return Values

<i>Med</i>	Either of the following: <ul style="list-style-type: none">• Row vector containing the median values from elements in each column in <i>DMObj</i> (when <i>Dim</i> = 1)• Column vector containing the median values from elements in each row in <i>DMObj</i> (when <i>Dim</i> = 2)
------------	--

Description *Med* = median(*DMObj*) returns the median values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. *Med* is a row vector containing the median values for elements in each column in *DMObj*.

Med = median(*DMObj*, *Dim*) returns the median values of the elements in the columns or rows of a DataMatrix object, as specified by *Dim*. If *Dim* = 1, returns *Med*, a row vector containing the median values for elements in each column in *DMObj*. If *Dim* = 2, returns *Med*, a column vector containing the median values for elements in each row in *DMObj*. Default *Dim* = 1.

Med = median(*DMObj*, *Dim*, *IgnoreNaN*) specifies if NaNs should be ignored. *IgnoreNaN* can be true (default) or false.

See Also

Bioinformatics Toolbox function: DataMatrix (object constructor)

Bioinformatics Toolbox object: DataMatrix object

Bioinformatics Toolbox methods of a DataMatrix object: max, mean, min, sum

min (DataMatrix)

Purpose Return minimum values in DataMatrix object

Syntax

```
M = min(DMObj1)  
[M, Indices] = min(DMObj1)  
[M, Indices, Names] = min(DMObj1)  
... = min(DMObj1, [], Dim)  
MA = min(DMObj1, DMObj2)
```

Arguments *DMObj1*, *DMObj2* DataMatrix objects, such as created by DataMatrix (object constructor).

Note *DMObj1* and *DMObj2* must be the same size, unless one is a scalar.

Dim Scalar specifying the dimension of *DMObj* to return the minimum values. Choices are:

- 1 — Default. Returns a row vector containing a minimum value for each column.
- 2 — Returns a column vector containing a minimum value for each row.

Return Values

<i>M</i>	One of the following: <ul style="list-style-type: none"> • Scalar specifying the minimum value in <i>DMObj</i> when it contains vector of data • Row vector containing the minimum value for each column in <i>DMObj</i> (when <i>Dim = 1</i>) • Column vector containing the minimum value for each row in <i>DMObj</i> (when <i>Dim = 2</i>)
<i>Indices</i>	Either of the following: <ul style="list-style-type: none"> • Positive integer specifying the index of the minimum value in a DataMatrix object containing a vector of data • Vector containing the indices for the minimum value in each column (if <i>Dim = 1</i>) or row (if <i>Dim = 2</i>) in a DataMatrix object containing a matrix of data
<i>Names</i>	Vector of the row names (if <i>Dim = 1</i>) or column names (if <i>Dim = 2</i>) corresponding to the minimum value in each column or each row of a DataMatrix object.
<i>MA</i>	Numeric array created from the minimum elements in either of the following: <ul style="list-style-type: none"> • Two DataMatrix objects • A DataMatrix object and a numeric array

Description

$M = \text{min}(DMObj1)$ returns the minimum value(s) in *DMObj1*, a DataMatrix object. If *DMObj1* contains a vector of data, *M* is a scalar. If *DMObj1* contains a matrix of data, *M* is a row vector containing a minimum value in each column.

$[M, Indices] = \text{min}(DMObj1)$ returns *Indices*, the indices of the minimum value(s) in *DMObj1*, a DataMatrix object. If *DMObj1* contains

min (DataMatrix)

a vector of data, *Indices* is a positive integer. If *DMObj1* contains a matrix of data, *Indices* is a vector containing the indices for the minimum value in each column (if *Dim* = 1) or row (if *Dim* = 2). If there are multiple minimum values in a column or row, the index for the first value is returned.

$[M, Indices, Names] = \text{min}(DMObj1)$ returns *Names*, a vector of the row names (if *Dim* = 1) or column names (if *Dim* = 2) corresponding to the minimum value in each column or each row of *DMObj1*, a DataMatrix object. If there is more than one minimum value in a column or row, the row or column name for the first value is returned.

$\dots = \text{min}(DMObj1, [], Dim)$ specifies which dimension to return the minimum values for, that is each column or each row in a DataMatrix object. If *Dim* = 1, returns *M*, a row vector containing the minimum value in each column. If *Dim* = 2, returns *M*, a column vector containing the minimum value in each row. Default *Dim* = 1.

$MA = \text{min}(DMObj1, DMObj2)$ returns *MA*, a numeric array containing the smaller of the two values from each position of *DMObj1* and *DMObj2*. *DMObj1* and *DMObj2* can both be DataMatrix objects, or one can be a DataMatrix object and the other a numeric array. They must be the same size, unless one is a scalar. *MA* has the same size (number of rows and columns) as the first nonscalar input.

See Also

Bioinformatics Toolbox function: DataMatrix (object constructor)

Bioinformatics Toolbox object: DataMatrix object

Bioinformatics Toolbox methods of a DataMatrix object: max, sum

Purpose

Find minimal spanning tree in biograph object

Syntax

```
[Tree, pred] = minspantree(BGObj)
[Tree, pred] = minspantree(BGObj, R)
[Tree, pred] = minspantree(..., 'Method', MethodValue, ...)
[Tree, pred] = minspantree(..., 'Weights', WeightsValue, ...)
```

Arguments

BGObj Biograph object created by biograph (object constructor).
R Scalar between 1 and the number of nodes.

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions” in the *Bioinformatics Toolbox User’s Guide*.

`[Tree, pred] = minspantree(BGObj)` finds an acyclic subset of edges that connects all the nodes in the undirected graph represented by an N-by-N adjacency matrix extracted from a biograph object, *BGObj*, and for which the total weight is minimized. Weights of the edges are all nonzero entries in the lower triangle of the N-by-N sparse matrix. Output *Tree* is a spanning tree represented by a sparse matrix. Output *pred* is a vector containing the predecessor nodes of the minimal spanning tree (MST), with the root node indicated by 0. The root node defaults to the first node in the largest connected component. This computation requires an extra call to the `graphconncomp` function.

`[Tree, pred] = minspantree(BGObj, R)` sets the root of the minimal spanning tree to node *R*.

`[Tree, pred] = minspantree(..., 'PropertyName', PropertyValue, ...)` calls `minspantree` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

minspantree (biograph)

`[Tree, pred] = minspantree(..., 'Method', MethodValue, ...)` lets you specify the algorithm used to find the minimal spanning tree (MST). Choices are:

- 'Kruskal' — Grows the minimal spanning tree (MST) one edge at a time by finding an edge that connects two trees in a spreading forest of growing MSTs. Time complexity is $O(E+X \cdot \log(N))$, where X is the number of edges no longer than the longest edge in the MST, and N and E are the number of nodes and edges respectively.
- 'Prim' — Default algorithm. Grows the minimal spanning tree (MST) one edge at a time by adding a minimal edge that connects a node in the growing MST with any other node. Time complexity is $O(E \cdot \log(N))$, where N and E are the number of nodes and edges respectively.

Note When the graph is unconnected, Prim's algorithm returns only the tree that contains R , while Kruskal's algorithm returns an MST for every component.

`[Tree, pred] = minspantree(..., 'Weights', WeightsValue, ...)` lets you specify custom weights for the edges. *WeightsValue* is a column vector having one entry for every nonzero value (edge) in the N -by- N sparse matrix. The order of the custom weights in the vector must match the order of the nonzero values in the N -by- N sparse matrix when it is traversed column-wise. By default, `minspantree` gets weight information from the nonzero entries in the N -by- N sparse matrix.

References

- [1] Kruskal, J.B. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society 7, 48-50.
- [2] Prim, R. (1957). Shortest Connection Networks and Some Generalizations. Bell System Technical Journal 36, 1389-1401.

[3] Siek, J.G. Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

Bioinformatics Toolbox functions: `biograph` (object constructor), `graphminspantree`

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object:
`allshortestpaths`, `conncomp`, `isdag`, `isomorphism`, `isspantree`,
`maxflow`, `shortestpath`, `topoorder`, `traverse`

minus (DataMatrix)

Purpose Subtract DataMatrix objects

Syntax

```
DMObjNew = minus(DMObj1, DMObj2)  
DMObjNew = DMObj1 - DMObj2  
DMObjNew = minus(DMObj1, B)  
DMObjNew = DMObj1 - B  
DMObjNew = minus(B, DMObj1)  
DMObjNew = B - DMObj1
```

Arguments

<i>DMObj1</i> , <i>DMObj2</i>	DataMatrix objects, such as created by DataMatrix (object constructor).
<i>B</i>	MATLAB numeric or logical array.

Return Values

<i>DMObjNew</i>	DataMatrix object created by subtraction.
-----------------	---

Description

DMObjNew = minus(*DMObj1*, *DMObj2*) or the equivalent *DMObjNew* = *DMObj1* - *DMObj2* performs an element-by-element subtraction of the DataMatrix object *DMObj2* from the DataMatrix object *DMObj1* and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = minus(*DMObj1*, *B*) or the equivalent *DMObjNew* = *DMObj1* - *B* performs an element-by-element subtraction of *B*, a numeric or logical array, from the DataMatrix object *DMObj1*, and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

$DMObjNew = \text{minus}(B, DMObj1)$ or the equivalent $DMObjNew = B - DMObj1$ performs an element-by-element subtraction of the DataMatrix object $DMObj1$ from B , a numeric or logical array, and places the results in $DMObjNew$, another DataMatrix object. $DMObj1$ and B must have the same size (number of rows and columns), unless B is a scalar. The size (number of rows and columns), row names, and column names for $DMObjNew$ are the same as $DMObj1$.

Note Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls $DMObjNew = \text{minus}(X, Y)$ for the syntax $DMObjNew = X - Y$ when X or Y is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a DataMatrix object: `plus`

ndims (DataMatrix)

Purpose	Return number of dimensions in DataMatrix object	
Syntax	$N = \text{ndims}(DMObj)$	
Arguments	<i>DMObj</i>	DataMatrix object, such as created by DataMatrix (object constructor).
Return Values	<i>N</i>	Positive integer representing the number of dimensions in <i>DMObj</i> . The number of dimensions in a DataMatrix object is always 2.
Description	$N = \text{ndims}(DMObj)$ returns the number of dimensions in <i>DMObj</i> , a DataMatrix object. The number of dimensions in a DataMatrix object is always 2.	
See Also	Bioinformatics Toolbox function: DataMatrix (object constructor) Bioinformatics Toolbox object: DataMatrix object	

Purpose	Test DataMatrix objects for inequality
Syntax	$T = ne(DMObj1, DMObj2)$ $T = DMObj1 \neq DMObj2$ $T = ne(DMObj1, B)$ $T = DMObj1 \neq B$ $T = ne(B, DMObj1)$ $T = B \neq DMObj1$
Arguments	<p><i>DMObj1</i>, <i>DMObj2</i> DataMatrix objects, such as created by DataMatrix (object constructor).</p> <p><i>B</i> MATLAB numeric or logical array.</p>
Return Values	<p><i>T</i> Logical matrix of the same size as <i>DMObj1</i> and <i>DMObj2</i> or <i>DMObj1</i> and <i>B</i>. It contains logical 1 (true) where elements in the first input are not equal to the corresponding element in the second input, and logical 0 (false) when they are equal.</p>
Description	<p>$T = ne(DMObj1, DMObj2)$ or the equivalent $T = DMObj1 \neq DMObj2$ compares each element in DataMatrix object <i>DMObj1</i> to the corresponding element in DataMatrix object <i>DMObj2</i>, and returns <i>T</i>, a logical matrix of the same size as <i>DMObj1</i> and <i>DMObj2</i>, containing logical 1 (true) where elements in <i>DMObj1</i> are not equal to the corresponding element in <i>DMObj2</i>, and logical 0 (false) when they are equal. <i>DMObj1</i> and <i>DMObj2</i> must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). <i>DMObj1</i> and <i>DMObj2</i> can have different Name properties.</p> <p>$T = ne(DMObj1, B)$ or the equivalent $T = DMObj1 \neq B$ compares each element in DataMatrix object <i>DMObj1</i> to the corresponding element in <i>B</i>, a numeric or logical array, and returns <i>T</i>, a logical matrix of the same size as <i>DMObj1</i> and <i>B</i>, containing logical 1 (true) where elements</p>

ne (DataMatrix)

in *DMObj1* are not equal to the corresponding element in *B*, and logical 0 (false) when they are equal. *DMObj1* and *B* must have the same size (number of rows and columns), unless one is a scalar.

$T = \text{ne}(B, \text{DMObj1})$ or the equivalent $T = B \neq \text{DMObj1}$ compares each element in *B*, a numeric or logical array, to the corresponding element in DataMatrix object *DMObj1*, and returns *T*, a logical matrix of the same size as *B* and *DMObj1*, containing logical 1 (true) where elements in *B* are not equal to the corresponding element in *DMObj1*, and logical 0 (false) when they are equal. *B* and *DMObj1* must have the same size (number of rows and columns), unless one is a scalar.

MATLAB calls $T = \text{ne}(X, Y)$ for the syntax $T = X \neq Y$ when *X* or *Y* is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a DataMatrix object: `eq`

Purpose	Return number of elements in DataMatrix object						
Syntax	$N = \text{numel}(\text{DMObj})$ $Ns = \text{numel}(\text{DMObj}, \text{Index1}, \text{Index2})$						
Arguments	<table><tr><td><i>DMObj</i></td><td>DataMatrix object, such as created by DataMatrix (object constructor).</td></tr><tr><td><i>Index1</i></td><td>A row or range of rows in <i>DMObj</i> specified by a positive integer or a range using the format <i>x</i>:<i>y</i>, where <i>x</i> is the first row and <i>y</i> is the last row.</td></tr><tr><td><i>Index2</i></td><td>A column or range of columns in <i>DMObj</i> specified by a positive integer or a range using the format <i>x</i>:<i>y</i>, where <i>x</i> is the first column and <i>y</i> is the last column.</td></tr></table>	<i>DMObj</i>	DataMatrix object, such as created by DataMatrix (object constructor).	<i>Index1</i>	A row or range of rows in <i>DMObj</i> specified by a positive integer or a range using the format <i>x</i> : <i>y</i> , where <i>x</i> is the first row and <i>y</i> is the last row.	<i>Index2</i>	A column or range of columns in <i>DMObj</i> specified by a positive integer or a range using the format <i>x</i> : <i>y</i> , where <i>x</i> is the first column and <i>y</i> is the last column.
<i>DMObj</i>	DataMatrix object, such as created by DataMatrix (object constructor).						
<i>Index1</i>	A row or range of rows in <i>DMObj</i> specified by a positive integer or a range using the format <i>x</i> : <i>y</i> , where <i>x</i> is the first row and <i>y</i> is the last row.						
<i>Index2</i>	A column or range of columns in <i>DMObj</i> specified by a positive integer or a range using the format <i>x</i> : <i>y</i> , where <i>x</i> is the first column and <i>y</i> is the last column.						
Return Values	<table><tr><td><i>N</i></td><td>Positive integer representing the number of elements in <i>DMObj</i>, a DataMatrix object.</td></tr><tr><td><i>Ns</i></td><td>Positive integer representing the number of subscripted elements in <i>DMObj</i>, a DataMatrix object.</td></tr></table>	<i>N</i>	Positive integer representing the number of elements in <i>DMObj</i> , a DataMatrix object.	<i>Ns</i>	Positive integer representing the number of subscripted elements in <i>DMObj</i> , a DataMatrix object.		
<i>N</i>	Positive integer representing the number of elements in <i>DMObj</i> , a DataMatrix object.						
<i>Ns</i>	Positive integer representing the number of subscripted elements in <i>DMObj</i> , a DataMatrix object.						
Description	<p>$N = \text{numel}(\text{DMObj})$ returns 1. To find the number of elements in <i>DMObj</i>, a DataMatrix object, use either of the following syntaxes:</p> <pre>prod(size(DMObj)) numel(DMObj, ':', ':')</pre> <p>$Ns = \text{numel}(\text{DMObj}, \text{Index1}, \text{Index2})$ returns the number of subscripted elements in <i>DMObj</i>, a DataMatrix object. <i>Index1</i> specifies a row or range of rows in <i>DMObj</i>. <i>Index2</i> specifies a column or range of columns in <i>DMObj</i>.</p>						
See Also	Bioinformatics Toolbox function: DataMatrix (object constructor)						

numel (DataMatrix)

Bioinformatics Toolbox object: DataMatrix object

Purpose Calculate pairwise patristic distances in phytree object

Syntax

```
D = pdist(Tree)
[D,C] = pdist(Tree)
pdist(..., 'PropertyName', PropertyValue,...)
pdist(..., 'Nodes', NodeValue)
pdist(..., 'Squareform', SquareformValue)
pdist(..., 'Criteria', CriteriaValue)
```

Arguments

Tree	Phylogenetic tree object created with the phytree constructor function.
NodeValue	Property to select the nodes. Enter either 'leaves' (default) or 'all'.
SquareformValue	Property to control creating a square matrix.

Description

`D = pdist(Tree)` returns a vector (*D*) containing the patristic distances between every possible pair of leaf nodes a phylogenetic tree object (*Tree*). The patristic distances are computed by following paths through the branches of the tree and adding the patristic branch distances originally created with `seqlinkage`.

The output vector *D* is arranged in the order $((2,1), (3,1), \dots, (M,1), (3,2), \dots (M,3), \dots (M,M-1))$ (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 (phytree)

`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 = phytread('pf00002.tree')
```

- 2 Calculate the tree distances between pairs of leaves.

```
dist = pdist(tr, 'nodes', 'leaves', 'squareform', true)
```

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreeread`, `phytreetool`, `seqlinkage`, `seqpdist`

Purpose Render clustergram heat map and dendrograms for clustergram object

Syntax

```
plot(CGobj)
plot(CGobj, HFig)
H = plot(...)
```

Arguments

<i>CGobj</i>	Clustergram object created with the function <code>clustergram</code> .
<i>HFig</i>	Handle to a MATLAB Figure window.

Description

`plot(CGobj)` renders a heat map and dendrograms for *CGobj*, a clustergram object, in a MATLAB Figure window.

`plot(CGobj, HFig)` renders a heat map and dendrograms for *CGobj*, a clustergram object, in a MATLAB Figure window with the handle *HFig*.

`H = plot(...)` returns the handle to the figure. The graphic properties are stored as application data in the figure handle.

Examples

Plot the clustergram object created in the Examples section of the `clustergram` function.

```
plot(cgo)
```

See Also

Bioinformatics Toolbox function: `clustergram` (object constructor)
Bioinformatics Toolbox object: `clustergram` object
Bioinformatics Toolbox methods of a clustergram object: `get`, `set`, `view`

plot (DataMatrix)

Purpose Draw 2-D line plot of DataMatrix object

Syntax
`plot(DMObj1)`
`plot(DMObj1, DMObj2)`
`plot(..., LineSpec)`

Arguments *DMObj1*, *DMObj2* DataMatrix objects, such as created by DataMatrix (object constructor).

Note If both *DMObj1* and *DMObj2* are input arguments, one of the inputs can be a MATLAB numeric array.

LineSpec String specifying a line style, marker symbol, and color of the plotted lines. For more information on these specifiers, see LineSpec.

Description `plot(DMObj1)` plots the columns of a DataMatrix object *DMObj1* versus their index.

`plot(DMObj1, DMObj2)` plots the data from *DMObj1* and *DMObj2*, two DataMatrix objects, or one DataMatrix object and one MATLAB numeric array.

- If *DMObj1* and *DMObj2* are both vectors, they must have the same number of elements, and `plot` plots one vector versus the other vector, creating a single line.
- If one is a vector and one a scalar, `plot` plots discrete points vertically or horizontally, at the scalar value.
- If one is a vector and one a matrix, the number of elements in the vector must equal either the number of rows or the number of columns in the matrix, and `plot` plots the vector versus each row or column in the matrix.

- If both are matrices, they must have the same size (number of rows and columns), and `plot` plots each column in *DMObj1* versus the corresponding column in *DMObj2*.

`plot(..., LineSpec)` plots all lines as defined by *LineSpec*, a character string specifying a line style, marker symbol, and/or color.

Note For a list of line style, marker, and color specifiers, see `LineSpec`.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

MATLAB function: `plot`

plot (phytree)

Purpose Draw phylogenetic tree

Syntax

```
plot(Tree)
plot(Tree, ActiveBranches)
plot(..., 'Type', TypeValue)
plot(..., 'Orientation', OrientationValue)
plot(..., 'BranchLabels', BranchLabelsValue)
plot(..., 'LeafLabels', LeafLabelsValue)
plot(..., 'TerminalLabels', TerminalLabelsValue)
```

Arguments

<i>Tree</i>	Phylogenetic tree object created with the <code>phytree</code> constructor function.
<i>ActiveBranches</i>	Branches veivable in the Figure window.
<i>TypeValue</i>	Property to select a method for drawing a phylogenetic tree. Enter 'square', 'angular', or 'radial'. The default value is 'square'.
<i>OrientationValue</i>	Property to orient a phylogram or cladogram tree. Enter 'top', 'bottom', 'left', or 'right'. The default value is 'left'.
<i>BranchLabelsValue</i>	Property to control displaying branch labels. Enter either true or false. The default value is false.
<i>LeafLabelsValue</i>	Property to control displaying leaf labels. Enter either true or false. The default value is false.
<i>TerminalLabels</i>	Property to control displaying terminal labels. Enter either true or false. The default value is false.

Description `plot(Tree)` draws a phylogenetic tree object into a 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 = phytread('pf00002.tree')
plot(tr, 'Type', 'radial')
```

Graph element properties can be modified as follows:

```
h=get(gcf, 'UserData')
set(h.branchNodeLabels, 'FontSize', 6, 'Color', [.5 .5 .5])
```

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreeread`, `phytreetool`, `seqlinkage`

Bioinformatics Toolbox method of `phytree` object: `view`

plus (DataMatrix)

Purpose

Add DataMatrix objects

Syntax

DMObjNew = plus(*DMObj1*, *DMObj2*)

DMObjNew = *DMObj1* + *DMObj2*

DMObjNew = plus(*DMObj1*, *B*)

DMObjNew = *DMObj1* + *B*

DMObjNew = plus(*B*, *DMObj1*)

DMObjNew = *B* + *DMObj1*

Arguments

DMObj1, *DMObj2* DataMatrix objects, such as created by DataMatrix (object constructor).

B MATLAB numeric or logical array.

Return Values

DMObjNew DataMatrix object created by addition.

Description

DMObjNew = plus(*DMObj1*, *DMObj2*) or the equivalent *DMObjNew* = *DMObj1* + *DMObj2* performs an element-by-element addition of the DataMatrix objects *DMObj1* and *DMObj2* and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = plus(*DMObj1*, *B*) or the equivalent *DMObjNew* = *DMObj1* + *B* performs an element-by-element addition of *DMObj1*, a DataMatrix object, and *B*, a numeric or logical array, and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

$DMObjNew = plus(B, DMObj1)$ or the equivalent $DMObjNew = B + DMObj1$ performs an element-by-element addition of B , a numeric or logical array, and $DMObj1$, a DataMatrix object, and places the results in $DMObjNew$, another DataMatrix object. $DMObj1$ and B must have the same size (number of rows and columns), unless B is a scalar. The size (number of rows and columns), row names, and column names for $DMObjNew$ are the same as $DMObj1$.

Note Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls $DMObjNew = plus(X, Y)$ for the syntax $DMObjNew = X + Y$ when X or Y is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a DataMatrix object: `minus`

power (DataMatrix)

Purpose

Array power DataMatrix objects

Syntax

DMObjNew = power(*DMObj1*, *DMObj2*)

DMObjNew = *DMObj1* .^ *DMObj2*

DMObjNew = power(*DMObj1*, *B*)

DMObjNew = *DMObj1* .^ *B*

DMObjNew = power(*B*, *DMObj1*)

DMObjNew = *B* .^ *DMObj1*

Arguments

DMObj1, *DMObj2* DataMatrix objects, such as created by DataMatrix (object constructor).

B MATLAB numeric or logical array.

Return Values

DMObjNew DataMatrix object created by array power.

Description

DMObjNew = power(*DMObj1*, *DMObj2*) or the equivalent *DMObjNew* = *DMObj1* .^ *DMObj2* performs an element-by-element power of the DataMatrix objects *DMObj1* and *DMObj2* and places the results in *DMObjNew*, another DataMatrix object. In other words, power raises each element in *DMObj1* by the corresponding element in *DMObj2*. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = power(*DMObj1*, *B*) or the equivalent *DMObjNew* = *DMObj1* .^ *B* performs an element-by-element power of the DataMatrix object *DMObj1* and *B*, a numeric or logical array, and places the results in *DMObjNew*, another DataMatrix object. In other words, power raises each element in *DMObj1* by the corresponding element in *B*. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

$DMObjNew = \text{power}(B, DMObj1)$ or the equivalent $DMObjNew = B .^ DMObj1$ performs an element-by-element power of B , a numeric or logical array, and the DataMatrix object $DMObj1$, and places the results in $DMObjNew$, another DataMatrix object. In other words, `power` raises each element in B by the corresponding element in $DMObj1$. $DMObj1$ and B must have the same size (number of rows and columns), unless B is a scalar. The size (number of rows and columns), row names, and column names for $DMObjNew$ are the same as $DMObj1$.

Note Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls $DMObjNew = \text{power}(X, Y)$ for the syntax $DMObjNew = X .^ Y$ when X or Y is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a DataMatrix object: `times`

prune (phytree)

Purpose Remove branch nodes from phylogenetic tree

Syntax
T2 = prune(T1, Nodes)
T2 = prune(T1, Nodes, 'Mode', 'Exclusive')

Arguments

T1	Phylogenetic object created with the phytree constructor function.
Nodes	Nodes to remove from tree.
Mode	Property to control the method of pruning. Enter either 'Inclusive' or 'Exclusive'. The default value is 'Inclusive'.

Description

T2 = prune(T1, Nodes) removes the nodes listed in the vector Nodes from the tree T1. prune removes any branch or leaf node listed in Nodes and all their descendants from the tree T1, and returns the modified tree T2. The parent nodes are connected to the 'brothers' as required. Nodes in the tree are labeled as [1:numLeaves] for the leaves and as [numLeaves+1:numLeaves+numBranches] for the branches. Nodes can also be a logical array of size [numLeaves+numBranches x 1] indicating the nodes to be removed.

T2 = prune(T1, Nodes, 'Mode', 'Exclusive') changes the property (Mode) for pruning to 'Exclusive' and removes only the descendants of the nodes listed in the vector Nodes. Nodes that do not have a predecessor become leaves in the list Nodes. In this case, pruning is the process of reducing a tree by turning some branch nodes into leaf nodes, and removing the leaf nodes under the original branch.

Examples

Load a phylogenetic tree created from a protein family

```
tr = phytreeread('pf00002.tree');  
view(tr)
```

% To :

Remove all the 'mouse' proteins

```
ind = getbyname(tr,'mouse');  
tr = prune(tr,ind);  
view(tr)
```

Remove potential outliers in the tree

```
[sel,sel_leaves] = select(tr,'criteria','distance',...  
                        'threshold',.3,...  
                        'reference','leaves',...  
                        'exclude','leaves',...  
                        'propagate','toleaves');  
tr = prune(tr,-sel_leaves)  
view(tr)
```

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreetool`

Bioinformatics Toolbox methods of `phytree` object: `select`, `get`

rdivide (DataMatrix)

Purpose

Right array divide DataMatrix objects

Syntax

```
DMObjNew = rdivide(DMObj1, DMObj2)  
DMObjNew = DMObj1 ./ DMObj2  
DMObjNew = rdivide(DMObj1, B)  
DMObjNew = DMObj1 ./ B  
DMObjNew = rdivide(B, DMObj1)  
DMObjNew = B ./ DMObj1
```

Arguments

DMObj1, *DMObj2* DataMatrix objects, such as created by DataMatrix (object constructor).

B MATLAB numeric or logical array.

Return Values

DMObjNew DataMatrix object created by right array division.

Description

DMObjNew = rdivide(*DMObj1*, *DMObj2*) or the equivalent *DMObjNew* = *DMObj1* ./ *DMObj2* performs an element-by-element right array division of the DataMatrix objects *DMObj1* and *DMObj2* and places the results in *DMObjNew*, another DataMatrix object. In other words, rdivide divides each element in *DMObj1* by the corresponding element in *DMObj2*. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = rdivide(*DMObj1*, *B*) or the equivalent *DMObjNew* = *DMObj1* ./ *B* performs an element-by-element right array division of the DataMatrix object *DMObj1* and *B*, a numeric or logical array, and places the results in *DMObjNew*, another DataMatrix object. In other words, rdivide divides each element in *DMObj1* by the corresponding element in *B*. *DMObj1* and *B* must have the same size (number of rows and

columns), unless B is a scalar. The size (number of rows and columns), row names, and column names for $DMObjNew$ are the same as $DMObj1$.

$DMObjNew = rdivide(B, DMObj1)$ or the equivalent $DMObjNew = B ./ DMObj1$ performs an element-by-element right array division of B , a numeric or logical array, and the DataMatrix object $DMObj1$, and places the results in $DMObjNew$, another DataMatrix object. In other words, `rdivide` divides each element in B by the corresponding element in $DMObj1$. $DMObj1$ and B must have the same size (number of rows and columns), unless B is a scalar. The size (number of rows and columns), row names, and column names for $DMObjNew$ are the same as $DMObj1$.

Note Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls $DMObjNew = rdivide(X, Y)$ for the syntax $DMObjNew = X ./ Y$ when X or Y is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a DataMatrix object: `ldivide`, `times`

reorder (phytree)

Purpose Reorder leaves of phylogenetic tree

Syntax

```
Tree1Reordered = reorder(Tree1, Order)
[Tree1Reordered, OptimalOrder] = reorder(Tree1, Order,
    'Approximate', ApproximateValue)
[Tree1Reordered, OptimalOrder] = reorder(Tree1, Tree2)
```

Arguments

<i>Tree1, Tree2</i>	Phytree objects.
<i>Order</i>	Vector with position indices for each leaf.
<i>ApproximateValue</i>	Controls the use of the optimal leaf-ordering calculation to find the closest order possible to the suggested one without dividing the clades or producing crossing branches. Enter <code>true</code> to use the calculation. Default is <code>false</code> .

Return Values

<i>Tree1Reordered</i>	Phytree object with reordered leaves.
<i>OptimalOrder</i>	Vector of position indices for each leaf in <i>Tree1Reordered</i> , determined by the optimal leaf-ordering calculation.

Description *Tree1Reordered* = `reorder(Tree1, Order)` reorders the leaves of the phylogenetic tree *Tree1*, without modifying its structure and distances, creating a new phylogenetic tree, *Tree1Reordered*. *Order* is a vector of position indices for each leaf. If *Order* is invalid, that is, if it divides the clades (or produces crossing branches), then `reorder` returns an error message.

`[Tree1Reordered, OptimalOrder] = reorder(Tree1, Order, 'Approximate', ApproximateValue)` controls the use of the optimal leaf-ordering calculation, which finds the best approximate order closest to the suggested one, without dividing the clades or producing crossing branches. Enter `true` to use the calculation and return

Tree1Reordered, the reordered tree, and *OptimalOrder*, a vector of position indices for each leaf in *Tree1Reordered*, determined by the optimal leaf-ordering calculation. Default is false.

```
[Tree1Reordered, OptimalOrder] = reorder(Tree1, Tree2)
```

uses the optimal leaf-ordering calculation to reorder the leaves in *Tree1* such that it matches the order of leaves in *Tree2* as closely as possible, without dividing the clades or producing crossing branches. *Tree1Reordered* is the reordered tree, and *OptimalOrder* is a vector of position indices for each leaf in *Tree1Reordered*, determined by the optimal leaf-ordering calculation

Examples

Reordering Leaves Using a Valid Order

- 1 Create and view a phylogenetic tree.

```
b = [1 2; 3 4; 5 6; 7 8; 9 10];  
tree = phytree(b)  
    Phylogenetic tree object with 6 leaves (5 branches)  
view(tree)
```

- 2 Reorder the leaves on the phylogenetic tree, and then view the reordered tree.

```
treeReordered = reorder(tree, [5, 6, 3, 4, 1, 2])  
view(treeReordered)
```

Finding Best Approximate Order When Using an Invalid Order

- 1 Create a phylogenetic tree by reading a Newick-formatted tree file (ASCII text file).

```
tree = phytread('pf00002.tree')  
    Phylogenetic tree object with 33 leaves (32 branches)
```

- 2 Create a row vector of the leaf names in alphabetical order.

```
[dummy, order] = sort(get(tree, 'LeafNames'));
```

reorder (phytree)

- 3 Reorder the phylogenetic tree to match as closely as possible the row vector of alphabetically ordered leaf names, without dividing the clades or having crossing branches.

```
treeReordered = reorder(tree,order,'approximate',true)
Phylogenetic tree object with 33 leaves (32 branches)
```

- 4 View the original and the reordered phylogenetic trees.

```
view(tree)
view(treeReordered)
```

Reordering Leaves to Match Leaf Order in Another Phylogenetic Tree

- 1 Create a phylogenetic tree by reading sequence data from a FASTA file, calculating the pairwise distances between sequences, and then using the neighbor-joining method.

```
seqs = fastaread('pf00002.fa')

seqs =

33x1 struct array with fields:
    Header
    Sequence

dist = seqpdist(seqs,'method','jukes-cantor','indels','pair');
NJtree = seqneighjoin(dist,'equivar',seqs)
Phylogenetic tree object with 33 leaves (32 branches)
```

- 2 Create another phylogenetic tree from the same sequence data and pairwise distances between sequences, using the single linkage method.

```
HCTree = seqlinkage(dist,'single',seqs)
Phylogenetic tree object with 33 leaves (32 branches)
```

- 3 Use the optimal leaf-ordering calculation to reorder the leaves in HCtree such that it matches the order of leaves in NJtree as closely as possible, without dividing the clades or having crossing branches.

```
HCtree_reordered = reorder(HCtree,NJtree)
Phylogenetic tree object with 33 leaves (32 branches)
```

- 4 View the reordered phylogenetic tree and the tree used to reorder it.

```
view(HCtree_reordered)
view(NJtree)
```

See Also

Bioinformatics Toolbox function: `phytree` (object constructor)

Bioinformatics Toolbox object: `phytree` object

Bioinformatics Toolbox methods of a `phytree` object: `get`, `getbyname`, `prune`

reroot (phytree)

Purpose Change root of phylogenetic tree

Syntax

```
Tree2 = reroot(Tree1)
Tree2 = reroot(Tree1, Node)
Tree2 = reroot(Tree1, Node, Distance)
```

Arguments

<i>Tree1</i>	Phylogenetic tree (phytree object) created with the function <code>phytree</code> .
<i>Node</i>	Node index returned by the phytree object method <code>getbyname</code> .
<i>Distance</i>	Distance from the reference branch.

Description

`Tree2 = reroot(Tree1)` changes the root of a phylogenetic tree (*Tree1*) using a midpoint method. The midpoint is the location where the mean values of the branch lengths, on either side of the tree, are equalized. The original root is deleted from the tree.

`Tree2 = reroot(Tree1, Node)` changes the root of a phylogenetic tree (*Tree1*) to a branch node using the node index (*Node*). The new root is placed at half the distance between the branch node and its parent.

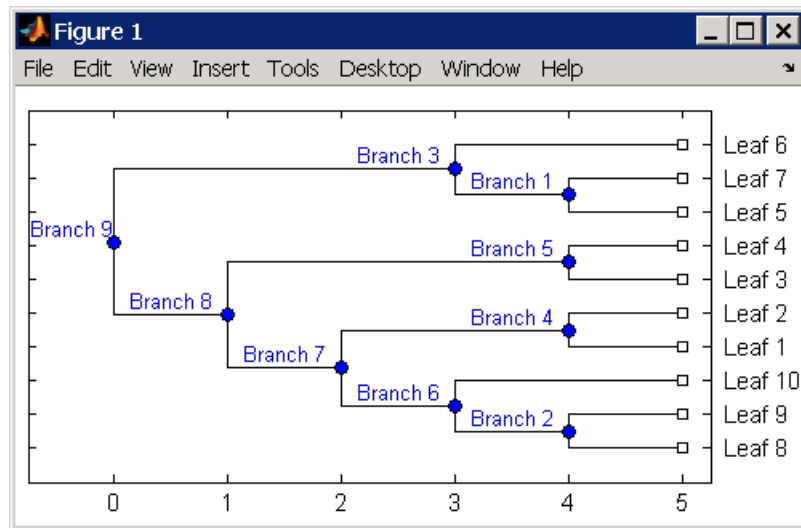
`Tree2 = reroot(Tree1, Node, Distance)` changes the root of a phylogenetic tree (*Tree1*) to a new root at a given distance (*Distance*) from the reference branch node (*Node*) toward the original root of the tree. Note: The new branch representing the root in the new tree (*Tree2*) is labeled 'Root'.

Examples

1 Create an ultrametric tree.

```
tr_1 = phytree([5 7;8 9;6 11; 1 2;3 4;10 12;...
               14 16; 15 17;13 18])
plot(tr_1,'branchlabels',true)
```

A figure with the phylogenetic tree displays.

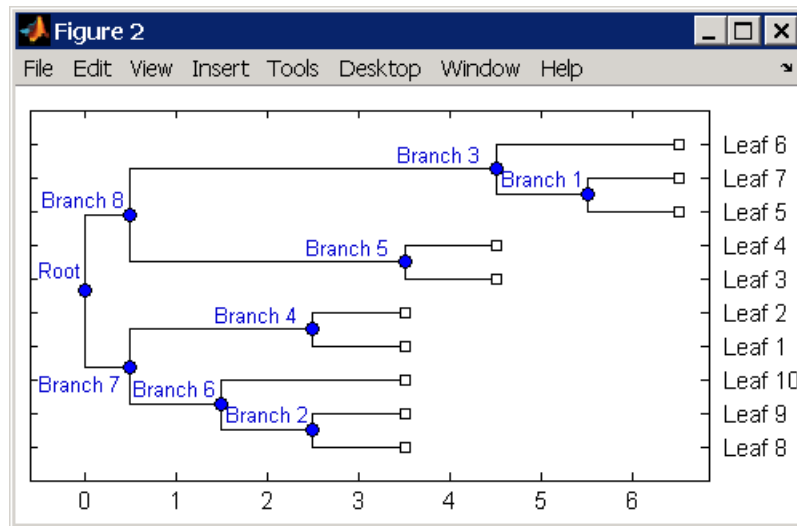


2 Place the root at 'Branch 7'.

```
sel = getbyname(tr_1,'Branch 7');  
tr_2 = reroot(tr_1,sel)  
plot(tr_2,'branchlabels',true)
```

A figure of a phylogenetic tree displays with the root moved to the center of branch 7.

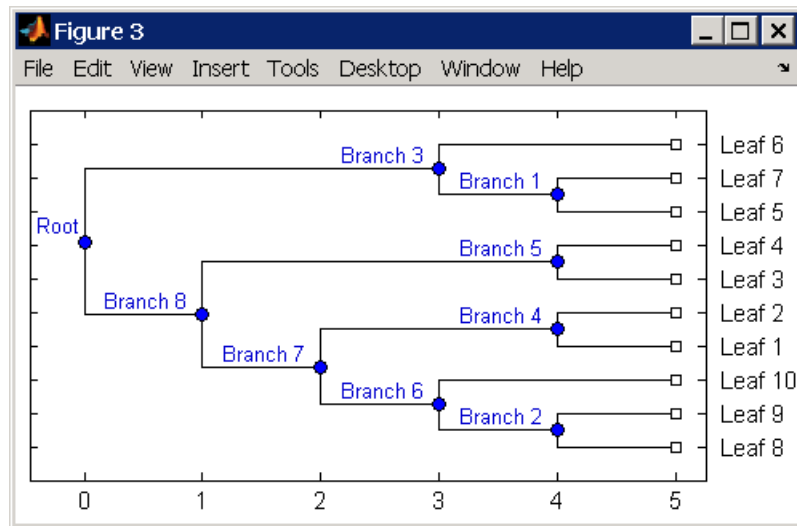
reroot (phytree)



- 3** Move the root to a branch that makes the tree as ultrametric as possible.

```
tr_3 = reroot(tr_2)
plot(tr_3, 'branchlabels', true)
```

A figure of the new tree displays with the root moved from the center of branch 7 to branch 8.



See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `seqneighjoin`

Bioinformatics Toolbox methods of `phytree` object: `get`, `getbyname`, `prune`, `select`

rownames (DataMatrix)

Purpose Retrieve or set row names of DataMatrix object

Syntax

```
ReturnRowNames = rownames(DObj)  
ReturnRowNames = rownames(DObj, RowIndices)  
DObjNew = rownames(DObj, RowIndices, RowNames)
```


Arguments

<i>DObj</i>	DataMatrix object, such as created by <code>DataMatrix</code> (object constructor).
<i>RowIndices</i>	One or more rows in <i>DObj</i> , specified by any of the following: <ul style="list-style-type: none">• Positive integer• Vector of positive integers• String specifying a row name• Cell array of strings• Logical vector
<i>RowNames</i>	Row names specified by any of the following: <ul style="list-style-type: none">• Numeric vector• Cell array of strings• Character array• Single string, which is used as a prefix for row names, with row numbers appended to the prefix• Logical <code>true</code> or <code>false</code> (default). If <code>true</code>, unique row names are assigned using the format <code>row1</code>, <code>row2</code>, <code>row3</code>, etc. If <code>false</code>, no row names are assigned.

Note The number of elements in *RowNames* must equal the number of elements in *RowIndices*.

rownames (DataMatrix)

Return Values

ReturnRowNames String or cell array of strings containing row names in *DObj*.

DObjNew DataMatrix object created with names specified by *RowIndices* and *RowNames*.

Description

ReturnRowNames = rownames(*DObj*) returns *ReturnRowNames*, a cell array of strings specifying the row names in *DObj*, a DataMatrix object.

ReturnRowNames = rownames(*DObj*, *RowIndices*) returns the row names specified by *RowIndices*. *RowIndices* can be a positive integer, vector of positive integers, string specifying a row name, cell array of strings, or a logical vector.

DObjNew = rownames(*DObj*, *RowIndices*, *RowNames*) returns *DObjNew*, a DataMatrix object with rows specified by *RowIndices* set to the names specified by *RowNames*. The number of elements in *RowIndices* must equal the number of elements in *RowNames*.

See Also

Bioinformatics Toolbox function: DataMatrix (object constructor)
Bioinformatics Toolbox object: DataMatrix object
Bioinformatics Toolbox method of a DataMatrix object: colnames

Purpose Select tree branches and leaves in phytree object

Syntax

```
S = select(Tree, N)
[S, Selleaves, Selbranches] = select(...)
select(..., 'Reference', ReferenceValue, ...)
select(..., 'Criteria', CriteriaValue, ...)
select(..., 'Threshold', ThresholdValue, ...)
select(..., 'Exclude', ExcludeValue, ...)
select(..., 'Propagate', PropagateValue, ...)
```

Arguments	<i>Tree</i>	Phylogenetic tree (phytree object) created with the function <code>phytree</code> .
	<i>N</i>	Number of closest nodes to the root node.
	<i>ReferenceValue</i>	Property to select a reference point for measuring distance.
	<i>CriteriaValue</i>	Property to select a criteria for measuring distance.
	<i>ThresholdValue</i>	Property to select a distance value. Nodes with distances below this value are selected.
	<i>ExcludeValue</i>	Property to remove (exclude) branch or leaf nodes from the output. Enter 'none', 'branches', or 'leaves'. The default value is 'none'.
	<i>PropagateValue</i>	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`.

select (phytree)

`[S, Selleaves, Selbranches] = select(...)` returns two additional logical vectors, one for the selected leaves and one for the selected branches.

`select(..., 'PropertyName', PropertyValue, ...)` calls `select` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotation marks and is case insensitive. These property name/property value pairs are as follows:

`select(..., 'Reference', ReferenceValue, ...)` changes the reference point(s) to measure the closeness. `Reference` can be the root (default) or leaves. When using leaves, a node can have multiple distances to its descendant leaves (nonultrametric tree). If this the case, `select` considers the minimum distance to any descendant leaf.

`select(..., 'Criteria', CriteriaValue, ...)` changes the criteria `select` uses to measure closeness. If `C = 'levels'` (default), the first criterion is branch levels and then patristic distance. If `C = 'distance'`, the first criterion is patristic distance and then branch levels.

`select(..., 'Threshold', ThresholdValue, ...)` selects all the nodes where closeness is less than or equal to the threshold value (*ThresholdValue*). Notice, you can also use either of the properties 'criteria' or 'reference', if `N` is not specified, then `N = infF`; otherwise you can limit the number of selected nodes by `N`.

`select(..., 'Exclude', ExcludeValue, ...)` when `ExcludeValue = 'branches'`, sets a postfilter that excludes all the branch nodes from `S`, or when `ExcludeValue = 'leaves'`, all the leaf nodes. The default is 'none'.

`select(..., 'Propagate', PropagateValue, ...)` activates a postfunctionality that propagates the selected nodes to the leaves when `P == 'toleaves'` or toward the root finding a common ancestor when `P == 'toroot'`. The default value is 'none'. `P` may also be 'both'. The 'Propagate' property acts after the 'Exclude' property.

Examples

```
% Load a phylogenetic tree created from a protein family:
tr = phytread('pf00002.tree');

% To find close products for a given protein (e.g. vipr2_human):
ind = getbyname(tr,'vipr2_human');
[sel,sel_leaves] = select(tr,'criteria','distance',...
                        'threshold',0.6,'reference',ind);
view(tr,sel_leaves)

% To find potential outliers in the tree, use
[sel,sel_leaves] = select(tr,'criteria','distance',...
                        'threshold',.3,...
                        'reference','leaves',...
                        'exclude','leaves',...
                        'propagate','toleaves');
view(tr,~sel_leaves)
```

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreetool`

Bioinformatics Toolbox methods of `phytree` object: `get`, `pdist`, `prune`

set (biograph)

Purpose Set property of biograph object

Syntax

```
set(BGobj)
set(BGobj, 'PropertyName')
set(BGobj, 'PropertyName', PropertyValue)
set(BGobj, 'Property1Name', Property1Value,
'Property2Name',
    Property2Value, ...)
```

Arguments

BGobj Biograph object created with the function `biograph`.

PropertyName Property name for a biograph object.

PropertyValue Value of the property specified by *PropertyName*.

Description

`set(BGobj)` displays possible values for all properties that have a fixed set of property values in *BGobj*, a biograph object.

`set(BGobj, 'PropertyName')` displays possible values for a specific property that has a fixed set of property values in *BGobj*, a biograph object.

`set(BGobj, 'PropertyName', PropertyValue)` sets the specified property of *BGobj*, a biograph object.

`set(BGobj, 'Property1Name', Property1Value, 'Property2Name', Property2Value, ...)` sets the specified properties of *BGobj*, a biograph object.

Properties of a Biograph Object

Property	Description
ID	String to identify the biograph object. Default is ''.
Label	String to label the biograph object. Default is ''.

Properties of a Biograph Object (Continued)

Property	Description
Description	String that describes the biograph object. Default is ''.
LayoutType	<p>String that specifies the algorithm for the layout engine. Choices are:</p> <ul style="list-style-type: none"> • 'hierarchical' (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges. • 'radial' — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges. • 'equilibrium' — Calculates layout by minimizing the energy in a dynamic spring system.
EdgeType	<p>String that specifies how edges display. Choices are:</p> <ul style="list-style-type: none"> • 'straight' • 'curved' (default) • 'segmented' <hr/> <p>Note Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with LayoutType equal to 'equilibrium' or 'radial' cannot produce curved or segmented edges.</p> <hr/>

set (biograph)

Properties of a Biograph Object (Continued)

Property	Description
Scale	Positive number that post-scales the node coordinates. Default is 1.
LayoutScale	Positive number that scales the size of the nodes before calling the layout engine. Default is 1.
EdgeTextColor	Three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.
EdgeFontSize	Positive number that sets the size of the edge font in points. Default is 8.
ShowArrows	Controls the display of arrows with the edges. Choices are 'on' (default) or 'off'.
ArrowSize	Positive number that sets the size of the arrows in points. Default is 8.
ShowWeights	Controls the display of text indicating the weight of the edges. Choices are 'on' (default) or 'off'.
ShowTextInNodes	String that specifies the node property used to label nodes when you display a biograph object using the view method. Choices are: <ul style="list-style-type: none">• 'Label' — Uses the Label property of the node object (default).• 'ID' — Uses the ID property of the node object.• 'None'
NodeAutoSize	Controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.

Properties of a Biograph Object (Continued)

Property	Description
NodeCallback	User-defined callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the <code>view</code> function to display the biograph object in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, <code>@(node) inspect(node)</code> , which displays the Property Inspector dialog box.
EdgeCallback	User-defined callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the <code>view</code> function to display the biograph object in the Biograph Viewer, you can double-click an edge to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, <code>@(edge) inspect(edge)</code> , which displays the Property Inspector dialog box.
CustomNodeDrawFcn	Function handle to a customized function to draw nodes. Default is <code>[]</code> .

set (biograph)

Properties of a Biograph Object (Continued)

Property	Description
Nodes	Read-only column vector with handles to node objects of a biograph object. The size of the vector is the number of nodes. For properties of node objects, see Properties of a Node Object on page 5-7.
Edges	Read-only column vector with handles to edge objects of a biograph object. The size of the vector is the number of edges. For properties of edge objects, see Properties of an Edge Object on page 5-9.

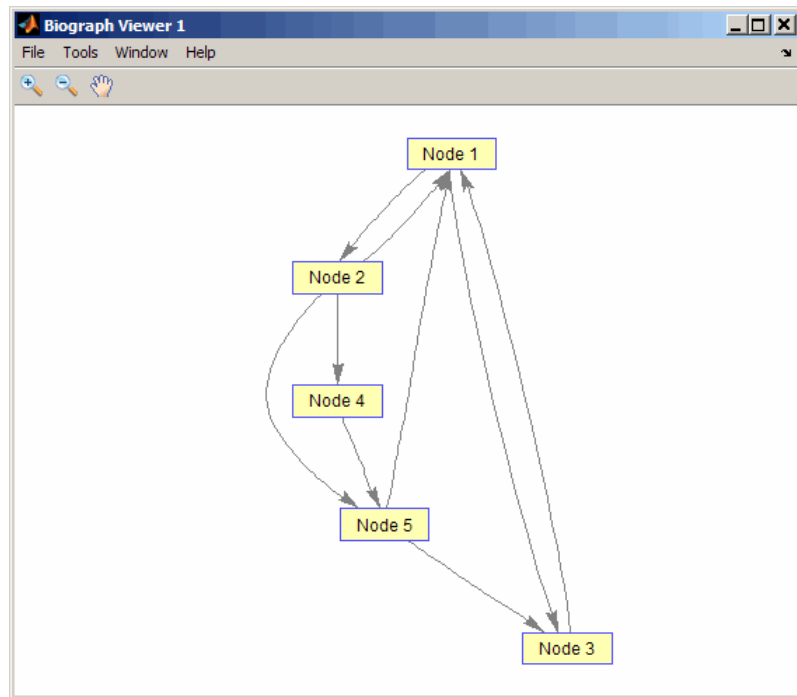
Examples

- 1 Create a biograph object with default node IDs.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];  
bg = biograph(cm)  
Biograph object with 5 nodes and 9 edges.
```

- 2 Use the view method to display the biograph object.

```
view(bg)
```



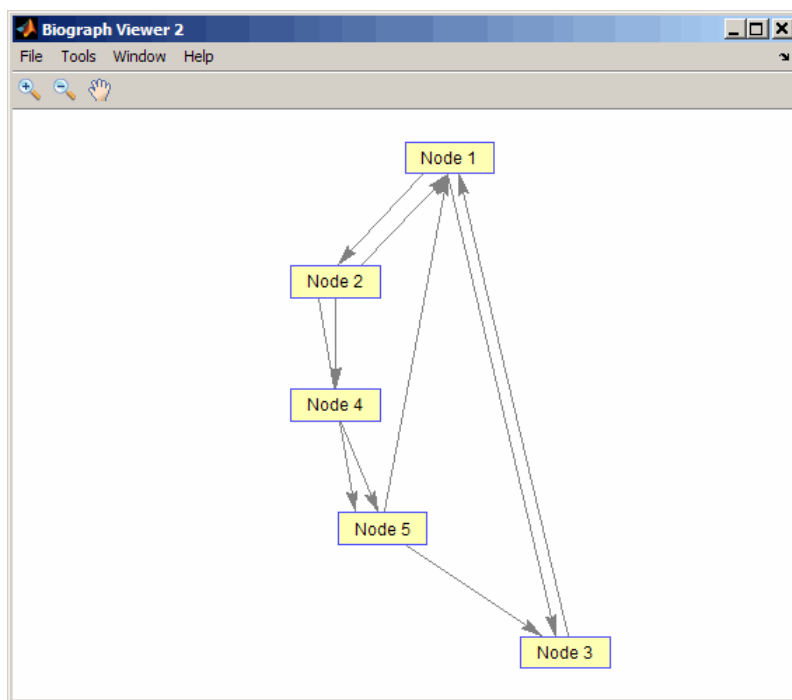
- 3 Use the set method to change the edge lines from curved to straight.

```
set(bg, 'EdgeType', 'straight')
```

- 4 Display the biograph object again.

```
view(bg)
```

set (biograph)



See Also

Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox method of a `biograph` object: `get`

Purpose Set property of clustergram object

Syntax

```
set(CGobj)
set(CGobj, 'PropertyName')
set(CGobj, 'PropertyName', PropertyValue)
set(CGobj, 'Property1Name', Property1Value,
'Property2Name',
Property2Value, ...)
```

Arguments

<i>CGobj</i>	Clustergram object created with the function clustergram.
--------------	---

PropertyName Property name for a clustergram object.

Description

Note You cannot set the properties of a clustergram object if you created it using the **Export Group to Workspace** command in the Clustergram window.

`set(CGobj)` displays possible values for all properties that have a fixed set of property values in *CGobj*, a clustergram object.

`set(CGobj, 'PropertyName')` displays possible values for a specific property that has a fixed set of property values in *CGobj*, a clustergram object.

`set(CGobj, 'PropertyName', PropertyValue)` sets the specified property of *CGobj*, a clustergram object.

`set(CGobj, 'Property1Name', Property1Value, 'Property2Name', Property2Value, ...)` sets the specified properties of *CGobj*, a clustergram object.

set (clustergram)

Properties of a Clustergram Object

Property	Description
RowLabels	Vector of numbers or cell array of text strings to label the rows in the dendrogram and heat map. Default is a vector of values 1 through M , where M is the number of rows in <i>Data</i> , the matrix of data used by the <code>clustergram</code> function to create the clustergram object.
ColumnLabels	Vector of numbers or cell array of text strings to label the columns in the dendrogram and heat map. Default is a vector of values 1 through N , where N is the number of columns in <i>Data</i> , the matrix of data used by the <code>clustergram</code> function to create the clustergram object.
Standardize	Numeric value that specifies the dimension for standardizing the values in <i>Data</i> , the matrix of data used to create the clustergram object. The standardized values are transformed so that the mean is 0 and the standard deviation is 1 in the specified dimension. Choices are: <ul style="list-style-type: none">• 1 — Standardize along the columns of data.• 2 — Standardize along the rows of data.• 3 — Do not perform standardization.

Properties of a Clustergram Object (Continued)

Property	Description
Cluster	<p>Numeric value that specifies the dimension for clustering the values in <i>Data</i>, the matrix of data used to create the clustergram object. Choices are:</p> <ul style="list-style-type: none"> • 1 — Cluster rows of data only. • 2 — Cluster columns of data only. • 3 — Cluster rows of data, then cluster columns of row-clustered data.
RowPdist	<p>String that specifies the distance metric to pass to the <code>pdist</code> function (Statistics Toolbox software) to use to calculate the pairwise distances between rows. For information on choices, see the <code>pdist</code> function.</p> <hr/> <p>Note If the distance metric requires extra arguments, then <i>RowPdistValue</i> is a cell array. For example, to use the Minkowski distance with exponent <i>P</i>, you would use {'minkowski', <i>P</i>}.</p> <hr/>

set (clustergram)

Properties of a Clustergram Object (Continued)

Property	Description
ColumnPdist	<p>String that specifies the distance metric to pass to the <code>pdist</code> function (Statistics Toolbox software) to use to calculate the pairwise distances between columns. For information on choices, see the <code>pdist</code> function.</p> <hr/> <p>Note If the distance metric requires extra arguments, then <i>ColumnPdistValue</i> is a cell array. For example, to use the Minkowski distance with exponent <i>P</i>, you would use <code>{'minkowski', P}</code>.</p> <hr/>
Linkage	<p>String or two-element cell array of strings that specifies the linkage method to pass to the <code>linkage</code> function (Statistics Toolbox software) to use to create the hierarchical cluster tree for rows and columns. If a two-element cell array of strings, the first element is used for linkage between rows, and the second element is used for linkage between columns. For information on choices, see the <code>linkage</code> function.</p>
Dendrogram	<p>Scalar or two-element numeric vector or cell array that specifies the <code>'colorthreshold'</code> property to pass to the <code>dendrogram</code> function (Statistics Toolbox software) to create the dendrogram plot. If a two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns. For more information, see the <code>dendrogram</code> function.</p>

Properties of a Clustergram Object (Continued)

Property	Description
OptimalLeafOrder	<p>Property to enable or disable the optimal leaf ordering calculation, which determines the leaf order that maximizes the similarity between neighboring leaves. Choices are <code>true</code> (enable) or <code>false</code> (disable).</p> <hr/> <p>Tip Disabling the optimal leaf ordering calculation can be useful when working with large data sets because this calculation uses a large amount of memory and can be very time consuming.</p> <hr/>
LogTrans	Controls the \log_2 transform of <i>Data</i> , the matrix of data used to create the clustergram object, from natural scale. Choices are <code>true</code> or <code>false</code> .
ColorMap	<p>Either of the following:</p> <ul style="list-style-type: none"> • <i>M</i>-by-3 matrix of RGB values • Name or function handle of a function that returns a colormap, such as <code>redgreencmap</code> or <code>redbluecmap</code>
DisplayRange	<p>Positive scalar that specifies the display range of standardized values.</p> <p>For example, if you specify <code>redgreencmap</code> for the 'ColorMap' property, pure red represents values \geq DisplayRange, and pure green represents values \leq -DisplayRange.</p>

set (clustergram)

Properties of a Clustergram Object (Continued)

Property	Description
SymmetricRange	Property to force the color scale of the heat map to be symmetric around zero. Choices are true or false.
Ratio	<p>Either of the following:</p> <ul style="list-style-type: none">• Scalar• Two-element vector <p>It specifies the ratio of space that the row and column dendrograms occupy relative to the heat map. If <code>Ratio</code> is a scalar, it is used as the ratio for both dendrograms. If <code>Ratio</code> is a two-element vector, the first element is used for the ratio of the row dendrogram width to the heat map width, and the second element is used for the ratio of the column dendrogram height to the heat map height. The second element is ignored for one-dimensional clustergrams.</p>
Impute	<p>Any of the following:</p> <ul style="list-style-type: none">• Name of a function that imputes missing data.• Handle to a function that imputes missing data.• Cell array where the first element is the name of or handle to a function that imputes missing data and the remaining elements are property name/property value pairs used as inputs to the function.

Properties of a Clustergram Object (Continued)

Property	Description
RowMarkers	<p>Optional structure array for annotating the groups (clusters) of rows determined by the <code>clustergram</code> function. Each structure in the array represents a group of rows and contains the following fields:</p> <ul style="list-style-type: none"> • <code>GroupNumber</code> — Number to annotate the row group. • <code>Annotation</code> — String specifying text to annotate the row group. • <code>Color</code> — String or three-element vector of RGB values specifying a color, which is used to label the row group. For more information on specifying colors, see <code>colorspec</code>. If this field is empty, default is 'blue'.
ColumnMarkers	<p>Optional structure array for annotating groups (clusters) of columns determined by the <code>clustergram</code> function. Each structure in the array represents a group of rows and contains the following fields:</p> <ul style="list-style-type: none"> • <code>GroupNumber</code> — Number to annotate the column group. • <code>Annotation</code> — String specifying text to annotate the column group. • <code>Color</code> — String or three-element vector of RGB values specifying a color, which is used to label the column group. For more information on specifying colors, see <code>colorspec</code>. If this field is empty, default is 'blue'.

set (clustergram)

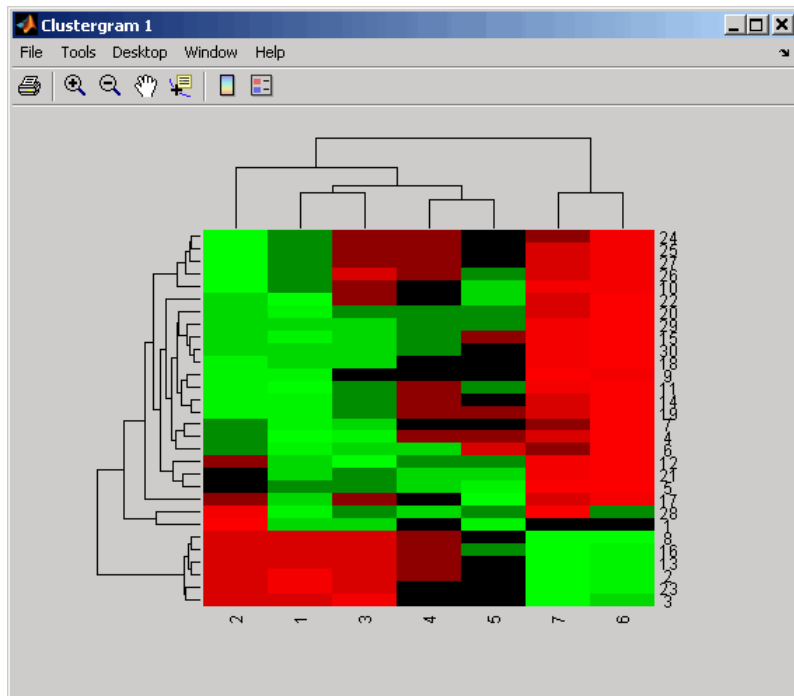
Examples

- 1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains filtered yeast data. This MAT-file includes three variables: `yeastvalues`, a matrix of gene expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`.

```
load filteredyeastdata
```

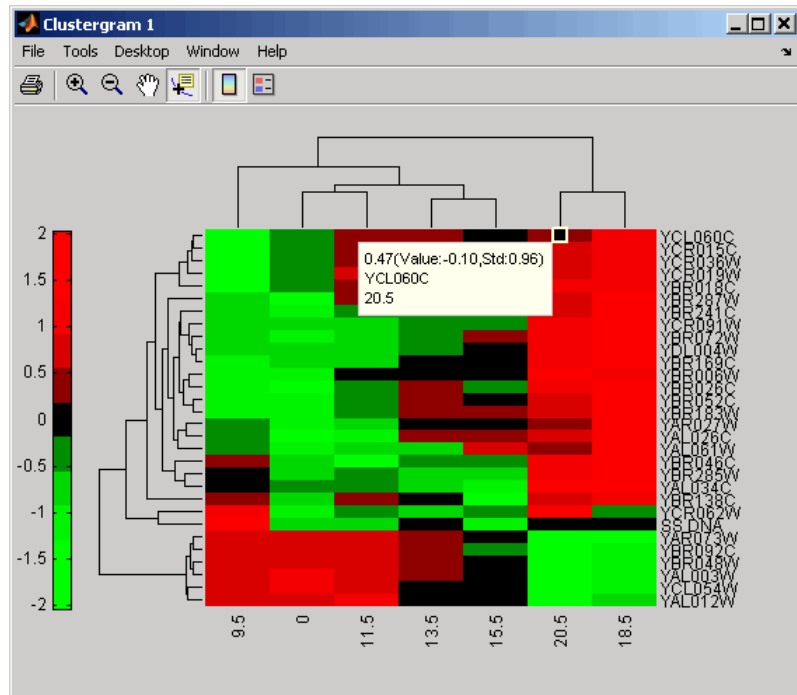
- 2 Create a clustergram object and display the dendrograms and heat map from the gene expression data in the first 30 rows of the `yeastvalues` matrix.

```
cgo = clustergram(yeastvalues(1:30,:))  
Clustergram object with 30 rows of nodes and 7 column of nodes.
```



- 3** Use the set method and the genes and times vectors to add meaningful row and column labels to the clustergram.

```
set(cgo, 'RowLabels', genes(1:30), 'ColumnLabels', times)
```



- 4** Reset the colormap of the heat map to redbluecmap.

```
set(cgo, 'Colormap', redbluecmap);
```


Purpose Set property of DataMatrix object

Syntax

```
set(DMObj)
set(DMObj, 'PropertyName')
DMObj = set(DMObj, 'PropertyName', PropertyValue)
DMObj = set(DMObj, 'Property1Name', Property1Value,
            'Property2Name', Property2Value, ...)
```

Arguments

DMObj DataMatrix object, such as created by DataMatrix (object constructor).

PropertyName Property name of a DataMatrix object.

PropertyValue Value of the property specified by *PropertyName*.

Description `set(DMObj)` displays possible values for all properties that have a fixed set of property values in *DMObj*, a DataMatrix object.

`set(DMObj, 'PropertyName')` displays possible values for a specific property that has a fixed set of property values in *DMObj*, a DataMatrix object.

`DMObj = set(DMObj, 'PropertyName', PropertyValue)` sets the specified property of *DMObj*, a DataMatrix object.

`DMObj = set(DMObj, 'Property1Name', Property1Value, 'Property2Name', Property2Value, ...)` sets the specified properties of *DMObj*, a DataMatrix object.

Properties of a DataMatrix Object

Property	Description
Name	String that describes the DataMatrix object. Default is ''.

set (DataMatrix)

Properties of a DataMatrix Object (Continued)

Property	Description
RowNames	Empty array or cell array of strings that specifies the names for the rows, typically gene names or probe identifiers. The number of elements in the cell array must equal the number of rows in the matrix. Default is an empty array.
ColNames	Empty array or cell array of strings that specifies the names for the columns, typically sample identifiers. The number of elements in the cell array must equal the number of columns in the matrix.
NRows	Positive number that specifies the number of rows in the matrix. <hr/> Note You cannot modify this property directly. You can access it using the <code>get</code> method. <hr/>
NCols	Positive number that specifies the number of columns in the matrix. <hr/> Note You cannot modify this property directly. You can access it using the <code>get</code> method. <hr/>

Properties of a DataMatrix Object (Continued)

Property	Description
NDims	<p>Positive number that specifies the number of dimensions in the matrix.</p> <hr/> <p>Note You cannot modify this property directly. You can access it using the <code>get</code> method.</p>
ElementClass	<p>String that specifies the class type, such as <code>single</code> or <code>double</code>.</p> <hr/> <p>Note You cannot modify this property directly. You can access it using the <code>get</code> method.</p>

Examples

- 1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: `yeastvalues`, a matrix of gene expression data, `genes`, a cell array of GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a vector of time values for labeling the columns in `yeastvalues`.

```
load filteredyeastdata
```

- 2 Import the microarray object package so that the `DataMatrix` constructor function will be available.

```
import bioma.data.*
```

- 3 Create a `DataMatrix` object from the gene expression data in the first 30 rows of the `yeastvalues` matrix.

```
dmo = DataMatrix(yeastvalues(1:30,:));
```

set (DataMatrix)

- 4 Use the `get` method to display the properties of the `DataMatrix` object, `dmo`.

```
get(dmo)

    Name: ''
   RowNames: []
   ColNames: []
     NRows: 30
      NCols: 7
      NDims: 2
ElementClass: 'double'
```

Notice that the `RowNames` and `ColNames` fields are empty.

- 5 Use the `set` method and the `genes` and `times` variables to specify row names and column names for the `DataMatrix` object, `dmo`.

```
dmo = set(dmo, 'RowNames', genes(1:30), 'ColNames', times)
```

- 6 Use the `get` method to display the properties of the `DataMatrix` object, `dmo`.

```
get(dmo)

    Name: ''
   RowNames: {30x1 cell}
   ColNames: {' 0' ' 9.5' '11.5' '13.5' '15.5' '18.5' }
     NRows: 30
      NCols: 7
      NDims: 2
ElementClass: 'double'
```

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a `DataMatrix` object: `get`

Purpose

Solve shortest path problem in biograph object

Syntax

```
[dist, path, pred] = shortestpath(BGObj, S)  
[dist, path, pred] = shortestpath(BGObj, S, T)  
[...] = shortestpath(..., 'Directed', DirectedValue, ...)  
[...] = shortestpath(..., 'Method', MethodValue, ...)  
[...] = shortestpath(..., 'Weights', WeightsValue, ...)
```

Arguments

<i>BGObj</i>	Biograph object created by biograph (object constructor).
<i>S</i>	Node in graph represented by an N-by-N adjacency matrix extracted from a biograph object, <i>BGObj</i> .
<i>T</i>	Node in graph represented by an N-by-N adjacency matrix extracted from a biograph object, <i>BGObj</i> .
<i>DirectedValue</i>	Property that indicates whether the graph represented by the N-by-N adjacency matrix extracted from a biograph object, <i>BGObj</i> , is directed or undirected. Enter <code>false</code> for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is <code>true</code> .

shortestpath (biograph)

- MethodValue* String that specifies the algorithm used to find the shortest path. Choices are:
- 'Bellman-Ford' — Assumes weights of the edges to be nonzero entries in the N-by-N adjacency matrix. Time complexity is $O(N * E)$, where N and E are the number of nodes and edges respectively.
 - 'BFS' — Breadth-first search. Assumes all weights to be equal, and nonzero entries in the N-by-N adjacency matrix to represent edges. Time complexity is $O(N + E)$, where N and E are the number of nodes and edges respectively.
 - 'Acyclic' — Assumes the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj*, to be a directed acyclic graph and that weights of the edges are nonzero entries in the N-by-N adjacency matrix. Time complexity is $O(N + E)$, where N and E are the number of nodes and edges respectively.
 - 'Dijkstra' — Default algorithm. Assumes weights of the edges to be positive values in the N-by-N adjacency matrix. Time complexity is $O(\log(N) * E)$, where N and E are the number of nodes and edges respectively.
- WeightsValue* Column vector that specifies custom weights for the edges in the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. It must have one entry for every nonzero value (edge) in the N-by-N adjacency matrix. The order of the custom weights in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, *shortestpaths* gets weight information from the nonzero entries in the N-by-N adjacency matrix.

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions” in the *Bioinformatics Toolbox User’s Guide*.

`[dist, path, pred] = shortestpath(BGObj, S)` determines the single-source shortest paths from node *S* to all other nodes in the graph represented by an *N*-by-*N* adjacency matrix extracted from a biograph object, *BGObj*. Weights of the edges are all nonzero entries in the *N*-by-*N* adjacency matrix. *dist* are the *N* distances from the source to every node (using *Inf*s for nonreachable nodes and 0 for the source node). *path* contains the winning paths to every node. *pred* contains the predecessor nodes of the winning paths.

`[dist, path, pred] = shortestpath(BGObj, S, T)` determines the single source-single destination shortest path from node *S* to node *T*.

`[...] = shortestpath(..., 'PropertyName', PropertyValue, ...)` calls `shortestpath` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

`[...] = shortestpath(..., 'Directed', DirectedValue, ...)` indicates whether the graph represented by the *N*-by-*N* adjacency matrix extracted from a biograph object, *BGObj*, is directed or undirected. Set *DirectedValue* to `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`.

`[...] = shortestpath(..., 'Method', MethodValue, ...)` lets you specify the algorithm used to find the shortest path. Choices are:

- 'Bellman-Ford' — Assumes weights of the edges to be nonzero entries in the *N*-by-*N* adjacency matrix. Time complexity is $O(N \cdot E)$, where *N* and *E* are the number of nodes and edges respectively.

shortestpath (biograph)

- 'BFS' — Breadth-first search. Assumes all weights to be equal, and nonzero entries in the N-by-N adjacency matrix to represent edges. Time complexity is $O(N+E)$, where N and E are the number of nodes and edges respectively.
- 'Acyclic' — Assumes the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj*, to be a directed acyclic graph and that weights of the edges are nonzero entries in the N-by-N adjacency matrix. Time complexity is $O(N+E)$, where N and E are the number of nodes and edges respectively.
- 'Dijkstra' — Default algorithm. Assumes weights of the edges to be positive values in the N-by-N adjacency matrix. Time complexity is $O(\log(N)*E)$, where N and E are the number of nodes and edges respectively.

[...] = `shortestpath(..., 'Weights', WeightsValue, ...)` lets you specify custom weights for the edges. *WeightsValue* is a column vector having one entry for every nonzero value (edge) in the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. The order of the custom weights in the vector must match the order of the nonzero values in the N-by-N adjacency matrix when it is traversed column-wise. This property lets you use zero-valued weights. By default, `shortestpath` gets weight information from the nonzero entries in the N-by-N adjacency matrix.

References

- [1] Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.
- [2] Bellman, R. (1958). On a Routing Problem. *Quarterly of Applied Mathematics 16(1)*, 87–90.
- [3] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). *The Boost Graph Library User Guide and Reference Manual*, (Upper Saddle River, NJ:Pearson Education).

See Also

Bioinformatics Toolbox functions: `biograph` (object constructor), `graphshortestpath`

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object:

allshortestpaths, conncomp, isdag, isomorphism, isspantree,
maxflow, minspanntree, topoorder, traverse

single (DataMatrix)

Purpose Convert DataMatrix object to single-precision array

Syntax

```
B = single(DMObj)  
B = single(DMObj, Rows)  
B = single(DMObj, Rows, Cols)
```

Arguments

<i>DMObj</i>	DataMatrix object, such as created by <code>DataMatrix</code> (object constructor).
<i>Rows</i> , <i>Cols</i>	Row(s) or column(s) in <i>DMObj</i> , specified by one of the following: <ul style="list-style-type: none">• Scalar• Vector of positive integers• String specifying a row or column name• Cell array of row or column names• Logical vector

Return Values

<i>B</i>	MATLAB numeric array.
----------	-----------------------

Description

B = `single(DMObj)` converts *DMObj*, a DataMatrix object, to a single-precision array, which it returns in *B*.

B = `single(DMObj, Rows)` converts a subset of *DMObj*, a DataMatrix object, specified by *Rows*, to a single-precision array, which it returns in *B*. *Rows* can be a positive integer, vector of positive integers, string specifying a row name, cell array of row names, or a logical vector.

B = `single(DMObj, Rows, Cols)` converts a subset of *DMObj*, a DataMatrix object, specified by *Rows* and *Cols*, to a single-precision array, which it returns in *B*. *Cols* can be a positive integer, vector of

positive integers, string specifying a column name, cell array of column names, or a logical vector.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a `DataMatrix` object: `double`

sortcols (DataMatrix)

Purpose Sort columns of DataMatrix object in ascending or descending order

Syntax

```
DMObjNew = sortcols(DMObj1)  
DMObjNew = sortcols(DMObj1, Row)  
DMObjNew = sortcols(DMObj1, 'ColName')  
DMObjNew = sortcols(DMObj1, ..., Mode)  
[DMObjNew, Indices] = sortcols(DMObj1, ...)
```

Arguments

<i>DMObj1</i>	DataMatrix object, such as created by DataMatrix (object constructor).
<i>Row</i>	One or more rows in <i>DMObj1</i> by which to sort the columns. Choices are: <ul style="list-style-type: none">• Positive integer• Vector of positive integers• String specifying a row name• Cell array of strings specifying multiple row names• Logical vector
'ColName'	String that specifies to sort the columns by the column names.
<i>Mode</i>	String specifying the order by which to sort the columns. Choices are 'ascend' (default) or 'descend'.

Return Values

<i>DMObjNew</i>	DataMatrix object created from sorting the columns of another DataMatrix object.
<i>Indices</i>	Index vector that links <i>DMObj1</i> to <i>DMObjNew</i> . In other words, <i>DMObjNew</i> = <i>DMObj1</i> (:,idx).

Description *DMObjNew* = sortcols(*DMObj1*) sorts the columns in *DMObj1* in ascending order based on the elements in the first row. For any

columns that have equal elements in a row, sorting is based on the row immediately below.

DMObjNew = `sortcols(DMObj1, Row)` sorts the columns in *DMObj1* in ascending order based on the elements in the specified row. Any columns that have equal elements in the specified row are sorted based on the elements in the next specified row.

DMObjNew = `sortcols(DMObj1, 'ColName')` sorts the columns in *DMObj1* in ascending order according to the column names.

DMObjNew = `sortcols(DMObj1, ..., Mode)` specifies the order of the sort. *Mode* can be 'ascend' (default) or 'descend'.

`[DMObjNew, Indices] = sortcols(DMObj1, ...)` returns *Indices*, an index vector that links *DMObj1* to *DMObjNew*. In other words, *DMObjNew* = *DMObj1*(:,idx).

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a `DataMatrix` object: `sortrows`

sortrows (DataMatrix)

Purpose Sort rows of DataMatrix object in ascending or descending order

Syntax

```
DObjNew = sortrows(DObj1)  
DObjNew = sortrows(DObj1, Column)  
DObjNew = sortrows(DObj1, 'RowName')  
DObjNew = sortrows(DObj1, ..., Mode)  
[DObjNew, Indices] = sortrows(DObj1, ...)
```

Arguments

<i>DObj1</i>	DataMatrix object, such as created by DataMatrix (object constructor).
<i>Column</i>	One or more columns in <i>DObj1</i> by which to sort the rows. Choices are: <ul style="list-style-type: none">• Positive integer• Vector of positive integers• String specifying a column name• Cell array of strings specifying multiple column names• Logical vector
'RowName'	String that specifies to sort the rows by the row names.
<i>Mode</i>	String specifying the order by which to sort the rows. Choices are 'ascend' (default) or 'descend'.

Return Values

<i>DObjNew</i>	DataMatrix object created from sorting the rows of another DataMatrix object.
<i>Indices</i>	Index vector that links <i>DObj1</i> to <i>DObjNew</i> . In other words, <i>DObjNew</i> = <i>DObj1</i> (<i>idx</i> ,:).

Description *DObjNew* = sortrows(*DObj1*) sorts the rows in *DObj1* in ascending order based on the elements in the first column. For any rows that have

equal elements in a column, sorting is based on the column immediately to the right.

DMObjNew = `sortrows(DMObj1, Column)` sorts the rows in *DMObj1* in ascending order based on the elements in the specified column. Any rows that have equal elements in the specified column are sorted based on the elements in the next specified column.

DMObjNew = `sortrows(DMObj1, 'RowName')` sorts the rows in *DMObj1* in ascending order according to the row names.

DMObjNew = `sortrows(DMObj1, ..., Mode)` specifies the order of the sort. *Mode* can be 'ascend' (default) or 'descend'.

`[DMObjNew, Indices] = sortrows(DMObj1, ...)` returns *Indices*, an index vector that links *DMObj1* to *DMObjNew*. In other words, *DMObjNew* = `DMObj1(idx,:)`.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox method of a `DataMatrix` object: `sortcols`

std (DataMatrix)

Purpose Return standard deviation values in DataMatrix object

Syntax

```
S = std(DMObj)
S = std(DMObj, Flag)
S = std(DMObj, Flag, Dim)
S = std(DMObj, Flag, Dim, IgnoreNaN)
```

Arguments

<i>DMObj</i>	DataMatrix object, such as created by <code>DataMatrix</code> (object constructor).
<i>Flag</i>	Scalar specifying how to normalize the data. Choices are: <ul style="list-style-type: none">• 0 — Default. Normalizes using a sample size of $N - 1$, unless $N = 1$, in which case, normalizes using a sample size of 1.• 1 — Normalizes using a sample size of N. N = the number of elements in each column or row, as specified by <i>Dim</i> . For more information on the normalization equations, see the MATLAB function <code>std</code> .
<i>Dim</i>	Scalar specifying the dimension of <i>DMObj</i> to calculate the standard deviations. Choices are: <ul style="list-style-type: none">• 1 — Default. Returns standard deviation values for elements in each column.• 2 — Returns standard deviation values for elements in each row.
<i>IgnoreNaN</i>	Specifies if NaNs should be ignored. Choices are <code>true</code> (default) or <code>false</code> .

Return Values

S

Either of the following:

- Row vector containing the standard deviation values from elements in each column in *DMObj* (when *Dim* = 1)
- Column vector containing the standard deviation values from elements in each row in *DMObj* (when *Dim* = 2)

Description

$S = \text{std}(\text{DMObj})$ returns the standard deviation values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. The data is normalized using a sample size of $N - 1$, where N = the number of elements in each column. S is a row vector containing the standard deviation values for elements in each column in *DMObj*.

$S = \text{std}(\text{DMObj}, \text{Flag})$ specifies how to normalize the data. If *Flag* = 0, normalizes using a sample size of $N - 1$. If *Flag* = 1, normalizes using a sample size of N . N = the number of elements in each column or row, as specified by *Dim*. For more information on the normalization equations, see the MATLAB function `std`. Default *Flag* = 0.

$S = \text{std}(\text{DMObj}, \text{Flag}, \text{Dim})$ returns the standard deviation values of the elements in the columns or rows of a DataMatrix object, as specified by *Dim*. If *Dim* = 1, returns S , a row vector containing the standard deviation values for elements in each column in *DMObj*. If *Dim* = 2, returns S , a column vector containing the standard deviation values for elements in each row in *DMObj*. Default *Dim* = 1.

$S = \text{std}(\text{DMObj}, \text{Flag}, \text{Dim}, \text{IgnoreNaN})$ specifies if NaNs should be ignored. *IgnoreNaN* can be true (default) or false.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a DataMatrix object: `mean`, `median`, `var`

subtree (phytree)

Purpose Extract phylogenetic subtree

Syntax `Tree2 = subtree(Tree1, Nodes)`

Description `Tree2 = subtree(Tree1, Nodes)` extracts a new subtree (*Tree2*) where the new root is the first common ancestor of the *Nodes* vector from *Tree1*. Nodes in the tree are indexed as [1:NUMLEAVES] for the leaves and as [NUMLEAVES+1:NUMLEAVES+NUMBRANCHES] for the branches. Nodes can also be a logical array of following sizes [NUMLEAVES+NUMBRANCHES x 1], [NUMLEAVES x 1] or [NUMBRANCHES x 1].

Examples **1** Load a phylogenetic tree created from a protein family.

```
tr = phytread('pf00002.tree')
```

2 Get the subtree that contains the VIPS and CGRR human proteins.

```
sel = getbyname(tr,{'vips_human','cgrh_human'});  
sel = any(sel,2);  
tr = subtree(tr,sel)  
view(tr);
```

See Also Bioinformatics Toolbox functions: `phytree` (object constructor)

Bioinformatics Toolbox methods of `phytree` object: `get`, `getbyname`, `prune`, `select`

Purpose	Return sum of elements in DataMatrix object						
Syntax	<pre>S = sum(DMObj) S = sum(DMObj, Dim) S = sum(DMObj, Dim, IgnoreNaN)</pre>						
Arguments	<table><tr><td><i>DMObj</i></td><td>DataMatrix object, such as created by DataMatrix (object constructor).</td></tr><tr><td><i>Dim</i></td><td>Scalar specifying the dimension of <i>DMObj</i> to calculate the sums. Choices are:<ul style="list-style-type: none">• 1 — Default. Returns sum of elements in each column.• 2 — Returns sum of elements in each row.</td></tr><tr><td><i>IgnoreNaN</i></td><td>Specifies if NaNs should be ignored. Choices are true (default) or false.</td></tr></table>	<i>DMObj</i>	DataMatrix object, such as created by DataMatrix (object constructor).	<i>Dim</i>	Scalar specifying the dimension of <i>DMObj</i> to calculate the sums. Choices are: <ul style="list-style-type: none">• 1 — Default. Returns sum of elements in each column.• 2 — Returns sum of elements in each row.	<i>IgnoreNaN</i>	Specifies if NaNs should be ignored. Choices are true (default) or false.
<i>DMObj</i>	DataMatrix object, such as created by DataMatrix (object constructor).						
<i>Dim</i>	Scalar specifying the dimension of <i>DMObj</i> to calculate the sums. Choices are: <ul style="list-style-type: none">• 1 — Default. Returns sum of elements in each column.• 2 — Returns sum of elements in each row.						
<i>IgnoreNaN</i>	Specifies if NaNs should be ignored. Choices are true (default) or false.						
Return Values	<table><tr><td><i>S</i></td><td>Either of the following:<ul style="list-style-type: none">• Row vector containing the sums of the elements in each column in <i>DMObj</i> (when <i>Dim</i> = 1)• Column vector containing the sums of the elements in each row in <i>DMObj</i> (when <i>Dim</i> = 2)</td></tr></table>	<i>S</i>	Either of the following: <ul style="list-style-type: none">• Row vector containing the sums of the elements in each column in <i>DMObj</i> (when <i>Dim</i> = 1)• Column vector containing the sums of the elements in each row in <i>DMObj</i> (when <i>Dim</i> = 2)				
<i>S</i>	Either of the following: <ul style="list-style-type: none">• Row vector containing the sums of the elements in each column in <i>DMObj</i> (when <i>Dim</i> = 1)• Column vector containing the sums of the elements in each row in <i>DMObj</i> (when <i>Dim</i> = 2)						
Description	<p><i>S</i> = sum(<i>DMObj</i>) returns the sum of the elements in the columns of a DataMatrix object, treating NaNs as missing values. <i>S</i> is a row vector containing the sums of the elements in each column in <i>DMObj</i>. If the values in <i>DMObj</i> are singles, then <i>S</i> is a single; otherwise, <i>S</i> is a double.</p> <p><i>S</i> = sum(<i>DMObj</i>, <i>Dim</i>) returns the sum of the elements in the columns or rows of a DataMatrix object, as specified by <i>Dim</i>. If <i>Dim</i> = 1, returns</p>						

sum (DataMatrix)

S , a row vector containing the sums of the elements in each column in $DMObj$. If $Dim = 2$, returns S , a column vector containing the sums of the elements in each row in $DMObj$. Default $Dim = 1$.

$S = \text{sum}(DMObj, Dim, IgnoreNaN)$ specifies if NaNs should be ignored. $IgnoreNaN$ can be true (default) or false.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a `DataMatrix` object: `max`, `min`

Purpose

Multiply DataMatrix objects

Syntax

```
DMObjNew = times(DMObj1, DMObj2)
```

```
DMObjNew = DMObj1 .* DMObj2
```

```
DMObjNew = times(DMObj1, B)
```

```
DMObjNew = DMObj1 .* B
```

```
DMObjNew = times(B, DMObj1)
```

```
DMObjNew = B .* DMObj1
```

Arguments

DMObj1, *DMObj2* DataMatrix objects, such as created by DataMatrix (object constructor).

B MATLAB numeric or logical array.

Return Values

DMObjNew DataMatrix object created by multiplication.

Description

DMObjNew = times(*DMObj1*, *DMObj2*) or the equivalent *DMObjNew* = *DMObj1* .* *DMObj2* performs an element-by-element multiplication of the DataMatrix objects *DMObj1* and *DMObj2* and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *DMObj2* must have the same size (number of rows and columns), unless one is a scalar (1-by-1 DataMatrix object). The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*, unless *DMObj1* is a scalar; then they are the same as *DMObj2*.

DMObjNew = times(*DMObj1*, *B*) or the equivalent *DMObjNew* = *DMObj1* .* *B* performs an element-by-element multiplication of the DataMatrix object *DMObj1* and *B*, a numeric or logical array, and places the results in *DMObjNew*, another DataMatrix object. *DMObj1* and *B* must have the same size (number of rows and columns), unless *B* is a scalar. The size (number of rows and columns), row names, and column names for *DMObjNew* are the same as *DMObj1*.

times (DataMatrix)

$DMObjNew = \text{times}(B, DMObj1)$ or the equivalent $DMObjNew = B .* DMObj1$ performs an element-by-element multiplication of B , a numeric or logical array, and the DataMatrix object $DMObj1$, and places the results in $DMObjNew$, another DataMatrix object. $DMObj1$ and B must have the same size (number of rows and columns), unless B is a scalar. The size (number of rows and columns), row names, and column names for $DMObjNew$ are the same as $DMObj1$.

Note Arithmetic operations between a scalar DataMatrix object and a nonscalar array are not supported.

MATLAB calls $DMObjNew = \text{times}(X, Y)$ for the syntax $DMObjNew = X .* Y$ when X or Y is a DataMatrix object.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a DataMatrix object: `minus`, `plus`

MATLAB functions: Arithmetic Operators `+` `-` `*` `/` `\` `^` `'`

Purpose Perform topological sort of directed acyclic graph extracted from biograph object

Syntax `order = topoorder(BGObj)`

Arguments

`BGObj` Biograph object created by biograph (object constructor).

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions” in the *Bioinformatics Toolbox User’s Guide*.

`order = topoorder(BGObj)` returns an index vector with the order of the nodes sorted topologically. In topological order, an edge can exist between a source node `u` and a destination node `v`, if and only if `u` appears before `v` in the vector `order`. `BGObj` is a biograph object from which an N-by-N adjacency matrix is extracted and represents a directed acyclic graph (DAG). In the N-by-N sparse matrix, all nonzero entries indicate the presence of an edge.

References

[1] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also Bioinformatics Toolbox functions: biograph (object constructor), graphtopoorder

Bioinformatics Toolbox object: biograph object

Bioinformatics Toolbox methods of a biograph object:
allshortestpaths, conncomp, isdag, isomorphism, isspanntree,
maxflow, minspanntree, shortestpath, traverse

traverse (biograph)

Purpose Traverse biograph object by following adjacent nodes

Syntax

```
[disc, pred, closed] = traverse(BGObj, S)
[...] = traverse(BGObj, S, ...'Depth', DepthValue, ...)
[...] = traverse(BGObj, S, ...'Directed', DirectedValue, ...)
[...] = traverse(BGObj, S, ...'Method', MethodValue, ...)
```

Arguments

<i>BGObj</i>	Biograph object created by biograph (object constructor).
<i>S</i>	Integer that indicates the source node in <i>BGObj</i> .
<i>DepthValue</i>	Integer that indicates a node in <i>BGObj</i> that specifies the depth of the search. Default is Inf (infinity).
<i>DirectedValue</i>	Property that indicates whether graph represented by an N-by-N adjacency matrix extracted from a biograph object, <i>BGObj</i> is directed or undirected. Enter <code>false</code> for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is <code>true</code> .
<i>MethodValue</i>	String that specifies the algorithm used to traverse the graph. Choices are: <ul style="list-style-type: none">'BFS' — Breadth-first search. Time complexity is $O(N+E)$, where N and E are number of nodes and edges respectively.'DFS' — Default algorithm. Depth-first search. Time complexity is $O(N+E)$, where N and E are number of nodes and edges respectively.

Description

Tip For introductory information on graph theory functions, see “Graph Theory Functions” in the *Bioinformatics Toolbox User’s Guide*.

`[disc, pred, closed] = traverse(BGObj, S)` traverses the directed graph represented by an N-by-N adjacency matrix extracted from a biograph object, *BGObj*, starting from the node indicated by integer *S*. In the N-by-N sparse matrix, all nonzero entries indicate the presence of an edge. *disc* is a vector of node indices in the order in which they are discovered. *pred* is a vector of predecessor node indices (listed in the order of the node indices) of the resulting spanning tree. *closed* is a vector of node indices in the order in which they are closed.

`[...] = traverse(BGObj, S, ...'PropertyName', PropertyValue, ...)` calls `traverse` with optional properties that use property name/property value pairs. You can specify one or more properties in any order. Each *PropertyName* must be enclosed in single quotes and is case insensitive. These property name/property value pairs are as follows:

`[...] = traverse(BGObj, S, ...'Depth', DepthValue, ...)` specifies the depth of the search. *DepthValue* is an integer indicating a node in the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. Default is `Inf` (infinity).

`[...] = traverse(BGObj, S, ...'Directed', DirectedValue, ...)` indicates whether the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj* is directed or undirected. Set *DirectedValue* to `false` for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is `true`.

`[...] = traverse(BGObj, S, ...'Method', MethodValue, ...)` lets you specify the algorithm used to traverse the graph represented by the N-by-N adjacency matrix extracted from a biograph object, *BGObj*. Choices are:

- 'BFS' — Breadth-first search. Time complexity is $O(N+E)$, where *N* and *E* are number of nodes and edges respectively.
- 'DFS' — Default algorithm. Depth-first search. Time complexity is $O(N+E)$, where *N* and *E* are number of nodes and edges respectively.

traverse (biograph)

References

[1] Sedgewick, R., (2002). Algorithms in C++, Part 5 Graph Algorithms (Addison-Wesley).

[2] Siek, J.G., Lee, L-Q, and Lumsdaine, A. (2002). The Boost Graph Library User Guide and Reference Manual, (Upper Saddle River, NJ:Pearson Education).

See Also

Bioinformatics Toolbox functions: `biograph` (object constructor), `graphtraverse`

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a `biograph` object:
`allshortestpaths`, `conncomp`, `isdag`, `isomorphism`, `isspantree`,
`maxflow`, `minspantree`, `shortestpath`, `topoorder`

Purpose

Return variance values in DataMatrix object

Syntax

```
V = var(DMObj)
V = var(DMObj, Flag)
V = var(DMObj, Wgt)
V = var(..., Dim)
V = var(..., Dim, IgnoreNaN)
```

Arguments

DMObj

DataMatrix object, such as created by DataMatrix (object constructor).

Flag

Scalar specifying how to normalize the data. Choices are:

- 0 — Default. Normalizes using a sample size of $N - 1$, unless $N = 1$, in which case, normalizes using a sample size of 1.
- 1 — Normalizes using a sample size of N .

N = the number of elements in each column or row, as specified by *Dim*. For more information on the normalization equations, see the MATLAB function `std`.

Wgt

Weight vector equal in length to the dimension over which `var` operates (specified by *Dim*). It is used to compute the variance.

var (DataMatrix)

<i>Dim</i>	Scalar specifying the dimension of <i>DMObj</i> to calculate the variances. Choices are: <ul style="list-style-type: none">• 1 — Default. Returns variance values for elements in each column.• 2 — Returns variance values for elements in each row.
<i>IgnoreNaN</i>	Specifies if NaNs should be ignored. Choices are true (default) or false.

Return Values

<i>V</i>	An unbiased estimator of the variance within the columns or rows of a DataMatrix object. It can be either of the following: <ul style="list-style-type: none">• Row vector containing the variance values from elements in each column in <i>DMObj</i> (when <i>Dim</i> = 1)• Column vector containing the variance values from elements in each row in <i>DMObj</i> (when <i>Dim</i> = 2)
----------	---

Description

$V = \text{var}(DMObj)$ returns the variance values of the elements in the columns of a DataMatrix object, treating NaNs as missing values. The data is normalized using a sample size of $N - 1$, where N = the number of elements in each column. V is a row vector containing the variance values for elements in each column in *DMObj*. The variance is the square of the standard deviation.

$V = \text{var}(DMObj, Flag)$ specifies how to normalize the data. If *Flag* = 0, normalizes using a sample size of $N - 1$. If *Flag* = 1, normalizes using a sample size of N . N = the number of elements in each column or row, as specified by *Dim*. For more information on the normalization equations, see the MATLAB function `std`. Default *Flag* = 0.

$V = \text{var}(DMObj, Wgt)$ computes the variance using Wgt , a weight vector whose length must equal the length of the dimension over which var operates (specified by Dim). All elements in Wgt must be nonnegative. The var function normalizes Wgt to sum of 1.

$V = \text{var}(\dots, Dim)$ returns the variance values of the elements in the columns or rows of a `DataMatrix` object, as specified by Dim . If $Dim = 1$, returns V , a row vector containing the variance values for elements in each column in $DMObj$. If $Dim = 2$, returns V , a column vector containing the variance values for elements in each row in $DMObj$. Default $Dim = 1$.

$V = \text{var}(\dots, Dim, IgnoreNaN)$ specifies if NaNs should be ignored. $IgnoreNaN$ can be `true` (default) or `false`.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox object: `DataMatrix` object

Bioinformatics Toolbox methods of a `DataMatrix` object: `mean`, `median`, `std`

vertcat (DataMatrix)

Purpose Concatenate DataMatrix objects vertically

Syntax
 $DMObjNew = \text{vertcat}(DMObj1, DMObj2, \dots)$
 $DMObjNew = (DMObj1; DMObj2; \dots)$
 $DMObjNew = \text{vertcat}(DMObj1, B, \dots)$
 $DMObjNew = (DMObj1, B, \dots)$

Arguments

$DMObj1, DMObj2$	DataMatrix objects, such as created by DataMatrix (object constructor).
B	MATLAB numeric or logical array.

Return Values

$DMObjNew$	DataMatrix object created by vertical concatenation.
------------	--

Description

$DMObjNew = \text{vertcat}(DMObj1, DMObj2, \dots)$ or the equivalent $DMObjNew = (DMObj1; DMObj2; \dots)$ vertically concatenates the DataMatrix objects $DMObj1$ and $DMObj2$ into $DMObjNew$, another DataMatrix object. $DMObj1$ and $DMObj2$ must have the same number of columns. The column names and the order of columns for $DMObjNew$ are the same as $DMObj1$. The column names of $DMObj2$ and any other DataMatrix object input arguments are not preserved. The row names for $DMObjNew$ are the row names of $DMObj1$, $DMObj2$, and other DataMatrix object input arguments.

$DMObjNew = \text{vertcat}(DMObj1, B, \dots)$ or the equivalent $DMObjNew = (DMObj1, B, \dots)$ vertically concatenates the DataMatrix object $DMObj1$ and a numeric or logical array B into $DMObjNew$, another DataMatrix object. $DMObj1$ and B must have the same number of columns. The column names for $DMObjNew$ are the same as $DMObj1$. The column names of $DMObj2$ and any other DataMatrix object input arguments are not preserved. The row names for $DMObjNew$ are the row names of $DMObj1$ and empty for the rows from B .

MATLAB calls $DMObjNew = \text{vertcat}(X1, X2, X3, \dots)$ for the syntax $DMObjNew = [X1; X2; X3; \dots]$ when any one of $X1, X2, X3$, etc. is a DataMatrix object.

See Also

Bioinformatics Toolbox function: DataMatrix (object constructor)

Bioinformatics Toolbox object: DataMatrix object

Bioinformatics Toolbox methods of a DataMatrix object: horzcat

view (biograph)

Purpose Draw figure from biograph object

Syntax `view(BGobj)`
`BGobjHandle = view(BGobj)`

Arguments *BGobj* Biograph object created with the function `biograph`.

Description `view(BGobj)` opens a Figure window and draws a graph represented by a biograph object (*BGobj*). When the biograph object is already drawn in the Figure window, this function only updates the graph properties.

`BGobjHandle = view(BGobj)` returns a handle to a deep copy of the biograph object (*BGobj*) in the Figure window. When updating an existing figure, you can use the returned handle to change object properties programmatically or from the command line. When you close the Figure window, the handle is no longer valid. The original biograph object (*BGobj*) is left unchanged.

Examples **1** Create a biograph object.

```
cm = [0 1 1 0 0;1 0 0 1 1;1 0 0 0 0;0 0 0 0 1;1 0 1 0 0];  
bg = biograph(cm)
```

2 Render the biograph object into a Handles Graphic figure and get back a handle.

```
h = view(bg)
```

3 Change the color of all nodes and edges.

```
set(h.Nodes, 'Color', [.5 .7 1])  
set(h.Edges, 'LineColor', [0 0 0])
```

See Also Bioinformatics Toolbox function: `biograph` (object constructor)

Bioinformatics Toolbox object: `biograph` object

Bioinformatics Toolbox methods of a biograph object: dolayout, get, getancestors, getdescendants, getedgesbynodeid, getnodesbyid, getrelatives, set, view

view (clustergram)

Purpose View clustergram heat map and dendrograms for clustergram object

Syntax `view(CGobj)`

Arguments `CGobj` Clustergram object created with the function `clustergram`.

Description `view(CGobj)` opens a Clustergram window and draws a clustergram representing a clustergram object, `CGobj`. The clustergram shows hierarchical clustering using a heat map and dendrograms.

Note You can further explore the heat map and dendrograms using the mouse, toolbar buttons, and menu items in the Clustergram window. For more information, see the Examples section of the `clustergram` function.

Examples View the clustergram object created in the Examples section of the `clustergram` function.

```
view(cgo)
```

See Also Bioinformatics Toolbox function: `clustergram` (object constructor)
Bioinformatics Toolbox object: `clustergram` object
Bioinformatics Toolbox methods of a clustergram object: `get`, `plot`, `set`

Purpose View phylogenetic tree

Syntax `view(Tree)`
`view(Tree, IntNodes)`

Arguments

Tree Phylogenetic tree (phytree object) created with the function `phytree`.

IntNodes Nodes from the phytree object to initially display in the *Tree*.

Description

`view(Tree)` opens the Phylogenetic Tree Tool window and draws a tree from data in a `phytree` object (*Tree*). The significant distances between branches and nodes are in the horizontal direction. Vertical distances have no significance and are selected only for display purposes. You can access tools to edit and analyze the tree from the Phylogenetic Tree Tool menu bar or by using the left and right mouse buttons.

`view(Tree, IntNodes)` opens the Phylogenetic Tree Tool window with an initial selection of nodes specified by *IntNodes*. *IntNodes* can be a logical array of any of the following sizes: `NumLeaves + NumBranches x 1`, `NumLeaves x 1`, or `NumBranches x 1`. *IntNodes* can also be a list of indices.

Example

```
tree = phytreeread('pf00002.tree')
view(tree)
```

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreeread`, `phytreetool`, `seqlinkage`, `seqneighjoin`

Bioinformatics Toolbox object: `phytree` object

Bioinformatics Toolbox method of `phytree` object: `plot`

weights (phytree)

Purpose Calculate weights for phylogenetic tree

Syntax `W = weights(Tree)`

Arguments

<code>Tree</code>	Phylogenetic tree (phytree object) created with the function <code>phytree</code> .
-------------------	---

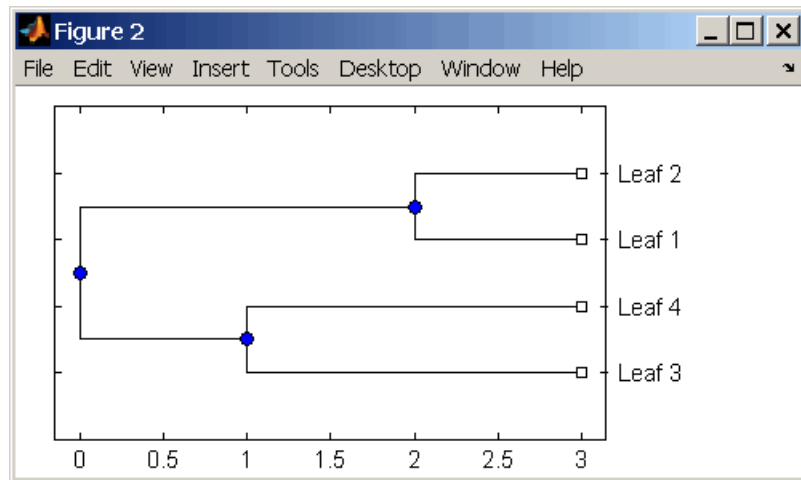
Description `W = weights(Tree)` calculates branch proportional weights for every leaf in a tree (`Tree`) using the Thompson-Higgins-Gibson method. The distance of every segment of the tree is adjusted by dividing it by the number of leaves it contains. The sequence weights are the result of normalizing to unity the new patristic distances between every leaf and the root.

Examples **1** Create an ultrametric tree with specified branch distances.

```
bd = [1 2 3]';  
tr_1 = phytree([1 2;3 4;5 6],bd)
```

2 View the tree.

```
view(tr_1)
```



3 Display the calculated weights.

```
weights(tr_1)
```

```
ans =
```

```
1.0000  
1.0000  
0.8000  
0.8000
```

References

[1] Thompson JD, Higgins DG, Gibson TJ (1994), "CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Research*, 22(22):4673-4680.

[2] Henikoff S, Henikoff JG (1994), "Position-based sequence weights," *Journal Molecular Biology*, 243(4):574-578.

See Also

Bioinformatics Toolbox functions: `multialign`, `phytree` (object constructor), `proalign`, `seqlinkage`

weights (phytree)

Object Reference

biograph object

Purpose Data structure containing generic interconnected data used to implement directed graph

Description A biograph object is a data structure containing generic interconnected data used to implement a directed graph. Nodes represent proteins, genes, or any other biological entity, and edges represent interactions, dependences, or any other relationship between the nodes. A biograph object also stores information, such as color properties and text label characteristics, used to create a 2-D visualization of the graph.

You create a biograph object using the object constructor function `biograph`. You can view a graphical representation of a biograph object using the `view` method.

Method Summary

Following are methods of a biograph object:

<code>allshortestpaths (biograph)</code>	Find all shortest paths in biograph object
<code>conncomp (biograph)</code>	Find strongly or weakly connected components in biograph object
<code>dolayout (biograph)</code>	Calculate node positions and edge trajectories
<code>get (biograph)</code>	Retrieve information about biograph object
<code>getancestors (biograph)</code>	Find ancestors in biograph object
<code>getdescendants (biograph)</code>	Find descendants in biograph object
<code>getedgesbynodeid (biograph)</code>	Get handles to edges in biograph object
<code>getmatrix (biograph)</code>	Get connection matrix from biograph object
<code>getnodesbyid (biograph)</code>	Get handles to nodes
<code>getrelatives (biograph)</code>	Find relatives in biograph object

isdag (biograph)	Test for cycles in biograph object
isomorphism (biograph)	Find isomorphism between two biograph objects
isspantree (biograph)	Determine if tree created from biograph object is spanning tree
maxflow (biograph)	Calculate maximum flow in biograph object
minspantree (biograph)	Find minimal spanning tree in biograph object
set (biograph)	Set property of biograph object
shortestpath (biograph)	Solve shortest path problem in biograph object
topoorder (biograph)	Perform topological sort of directed acyclic graph extracted from biograph object
traverse (biograph)	Traverse biograph object by following adjacent nodes
view (biograph)	Draw figure from biograph object

Following are methods of a node object:

getancestors (biograph)	Find ancestors in biograph object
getdescendants (biograph)	Find descendants in biograph object
getrelatives (biograph)	Find relatives in biograph object

Property Summary

A biograph object contains two kinds of objects, node objects and edge objects, that have their own properties. For a list of the properties of node objects and edge objects, see the following tables.

biograph object

Properties of a Biograph Object

Property	Description
ID	String to identify the biograph object. Default is ''.
Label	String to label the biograph object. Default is ''.
Description	String that describes the biograph object. Default is ''.
LayoutType	String that specifies the algorithm for the layout engine. Choices are: <ul style="list-style-type: none">• 'hierarchical' (default) — Uses a topological order of the graph to assign levels, and then arranges the nodes from top to bottom, while minimizing crossing edges.• 'radial' — Uses a topological order of the graph to assign levels, and then arranges the nodes from inside to outside of the circle, while minimizing crossing edges.• 'equilibrium' — Calculates layout by minimizing the energy in a dynamic spring system.

Properties of a Biograph Object (Continued)

Property	Description
EdgeType	<p>String that specifies how edges display. Choices are:</p> <ul style="list-style-type: none"> 'straight' 'curved' (default) 'segmented' <hr/> <p>Note Curved or segmented edges occur only when necessary to avoid obstruction by nodes. Biograph objects with LayoutType equal to 'equilibrium' or 'radial' cannot produce curved or segmented edges.</p> <hr/>
Scale	Positive number that post-scales the node coordinates. Default is 1.
LayoutScale	Positive number that scales the size of the nodes before calling the layout engine. Default is 1.
EdgeTextColor	Three-element numeric vector of RGB values. Default is [0, 0, 0], which defines black.
EdgeFontSize	Positive number that sets the size of the edge font in points. Default is 8.
ShowArrows	Controls the display of arrows with the edges. Choices are 'on' (default) or 'off'.
ArrowSize	Positive number that sets the size of the arrows in points. Default is 8.
ShowWeights	Controls the display of text indicating the weight of the edges. Choices are 'on' (default) or 'off'.

biograph object

Properties of a Biograph Object (Continued)

Property	Description
ShowTextInNodes	<p>String that specifies the node property used to label nodes when you display a biograph object using the view method. Choices are:</p> <ul style="list-style-type: none">• 'Label' — Uses the Label property of the node object (default).• 'ID' — Uses the ID property of the node object.• 'None'
NodeAutoSize	<p>Controls precalculating the node size before calling the layout engine. Choices are 'on' (default) or 'off'.</p>
NodeCallback	<p>User-defined callback for all nodes. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the view function to display the biograph object in the Biograph Viewer, you can double-click a node to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, @(node) inspect(node), which displays the Property Inspector dialog box.</p>

Properties of a Biograph Object (Continued)

Property	Description
EdgeCallback	User-defined callback for all edges. Enter the name of a function, a function handle, or a cell array with multiple function handles. After using the <code>view</code> function to display the biograph object in the Biograph Viewer, you can double-click an edge to activate the first callback, or right-click and select a callback to activate. Default is the anonymous function, <code>@(edge) inspect(edge)</code> , which displays the Property Inspector dialog box.
CustomNodeDrawFcn	Function handle to a customized function to draw nodes. Default is <code>[]</code> .
Nodes	Read-only column vector with handles to node objects of a biograph object. The size of the vector is the number of nodes. For properties of node objects, see Properties of a Node Object on page 5-7.
Edges	Read-only column vector with handles to edge objects of a biograph object. The size of the vector is the number of edges. For properties of edge objects, see Properties of an Edge Object on page 5-9.

Properties of a Node Object

Property	Description
ID	Character string defined when the biograph object is created, either by the <code>NodeIDs</code> input argument or internally by the <code>biograph</code> constructor function. You can modify this property using the <code>set</code> method, but each node object's ID must be unique.

biograph object

Properties of a Node Object (Continued)

Property	Description
Label	String for labeling a node when you display a biograph object using the view method. Default is ''.
Description	String that describes the node. Default is ''.
Position	Two-element numeric vector of x - and y -coordinates, for example, [150, 150]. If you do not specify this property, default is initially [], then when the layout algorithms are executed, it becomes a two-element numeric vector of x - and y -coordinates computed by the layout engine.
Shape	String that specifies the shape of the nodes. Choices are: <ul style="list-style-type: none">• 'box' (default)• 'ellipse'• 'circle'• 'rectangle'• 'diamond'• 'trapezium'• 'invtrapezium'• 'house'• 'inverse'• 'parallelogram'

Properties of a Node Object (Continued)

Property	Description
Size	Two-element numeric vector calculated before calling the layout engine using the actual font size and shape of the node. Default is [10, 10].
Color	Three-element numeric vector of RGB values that specifies the fill color of the node. Default is [1, 1, 0.7], which defines yellow.
LineWidth	Positive number. Default is 1.
LineColor	Three-element numeric vector of RGB values that specifies the outline color of the node. Default is [0.3, 0.3, 1], which defines blue.
FontSize	Positive number that sets the size of the node font in points. Default is 8.
TextColor	Three-element numeric vector of RGB values that specifies the color of the node labels. Default is [0, 0, 0], which defines black.
UserData	Miscellaneous, user-defined data that you want to associate with the node. The node does not use this property, but you can access and specify it using the get and set functions. Default is [].

Properties of an Edge Object

Property	Description
ID	Character string automatically generated from the node IDs when the biograph object is created by the biograph constructor function. You can modify this property using the set method, but each edge object's ID must be unique.

biograph object

Properties of an Edge Object (Continued)

Property	Description
Label	String for labeling an edge when you display a biograph object using the view method. Default is ''.
Description	String that describes the edge. Default is ''.
Weight	Value that represents the weight (cost, distance, length, or capacity) associated with the edge. Default is 1.
LineWidth	Positive number. Default is 1.
LineColor	Three-element numeric vector of RGB values that specifies the color of the edge. Default is [0.5, 0.5, 0.5], which defines gray.
UserData	Miscellaneous, user-defined data that you want to associate with the edge. The edge does not use this property, but you can access and specify it using the get and set functions. Default is [].

Examples

Determining Properties and Property Values of a Biograph Object

You can display all properties and their current values of a biograph object, *BGobj*, by using the following syntax:

```
get(BGobj)
```

You can return all properties and their current values of *BGobj*, a biograph object, to *BGstruct*, a scalar structure in which each field name is a property of a biograph object, and each field contains the value of that property, by using the following syntax:

```
BGstruct = get(BGobj)
```

You can return the value of a specific property of a biograph object, *BGobj*, by using either of the following syntaxes:

```
PropertyValue = get(BGobj, 'PropertyName')
```

```
PropertyValue = BGobj.PropertyName
```

You can return the value of specific properties of a biograph object, *BGobj*, by using the following syntax:

```
[Property1Value, Property2Value, ...] = get(BGobj,  
'Property1Name', 'Property2Name', ...)
```

Determining Possible Values of Biograph Object Properties

You can display possible values for all properties that have a fixed set of property values in a biograph object, *BGobj*, by using the following syntax:

```
set(BGobj)
```

You can display possible values for a specific property that has a fixed set of property values in a biograph object, *BGobj*, by using the following syntax:

You can access allowed values for any property that has a finite set of choices by using the following syntax:

```
set(BGobj, 'PropertyName')
```

Specifying Properties of a Biograph Object

You can set a specific property of a biograph object, *BGobj*, by using either of the following syntaxes:

```
set(BGobj, 'PropertyName', PropertyValue)
```

```
BGobj.PropertyName = PropertyValue
```

You can set multiple properties of a biograph object, *BGobj*, by using the following syntax:

```
set(BGobj, 'PropertyName1', PropertyValue1,  
'PropertyName2', PropertyValue2, ...)
```

See Also

Bioinformatics Toolbox function: `biograph` (object constructor)

biograph object

Bioinformatics Toolbox methods of a biograph object:
allshortestpaths, conncomp, dolayout, get, getancestors,
getdescendants, getedgesbynodeid, getmatrix, getnodesbyid,
getrelatives, isdag, isomorphism, isspantree, maxflow,
minspantree, set, shortestpath, topoorder, traverse, view

Purpose

Object containing hierarchical clustering analysis data

Description

A clustergram object contains hierarchical clustering analysis data that you can view in a heat map and dendrograms.

You create a clustergram object using the object constructor function `clustergram`. You can view a graphical representation of the clustergram object in a heat map and dendrograms using the `view` method.

Method Summary

Following are methods of a clustergram object:

<code>get (clustergram)</code>	Retrieve information about clustergram object
<code>plot (clustergram)</code>	Render clustergram heat map and dendrograms for clustergram object
<code>set (clustergram)</code>	Set property of clustergram object
<code>view (clustergram)</code>	View clustergram heat map and dendrograms for clustergram object

Property Summary

Properties of a Clustergram Object

Property	Description
<code>RowLabels</code>	Vector of numbers or cell array of text strings to label the rows in the dendrogram and heat map. Default is a vector of values 1 through M , where M is the number of rows in <i>Data</i> , the matrix of data used by the <code>clustergram</code> function to create the clustergram object.

clustergram object

Properties of a Clustergram Object (Continued)

Property	Description
ColumnLabels	Vector of numbers or cell array of text strings to label the columns in the dendrogram and heat map. Default is a vector of values 1 through M , where M is the number of columns in <i>Data</i> , the matrix of data used by the <code>clustergram</code> function to create the clustergram object.
RowGroupNames	A cell array of text strings containing the names of the row groups exported to a clustergram object created using the Export Group to Workspace command in the Clustergram window.
RowNodeNames	A cell array of text strings containing the names of the row nodes exported to a clustergram object created using the Export Group to Workspace command in the Clustergram window.
ColumnGroupNames	A cell array of text strings containing the names of the column groups exported to a clustergram object created using the Export Group to Workspace command in the Clustergram window.
ColumnNodeNames	A cell array of text strings containing the names of the column nodes exported to a clustergram object created using the Export Group to Workspace command in the Clustergram window.

Properties of a Clustergram Object (Continued)

Property	Description
ExprValues	An M by N matrix of data, where M and N are the number of row nodes and of column nodes respectively, exported to a clustergram object created using the Export Group to Workspace command in the Clustergram window. If the matrix contains gene expression data, typically each row corresponds to a gene and each column corresponds to sample.
Standardize	Numeric value that specifies the dimension for standardizing the values in the data. The standardized values are transformed so that the mean is 0 and the standard deviation is 1 in the specified dimension. Choices are: <ul style="list-style-type: none"> • 1 — Standardize along the columns of data. • 2 (default) — Standardize along the rows of data. • 3 — Do not perform standardization.
Cluster	Numeric value that specifies the dimension for clustering the values in the data. Choices are: <ul style="list-style-type: none"> • 1 — Cluster along the columns of data only, which results in clustered rows. • 2 — Cluster along the rows of data only, which results in clustered columns. • 3 (default) — Cluster along the columns of data, then cluster along the rows of row-clustered data.

clustergram object

Properties of a Clustergram Object (Continued)

Property	Description
RowPdist	<p>String that specifies the distance metric to pass to the <code>pdist</code> function (Statistics Toolbox software) to use to calculate the pairwise distances between rows. For information on choices, see the <code>pdist</code> function. Default is 'euclidean'.</p> <hr/> <p>Note If the distance metric requires extra arguments, then <i>RowPdistValue</i> is a cell array. For example, to use the Minkowski distance with exponent <i>P</i>, you would use {'minkowski', <i>P</i>}.</p> <hr/>
ColumnPdist	<p>String that specifies the distance metric to pass to the <code>pdist</code> function (Statistics Toolbox software) to use to calculate the pairwise distances between columns. For information on choices, see the <code>pdist</code> function. Default is 'euclidean'.</p> <hr/> <p>Note If the distance metric requires extra arguments, then <i>ColumnPdistValue</i> is a cell array. For example, to use the Minkowski distance with exponent <i>P</i>, you would use {'minkowski', <i>P</i>}.</p> <hr/>

Properties of a Clustergram Object (Continued)

Property	Description
Linkage	String or two-element cell array of strings that specifies the linkage method to pass to the <code>linkage</code> function (Statistics Toolbox software) to use to create the hierarchical cluster tree for rows and columns. If a two-element cell array of strings, the first element is used for linkage between rows, and the second element is used for linkage between columns. For information on choices, see the <code>linkage</code> function. Default is 'average'.
Dendrogram	Scalar or two-element numeric vector or cell array of strings that specifies the 'colorthreshold' property to pass to the <code>dendrogram</code> function (Statistics Toolbox software) to create the dendrogram plot. If a two-element numeric vector or cell array, the first element is for the rows, and the second element is for the columns. For more information, see the <code>dendrogram</code> function.

clustergram object

Properties of a Clustergram Object (Continued)

Property	Description
OptimalLeafOrder	<p>Property to enable or disable the optimal leaf ordering calculation, which determines the leaf order that maximizes the similarity between neighboring leaves. Choices are <code>true</code> (enable) or <code>false</code> (disable). Default depends on the size of <i>Data</i>, the matrix of data used to create the clustergram object. If the number of rows or columns in <i>Data</i> is greater than 1000, default is <code>false</code>; otherwise, default is <code>true</code>.</p> <hr/> <p>Note Disabling the optimal leaf ordering calculation can be useful when working with large data sets because this calculation uses a large amount of memory and can be very time consuming.</p> <hr/>
ColorMap	<p>Either of the following:</p> <ul style="list-style-type: none">• M-by-3 matrix of RGB values• Name or function handle of a function that returns a colormap, such as <code>redgreencmap</code> or <code>redbluecmap</code> <p>Default is <code>redgreencmap</code>.</p>

Properties of a Clustergram Object (Continued)

Property	Description
DisplayRange	<p>Positive scalar that specifies the display range of standardized values. Default is 3, which means there is a color variation for values between -3 and 3, but values >3 will be the same color as 3, and values < -3 will be the same color as -3.</p> <p>For example, if you specify redgreencmap for the 'ColorMap' property, pure red represents values $\geq DisplayRangeValue$, and pure green represents values $\leq -DisplayRangeValue$.</p>
SymmetricRange	Property to force the color scale of the heat map to be symmetric around zero. Choices are true (default) or false.
LogTrans	Controls the \log_2 transform of the data from natural scale. Choices are true or false (default).

clustergram object

Properties of a Clustergram Object (Continued)

Property	Description
Ratio	<p>Either of the following:</p> <ul style="list-style-type: none">• Scalar• Two-element vector <p>It specifies the ratio of space that the row and column dendrograms occupy relative to the heat map. If <i>RatioValue</i> is a scalar, it is used as the ratio for both dendrograms. If <i>RatioValue</i> is a two-element vector, the first element is used for the ratio of the row dendrogram width to the heat map width, and the second element is used for the ratio of the column dendrogram height to the heat map height. The second element is ignored for one-dimensional clustergrams. Default is 1/5.</p>
Impute	<p>Any of the following:</p> <ul style="list-style-type: none">• Name of a function that imputes missing data.• Handle to a function that imputes missing data.• Cell array where the first element is the name of or handle to a function that imputes missing data and the remaining elements are property name/property value pairs used as inputs to the function.

Properties of a Clustergram Object (Continued)

Property	Description
RowMarker	<p>Optional structure array for annotating the groups (clusters) of rows determined by the <code>clustergram</code> function. Each structure in the array represents a group of rows and contains the following fields:</p> <ul style="list-style-type: none"> • GroupNumber — The row group number to annotate. • Annotation — String specifying text to annotate the row group. • Color — String or three-element vector of RGB values specifying a color, which is used to label the row group. For more information on specifying colors, see <code>colorspec</code>. If this field is empty, default is 'blue'.
ColumnMarker	<p>Optional structure array for annotating groups (clusters) of columns determined by the <code>clustergram</code> function. Each structure in the array represents a group of columns and contains the following fields:</p> <ul style="list-style-type: none"> • GroupNumber — The column group number to annotate. • Annotation — String specifying text to annotate the column group. • Color — String or three-element vector of RGB values specifying a color, which is used to label the column group. For more information on specifying colors, see <code>colorspec</code>. If this field is empty, default is 'blue'.

clustergram object

Examples

Determining Properties and Property Values of a Clustergram Object

You can display all properties and their current values of a clustergram object, *CGobj*, by using the following syntax:

```
get(CGobj)
```

You can return all properties and their current values of *CGobj*, a clustergram object, to *CGstruct*, a scalar structure in which each field name is a property of a clustergram object, and each field contains the value of that property, by using the following syntax:

```
CGstruct = get(CGobj)
```

You can return the value of a specific property of a clustergram object, *CGobj*, by using either of the following syntaxes:

```
PropertyValue = get(CGobj, 'PropertyName')
```

```
PropertyValue = CGobj.PropertyName
```

You can return the value of specific properties of a clustergram object, *CGobj*, by using the following syntax:

```
[Property1Value, Property2Value, ...] = get(CGobj,  
'Property1Name', 'Property2Name', ...)
```

Determining Possible Values of Clustergram Object Properties

You can display possible values for all properties that have a fixed set of property values in a clustergram object, *CGobj*, by using the following syntax:

```
set(CGobj)
```

You can display possible values for a specific property that has a fixed set of property values in a clustergram object, *CGobj*, by using the following syntax:

```
set(CGobj, 'PropertyName')
```

Specifying Properties of a Clustergram Object

You can set a specific property of a clustergram object, *CGobj*, by using either of the following syntaxes:

```
set(CGobj, 'PropertyName', PropertyValue)
```

```
CGobj.PropertyName = PropertyValue
```

You can set multiple properties of a clustergram object, *CGobj*, by using the following syntax:

```
set(CGobj, 'PropertyName1', PropertyValue1,  
'PropertyName2', PropertyValue2, ...)
```

See Also

Bioinformatics Toolbox function: `clustergram` (object constructor)

Bioinformatics Toolbox methods of a clustergram object: `get`, `plot`, `set`, `view`

MATLAB function: `display`

DataMatrix object

Purpose

Data structure encapsulating data and metadata from microarray experiment so that it can be indexed by gene or probe identifiers and by sample identifiers

Description

A DataMatrix object is a data structure encapsulating measurement data and feature metadata from a microarray experiment so that it can be indexed by gene or probe identifiers and by sample identifiers. A DataMatrix object stores experimental data in a matrix, with rows typically corresponding to gene names or probe identifiers, and columns typically corresponding to sample identifiers. A DataMatrix object also stores metadata, such as the gene names or probe identifiers and sample identifiers, in row names and column names.

You create a DataMatrix object using the object constructor function `DataMatrix`.

Property Summary

Properties of a DataMatrix Object

Property	Description
Name	String that describes the DataMatrix object. Default is ''.
RowNames	Empty array or cell array of strings that specifies the names for the rows, typically gene names or probe identifiers. The number of elements in the cell array must equal the number of rows in the matrix. Default is an empty array.
ColNames	Empty array or cell array of strings that specifies the names for the columns, typically sample identifiers. The number of elements in the cell array must equal the number of columns in the matrix.

Properties of a DataMatrix Object (Continued)

Property	Description
NRows	<p>Read-only. Positive number that specifies the number of rows in the matrix.</p> <hr/> <p>Note You cannot modify this property directly. You can access it using the <code>get</code> method.</p> <hr/>
NCols	<p>Read-only. Positive number that specifies the number of columns in the matrix.</p> <hr/> <p>Note You cannot modify this property directly. You can access it using the <code>get</code> method.</p> <hr/>
NDims	<p>Read-only. Positive number that specifies the number of dimensions in the matrix.</p> <hr/> <p>Note You cannot modify this property directly. You can access it using the <code>get</code> method.</p> <hr/>
ElementClass	<p>Read-only. String that specifies the class type of the elements in the DataMatrix object, such as <code>single</code> or <code>double</code>.</p> <hr/> <p>Note You cannot modify this property directly. You can access it using the <code>get</code> method.</p> <hr/>

DataMatrix object

Method Summary

General Methods of a DataMatrix Object

Method	Description
colnames	Retrieve or set column names of DataMatrix object.
disp	Display DataMatrix object.
display	Display DataMatrix object, printing DataMatrix object name. To invoke this method, enter the name of a DataMatrix object at the command prompt.
double	Convert DataMatrix object to double-precision array.
get	Retrieve information about DataMatrix object.
isempty	Determine if DataMatrix object is empty.
isfinite	Determine if DataMatrix object elements are finite.
isinf	Determine if DataMatrix object elements are infinite.
isnan	Determine if DataMatrix object elements are NaN.
isscalar	Determine if DataMatrix object is scalar.
isequal	Test DataMatrix objects for equality.
isequalwithequalnans	Test DataMatrix objects for equality, treating NaNs as equal.
isvector	Determine if DataMatrix object is vector.
length	Return length of DataMatrix object.
ndims	Return number of dimensions in DataMatrix object.

General Methods of a DataMatrix Object (Continued)

Method	Description
numel	Return number of elements in DataMatrix object.
plot	Draw 2-D line plot of DataMatrix object.
rownames	Retrieve or set row names of DataMatrix object.
set	Set property of DataMatrix object.
single	Convert DataMatrix object to single-precision array.
size	Return size of DataMatrix object.

Methods for Manipulating the Data in a DataMatrix Object

Method	Description
cat	Concatenate DataMatrix objects. The horzcat and vertcat methods implement special cases.
horzcat	Concatenate DataMatrix objects horizontally.
sortcols	Sort columns of DataMatrix object in ascending or descending order.
sortrows	Sort rows of DataMatrix object in ascending or descending order.
subsasgn	Subscripted assignment for DataMatrix object. To invoke this method, use parentheses or dot indexing described in “Accessing DataMatrix Objects” in the Bioinformatics Toolbox User’s Guide.

DataMatrix object

Methods for Manipulating the Data in a DataMatrix Object (Continued)

Method	Description
subsref	Subscripted reference for DataMatrix object. To invoke this method, use parentheses or dot indexing described in “Accessing DataMatrix Objects” in the Bioinformatics Toolbox User’s Guide.
transpose	Transpose DataMatrix object.
vertcat	Concatenate DataMatrix objects vertically.

Descriptive Statistics and Statistical Learning Methods

Method	Description
kmeans	K-means clustering.
max	Return maximum values in DataMatrix object.
mean	Return average or mean values in DataMatrix object.
median	Return median values in DataMatrix object.
min	Return minimum values in DataMatrix object.
nanmax	Return maximum values in DataMatrix object ignoring NaN values.
nanmean	Return average or mean values in DataMatrix object ignoring NaN values.
nanmedian	Return median values in DataMatrix object ignoring NaN values.
nanmin	Return minimum values in DataMatrix object ignoring NaN values.

Descriptive Statistics and Statistical Learning Methods (Continued)

Method	Description
nanstd	Return standard deviation values in DataMatrix object ignoring NaN values.
nansum	Return sum of elements in DataMatrix object ignoring NaN values.
nanvar	Return variance values in DataMatrix object ignoring NaN values.
pdist	Pairwise distance.
princomp	Principal component analysis on data.
std	Return standard deviation values in DataMatrix object.
sum	Return sum of elements in DataMatrix object.
var	Return variance values in DataMatrix object.

Unary Methods – Exponential

Method	Description
exp	Exponential.
log	Natural logarithm.
log10	Common (base 10) logarithm.
log2	Base 2 logarithm and dissect floating-point numbers into exponent and mantissa.
pow2	Base 2 power and scale floating-point numbers.
sqrt	Square root.

DataMatrix object

Unary Methods – Integer

Method	Description
ceil	Round DataMatrix object toward infinity.
fix	Round DataMatrix object toward zero.
floor	Round DataMatrix object toward minus infinity.
round	Round DataMatrix object to nearest integer.

Unary Methods – Custom

Method	Description
dmarrayfun	Apply function to each element in DataMatrix object.

Binary Methods – Arithmetic Operator

Operator	Method	Description
+	plus	Add DataMatrix objects
-	minus	Subtract DataMatrix objects.
.*	times	Multiply DataMatrix objects.
./	rdivide	Right array divide DataMatrix objects.
.\	ldivide	Left array divide DataMatrix objects.
.^	power	Array power DataMatrix objects.

Binary Methods – Relational Operator

Operator	Method	Description
<	lt	Test DataMatrix objects for less than.

Binary Methods – Relational Operator (Continued)

Operator	Method	Description
<=	le	Test DataMatrix objects for less than or equal to.
>	gt	Test DataMatrix objects for greater than.
>=	ge	Test DataMatrix objects for greater than or equal to.
==	eq	Test DataMatrix objects for equality.
~=	ne	Test DataMatrix objects for inequality.

Binary Methods – Custom

Method	Description
dmbxfun	Apply element-by-element binary operation to two DataMatrix objects with singleton expansion enabled.

Examples

Determining Properties and Property Values of a DataMatrix Object

You can display all properties and their current values of a DataMatrix object, *DMobj*, by using the following syntax:

```
get(DMobj)
```

You can return all properties and their current values of *DMobj*, a DataMatrix object, to *DMstruct*, a scalar structure in which each field name is a property of a DataMatrix object, and each field contains the value of that property, by using the following syntax:

```
DMstruct = get(DMobj)
```

DataMatrix object

You can return the value of a specific property of a DataMatrix object, *DMObj*, by using either of the following syntaxes:

```
PropertyValue = get(DMObj, 'PropertyName') )
```

```
PropertyValue = DMObj.PropertyName
```

You can return the value of specific properties of a DataMatrix object, *DMObj*, by using the following syntax:

```
[Property1Value, Property2Value, ...] = get(DMObj,  
'Property1Name', 'Property2Name', ...)
```

Determining Possible Values of DataMatrix Object Properties

You can display possible values for all properties that have a fixed set of property values in a DataMatrix object, *DMObj*, by using the following syntax:

```
set(DMObj)
```

You can display possible values for a specific property that has a fixed set of property values in a DataMatrix object, *DMObj*, by using the following syntax:

```
set(DMObj, 'PropertyName' )
```

Specifying Properties of a DataMatrix Object

You can set a specific property of a DataMatrix object, *DMObj*, by using either of the following syntaxes:

```
DMObj = set(DMObj, 'PropertyName', PropertyValue)
```

```
DMObj.PropertyName = PropertyValue
```

You can set multiple properties of a DataMatrix object, *DMObj*, by using the following syntax:

```
set(DMObj, 'PropertyName1', PropertyValue1, 'PropertyName2',  
PropertyValue2, ...)
```

Note For more examples of creating and using DataMatrix objects, see “Working with DataMatrix Objects” in the Bioinformatics Toolbox User’s Guide.

See Also

Bioinformatics Toolbox function: `DataMatrix` (object constructor)

Bioinformatics Toolbox methods of a DataMatrix object: `colnames`, `disp`, `dmarrayfun`, `dmsxfun`, `double`, `eq`, `ge`, `get`, `gt`, `horzcat`, `isequal`, `isequalwithqualnans`, `ldivide`, `le`, `lt`, `max`, `mean`, `median`, `min`, `minus`, `ndims`, `ne`, `numel`, `plot`, `plus`, `power`, `rdivide`, `rownames`, `set`, `single`, `sortcols`, `sortrows`, `std`, `sum`, `times`, `var`, `vertcat`

geneont object

Purpose Data structure containing Gene Ontology (GO) information

Description A geneont object is a data structure containing Gene Ontology information. Gene Ontology terms can be explored and traversed through “is_a” and “part_of” relationships.

Method Summary Following are methods of a geneont object:

getancestors (geneont)	Find terms that are ancestors of specified Gene Ontology (GO) term
getdescendants (geneont)	Find terms that are descendants of specified Gene Ontology (GO) term
getmatrix (geneont)	Convert geneont object into relationship matrix
getrelatives (geneont)	Find terms that are relatives of specified Gene Ontology (GO) term

Property Summary

Properties of a geneont Object

Property	Description
default_namespace	Read-only string containing the namespace to which terms are assigned.
format_version	Read-only string containing the version of the encoding of the OBO flat format file.
date	Read-only string containing the date the OBO file was last updated.
Terms	Read-only column vector with handles to term objects of a geneont object. For properties of term objects, see Properties of Terms Objects on page 5-35.

Properties of Terms Objects

Property	Description
id	Numeric value that corresponds to the GO ID of the GO term. Tip You can use the num2goid function to convert id to a GO ID string.
name	String representing the name of the GO term.
ontology	String limited to 'molecular function', 'biological process', or 'cellular component'.
definition	String that defines the GO term.
synonym	Numeric array containing GO IDs of GO terms that are synonyms of this GO term.
is_a	Numeric array containing GO IDs of GO terms that have an “is_a” relationship with this GO term.
part_of	Numeric array containing GO IDs that of GO terms that have a “part_of” relationship with this GO term.
obsolete	Boolean value that indicates if the GO term is obsolete (1) or not obsolete (0).

See Also

Bioinformatics Toolbox functions: `geneont` (object constructor), `goannotread`, `num2goid`

Bioinformatics Toolbox methods of `geneont` object: `getancestors`, `getdescendants`, `getmatrix`, `getrelatives`

phytree object

Purpose Data structure containing phylogenetic tree

Description A phytree object is a data structure containing a phylogenetic tree. Phylogenetic trees are binary rooted trees, which means that each branch is the parent of two other branches, two leaves, or one branch and one leaf. A phytree object can be ultrametric or nonultrametric.

Method Summary Following are methods of a phytree object:

get (phytree)	Retrieve information about phylogenetic tree object
getbyname (phytree)	Branches and leaves from phytree object
getcanonical (phytree)	Calculate canonical form of phylogenetic tree
getmatrix (phytree)	Convert phytree object into relationship matrix
getnewickstr (phytree)	Create Newick-formatted string
pdist (phytree)	Calculate pairwise patristic distances in phytree object
plot (phytree)	Draw phylogenetic tree
prune (phytree)	Remove branch nodes from phylogenetic tree
reorder (phytree)	Reorder leaves of phylogenetic tree
reroot (phytree)	Change root of phylogenetic tree
select (phytree)	Select tree branches and leaves in phytree object
subtree (phytree)	Extract phylogenetic subtree

view (phytree)	View phylogenetic tree
weights (phytree)	Calculate weights for phylogenetic tree

Property Summary

Note You cannot modify these properties directly. You can access these properties using the `get` method.

Property	Description
NumLeaves	Number of leaves
NumBranches	Number of branches
NumNodes	Number of nodes (NumLeaves + NumBranches)
Pointers	Branch to leaf/branch connectivity list
Distances	Edge length for every leaf/branch
LeafNames	Names of the leaves
BranchNames	Names of the branches
NodeNames	Names of all the nodes

See Also

Bioinformatics Toolbox functions: `phytree` (object constructor), `phytreeread`, `phytreetool`, `phytreewrite`, `seqlinkage`, `seqneighjoin`, `seqpdist`

Bioinformatics Toolbox methods of `phytree` object: `get`, `getbyname`, `getcanonical`, `getmatrix`, `getnewickstr`, `pdist`, `plot`, `prune`, `reroot`, `select`, `subtree`, `view`, `weights`

phytree object

A

aa2int function
reference 2-2

aa2nt function
reference 2-6

aaccount function
reference 2-13

affygrma function
reference 2-19

affyinvarsetnorm function
reference 2-30

affyprobeaffinities function
reference 2-38

affyprobeseqread function
reference 2-45

affyread function
reference 2-50

affyrma function
reference 2-69

affysnpannotread function
reference 2-76

affysnpintensitysplit function
reference 2-80

affysnpquartets function
reference 2-85

agferead function
reference 2-88

allshortestpaths method
reference 4-2

aminolookup function
reference 2-90

atomiccomp function
reference 2-95

B

basecount function
reference 2-97

baselookup function
reference 2-101

biograph constructor
reference 2-106

biograph object
reference 5-2

blastformat function
reference 2-115

blastlocal function
reference 2-121

blastncbi function
reference 2-134

blastread function
reference 2-149

blastreadlocal function
reference 2-155

blosum function
reference 2-163

C

celintensityread function
reference 2-166

cghcbs function
reference 2-172 2-186

chromosomeplot function
reference 2-203

classperf function
reference 2-219

cleave function
reference 2-233

cleavelookup function
reference 2-239

clustergram function
reference 2-243

clustergram object
reference 5-13

codonbias function
reference 2-268

codoncount function
reference 2-275

colnames (DataMatrix) method
reference 4-5
conncomp method
reference 4-8
cpgisland function
reference 2-280
crossvalind function
reference 2-284
cytobandread function
reference 2-287

D

DataMatrix constructor
reference 2-290
DataMatrix object
constructing 2-290
methods 5-26
properties 5-24
reference 5-24
dayhoff function
reference 2-298
dimercount function
reference 2-299
disp (DataMatrix) method
reference 4-11
dmarrayfun (DataMatrix) method
reference 4-12
dmbsxfun (DataMatrix) method
reference 4-17
dna2rna function
reference 2-303

dnds function
reference 2-304
dndsml function
reference 2-312
dolayout method
reference 4-19
double (DataMatrix) method
reference 4-22

E

emblread function
reference 2-318
eq (DataMatrix) method
reference 4-24
evalrasmolscript function
reference 2-322
exprprofrange function
reference 2-324
exprprofvar function
reference 2-325

F

fastaread function
reference 2-326
fastawrite function
reference 2-329
featuresmap
reference 2-333
featuresparse
reference 2-343

functions

- aa2int 2-2
- aa2nt 2-6
- aacount 2-13
- affygcma 2-19
- affyinvvarsetnorm 2-30
- affyprobeaffinities 2-38
- affyprobeseqread 2-45
- affyread 2-50
- affyrma 2-69
- affysnpannotread 2-76
- affysnpintensitiesplit 2-80
- affysnpquartets 2-85
- agferead 2-88
- aminolookup 2-90
- atomiccomp 2-95
- basecount 2-97
- baselookup 2-101
- biograph constructor 2-106
- blastformat 2-115
- blastlocal 2-121
- blastncbi 2-134
- blastread 2-149
- blastreadlocal 2-155
- blosum 2-163
- celintensityread 2-166
- cghcbs 2-172 2-186
- chromosomeplot 2-203
- classperf 2-219
- cleave 2-233
- cleavelookup 2-239
- clustergram 2-243
- codonbias 2-268
- codoncount 2-275
- cpgisland 2-280
- crossvalind 2-284
- cytobandread 2-287
- DataMatrix constructor 2-290
- dayhoff 2-298
- dimercount 2-299
- dna2rna 2-303
- dnds 2-304
- dndsm1 2-312
- emblread 2-318
- evalrasmolscript 2-322
- exprprofrange 2-324
- exprprofrange 2-325

G

- galread function
 - reference 2-349
- gcrma function
 - reference 2-350
- gcrmabackadj function
 - reference 2-359
- ge (DataMatrix) method
 - reference 4-26
- genbankread function
 - reference 2-368
- geneentropyfilter function
 - reference 2-370
- genelowvalfilter function
 - reference 2-372
- geneont function
 - reference 2-375
- geneont object
 - reference 5-34
- generangefilter function
 - reference 2-378
- geneticcode function
 - reference 2-381
- genevarfilter function
 - reference 2-384
- genpeptread function
 - reference 2-386
- geoseriesread function
 - reference 2-389
- geosoftread function
 - reference 2-392
- get (biograph) method
 - reference 4-28
- get (clustergram) method
 - reference 4-34
- get (DataMatrix) method
 - reference 4-45
- get (phytree) method
 - reference 4-48
- getancestors method
 - biograph object 4-50
 - geneont object 4-53
- getblast function
 - reference 2-395
- getbyname method
 - reference 4-58
- getcanonical method
 - reference 4-60
- getdescendants method
 - biograph object 4-62
 - geneont object 4-65
- getedgesbynodeid method
 - reference 4-70
- getembl function
 - reference 2-403
- getgenbank function
 - reference 2-406
- getgenpept function
 - reference 2-411
- getgeodata function
 - reference 2-416
- gethmmalignment function
 - reference 2-418
- gethmmprof function
 - reference 2-422
- gethmmtree function
 - reference 2-428
- getmatrix (biograph) method
 - reference 4-72
- getmatrix (geneont) method
 - reference 4-73
- getmatrix (phytree) method
 - reference 4-74
- getnewickstr method
 - reference 4-75
- getnodesbyid method
 - reference 4-77
- getpdb function
 - reference 2-431

getrelatives method
 biograph object 4-78
 geneont object 4-79
goannotread function
 reference 2-438
gonnet function
 reference 2-442
gprread function
 reference 2-443
graphallshortestpaths function
 reference 2-445
graphconncomp function
 reference 2-452
graphisdag function
 reference 2-459
graphisomorphism function
 reference 2-465
graphisspantree function
 reference 2-472
graphmaxflow function
 reference 2-474
graphminspanntree function
 reference 2-482
graphpred2path function
 reference 2-488
graphshortestpath function
 reference 2-492
graphtopoorder function
 reference 2-504
graphtraverse function
 reference 2-508
gt (DataMatrix) method
 reference 4-85

H

hmmprofalign function
 reference 2-517
hmmprofestimate function
 reference 2-520

hmmprofgenerate function
 reference 2-523
hmmprofmerge function
 reference 2-525
hmmprofstruct function
 reference 2-527
horzcat (DataMatrix) method
 reference 4-87

I

ilmnbslookup function
 reference 2-539
ilmnbsread function
 reference 2-549
imageneread function
 reference 2-555
int2aa function
 reference 2-558
int2nt function
 reference 2-561
isdag method
 reference 4-89
isequal (DataMatrix) method
 reference 4-90
isequalwithequalnans (DataMatrix) method
 reference 4-92
isoelectric function
 reference 2-564
isomorphism method
 reference 4-94
isspantree method
 reference 4-96

J

jcampread function
 reference 2-567
joinseq function
 reference 2-570

K

knnclassify function
reference 2-571
knnimpute function
reference 2-579

L

ldivide (DataMatrix) method
reference 4-97
le (DataMatrix) method
reference 4-99
lt (DataMatrix) method
reference 4-101

M

maboxplot function
reference 2-583
mafdr function
reference 2-587
magetfield function
reference 2-595
maimage function
reference 2-596

mainvarsetnorm function
reference 2-598
mairplot function
reference 2-606
maloglog function
reference 2-618
malowess function
reference 2-620
manorm function
reference 2-622
mapcaplot function
reference 2-625
mattest function
reference 2-628
mavolcanoplot function
reference 2-637
max (DataMatrix) method
reference 4-103
maxflow method
reference 4-106
mean (DataMatrix) method
reference 4-110
median (DataMatrix) method
reference 4-112

methods

allshortestpaths 4-2
 colnames (DataMatrix) 4-5
 conncomp 4-8
 disp (DataMatrix) 4-11
 dmarrayfun (DataMatrix) 4-12
 dmbsxfun (DataMatrix) 4-17
 dolayout 4-19
 double (DataMatrix) 4-22
 eq (DataMatrix) 4-24
 ge (DataMatrix) 4-26
 get (biograph) 4-28
 get (clustergram) 4-34
 get (DataMatrix) 4-45
 get (phytree) 4-48
 getancestors (biograph) 4-50
 getancestors (geneont) 4-53
 getbyname 4-58
 getcanonical 4-60
 getdescendants (biograph) 4-62
 getdescendants (geneont) 4-65
 getedgesbynodeid 4-70
 getmatrix (biograph) 4-72
 getmatrix (geneont) 4-73
 getmatrix (phytree) 4-74
 getnewickstr 4-75
 getnodesbyid 4-77
 getrelatives (biograph) 4-78
 getrelatives (geneont) 4-79
 gt (DataMatrix) 4-85
 horzcat (DataMatrix) 4-87
 isdag 4-89
 isequal (DataMatrix) 4-90
 isequalwithequalnans (DataMatrix) 4-92
 isomorphism 4-94
 isspantree 4-96
 ldivide (DataMatrix) 4-97
 le (DataMatrix) 4-99
 lt (DataMatrix) 4-101
 max (DataMatrix) 4-103
 maxflow 4-106
 mean (DataMatrix) 4-110
 median (DataMatrix) 4-112
 min (DataMatrix) 4-114
 minspantree 4-117
 minus (DataMatrix) 4-120
 ndims (DataMatrix) 4-122

min (DataMatrix) method
 reference 4-114
 minspantree method
 reference 4-117
 minus (DataMatrix) method
 reference 4-120
 molviewer function
 reference 2-647
 molweight function
 reference 2-646
 msalign function
 reference 2-655
 msbackadj function
 reference 2-669
 msdotplot function
 reference 2-674
 msheatmap function
 reference 2-679
 mslowess function
 reference 2-689
 msnorm function
 reference 2-693
 mspalign function
 reference 2-697
 mspeaks function
 reference 2-706
 mspresample function
 reference 2-719
 msresample function
 reference 2-727
 mssgolay function
 reference 2-732
 msviewer function
 reference 2-734
 multialign function
 reference 2-737
 multialignread function
 reference 2-746
 multialignviewer function
 reference 2-748
 multialignwrite function
 reference 2-749
 mzcdf2peaks function
 reference 2-753
 mzcdfinfo function
 reference 2-757
 mzcdfinfo function

O

objects

- biograph 5-2
- clustergram 5-13
- DataMatrix 5-24
- geneont 5-34
- phytree 5-36

oligoprop function

- reference 2-808

optimalleaforder function

- reference 2-817

P

palindromes function

- reference 2-821

pam function

- reference 2-823

pbdistplot function

- reference 2-825

pbbread function

- reference 2-827

pbdsuperpose function

- reference 2-834

pbdtransform function

- reference 2-844

pdbwrite function

- reference 2-848

pdist method

- reference 4-127

pfamhmmread function

- reference 2-851

phytree constructor

- reference 2-856

phytree object

- reference 5-36

phytreeread function

- reference 2-860

phytreetool function

- reference 2-861

phytreewrite function

- reference 2-863

plot (clustergram) method

- reference 4-129

plot (DataMatrix) method

- reference 4-130

plot method

- reference 4-132

plus (DataMatrix) method

- reference 4-134

power (DataMatrix) method

- reference 4-136

probelibraryinfo function

- reference 2-865

probesetlink function

- reference 2-867

probesetlookup function

- reference 2-870

probesetplot function

- reference 2-872

probesetvalues function

- reference 2-877

profalign function

- reference 2-883

proteinplot function

- reference 2-886

proteinpropplot function

- reference 2-889

prune method

- reference 4-138

Q

quantilenorm function

- reference 2-895

R

ramachandran function

- reference 2-896

randfeatures function
reference 2-911

randseq function
reference 2-914

rankfeatures function
reference 2-917

rdivide (DataMatrix) method
reference 4-140

rebasecuts function
reference 2-922

redbluecmap function
reference 2-924

redgreencmap function
reference 2-927

reorder method
reference 4-142

reroot method
reference 4-146

restrict function
reference 2-930

revgeneticcode function
reference 2-933

rmabackadj function
reference 2-938

rmasummary function
reference 2-943

rna2dna function
reference 2-949

rnaconvert function
reference 2-950

rnafold function
reference 2-952

rnaplot function
reference 2-956

rownames (DataMatrix) method
reference 4-150

S

samplealign function
reference 2-972

scfread function
reference 2-991

select method
reference 4-153

seq2regex function
reference 2-994

seqcomplement function
reference 2-998

seqconsensus function
reference 2-999

seqdisp function
reference 2-1001

seqdotplot function
reference 2-1003

seqinsertgaps function
reference 2-1005

seqlinkage function
reference 2-1008

seqlogo function
reference 2-1010

seqmatch function
reference 2-1017

seqneighjoin function
reference 2-1018

seqpdist function
reference 2-1021

seqprofile function
reference 2-1032

seqrcomplement function
reference 2-1035

seqreverse function
reference 2-1036

seqshoworfs function
reference 2-1037

seqshowwords function
reference 2-1043

- seqtool function
 - reference 2-1046
- seqwordcount function
 - reference 2-1048
- set (biograph) method
 - reference 4-156
- set (clustergram) method
 - reference 4-163
- set (DataMatrix) method
 - reference 4-173
- shortestpath method
 - reference 4-177
- showalignment function
 - reference 2-1050
- showhmmprof function
 - reference 2-1056
- single (DataMatrix) method
 - reference 4-182
- sortcols (DataMatrix) method
 - reference 4-184
- sortrows (DataMatrix) method
 - reference 4-186
- sptread function
 - reference 2-1058
- std (DataMatrix) method
 - reference 4-188
- subtree method
 - reference 4-190
- sum (DataMatrix) method
 - reference 4-191
- support vector machines
 - svmclassify function 2-1060
 - svmsmoset function 2-1067
 - svmtrain function 2-1071
- svmclassify function
 - reference 2-1060
- svmsmoset function
 - reference 2-1067

- svmtrain function
 - reference 2-1071
- swalign function
 - reference 2-1087

T

- times (DataMatrix) method
 - reference 4-193
- topoorder method
 - reference 4-195
- traceplot function
 - reference 2-1094
- traverse method
 - reference 4-196

V

- var (DataMatrix) method
 - reference 4-199
- vertcat (DataMatrix) method
 - reference 4-202
- view (biograph) method
 - reference 4-204
- view (clustergram) method
 - reference 4-206
- view (phytree) method
 - reference 4-207

W

- weights method
 - reference 4-208

Z

- zonebackadj function
 - reference 2-1095