Bioinformatics Toolbox

For Use with MATLAB®

- Computation
- Visualization
- Programming

User's Guide

Version 1

The MathWorks

How to Contact The MathWorks:

	`www.mathworks.com`	Web
	`comp.soft-sys.matlab`	Newsgroup
@	`support@mathworks.com`	Technical Support
	`suggest@mathworks.com`	Product enhancement suggestions
	`bugs@mathworks.com`	Bug reports
	`doc@mathworks.com`	Documentation error reports
	`service@mathworks.com`	Order status, license renewals, passcodes
	`info@mathworks.com`	Sales, pricing, and general information
☎	508-647-7000	Phone
	508-647-7001	Fax
✉	The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098	Mail

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

Bioinformatics Toolbox User's Guide

Printing History:

September 2003	Online only	New for Version 1.0 (Release 13SP1+)
June 2004	Online only	Updated for Version 1.1 (Release 14)

Contents

Getting Started

This chapter is an overview of the functions and features in the Bioinformatics Toolbox. An introduction to these features will help you to develop a conceptual model for working with the toolbox and your biological data.

"What Is the Bioinformatics Toolbox?" (p. 1-2)	Description of this toolbox and the intended user
"Installation" (p. 1-5)	Required software and additional software for developing advanced algorithms
"Features and Functions" (p. 1-7)	Functions grouped into categories that support bioinformatic tasks

What Is the Bioinformatics Toolbox?

The Bioinformatics Toolbox extends MATLAB® to provide an integrated software environment for genome and proteome analysis. Together, MATLAB and the Bioinformatics Toolbox give scientists and engineers a set of computational tools to solve problems and build applications in drug discovery, genetic engineering, and biological research.

You can use the basic bioinformatic functions provided with this toolbox to create more complex algorithms and applications. These robust and well tested functions are the functions that you would otherwise have to create yourself.

- Connecting to Web accessible databases
- Reading and converting between multiple data formats
- Determining statistical characteristics of data
- Manipulating and aligning sequences
- Modeling patterns in biological sequences using Hidden Markov Model (HMM) profiles
- Reading, normalizing, and visualizing microarray data
- Creating and manipulating phylogenetic tree data
- Interfacing with other bioinformatic software (BioPearl and BioJava)

The field of bioinformatics is rapidly growing and will become increasingly important as biology becomes a more analytical science. The Bioinformatics Toolbox provides an open environment that you can customize for development and deployment of the analytical tools you and scientists will need.

Prototype and develop algorithms — Prototype new ideas in an open and extendable environment. Develop algorithms using efficient string processing and statistical functions, view the source code for existing functions, and use the code as a template for improving or creating your own functions. See “Prototype and Development Environment” on page 1-12.

Visualize data — Visualize sequence alignments, gene expression data, phylogenetic trees, and protein structure analyses. See "Data Visualization" on page 1-12.

Share and deploy applications — Use an interactive GUI builder to develop a custom graphical front end for your data analysis programs. Create stand-alone applications that run separate from MATLAB. See "Algorithm Sharing and Application Deployment" on page 1-12.

Expected User

The Bioinformatics Toolbox is for computational biologists and research scientists who need to develop new or implement published algorithms, visualize results, and create stand-alone applications.

- **Industry/Professional** — Increasingly, drug discovery methods are being supported by engineering practice. This toolbox supports tool builders who want to create applications for the biotechnology and pharmaceutical industry.
- **Education/Student** — This toolbox is well suited for learning and teaching genome and proteome analysis techniques. Educators and students can concentrate on bioinformatic algorithms instead of programming basic functions such as reading and writing to files.

While the toolbox includes many bioinformatics functions, it is not intended to be a complete set of tools for scientists to analyze their biological data. However, MATLAB is the ideal environment for you to rapidly design and prototype the tools you will need.

Installation

You don't need to do anything special when installing the Bioinformatics Toolbox. Install the toolbox from a CD or Web release using The MathWorks installer.

- "Required Software" on page 1-5 — List of MathWorks products you need to purchase with the Bioinformatics Toolbox
- "Additional Software" on page 1-5 — List of toolboxes from The MathWorks for advanced algorithm development

Required Software

The Bioinformatics Toolbox requires the following products from the MathWorks to be installed on your computer:

MATLAB	Provides a command-line interface and integrated software environment for the Bioinformatics Toolbox.Version 1.1 of the Bioinformatics Toolbox requires MATLAB Version 7 on the Release 14 CD.
Statistics Toolbox	Provides basic statistics and probability functions that the functions in the Bioinformatics Toolbox use.Version 1.1 of the Bioinformatics Toolbox requires the Statistics Toolbox Version 5 on the Release 14 CD or downloaded from the Web.

Additional Software

MATLAB and the Bioinformatics Toolbox provide an open and extensible software environment. In this environment you can interactively explore ideas, prototype new algorithms, and develop complete solutions to problems in bioinformatics. The MATLAB language facilitates the use of computation, visualization, prototyping, and deployment.

Using the Bioinformatics Toolbox in combination with other MATLAB toolboxes, will allow your to solve multidisciplinary problems.

Signal Processing Toolbox	Process signal data from bioanalytical instrumentation. Examples include acquisition of fluorescence data for DNA sequence analyzers, fluorescence data for microarray scanners, and mass spectrometric data from protein analyses.
Image Processing Toolbox	Create complex and custom image processing algorithms for data from microarray scanners.
Optimization Toolbox	Use nonlinear optimization for predicting the secondary structure of proteins and the structure of other biological macromolecules.
Neural Network Toolbox	Use neural networks to solve problems where algorithms are not available. For example, you can train neural networks for pattern recognition using large sets of sequence data.
Database Toolbox	Create your own in-house databases for sequence data with custom annotations.
MATLAB Compiler	Create stand-alone applications from MATLAB GUI applications, and create dynamic link libraries from MATLAB functions for use with any programming environment.
MATLAB® Builder for COM	Create COM objects to use with any COM-based programming environment.
MATLAB® Builder for Excel	Create Excel add-in functions from MATLAB functions to use with Excel spreadsheets.

Features and Functions

The Bioinformatics Toolbox includes many functions to help you with genome and proteome analysis. Most functions are implemented in M-Code (the MATLAB programming language) with the source available for you to view. This open environment lets you explore and customize the existing toolbox algorithms or develop your own.

- “Data Formats and Databases” on page 1-7 — Access online databases, copy data into the MATLAB workspace, and read and write to files with standard bioinformatic formats.
- “Sequence Alignments” on page 1-9 — Compare nucleotide or amino acid sequences using pairwise and multiple sequence alignment functions.
- “Sequence Utilities and Statistics ” on page 1-9 — Manipulate sequences and determine physical, chemical, and biological characteristics.
- “Microarray Analysis” on page 1-10 — Read, filter, normalize, and visualize microarray data.
- “Protein Structure Analysis” on page 1-10 — Determine protein characteristics and simulate enzyme cleavage reactions.
- “Phylogenetic Analysis” on page 1-11 — Explore phylogenetic data with functions and a GUI to draw phylograms (trees)
- “Prototype and Development Environment” on page 1-12 — Create new algorithms, try new ideas, and compare alternatives.
- “Data Visualization” on page 1-12 — Visually compare pairwise and multiply aligned sequences, gene expression data from microarrays, and plot nucleic acid and protein characteristics.
- “Algorithm Sharing and Application Deployment” on page 1-12 — Create GUIs and stand-alone applications.

Data Formats and Databases

The Bioinformatics Toolbox supports access to many of the databases on the Web and other online sources. It also reads many common genome file formats so that you do not have to write and maintain your own file readers.

Web-based databases — You can directly access public databases on the Web and copy sequence and gene expression information into MATLAB.

Currently supported sequence databases are GenBank (`getgenbank`), GenPept (`getgenpept`), European Molecular Biology Laboratory EMBL (`getembl`), Protein Sequence Database PIR-PSD (`getpir`), and Protein Data Bank PDB (`getpdb`). You can also access data from the NCBI Gene Expression Omnibus (GEO) web site by using a single function (`getgeodata`).

Get multiple aligned sequences (`gethmmalignment`), hidden Markov model profiles (`gethmmprof`), and phylogenetic tree data (gethmmtree) from the PFAM database.

Raw data — Read and visualize data generated from gene sequencing instruments in MATLAB (`scfread`, `joinseq`, `traceplot`).

Reading data formats — The toolbox provides a number of functions for reading data from common file formats.

- Sequence data: GenBank (`genbankread`), GenPept (`genpeptread`), EMBL (`emblread`), PIR-PSD (`pirread`), PDB (`pdbread`), and FASTA (

  `fastaread`
- Multiply aligned sequences: ClustalW and GCG formats (`multialignread`).
- Gene expression data from microarrays: Gene Expression Omnibus (GEO) data (`geosoftread`), GenePix data (`galread`, `gprread`), and SPOT data (`sptread`), Affymetrix data (`affyread`)

  Note: The function `affyread` only works on PC supported platforms.
- Hidden Markov Model profiles: PFAM-HMM file (`pfamhmmread`).

Writing data formats — The functions for getting data from the Web include the option to save the data to a file. However, there is a function to write data to a file using the FASTA format (`fastawrite`).

MATLAB has built-in support for other industry-standard file formats including Microsoft Excel and comma-separated value (CSV) files. Additional functions perform ASCII and low-level binary I/O, allowing you to develop custom functions for working with any data format.

Sequence Alignments

You can select from a list of analysis methods to perform pairwise or multiple sequence alignment.

Pairwise sequence alignment — The toolbox provides efficient MATLAB implementations of standard algorithms such as the Needleman-Wunsch (nwalign) and Smith-Waterman (swalign) algorithms for pairwise sequence alignment. The toolbox also includes standard scoring matrices such as the PAM and BLOSUM families of matrices (blosum, dayhoff, gonnet, nuc44, pam).

Sequence profile alignment — The toolbox provides efficient MATLAB implementations for profile hidden Markov model algorithms (gethmmprof, gethmmalignment, pfamhmmread, hmmprofalign, hmmprofestimate, hmmprofgenerate, hmmprofmerge, hmmprofstruct, showhmmprof).

Visualizing sequence alignments — Once you have analyzed your data, there are several tools for visualizing your sequence alignments (seqdotplot, showalignment, seqshowwords, seqshoworfs).

Biological codes — Look up the letters or numerical equivalents for commonly used biological codes (aminolookup, geneticcode, revgeneticcode).

Sequence Utilities and Statistics

You can manipulate and analyze your sequence to gain a deeper understanding of your data.

Sequence manipulation — The toolbox provides routines for common operations such as converting DNA or RNA sequences to amino acid sequences that are basic to working with nucleic acid or protein sequences (aa2int, aa2nt, dna2rna, rna2dna, int2aa, int2nt, nt2aa, nt2int, seqcomplement, seqrcomplement, seqreverse).

You can manipulate your sequence by performing an in-silico digestion with restriction endonucleases (restrict) and proteases (cleave).

Sequence statistics — You can determine various statistics about a sequences (aacount, basecount, codoncount, dimercount, nmercount, ntdensity), search for specific patterns within a sequence (seqshowwords, seqwordcount), or search for open reading frames (seqshoworfs). In addition, you can create random sequences for test cases (randseq).

Additional functions in MATLAB efficiently handle string operations with regular expressions (regexp, seq2regexp) to look for specific patterns in a sequence, and look for possible cleavage sites in a DNA/RNA sequence by searching for palindromes (palindromes).

Microarray Analysis

MATLAB is widely used for microarray data analysis. However, the standard normalization and visualization tools that scientists use can be difficult to implement. The Bioinformatics Toolbox includes these standard functions.

Microarray normalization — The toolbox provides a number of methods for normalizing microarray data, such as lowess normalization (malowess), global mean normalization (mameannorm), and median absolute deviation (MAD) normalization (mamadnorm). You can use filtering functions to clean raw data before analysis (geneentropyfilter, genelowvalfilter, generangefilter, genevarfilter), and calculate the range and variance of values (exprprofrange, exprprofvar).

Microarray visualization — The toolbox contains routines for visualizing microarray data. These routines include spacial plots of microarray data (maimage, redgreencmap), box plots (maboxplot), loglog plot (maloglog), and intensity-ratio plots (mairplot). You can also view clustered expression profiles (clustergram, redgreencmap)

The toolbox accesses statistical routines to perform cluster analysis and to visualize the results.

MATLAB visualization tools let you view your data through statistical visualizations such as dendrograms, classification, and regression trees.

Protein Structure Analysis

You can use a collection of protein analysis methods to extract information from your data. The toolbox provides functions to calculate various properties of a protein sequence such as the atomic composition (atomiccomp) and the molecular weight (molweight). You can cleave a protein with an enzyme (`cleave`) and create distance and Ramachandran plots for PDB data (`pdbdistplot`, `ramachandran`). The toolbox contains a graphical user interface for protein analysis (`proteinplot`). After analyzing the data, you can create revealing visualizations of your results.

Phylogenetic Analysis

Functions for phylogenetic tree building and analysis.

- phytreeread — Read a Newick formatted tree file into the MATLAB workspace and return a `phytree` object with data from the file. Data in the file uses the Newick (New Hampshire) format for describing trees.
- phytreewrite — Copy the contents of a `phytree` object from the MATLAB workspace to a file.
- phytreetool — 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.
- seqpdist — Calculate the pairwise distance between biological sequences.
- seqlinkage — Construct a phylogenetic tree from pairwise distances.

MALTLAB object and methods for manipulating phylogenetic tree data.

- phytree — Function to create a phytree object.
- phytree/get — Get property values from a phytree object
- phytree/getbyname — Get node names from a phytree object.
- phytree/pdist — Calculate the patristic distances between pairs of leaf nodes.
- phytree/plot — Draw a phylogenetic tree object in a MATLAB figure window as a phylogram, cladogram, or radial tree.
- phytree/prune — Remove nodes from a phylogenetic tree.
- phytree/select — Select branches and leaves from a phylogenetic tree using a specified criteria.
- phytree/view — Opens a phylogenetic tree in a phytreetool window.

Prototype and Development Environment

MATLAB is a prototyping and development environment where you can create algorithms and easily compare alternatives.

- **Integrated environment** — Explore biological data in an environment that integrates programming and visualization. Create reports and plots with the built-in functions for math, graphics, and statistics.
- **Open environment** — Access the source code for the Bioinformatics Toolbox functions, The toolbox includes many of the basic bioinformatics functions you will need to use, and it includes prototypes for some of the more advanced functions. Modify these functions to create your own custom solutions.
- **Interactive programming language** — Test your ideas by typing functions that are interpreted interactively with a language whose basic data element is an array. The arrays do not require dimensioning and allow you to solve many technical computing problems,

  Using matrixes for sequences or groups of sequences allows you to work efficiently with sequences and not worry about writing loops or other programming controls.
- **Programming tools** — Use a visual debugger for algorithm development and refinement and an algorithm performance profiler to accelerate development

Data Visualization

In addition, MATLAB 2D and volume visualization features let you create custom graphical representations of multidimensional data sets. You can also create montages and overlays, and export finished graphics to a PostScript image file or copy directly into Microsoft PowerPoint.

Algorithm Sharing and Application Deployment

The open MATLAB environment lets you share your analysis solutions with other MATLAB users, and it includes tools to create custom software applications. With the addition of the MATLAB Compiler, you can create stand-alone applications independent from MATLAB, and with the addition of the MATLAB COM Builder, you can create GUIs and stand-alone applications within other programming environments.

- **Share algorithms with other MATLAB users** — You can share data analysis algorithms created in the MATLAB language across all MATLAB supported platforms by giving M-files to other MATLAB users, Also, you can create GUIs within MATLAB using the Graphical User Interface Development Environment (GUIDE).
- **Deploy MATLAB GUIs** — Create a GUI within MATLAB using GUIDE, and then use the MATLAB Compiler to create a stand-alone GUI application that runs separate from MATLAB.
- **Create dynamic link libraries (DLL)** — Use the MATLAB compiler to create dynamic link libraries (DLLs) for your functions, and then link these libraries to other programming environments such as C and C++.
- **Create COM objects** — Use the MATLAB COM Builder to create COM objects, and then use a COM compatible programming environment (Visual Basic) to create a stand-alone application.
- **Create Excel add-ins** — Use the MATLAB Excel Builder to create Excel add-in functions, and then use the add-in functions with Excel spreadsheets.

Sequence Analysis

Sequence analysis is the process you use to find information about a nucleotide or amino acid sequence using computational methods. Common tasks in sequence analysis are identifying genes, determining the similarity of two genes, determining the protein coded by a gene, and determining the function of a gene by finding a similar gene in another organism with a know function.

"Example: Sequence Statistics" (p. 2-2)	Starting with a DNA sequence, calculate statistics for the nucleotide content.
"Example: Sequence Alignment" (p. 2-17)	Starting with a DNA sequence for a human gene, locate and verify a corresponding gene in a model organism.

Example: Sequence Statistics

After sequencing a piece of DNA, one of the first tasks is to investigate the nucleotide content in the sequence. Starting with a DNA sequence, this example uses sequence statistics functions to determine mono-, di-, and trinucleotide content, and to locate open reading frames.

- “Determining Nucleotide Content” on page 2-2 — Use the MATLAB Help browser to search the Web for information.
- “Getting Sequence Information into MATLAB” on page 2-4 — Find a nucleotide sequence in a public database and read the sequence information into MATLAB.
- “Determining Nucleotide Composition” on page 2-5 — Determine the monomers and dimers, and then visualize data in graphs and bar plots.
- “Determining Codon Composition” on page 2-8 — Look at codons for the six reading frames.
- “Open Reading Frames” on page 2-11 — Locate the open reading frames using a specific genetic code.
- “Amino Acid Conversion and Composition” on page 2-14 — Extract the protein-coding sequence from a gene sequence and convert it to the amino acid sequence for the protein.

Determining Nucleotide Content

In this example you are interested in studying the human mitochondrial genome. While many genes that code for mitochondrial proteins are found in the cell nucleus, the mitochondrial has genes that code for proteins used to produce energy.

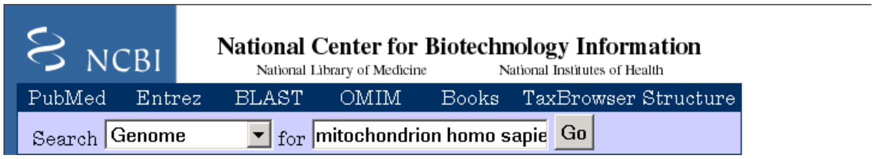
First research information about the human mitochondria and find the nucleotide sequence for the genome. Next, look at the nucleotide content for the entire sequence. And finally, determine open reading frames and extract specific gene sequences.

1 Use the MATLAB Help browser to explore the Web. In the **MATLAB Command Window**, type

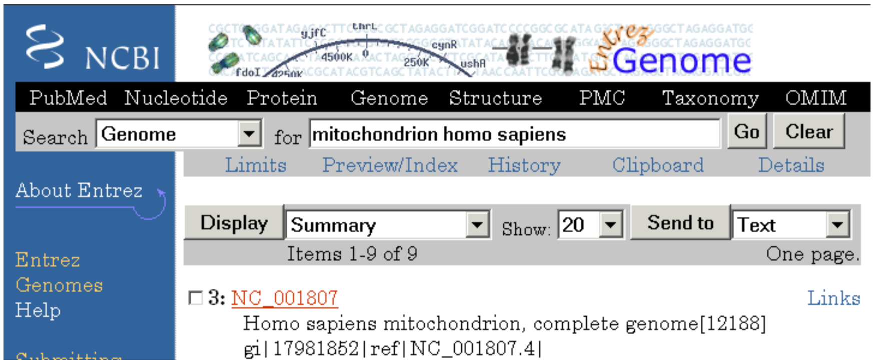
```
web('http://www.ncbi.nlm.nih.gov/')
```

A separate browser window opens with the home page for the NCBI Web site.

2 Search the NCBI Web site for information. For example, to search for the human mitochondrion genome, from the **Search** list, select `Genome`, and in the **for** box, enter `mitochondrion homo sapiens`.

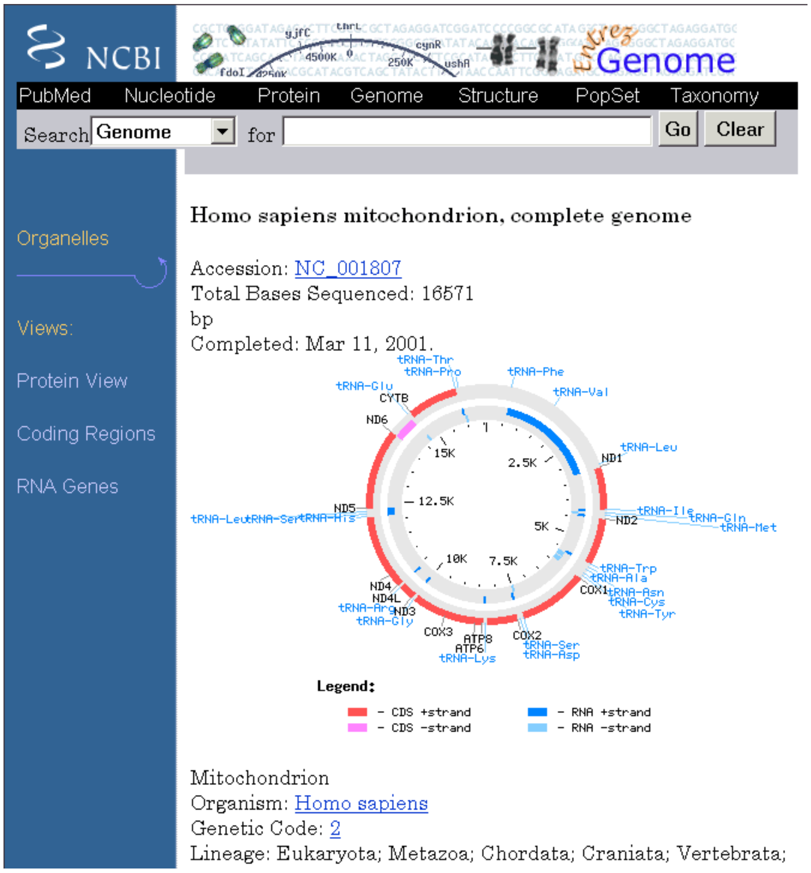

NCBI
National Center for Biotechnology Information
National Library of Medicine
National Institutes of Health
PubMed Entrez BLAST OMIM Books TaxBrowser Structure
Search
Genome
for
mitochondrion homo sapie
Go

The NCBI Web search returns a list of links to relevant pages.


NCBI
Entrez Genome
PubMed Nucleotide Protein Genome Structure PMC Taxonomy OMIM
Search
Genome
for
mitochondrion homo sapiens
Go
Clear
Limits Preview/Index History Clipboard Details
About Entrez
Entrez Genomes Help
Display
Summary
Show: 20
Send to
Text
Items 1-9 of 9
One page.
3: NC_001807
Links
Homo sapiens mitochondrion, complete genome[12188]
gi|17981852|ref|NC_001807.4|

3 Select a result page. For example, click the link labeled **NC_001807**.

The MATLAB Help browser displays the NCBI page for the human mitochondrial genome.



Homo sapiens mitochondrion, complete genome

Accession: NC_001807
Total Bases Sequenced: 16571
bp
Completed: Mar 11, 2001.

Mitochondrion
Organism: Homo sapiens
Genetic Code: 2
Lineage: Eukaryota; Metazoa; Chordata; Craniata; Vertebrata;

Getting Sequence Information into MATLAB

Many public data bases for nucleotide sequences are accessible from the Web. The MATLAB command window provides an integrated environment for bringing sequence information into MATLAB.

The consensus sequence for the human mitochondrial genome has the GenBank accession number `NC_001807`. Since the whole GenBank entry is quite large and you might only be interested in the sequence, you can get just the sequence information.

1 Get sequence information from a Web database.For example, to get sequence information for the human mitochondrial genome, in the **MATLAB Command Window**, type

```
mitochondria = getgenbank('NC_001807','SequenceOnly',true);
```

MATLAB gets the nucleotide sequence from the GenBank database and creates a character array.

```
mitochondria =
gatcacaggtctatcaccctattaaccactcacgggagctctccatgcat
ttggtattttcgtctgggggtgtgcacgcgatagcattgcgagacgctg
gagccggagcaccctatgtcgcagtatctgtctttgattcctgcctcatt
ctattatttatcgcacctacgttcaatattacaggcgaacatacctacta
aagt . . .
```

2 If you don't have a Web connection, you can load the data from a MAT-file included with the Bioinformatics Toolbox, using the command

```
load mitochondria
```

MATLAB loads the sequence `mitochondria` into the MATLAB workspace.

3 Get information about the sequence. Type

```
whos mitochondria
```

MATLAB displays information about the size of the sequence.

```
Name                  Size                    Bytes  Class
 mitochondria          1x16571                  33142  char array

Grand total is 16571 elements using 33142 bytes
```

Determining Nucleotide Composition

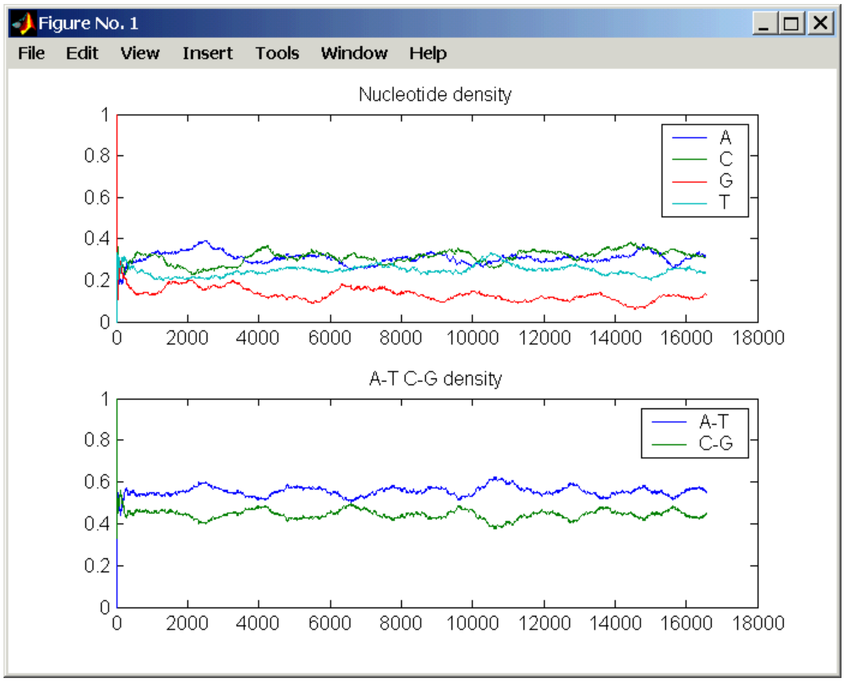
Sections of a DNA sequence with a high percent of A+T nucleotides usually indicates intergenic parts of the sequence, while low A+T and higher G+C nucleotide percentages indicate possible genes. Many times high CG dinucleotide content is located before a gene.

After you read a sequence into MATLAB, you can use the sequence statistics functions to determine if your sequence has the characteristics of a protein-coding region. This procedure uses the human mitochondrial genome as an example. See “Getting Sequence Information into MATLAB” on page 2-4.

1 Plot monomer densities and combined monomer densities in a graph. In the **MATLAB Command** window, type

```
ntdensity(mitochondria)
```

This graph shows that the genome is A+T rich.



2 Count the nucleotides using the function `basecount.basecount(mitochondria)`

A list of nucleotide counts is shown for the 5’-3’ strand.ans =

```
A: 5113
C: 5192
G: 2180
T: 4086
```

3 Count the nucleotides in the reverse complement of a sequence using the function `seqrcomplement`.

```
basecount(seqrcomplement(mitochondria))
```

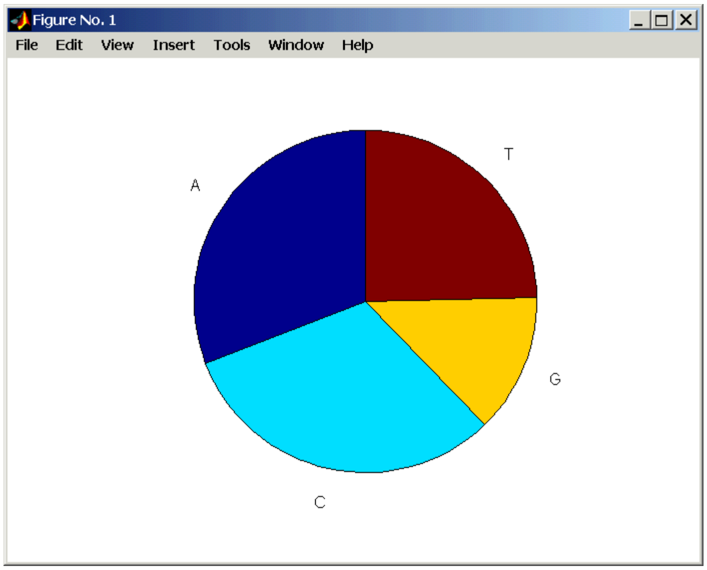
As expected, the nucleotide counts on the reverse complement strand are complementary to the 5’-3’ strand.

```
ans =
    A: 4086
    C: 2180
    G: 5192
    T: 5113
```

4 Use the function `basecount` with the `chart` option to visualize the nucleotide distribution.

```
basecount(mitochondria,'chart','pie');
```

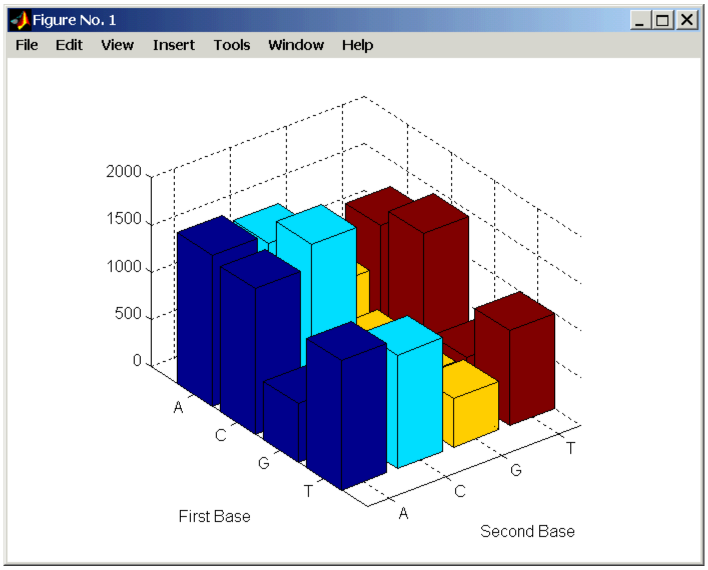
MATLAB draws a pie chart in a figure window.



5 Count the dimers in a sequence and display the information in a bar chart.

```
dimercount(mitochondria,'chart','bar')
```

MATLAB lists the dimer counts and draws a bar chart.



Determining Codon Composition

Trinucleotides (codon) code for an amino acid, and there are 64 possible codons in a nucleotide sequence. Knowing the percent of codons in your sequence can be helpful when you are comparing with tables for expected codon usage.

After you read a sequence into MATLAB, you can analyze the sequence for codon composition. This procedure uses the human mitochondria genome as an example. See “Getting Sequence Information into MATLAB” on page 2-4.

1 Count codons in a nucleotide sequence. In the **MATLAB Command Window**, type

```
codoncount(mitochondria)
```

MATLAB displays the codon counts for the first reading frame.

```
AAA-172  AAC-157  AAG-67  AAT-123
ACA-153  ACC-163  ACG-42  ACT-130
AGA-58   AGC-90   AGG-50  AGT-43
ATA-132  ATC-103  ATG-57  ATT-96
CAA-166  CAC-167  CAG-68  CAT-135
CCA-146  CCC-215  CCG-50  CCT-182
CGA-33   CGC-60   CGG-18  CGT-20
CTA-187  CTC-126  CTG-52  CTT-98
GAA-68   GAC-62   GAG-47  GAT-39
GCA-67   GCC-87   GCG-23  GCT-61
GGA-53   GGC-61   GGG-23  GGT-25
GTA-61   GTC-49   GTG-26  GTT-36
TAA-136  TAC-127  TAG-82  TAT-107
TCA-143  TCC-126  TCG-37  TCT-103
TGA-64   TGC-35   TGG-27  TGT-25
TTA-115  TTC-113  TTG-37  TTT-99
```

2 Count the codons in all six reading frames and plot the results in a heat map.

```
for frame = 1:3
    figure('color',[1 1 1])
    subplot(2,1,1);
    codoncount(mitochondria,'frame',frame,'figure',true);
    title(sprintf('Codons for frame %d',frame));
    subplot(2,1,2);
    codoncount(mitochondria,'reverse',true,
                            'frame',frame,
                            'figure',true);
    title(sprintf('Codons for reverse frame %d',frame));
end
```

MATLAB draws heat maps to visualize all 64 codons in the six reading frames.

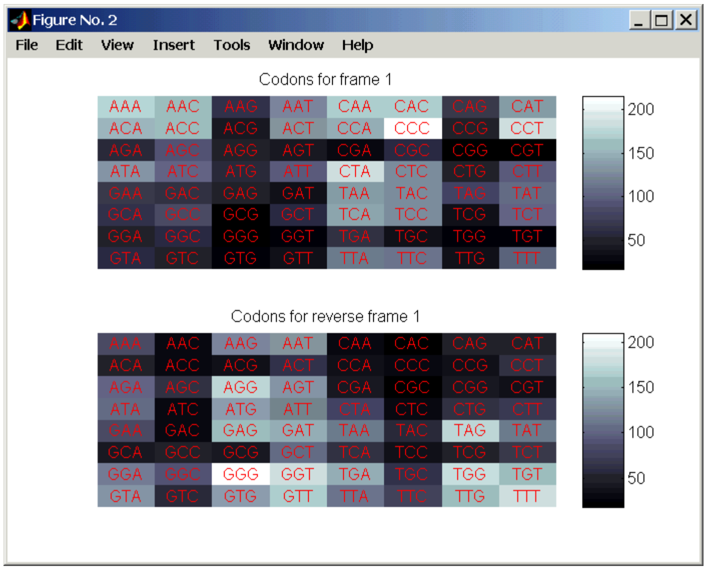


Figure No. 2
File Edit View Insert Tools Window Help
Codons for frame 1
AAA AAC AAG AAT CAA CAC CAG CAT
ACA ACC ACG ACT CCA CCC CCG CCT
AGA AGC AGG AGT CGA CGC CGG CGT
ATA ATC ATG ATT CTA CTC CTG CTT
GAA GAC GAG GAT TAA TAC TAG TAT
GCA GCC GCG GCT TCA TCC TCG TCT
GGA GGC GGG GGT TGA TGC TGG TGT
GTA GTC GTG GTT TTA TTC TTG TTT
200
150
100
50
Codons for reverse frame 1
AAA AAC AAG AAT CAA CAC CAG CAT
ACA ACC ACG ACT CCA CCC CCG CCT
AGA AGC AGG AGT CGA CGC CGG CGT
ATA ATC ATG ATT CTA CTC CTG CTT
GAA GAC GAG GAT TAA TAC TAG TAT
GCA GCC GCG GCT TCA TCC TCG TCT
GGA GGC GGG GGT TGA TGC TGG TGT
GTA GTC GTG GTT TTA TTC TTG TTT
200
150
100
50

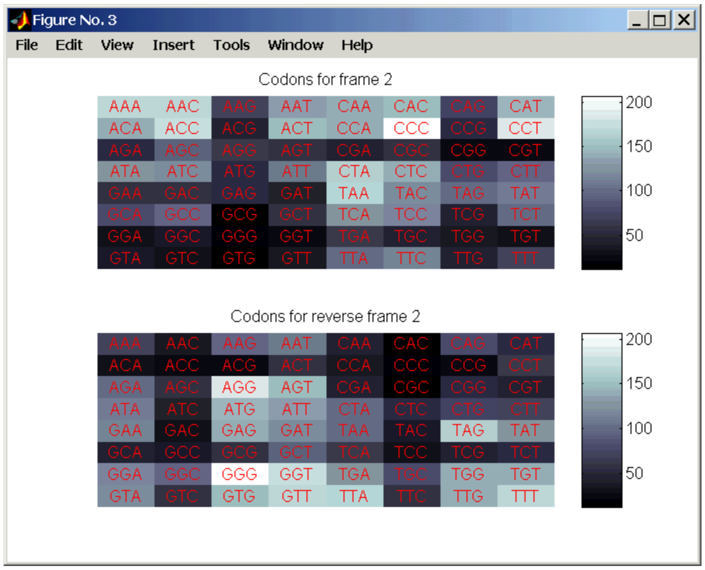
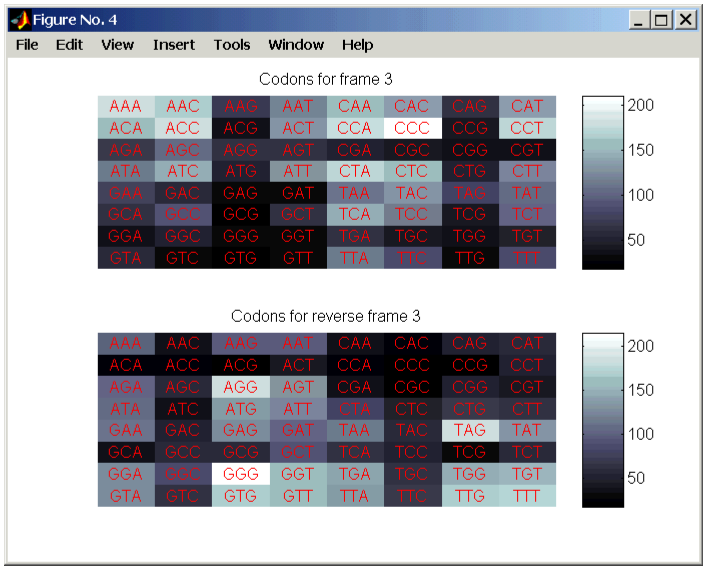


Figure No. 3
File Edit View Insert Tools Window Help
Codons for frame 2
AAA AAC AAG AAT CAA CAC CAG CAT
ACA ACC ACG ACT CCA CCC CCG CCT
AGA AGC AGG AGT CGA CGC CGG CGT
ATA ATC ATG ATT CTA CTC CTG CTT
GAA GAC GAG GAT TAA TAC TAG TAT
GCA GCC GCG GCT TCA TCC TCG TCT
GGA GGC GGG GGT TGA TGC TGG TGT
GTA GTC GTG GTT TTA TTC TTG TTT
200
150
100
50
Codons for reverse frame 2
AAA AAC AAG AAT CAA CAC CAG CAT
ACA ACC ACG ACT CCA CCC CCG CCT
AGA AGC AGG AGT CGA CGC CGG CGT
ATA ATC ATG ATT CTA CTC CTG CTT
GAA GAC GAG GAT TAA TAC TAG TAT
GCA GCC GCG GCT TCA TCC TCG TCT
GGA GGC GGG GGT TGA TGC TGG TGT
GTA GTC GTG GTT TTA TTC TTG TTT
200
150
100
50



Open Reading Frames

Determining the protein-coding sequence for a eukaryotic gene can be a difficult task because introns (noncoding sections) are mixed with exons. However, prokaryotic genes generally do not have introns and mRNA sequences have the introns removed. Identifying the start and stop codons for translation determines the protein-coding section or open reading frame (ORF) in a sequence. Once you know the ORF for a gene or mRNA, you can translate a nucleotide sequence to its corresponding amino acid sequence.

After you read a sequence into MATLAB, you can analyze the sequence for open reading frames. This procedure uses the human mitochondria genome as an example. See “Getting Sequence Information into MATLAB” on page 2-4.

1 Display open reading frames (ORFs) in a nucleotide sequence. In the **MATLAB Command** window, type

```
showorfs(mitochondria);
```

If you compare this output to the genes shown on the NCBI page for NC_001807, there are fewer genes than expected. This is because vertebrate mitochondria use a genetic code slightly different from the standard genetic code. For a table of genetic codes, see Genetic Code on page 6-4.

2 Display ORFs using the `Vertebrate Mitochondrial` code.

```
orfs= seqshoworfs(mitochondria,
                  'GeneticCode','Vertebrate Mitochondrial',
                  'alternativestart',true);
```

Notice that there are now two large ORFs on the first reading frame. One starts at position 4471 and the other starts at 5905. These correspond to the genes ND2 (NADH dehydrogenase subunit 2 [Homo sapiens]) and COX1 (cytochrome c oxidase subunit I) genes.

3 Find the corresponding stop codon. The start and stop positions for ORFs have the same indices as the start positions in the fields `Start` and `Stop`.

```
ND2Start = 4471;
StartIndex = find(orfs(1).Start == ND2Start)
ND2Stop = orfs(1).Stop(StartIndex)
```

MATLAB displays the stop position.

```
ND2Stop =
        5512
```

4 Using the sequence indices for the start and stop of the gene, extract the subsequence from the sequence.

```
ND2Seq = mitochondria(ND2Start:ND2Stop);
codoncount (ND2Seq)
```

The subsequence (protein-coding region) is stored in `ND2Seq` and displayed on the screen.

```
attaatcccctggcccaacccgtcatctactctaccatctttgcaggcac
actcatcacagcgctaagctcgcactgatttttacctgagtaggcctag
aaataaacatgctagcttttattccagttctaaccaaaaaaataaaccct
cgttccacagaagctgccatcaagtatttcctcacgcaagcaaccgcatc
cataatccttc . . .
```

5 Determine the codon distribution.

```
codoncount (ND2Seq)
```

The codon count shows a high amount of ACC, ATA, CTA, and ATC.

```
AAA-10  AAC-14  AAG-2   AAT-6
ACA-11  ACC-24  ACG-3   ACT-5
AGA-0   AGC-4   AGG-0   AGT-1
ATA-22  ATC-24  ATG-2   ATT-8
CAA-8   CAC-3   CAG-2   CAT-1
CCA-4   CCC-12  CCG-2   CCT-5
CGA-0   CGC-3   CGG-0   CGT-1
CTA-26  CTC-18  CTG-4   CTT-7
GAA-5   GAC-0   GAG-1   GAT-0
GCA-8   GCC-7   GCG-1   GCT-4
GGA-5   GGC-7   GGG-0   GGT-1
GTA-3   GTC-2   GTG-0   GTT-3
TAA-0   TAC-8   TAG-0   TAT-2
TCA-7   TCC-11  TCG-1   TCT-4
TGA-10  TGC-0   TGG-1   TGT-0
TTA-8   TTC-7   TTG-1   TTT-8
```

6 Look up the amino acids for codons ATA, CTA, ACC, and ATC.

```
aminolookup('code',nt2aa('ATA'))
aminolookup('code',nt2aa('CTA'))
aminolookup('code',nt2aa('ACC'))
aminolookup('code',nt2aa('ATC'))
```

MATLAB displays the following

```
Ile isoleucine
Leu leucine
Thr threonine
Ile isoleucine
```

Amino Acid Conversion and Composition

Determining the relative amino acid composition of a protein will give you a characteristic profile for the protein. Often, this profile is enough information to identify a protein. Using the amino acid composition, atomic composition, and molecular weight, you can also search public databases for similar proteins.

After you locate an open reading frame (ORF) in a gene, you can convert it to an amino sequence and determine its amino acid composition. This procedure uses the human mitochondria genome as an example. See “Open Reading Frames” on page 2-11.

1 Convert a nucleotide sequence to an amino acid sequence. In this example only the protein-coding sequence between the start and stop codons is converted.

   ```
   ND2AASeq = nt2aa(ND2Seq,'geneticcode','Vertebrate Mitochondrial');
   ```

   The sequence is converted using the `Vertebrate Mitochondrial` genetic code. Because the property `AlternativeStartCodons` is set to `'true'` by default, the first codon `att` is converted to `M` instead of `I`.

   ```
   MNPLAQPVIYSTIFAGTLITALSSHWFFTWVGLEMNMLAFIPVLTKKMNP
   RSTEAAIKYFLTQATASMILLMAILFNNMLSGQWTMTNTTNQYSSLMIMM
   AMAMKLGMAPFHFWVPEVTQGTPLTSGLLLLTWQKLAPISIMYQISPSLN
   VSLLLTLSILSIMAGSWGGLNQTQLRKILAYSSITHMGWMMAVLPYNPNM
   TILNLTIYIILTTTAFLLLNLNSSTTTLLLSRTWNKLTWLTPLIPSTLLS
   LGGLPPLTGFLPKWAIIEEFTKNNSLIIPTIMATITLLNLYFYLRLIYST
   SITLLPMSNNVKMKWQFEHTKPTPFLPTLIALTTLLLPISPFMLMIL
   ```

2 Compare your conversion with the published conversion in GenPept.

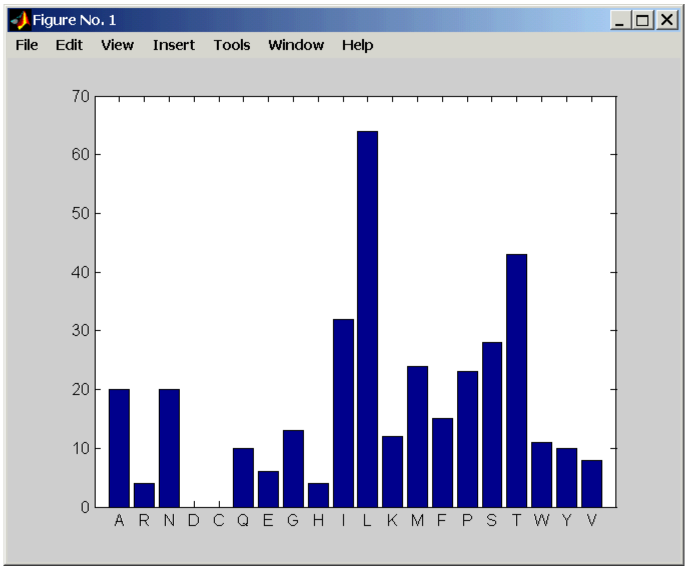
   ```
   ND2protein = getgenpept('NP_536844','sequenceonly',true)
   ```

   MATLAB gets the published conversion from the NCBI database and reads it into the MATLAB workspace.

3 Count the amino acids in the protein sequence.

   ```
   aacount(ND2AASeq, 'chart','bar')
   ```

MATLAB draws a bar graph. Notice the high content for leucine, threonine and isoleucine, and also notice the lack of cysteine and aspartic acid.



4 Determine the atomic composition and molecular weight of the protein.

```
atomiccomp(ND2AASeq)
molweight (ND2AASeq)
```

MATLAB displays the following.

```
ans =
    C: 1818
    H: 3574
    N: 420
    O: 817
    S: 25

ans =
  3.8960e+004
```

If this sequence was unknown, you could use this information to identify the protein by comparing it with the atomic composition of other proteins in a database.

Example: Sequence Alignment

Determining the similarity between two sequences is a common task in computational biology. Starting with a nucleotide sequence for a human gene, this example uses alignment algorithms to locate a similar gene in another organism.

- "Finding a Model Organism to Study" on page 2-17 — Use the MATLAB Help browser to search the Web for information.
- "Getting Sequence Information from a Public Database" on page 2-19 — Find the nucleotide sequence for a human gene in a public database and read the sequence information into MATLAB.
- "Searching a Public Database for Related Genes" on page 2-21' — Find the nucleotide sequence for a mouse gene related to a human gene, and read the sequence information into MATLAB.
- "Locating Protein Coding Sequences" on page 2-23 — Convert a sequence from nucleotides to amino acids and identify the open reading frames.
- "Comparing Amino Acid Sequences" on page 2-26 — Use global and local alignment functions to compare two amino acid sequences.

Finding a Model Organism to Study

In this example, you are interested in studying Tay-Sachs disease. Tay-Sachs is an autosomal recessive disease caused by the absence of the enzyme beta-hexosaminidase A (Hex A). This enzyme is responsible for the breakdown of gangliosides (GM2) in brain and nerve cells.

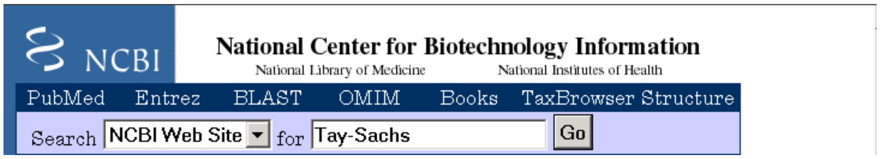
First, to research information about Tay-Sachs and the enzyme that is associated with this disease, then find the nucleotide sequence for the human gene that codes for the enzyme, and finally find a corresponding gene in another organism to use as a model for study.

1 Use the MATLAB Help browser to explore the Web. In the **MATLAB Command Window**, type

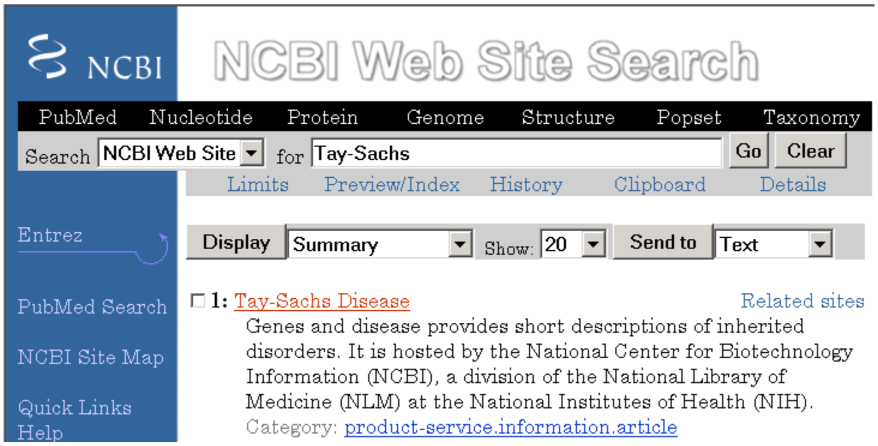
```
web('http://www.ncbi.nlm.nih.gov/')
```

The MATLAB Help browser opens with the home page for the NCBI web site.

2 Search the NCBI Web site for information. For example, to search for Tay-Sachs, from the **Search** list, select `NCBI Web Site`, and in the **for** box, enter `Tay-Sachs`.


NCBI
National Center for Biotechnology Information
National Library of Medicine
National Institutes of Health
PubMed Entrez BLAST OMIM Books TaxBrowser Structure
Search NCBI Web Site for Tay-Sachs Go

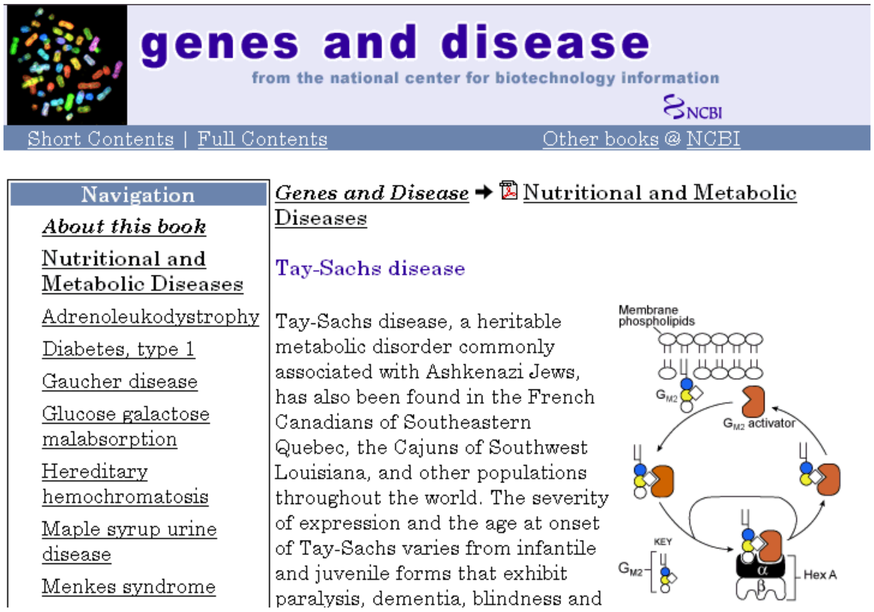
The NCBI Web search returns a list of links to relevant pages.


NCBI
NCBI Web Site Search
PubMed Nucleotide Protein Genome Structure Popset Taxonomy
Search NCBI Web Site for Tay-Sachs Go Clear
Limits Preview/Index History Clipboard Details
Entrez
PubMed Search
NCBI Site Map
Quick Links
Help
Display Summary Show: 20 Send to Text

1: Tay-Sachs Disease Related sites
Genes and disease provides short descriptions of inherited disorders. It is hosted by the National Center for Biotechnology Information (NCBI), a division of the National Library of Medicine (NLM) at the National Institutes of Health (NIH).
Category: product-service.information.article

3 Select a result page. For example, click the link labeled **Tay-Sachs Disease**

A page in the genes and diseases section of the NCBI Web site opens. This section provides a comprehensive introduction to medical genetics. In particular, this page contains an introduction and pictorial representation of the enzyme Hex A and its role in the metabolism of the lipid GM2 ganglioside.


genes and disease
from the national center for biotechnology information
NCBI
Short Contents | Full Contents   Other books @ NCBI

Navigation
About this book
Nutritional and Metabolic Diseases
Adrenoleukodystrophy
Diabetes, type 1
Gaucher disease
Glucose galactose malabsorption
Hereditary hemochromatosis
Maple syrup urine disease
Menkes syndrome

Genes and Disease ➔ Nutritional and Metabolic Diseases

Tay-Sachs disease

Tay-Sachs disease, a heritable metabolic disorder commonly associated with Ashkenazi Jews, has also been found in the French Canadians of Southeastern Quebec, the Cajuns of Southwest Louisiana, and other populations throughout the world. The severity of expression and the age at onset of Tay-Sachs varies from infantile and juvenile forms that exhibit paralysis, dementia, blindness and

4 After completing your research, you have concluded the following:

The gene HEXA codes for the alpha subunit of the dimer enzyme hexosaminidase A (Hex A), while the gene HEXB codes for the beta subunit of the enzyme. A third gene, GM2A, codes for the activator protein GM2. However, it is a mutation in the gene HEXA that causes Tay-Sachs.

Getting Sequence Information from a Public Database

Many public databases for nucleotide sequences (for example, GenBank, EMBL-EBI) are accessible from the Web. The MATLAB Command Window with the MATLAB Help browser provide an integrated environment for searching the Web and bringing sequence information into MATLAB.

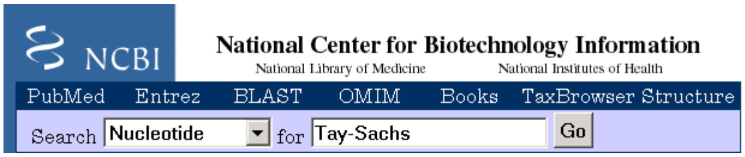
After you locate a sequence, you need to move the sequence data into the MATLAB workspace.

1 Open the MATLAB Help browser to the NCBI web site. In the **MATLAB Command Widow**, type

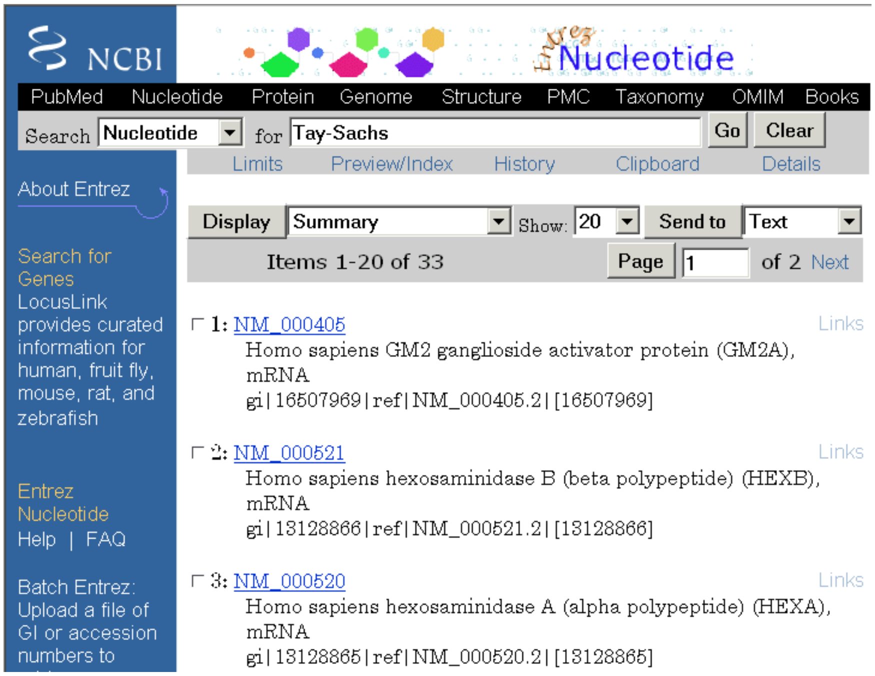
```
web('http://www.ncbi.nlm.nih.gov/')
```

The MATLAB Help browser window opens with the NCBI home page.

2 Search for the gene you are interested in studying. For example, from the **Search** list, select `Nucleotide`, and in the **for** box enter `Tay-Sachs`.


NCBI
National Center for Biotechnology Information
National Library of Medicine
National Institutes of Health
PubMed Entrez BLAST OMIM Books TaxBrowser Structure
Search Nucleotide for Tay-Sachs Go

The search returns entries for the genes that code the alpha and beta subunits of the enzyme hexosaminidase A (Hex A), and the gene that codes the activator enzyme. The NCBI reference for the human gene HEXA has accession number `NM_000520`.


NCBI
Entrez Nucleotide
PubMed Nucleotide Protein Genome Structure PMC Taxonomy OMIM Books
Search Nucleotide for Tay-Sachs Go Clear
Limits Preview/Index History Clipboard Details
About Entrez
Display Summary Show: 20 Send to Text
Items 1-20 of 33
Page 1 of 2 Next

Search for Genes
LocusLink provides curated information for human, fruit fly, mouse, rat, and zebrafish

Entrez Nucleotide
Help | FAQ

Batch Entrez: Upload a file of GI or accession numbers to

1: NM_000405 Links
Homo sapiens GM2 ganglioside activator protein (GM2A), mRNA
gi|16507969|ref|NM_000405.2|[16507969]

2: NM_000521 Links
Homo sapiens hexosaminidase B (beta polypeptide) (HEXB), mRNA
gi|13128866|ref|NM_000521.2|[13128866]

3: NM_000520 Links
Homo sapiens hexosaminidase A (alpha polypeptide) (HEXA), mRNA
gi|13128865|ref|NM_000520.2|[13128865]

3 Get sequence data into MATLAB. For example, to get sequence information for the human gene HEXA, type

```
humanHEXA = getgenbank('NM_000520')
```

Note that blank spaces in GenBank accession numbers use the underline character. Entering `'NM 00520'` returns the wrong entry.

The human gene is loaded into the MATLAB workspace as a structure.

```
humanHEXA =
                LocusName: 'HEXA'
      LocusSequenceLength: '2255'
     LocusNumberofStrands: ''
            LocusTopology: 'linear'
        LocusMoleculeType: 'mRNA'
     LocusGenBankDivision: 'PRI'
   LocusModificationDate: '10-MAY-2002'
               Definition: [1x63 char]
                Accession: 'NM_000520'
                  Version: '     NM_000520.2'
                       GI: '13128865'
                 Keywords: '.'
                  Segment: []
                   Source: [1x87 char]
           SourceOrganism: [2x65 char]
                Reference: {1x7 cell}
                  Comment: [15x67 char]
                 Features: [71x79 char]
                BaseCount: [1x1 struct]
                 Sequence: [1x2255 char]
```

Searching a Public Database for Related Genes

The sequence and function of many genes is conserved during the evolution of species through homologous genes. Homologous genes are genes that have a common ancestor and similar sequences. One goal of searching a public database is to find similar genes. If you are able to locate a sequence in a database that is similar to your unknown gene or protein, it is likely that the function and characteristics of the known and unknown genes are the same.

After finding the nucleotide sequence for a human gene, you can do a BLAST search or search in the genome of another organism for the corresponding gene. This procedure uses the mouse genome as an example.

1 Open the MATLAB Help browser to the NCBI Web site. In the **MATLAB Command** window, type

```
web('http://www.ncbi.nlm.nih.gov')
```

2 Search the nucleotide database for the gene or protein you are interested in studying. For example, from the **Search** list, select `Nucleotide`, and in the **for** box enter `hexosaminidase A`.

The search returns entries for the mouse and human genomes. The NCBI reference for the mouse gene HEXA has accession number `AK080777`.

3 Get sequence information for the mouse gene into MATLAB. Type

```
mouseHEXA = getgenbank('AK08077')
```

The mouse gene sequence is loaded into the MATLAB workspace as a structure.

```
mouseHEXA =
                LocusName: 'AK080777'
     LocusSequenceLength: '1839'
    LocusNumberofStrands: ''
           LocusTopology: 'linear'
       LocusMoleculeType: 'mRNA'
    LocusGenBankDivision: 'HTC'
   LocusModificationDate: '05-DEC-2002'
              Definition: [1x67 char]
               Accession: [1x201 char]
                 Version: '     AK080777.1'
                      GI: '26348756'
                Keywords: 'HTC; CAP trapper.'
                 Segment: []
                  Source: [1x93 char]
          SourceOrganism: [2x66 char]
               Reference: {1x6 cell}
                 Comment: [12x66 char]
                Features: [31x79 char]
               BaseCount: [1x1 struct]
                Sequence: [1x1839 char]
```

Locating Protein Coding Sequences

A nucleotide sequence includes regulatory sequences before and after the protein coding section. By analyzing this sequence, you can determine the nucleotides that code for the amino acids in the final protein.

After you have a list of genes you are interested in studying, you can determine the protein coding sequences. This procedure uses the human gene HEXA and mouse gene HEXA as an example.

1 If you did not retrieve gene data from the Web, you can load example data from a MAT-file included with the Bioinformatics Toolbox. In the **MATLAB Command** window, type

   ```
   load hexosaminidase
   ```

   MATLAB loads the structures humanHEXA and mouseHEXA into the MATLAB workspace.

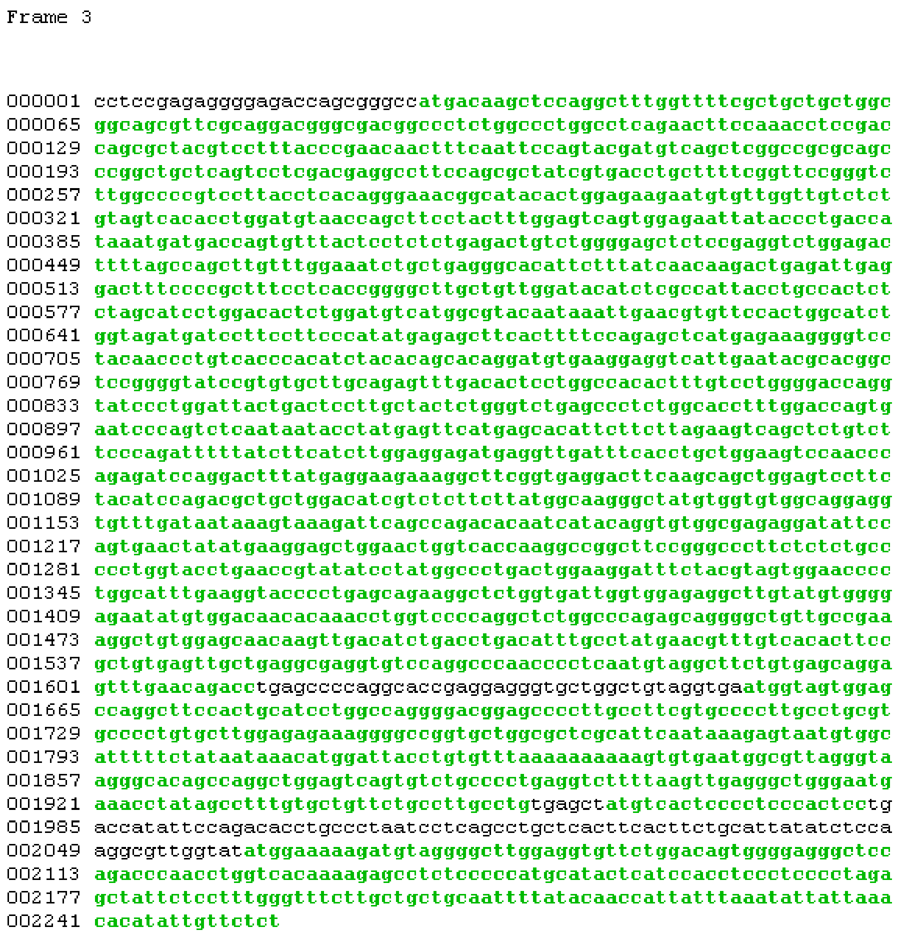
2 Look for open reading frames in the human gene. For example, for the human gene HEXA, type

```
humanORFs=seqshoworfs(humanHEXA.Sequence)
```

`seqshoworfs` creates the output structure `humanORFs`. This structure gives the position of the start and stop codons for all open reading frames (ORFs) on each reading frame.

```
humanORFs =

1x3 struct array with fields:
    Start
    Stop
```

The Help browser opens with a listing for the three reading frames with the ORFs colored blue, red, and green. Notice that the longest ORF is on the third reading frame.



```
Frame 3


000001 cctccgagaggggagaccagcgggccatgacaagctccaggctttggttttcgctgctgctggc
000065 ggcagcgttcgcaggacgggcgacggccctctggccctggcctcagaacttccaaacctccgac
000129 cagcgctacgtcctttacccgaacaactttcaattccagtacgatgtcagctcggccgcgcagc
000193 ccggctgctcagtcctcgacgaggccttccagcgctatcgtgacctgcttttcggttccgggtc
000257 ttggccccgtccttacctcacagggaaacggcatacactggagaagaatgtgttggttgtctct
000321 gtagtcacacctggatgtaaccagcttcctactttggagtcagtggagaattataccctgacca
000385 taaatgatgaccagtgtttactcctctctgagactgtctggggagctctccgaggtctggagac
000449 ttttagccagcttgtttggaaatctgctgagggcacattctttatcaacaagactgagattgag
000513 gactttccccgctttcctcaccggggcttgctgttggatacatctcgccattacctgccactct
000577 ctagcatcctggacactctggatgtcatggcgtacaataaattgaacgtgttccactggcatct
000641 ggtagatgatccttccttcccatatgagagcttcacttttccagagctcatgagaaaggggtcc
000705 tacaaccctgtcacccacatctacacagcacaggatgtgaaggaggtcattgaatacgcacggc
000769 tccggggtatccgtgtgcttgcagagtttgacactcctggccacactttgtcctggggaccagg
000833 tatccctggattactgactccttgctactctgggtctgagccctctggcacctttggaccagtg
000897 aatcccagtctcaataatacctatgagttcatgagcacattcttcttagaagtcagctctgtct
000961 tcccagattttatcttcatcttggaggagatgaggttgatttcacctgctggaagtccaaccc
001025 agagatccaggactttatgaggaagaaaggcttcggtgaggacttcaagcagctggagtccttc
001089 tacatccagacgctgctggacatcgtctcttcttatggcaagggctatgtggtgtggcaggagg
001153 tgtttgataataaagtaaagattcagccagacacaatcatacaggtgtggcgagaggatattcc
001217 agtgaactatatgaaggagctggaactggtcaccaaggccggcttccgggcccttctctctgcc
001281 ccctggtacctgaaccgtatatcctatggccctgactggaaggatttctacgtagtggaacccc
001345 tggcatttgaaggtacccctgagcagaaggctctggtgattggtggagaggcttgtatgtgggg
001409 agaatatgtggacaacacaaacctggtccccaggctctggcccagagcaggggctgttgccgaa
001473 aggctgtggagcaacaagttgacatctgacctgacatttgcctatgaacgtttgtcacacttcc
001537 gctgtgagttgctgaggcgaggtgtccaggcccaacccctcaatgtaggcttctgtgagcagga
001601 gtttgaacagacctgagccccaggcaccgaggagggtgctggctgtaggtgaatggtagtggag
001665 ccaggcttccactgcatcctggccaggggacggagccccttgccttcgtgccccttgcctgcgt
001729 gcccctgtgcttggagagaaaggggccggtgctggcgctcgcattcaataaagagtaatgtggc
001793 atttttctataataaacatggattacctgtgtttaaaaaaaaaagtgtgaatggcgttagggta
001857 agggcacagccaggctggagtcagtgtctgcccctgaggtcttttaagttgagggctgggaatg
001921 aaacctatagcctttgtgctgttctgccttgcctgtgagctatgtcactcccctcccactcctg
001985 accatattccagacacctgccctaatcctcagcctgctcacttcacttctgcattatatctcca
002049 aggcgttggtatatggaaaaagatgtagggcttggaggtgttctggacagtggggagggctcc
002113 agacccaacctggtcacaaaagagcctctcccccatgcatactcatccacctccctcccctaga
002177 gctattctcctttgggtttcttgctgctgcaattttatacaaccattatttaaatattattaaa
002241 cacatattgttctct
```

3 Locate open reading frames (ORFs) on the mouse gene. Type

```
mouseORFs = seqshoworfs(mouseHEXA.Sequence)
```

`seqshoworfs` creates the structure `mouseORFS`.

```
mouseORFs =

1x3 struct array with fields:
    Start
    Stop
```

The mouse gene shows the longest ORF on the first reading frame.

```
Frame 1


000001 gctgctggaaggggagctggccggtgggccatggccggctgcaggctctgggtttcgctgctgc
000065 tggcggcggcgttggcttgcttggccacggcactgtggccgtggccccagtacatccaaaccta
000129 ccaccggcgctacaccctgtaccccaacaacttccagttccggtaccatgtcagttcggccgcg
000193 caggcgggctgcgtcgtcctcgacgaggcctttcgacgctaccgtaacctgctcttcggttccg
000257 gctcttggccccgacccagcttctcaaataaacagcaaacgttggggaagaacattctggtggt
000321 ctccgtcgtcacagctgaatgtaatgaatttcctaatttggagtcggtagaaaattacaccctа
000385 accattaatgatgaccagtgtttactcgcctctgagactgtctggggcgctctccgaggtctgg
000449 agactttcagtcagcttgtttggaaatcagctgagggcacgttctttatcaacaagacaaagat
000513 taaagactttcctcgattccctcaccggggcgtactgctggatacatctcgccattacctgcca
000577 ttgtctagcatcctggatacactggatgtcatggcatacaataaattcaacgtgttccactggc
000641 acttggtggacgactcttccttcccatatgagagcttcactttcccagagctcaccagaaaggg
000705 gtccttcaaccctgtcactcacatctacacagcacaggatgtgaaggaggtcattgaatacgca
000769 aggcttcggggtatccgtgtgctggcagaatttgacactcctggccacactttgtcctgggggc
000833 caggtgcccctgggttattaacaccttgctactctgggtctcatctctctggcacatttggacc
000897 ggtgaaccccagtctcaacagcacctatgacttcatgagcacactcttcctggagatcagctca
000961 gtcttcccggacttttatctccacctgggaggggatgaagtcgacttcacctgctggaagtcca
001025 accccaacatccaggccttcatgaagaaaaagggctttactgacttcaagcagctggagtcctt
001089 ctacatccagacgctgctggacatcgtctctgattatgacaagggctatgtggtgtggcaggag
001153 gtatttgataataaagtgaaggttcggccagatacaatcatacaggtgtggcgggaagaaatgc
001217 cagtagagtacatgttggagatgcaagatatcaccagggctggcttccgggccctgctgtctgc
001281 tccctggtacctgaaccgtgtaaagtatggccctgactggaaggacatgtacaaagtggagccc
001345 ctggcgtttcatggtacgcctgaacagaaggctctggtcattggaggggaggcctgtatgtggg
001409 gagagtatgtggacagcaccaacctggtccccagactctggcccagagcgggtgccgtcgctga
001473 gagactgtggagcagtaacctgacaactaatatagactttgcctttaaacgtttgtcgcatttc
001537 cgttgtgagctggtgaggagaggaatccaggcccagcccatcagtgtaggctgctgtgagcagg
001601 agtttgagcagacttgagccaccagtgctgaacacccaggaggttgctgtcctttgagtcagct
001665 gcgctgagcacccaggagggtgctggccttaagagagcaggtcccggggcagggctaatctttc
001729 actgcctcccggccaggggagagcaccccttgcccgtgtgcccctgtgactacagagaaggagg
001793 ctggtgctggcactggtgttcaataaagatctatgtggcattttctc
```

Comparing Amino Acid Sequences

You could use alignment functions to look for similarities between two nucleotide sequences, but alignment functions return more biologically meaningful results when you are using amino acid sequences.

After you have located the open reading frames on your nucleotide sequences, you can convert the protein coding sections of the nucleotide sequences to their corresponding amino acid sequences, and then you can compare them for similarities.

1 Using the identified open reading frames, convert the DNA sequence to the amino acid sequences. Type

```
mouseProtein = nt2aa(mouseHEXA.Sequence)
```

Remember that the human HEXA gene was on the third reading frame, so you need to indicate which frame to use.

```
humanProtein = nt2aa(humanHEXA.Sequence,'frame',3)
```

2 Draw a dot plot comparing the human and mouse amino acid sequences. Type

```
seqdotplot(mouseProtein,humanProtein,4,3)
ylabel('Mouse hexosaminidase A (alpha subunit)')
xlabel('Human hexosaminidase A (alpha subunit)')
```

Dot plots are one of the easiest ways to look for similarity between sequences. The diagonal line shown below indicates that there may be a good alignment between the two sequences.

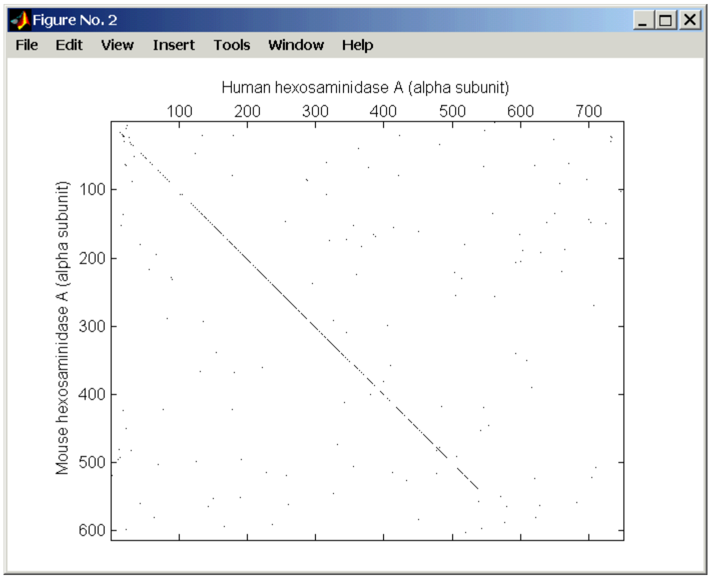


Figure No. 2
File Edit View Insert Tools Window Help
Human hexosaminidase A (alpha subunit)
100 200 300 400 500 600 700
Mouse hexosaminidase A (alpha subunit)
100 200 300 400 500 600

3 Globally align the two amino acid sequences, using the Needleman-Wunsch algorithm. Type

```
[GlobalScore, GlobalAlignment = nwalign(humanProtein,
                                         mouseProtein)
showalignment(GlobalAlignment)
```

`showalignment` displays the global alignment of the two sequences in the Help browser. Notice that the calculated identity between the two sequences is `64.5` %.

```
Identities = 486/753 (65%), Positives = 570/753 (76%)
  1  SE-RGDQR-AMTSSRLWFSLLLAAAFAGRATALWPWPQNFQTSDQRYVLYPNNFQFQYDVSSAA
     :  ||  | ||:: ||| |||||||:|  ||||||||| :||  :||:||||||||:| |||||
  1  AAGRGAGRWAMAGCRLWVSLLLAAALACLATALWPWPQYIQTYHRRYTLYPNNFQFRYHVSSAA

 63  QPGCSVLDEAFQRYRDLLFGSGSWPRPYLTGKRHTLEKNVLVVSVVTPGCNQLPTLESVENYTL
     | || ||||||:|||:||||||||||| :::|::|| ||:|||||||  ||::|:|||||||||
 65  QAGCVVLDEAFRRYRNLLFGSGSWPRPSFSNKQQTLGKNILVVSVVTAECNEFPNLESVENYTL

127  TINDDQCLLLSETVWGALRGLETFSQLVWKSAEGTFFINKTEIEDFPRFPHRGLLLDTSRHYLP
     ||||||||| |||||||||||||||||||||||||||||||:|:|||||||||:||||||||||
129  TINDDQCLLASETVWGALRGLETFSQLVWKSAEGTFFINKTKIKDFPRFPHRGVLLDTSRHYLP

191  LSSILDTLDVMAYNKLNVFHWHLVDDPSFPYESFTFPELMRKGSYNPVTHIYTAQDVKEVIEYA
     |||||||||||||||:|||||||||| |||||||||||| ||||:|||||||||||||||||||
193  LSSILDTLDVMAYNKFNVFHWHLVDDSSFPYESFTFPELTRKGSFNPVTHIYTAQDVKEVIEYA

255  RLRGIRVLAEFDTPGHTLSWGPGIPGLLTPCYSGSEPSGTFGPVNPSLNNTYEFMSTFFLEVSS
     ||||||||||||||||||||||| |||||||||||: ||||||||||||:||:||||:|||:||
257  RLRGIRVLAEFDTPGHTLSWGPGAPGLLTPCYSGSHLSGTFGPVNPSLNSTYDFMSTLFLEISS

319  VFPDFYLHLGGDEVDFTCWKSNPEIQDFMRKKGFGEDFKQLESFYIQTLLDIVSSYGKGYVVWQ
     |||||||||||||||||||||||:|| ||:||||  ||||||||||||||||||:| |||||||
321  VFPDFYLHLGGDEVDFTCWKSNPNIQAFMKKKGF-TDFKQLESFYIQTLLDIVSDYDKGYVVWQ

383  EVFDNKVKIQPDTIIQVWREDIPVNYMKELELVTKAGFRALLSAPWYLNRISYGPDWKDFYVVE
     ||||||||::||||||||||::||:|| |:: :|:|||||||||||||||::|||||||:| ||
384  EVFDNKVKVRPDTIIQVWREEMPVEYMLEMQDITRAGFRALLSAPWYLNRVKYGPDWKDMYKVE

447  PLAFEGTPEQKALVIGGEACMWGEYVDNTNLVPRLWPRAGAVAERLWSNKLTSDLTFAYERLSH
     ||||:||||||||||||||||||||||:||||||||||||||||||||::||::: ||::||||
448  PLAFHGTPEQKALVIGGEACMWGEYVDSTNLVPRLWPRAGAVAERLWSSNLTTNIDFAFKRLSH

511  FRCELLRRGVQAQPLNVGFCEQEFEQT*APGTEEGAGCR*MVVEPGFHCILARGRSPLPSCPLP
     |||||:|||:||||::|| ||||||||||  |  :|  :    :||       |      |
512  FRCELVRRGIQAQPISVGCCEQEFEQT*A--T--SA--E----HPG-------G------C---

575  ACPCAWRERGRCWRSHSIKSNVAFFYNKHGLPVFKKKSVNGVRVRAQPGWSQCLPLRSFKLRAG
      ||           |: ::   |        |   :: |  : :| :    | :|  : :   |
550  -CP---------L-SQ-LR--*A--------P---RR-V--LALR-E----Q-VP--G-Q---G

639  NETYSLCAVLPCL*AMSLPSHS*PYSRHLP*SSACSLHFCIISPRRWYMEKDVGAWRCSGQWGG
       :::         | | |::|   :   |    |    |  :|     ||::||    |   |
574  -*SFT---------A-SRPGES---T---P----CP---C--APVT--TEKEAGA----GT--G

703  LQTQPGHKRASPPCILIHLPPLELFSFGFLAAAILYNHYLNIIKHILFS
     :  |    |                |        ::  |:       |
604  V--Q---*R----------------S--------MW-HF-------L--
```

The alignment is very good for the first 550 nucleotides, after which the two sequences appear to be unrelated. Notice that there is a stop (*) in the sequence at this point. If you shorten the sequence to include only the amino acids that are in the protein (after the first methionine and before the first stop) you might get a better alignment.

4 Trim the sequence from the first start amino acid (usually M) to the first stop (first *) and then try alignment again. Find the indices for the stops in the sequences.

```
humanStops = find(humanProtein == '*')

humanStops =
   538   550   652   661   669


mouseStops = find(mouseProtein =='*')

mouseStops =

   539   557   574   606
```

Looking at the amino acid sequence for `humanProtein`, the first M is at position 9, while the first M for the mouse protein is at 11.

5 Truncate the sequence to include only amino acids in the protein and the stop.

```
humanProteinORF = humanProtein(9:humanStops(1));

humanProteinORF =
MTSSRLWFSLLLAAAFAGRATALWPWPQNFQTSDQRYVLYPNNFQFQYDV
SSAAQPGCSVLDEAFQRYRDLLFGSGSWPRPYLTGKRHTLEKNVLVVSVV
TPGCNQLPTLESVENYTLTINDDQCLLLSETVWGALRGLETFSQLVWKSA
EGTFFINKTEIEDFPRFPHRGLLLDTSRHYLPLSSILDTLDVMAYNKLNV
FHWHLVDDPSFPYESFTFPELMRKGSYNPVTHIYTAQDVKEVIEYARLRG
IRVLAEFDTPGHTLSWGPGIPGLLTPCYSGSEPSGTFGPVNPSLNNTYEF
MSTFFLEVSSVFPDFYLHLGGDEVDFTCWKSNPEIQDFMRKKGFGEDFKQ
LESFYIQTLLDIVSSYGKGYVVWQEVFDNKVKIQPDTIIQVWREDIPVNY
MKELELVTKAGFRALLSAPWYLNRISYGPDWKDFYVVEPLAFEGTPEQKA
LVIGGEACMWGEYVDNTNLVPRLWPRAGAVAERLWSNKLTSDLTFAYERL
SHFRCELLRRGVQAQPLNVGFCEQEFEQT*
```

```
mouseProteinORF = mouseProtein(11:mouseStops(1))

mouseProteinORF =
MAGCRLWVSLLLAAALACLATALWPWPQYIQTYHRRYTLYPNNFQFRYHV
SSAAQAGCVVLDEAFRRYRNLLFGSGSWPRPSFSNKQQTLGKNILVVSVV
TAECNEFPNLESVENYTLTINDDQCLLASETVWGALRGLETFSQLVWKSA
EGTFFINKTKIKDFPRFPHRGVLLDTSRHYLPLSSILDTLDVMAYNKFNV
FHWHLVDDSSFPYESFTFPELTRKGSFNPVTHIYTAQDVKEVIEYARLRG
IRVLAEFDTPGHTLSWGPGAPGLLTPCYSGSHLSGTFGPVNPSLNSTYDF
MSTLFLEISSVFPDFYLHLGGDEVDFTCWKSNPNIQAFMKKKGFTDFKQL
ESFYIQTLLDIVSDYDKGYVVWQEVFDNKVKVRPDTIIQVWREEMPVEYM
LEMQDITRAGFRALLSAPWYLNRVKYGPDWKDMYKVEPLAFHGTPEQKAL
VIGGEACMWGEYVDSTNLVPRLWPRAGAVAERLWSSNLTTNIDFAFKRLS
HFRCELVRRGIQAQPISVGCCEQEFEQT*
```

6 Globally align the trimmed amino acid sequences. Type

```
[Score, Alignment] = nwalign(humanProteinORF,
    mouseProteinORF);
showalignment(Alignment)
```

showalignment displays the results for the second global alignment. Notice that the percent identity for the untrimmed sequences is 54% and with trimmed sequences 83.3 percent.

```
Identities = 445/529 (84%), Positives = 501/529 (95%)
  1   MTSSRLWFSLLLAAAFAGRATALWPWPQNFQTSDQRYVLYPNNFQFQYDVSSAAQPGCSVLDEA
      |:: ||| |||||||:|  ||||||||| :||  :||:||||||||:| |||||| || |||||
  1   MAGCRLWVSLLLAAALACLATALWPWPQYIQTYHRRYTLYPNNFQFRYHVSSAAQAGCVVLDEA

 65   FQRYRDLLFGSGSWPRPYLTGKRHTLEKNVLVVSVVTPGCNQLPTLESVENYTLTINDDQCLLL
      |:|||:||||||||||| :::|::|| ||:|||||||  ||::|:||||||||||||||||||
 65   FRRYRNLLFGSGSWPRPSFSNKQQTLGKNILVVSVVTAECNEFPNLESVENYTLTINDDQCLLA

129   SETVWGALRGLETFSQLVWKSAEGTFFINKTEIEDFPRFPHRGLLLDTSRHYLPLSSILDTLDV
      |||||||||||||||||||||||||||||||:|:|||||||||:||||||||||||||||||||
129   SETVWGALRGLETFSQLVWKSAEGTFFINKTKIKDFPRFPHRGVLLDTSRHYLPLSSILDTLDV

193   MAYNKLNVFHWHLVDDPSFPYESFTFPELMRKGSYNPVTHIYTAQDVKEVIEYARLRGIRVLAE
      |||||:|||||||||| |||||||||||| ||||:|||||||||||||||||||||||||||||
193   MAYNKFNVFHWHLVDDSSFPYESFTFPELTRKGSFNPVTHIYTAQDVKEVIEYARLRGIRVLAE

257   FDTPGHTLSWGPGIPGLLTPCYSGSEPSGTFGPVNPSLNNTYEFMSTFFLEVSSVFPDFYLHLG
      ||||||||||||| |||||||||||: ||||||||||||:||:||||:|||:||||||||||||
257   FDTPGHTLSWGPGAPGLLTPCYSGSHLSGTFGPVNPSLNSTYDFMSTLFLEISSVFPDFYLHLG

321   GDEVDFTCWKSNPEIQDFMRKKGFGEDFKQLESFYIQTLLDIVSSYGKGYVVWQEVFDNKVKIQ
      |||||||||||||:|| ||:||||  ||||||||||||||||||:| |||||||||||||||::
321   GDEVDFTCWKSNPNIQAFMKKKGF-TDFKQLESFYIQTLLDIVSDYDKGYVVWQEVFDNKVKVR

385   PDTIIQVWREDIPVNYMKELELVTKAGFRALLSAPWYLNRISYGPDWKDFYVVEPLAFEGTPEQ
      ||||||||||::||:|| |:: :|:|||||||||||||||::|||||||:| ||||||:|||||
384   PDTIIQVWREEMPVEYMLEMQDITRAGFRALLSAPWYLNRVKYGPDWKDMYKVEPLAFHGTPEQ

449   KALVIGGEACMWGEYVDNTNLVPRLWPRAGAVAERLWSNKLTSDLTFAYERLSHFRCELLRRGV
      |||||||||||||||||:||||||||||||||||||||::||::: ||::|||||||||:|||:
448   KALVIGGEACMWGEYVDSTNLVPRLWPRAGAVAERLWSSNLTTNIDFAFKRLSHFRCELVRRGI

513   QAQPLNVGFCEQEFEQT
      ||||::|| ||||||||
512   QAQPISVGCCEQEFEQT
```

7 Another way to truncate an amino acid sequence to only those amino acids in the protein is to first truncate the nucleotide sequence with indices from the function `seqshoworfs`. Remember that the ORF for the human HEXA gene was on the third reading frame, and the ORF for the mouse HEXA was on the first reading frame.

```
humanORFs = seqshoworfs(humanHEXA.Sequence);
mouseORFs = seqshoworfs(humanHEXA.Sequence);

humanPORF = nt2aa(humanHEXA.Sequence(humanORFs(3).Start(1):
    humanORFs(3)Stop(1)))
mousePORF = nt2aa(mouseHEXA.Sequence(mouseORFs(1).Start(1):
    mouseORFs(1)Stop(1)))
[Scale, Alignment] = nwalign(humanPORF, mousePORF)
```

Show the alignment in the Help browser.

```
showalignment(Alignment)
```

The result from first truncating a nucleotide sequence before converting to an amino acid sequence is the same as the result from truncating the amino acid sequence after conversion. See the result in step 6.

An alternative method to working with subsequences is to use a local alignment function with the nontruncated sequences.

8 Locally align the two amino acid sequences using a Smith-Waterman algorithm. Type

```
[LocalScore, LocalAlignment = swalign(humanProtein,
    mouseProtein)

LocalScore =
        1057

LocalAlignment
RGDQR-AMTSSRLWFSLLLAAAFAGRATALWPWPQNFQTSDQRYV . . .
||  | ||:: ||| |||||||:|  ||||||||| :||  :||: . . .
RGAGRWAMAGCRLWVSLLLAAALACLATALWPWPQYIQTYHRRYT . . .
```

swalign displays the local alignment of two sequences in the Help browser.

9 Show the alignment in color.

```
showalignment(LocalAlignment)
```

```
Identities = 454/547 (83%), Positives = 514/547 (94%)
  1   RGDQR-AMTSSRLWFSLLLAAAFAGRATALWPWPQNFQTSDQRYVLYPNNFQFQYDVSSAAQPG
      ||  | ||:: ||| |||||||:|  ||||||||| :||  :||:||||||||:| |||||| |
  1   RGAGRWAMAGCRLWVSLLLAAALACLATALWPWPQYIQTYHRRYTLYPNNFQFRYHVSSAAQAG

 64   CSVLDEAFQRYRDLLFGSGSWPRPYLTGKRHTLEKNVLVVSVVTPGCNQLPTLESVENYTLTIN
      | ||||||:|||:||||||||||| :::|::|| ||:|||||||  ||::|:||||||||||||
 65   CVVLDEAFRRYRNLLFGSGSWPRPSFSNKQQTLGKNILVVSVVTAECNEFPNLESVENYTLTIN

128   DDQCLLLSETVWGALRGLETFSQLVWKSAEGTFFINKTEIEDFPRFPHRGLLLDTSRHYLPLSS
      |||||| |||||||||||||||||||||||||||||||:|:|||||||||:|||||||||||||
129   DDQCLLASETVWGALRGLETFSQLVWKSAEGTFFINKTKIKDFPRFPHRGVLLDTSRHYLPLSS

192   ILDTLDVMAYNKLNVFHWHLVDDPSFPYESFTFPELMRKGSYNPVTHIYTAQDVKEVIEYARLR
      ||||||||||||:|||||||||| |||||||||||| ||||:||||||||||||||||||||||
193   ILDTLDVMAYNKFNVFHWHLVDDSSFPYESFTFPELTRKGSFNPVTHIYTAQDVKEVIEYARLR

256   GIRVLAEFDTPGHTLSWGPGIPGLLTPCYSGSEPSGTFGPVNPSLNNTYEFMSTFFLEVSSVFP
      |||||||||||||||||||| |||||||||||: ||||||||||||:||:||||:|||:|||||
257   GIRVLAEFDTPGHTLSWGPGAPGLLTPCYSGSHLSGTFGPVNPSLNSTYDFMSTLFLEISSVFP

320   DFYLHLGGDEVDFTCWKSNPEIQDFMRKKGFGEDFKQLESFYIQTLLDIVSSYGKGYVVWQEVF
      ||||||||||||||||||||:|| ||:||||  ||||||||||||||||||:| ||||||||||
321   DFYLHLGGDEVDFTCWKSNPNIQAFMKKKGF-TDFKQLESFYIQTLLDIVSDYDKGYVVWQEVF

384   DNKVKIQPDTIIQVWREDIPVNYMKELELVTKAGFRALLSAPWYLNRISYGPDWKDFYVVEPLA
      |||||::||||||||||::||:|| |:: :|:|||||||||||||||::|||||||:| |||||
384   DNKVKVRPDTIIQVWREEMPVEYMLEMQDITRAGFRALLSAPWYLNRVKYGPDWKDMYKVEPLA

448   FEGTPEQKALVIGGEACMWGEYVDNTNLVPRLWPRAGAVAERLWSNKLTSDLTFAYERLSHFRC
      |:||||||||||||||||||||||:||||||||||||||||||||::||::: ||::|||||||
448   FHGTPEQKALVIGGEACMWGEYVDSTNLVPRLWPRAGAVAERLWSSNLTTNIDFAFKRLSHFRC

512   ELLRRGVQAQPLNVGFCEQEFEQT*APGTEEGAGC
      ||:|||:||||::|| |||||||||| ::|: :||
512   ELVRRGIQAQPISVGCCEQEFEQT*ATSAEHPGGC
```

Microarray Analysis

You can use gene expression profiles from microarray data to research the function of cells, compare the differences between healthy and diseased tissue, and observe changes with the application of drugs.

The examples in this chapter will help you to become more familiar with the functions in the Bioinformatics Toolbox for analyzing and visualizing gene expression patterns.

Example: Visualizing Microarray Data

This example looks at the various ways to visualize microarray data. The microarray data for this example is from Brown, V.M., Ossadtchi, A., Khan, A.H., Yee, S., Lacan, G., Melega, W.P., Cherry, S.R., Leahy, R.M., and Smith, D.J.; "Multiplex three dimensional brain gene expression mapping in a mouse model of Parkinson's disease"; Genome Research 12(6): 868-884 (2002).

Overview of the Mouse Example

The microarray data used in this example is available in a web supplement to the paper by Brown et al. from

```
http://labs.pharmacology.ucla.edu/smithlab/index.html
```

The microarray data is also available on the Gene Expression Omnibus Web site at

```
http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE30
```

The GenePix GPR formatted file `mouse_a1pd.gpr` contains the data for one of the microarrays used in the study. This is data from voxel A1 of the brain of a mouse in which a pharmacological model of Parkinson's disease (PD) was induced using methamphetamine. The voxel sample was labeled with Cy3 (green) and the control, RNA from a total (not voxelated) normal mouse brain, was labeled with Cy5 (red). GPR formatted files provide a large amount of information about the array, including the mean, median, and standard deviation of the foreground and background intensities of each spot at the 635 nm wavelength (the red, Cy5 channel) and the 532 nm wavelength (the green, Cy3 channel).

Exploring the Microarray Data Set

This procedure uses data from a study about gene expression in mouse brains as an example. See “Overview of the Mouse Example” on page 3-2.

1 Read data from a file into a MATLAB structure. For example, in the **MATLAB Command Window**, type

```
pd = gprread('mouse_a1pd.gpr')
```

MATLAB displays information about the structure:

```
pd =
         Header: [1x1 struct]
           Data: [9504x38 double]
         Blocks: [9504x1 double]
        Columns: [9504x1 double]
           Rows: [9504x1 double]
          Names: {9504x1 cell}
            IDs: {9504x1 cell}
    ColumnNames: {38x1 cell}
        Indices: [132x72 double]
          Shape: [1x1 struct]
```

2 Access the fields of a structure using `StructureName.FieldName`. For example, you can access the field `ColumnNames` of the structure `pd` by typing

```
pd.ColumnNames
```

The column names are shown below.

```
ans =
    'X'
    'Y'
    'Dia.'
    'F635 Median'
    'F635 Mean'
    'F635 SD'
    'B635 Median'
    'B635 Mean'
    'B635 SD'
```

```
'% > B635+1SD'
'% > B635+2SD'
'F635 % Sat.'
'F532 Median'
'F532 Mean'
'F532 SD'
'B532 Median'
'B532 Mean'
'B532 SD'
'% > B532+1SD'
'% > B532+2SD'
'F532 % Sat.'
'Ratio of Medians'
'Ratio of Means'
'Median of Ratios'
'Mean of Ratios'
'Ratios SD'
'Rgn Ratio'
'Rgn R2'
'F Pixels'
'B Pixels'
'Sum of Medians'
'Sum of Means'
'Log Ratio'
'F635 Median - B635'
'F532 Median - B532'
'F635 Mean - B635'
'F532 Mean - B532'
'Flags'
```

3 Access the names of the genes. For example, to list the first 20 gene names, type

```
pd.Names(1:20)
```

A list of the first 20 gene names is displayed:

```
ans =
    'AA467053'
    'AA388323'
    'AA387625'
    'AA474342'
    'Myo1b'
    'AA473123'
    'AA387579'
    'AA387314'
    'AA467571'
            ''
    'Spop'
    'AA547022'
    'AI508784'
    'AA413555'
    'AA414733'
            ''
    'Snta1'
    'AI414419'
    'W14393'
    'W10596'
```

Spatial Images of Microarray Data

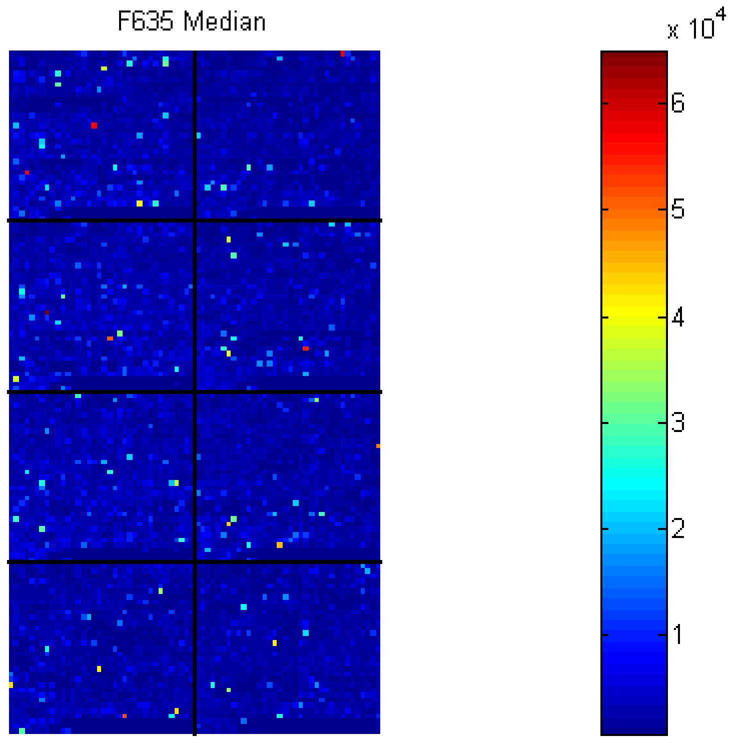
The function `maimage` can take a microarray data structure and create a pseudocolor image of the data arranged in the same order as the spots on the array. In other words, `maimage` plots a spatial plot of the microarray.

This procedure uses data from a study of gene expression in mouse brains. For a list of field names in the MATLAB structure `pd`, see “Exploring the Microarray Data Set” on page 3-3.

1 Plot the median values for the red channel. For example, to plot data from the field `F635 Median`, type

```
figure
maimage(pd,'F635 Median')
```

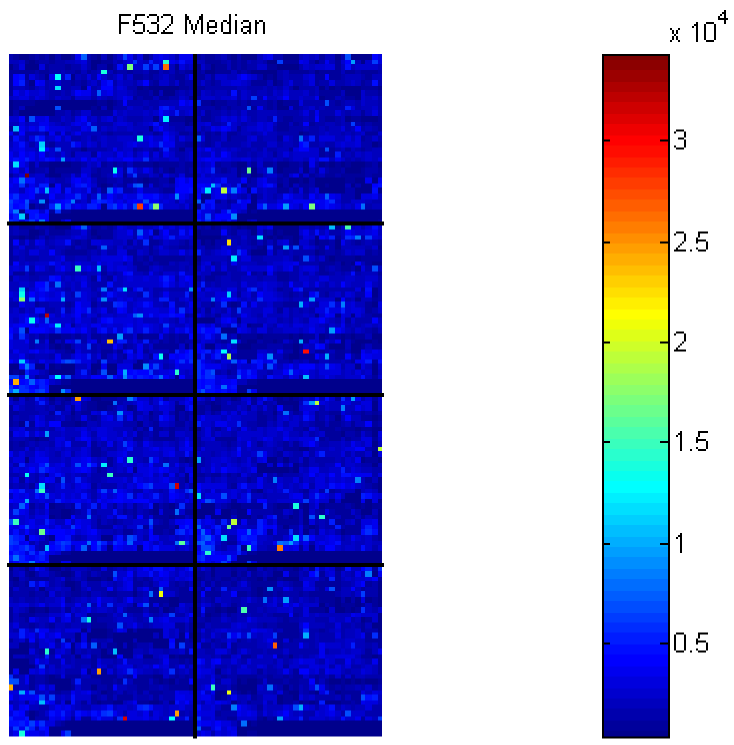
MATLAB plots an image showing the median pixel values for the foreground of the red (Cy5) channel.



2 Plot the median values for the green channel. For example, to plot data from the field `F532 Median`, type

```
figure
maimage(pd,'F532 Median')
```

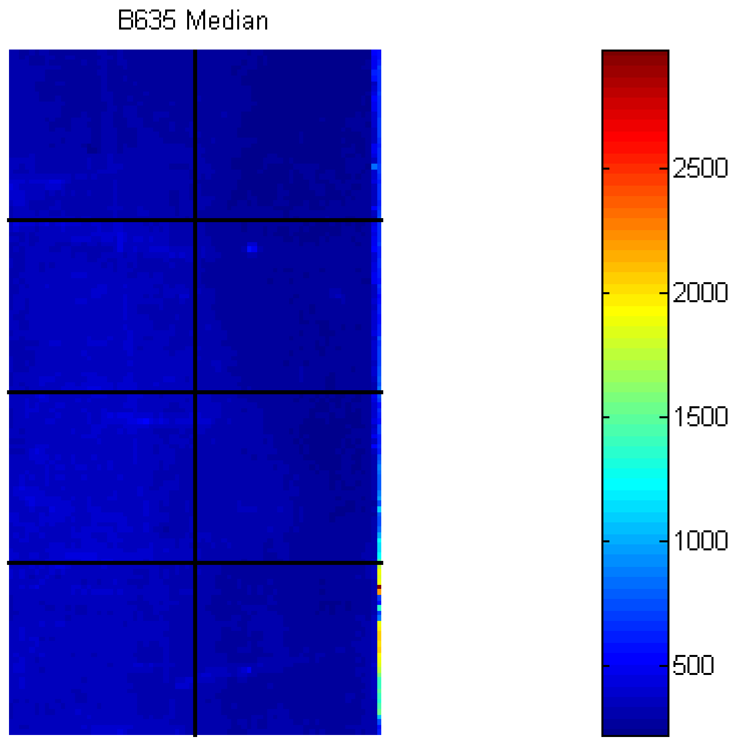
MATLAB plots an image showing the median pixel values of the foreground of the green (Cy3) channel.



3 Plot the median values for the red background. The field `B635 Median` shows the median values for the background of the red channel.

```
figure
maimage(pd,'B635 Median')
```

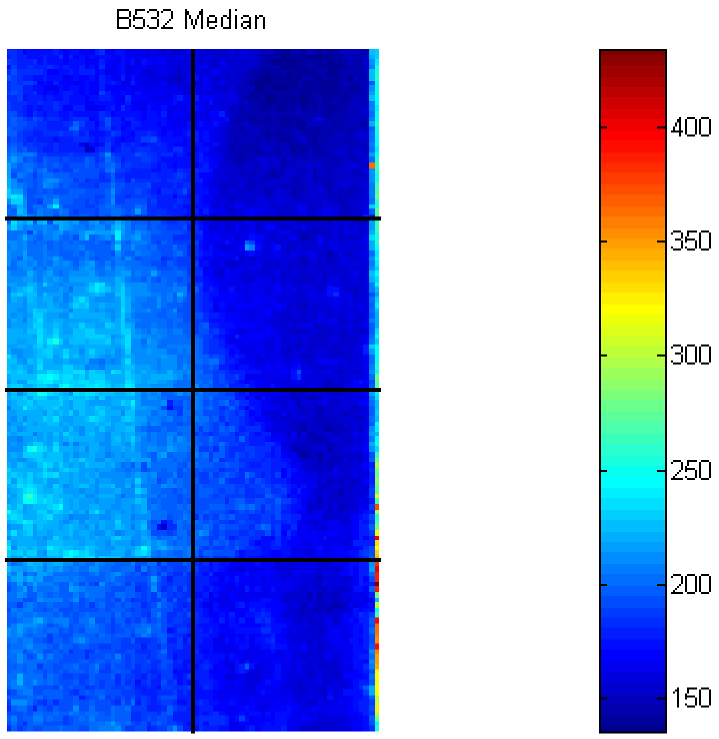
MATLAB plots an image for the background of the red channel. Notice the very high background levels down the right side of the array.



4 Plot the medial values for the green background. The field `B532 Median` shows the median values for the background of the green channel.

```
figure
maimage(pd,'B532 Median')
```

MATLAB plots an image for the background of the green channel.



5 The first array was for the Parkinson's disease model mouse. Now read in the data for the same brain voxel but for the untreated control mouse. In this case, the voxel sample was labeled with Cy3 and the control, total brain (not voxelated), was labeled with Cy5.

```
wt = gprread('mouse_a1wt.gpr')
```

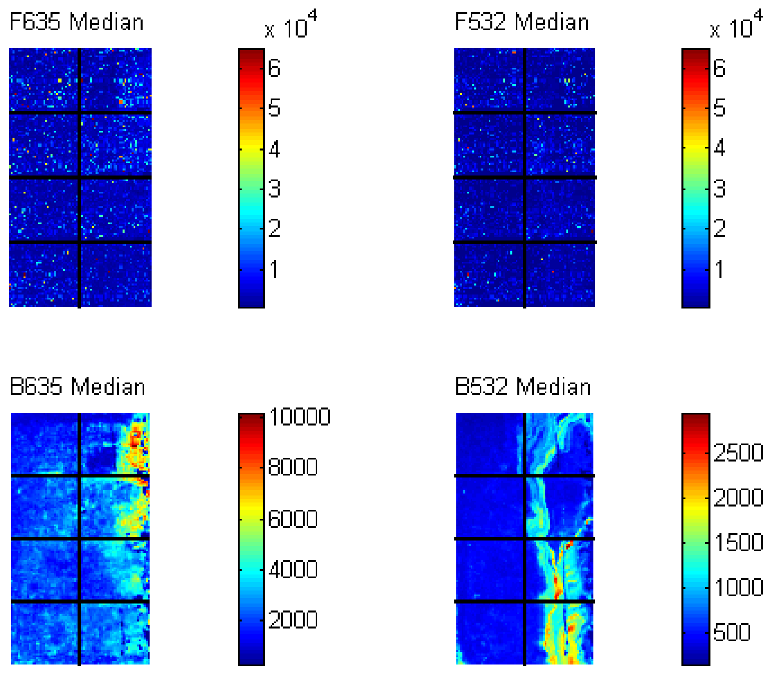
MATLAB creates a structure and displays information about the structure.

```
wt = 
         Header: [1x1 struct]
           Data: [9504x38 double]
         Blocks: [9504x1 double]
        Columns: [9504x1 double]
           Rows: [9504x1 double]
          Names: {9504x1 cell}
            IDs: {9504x1 cell}
    ColumnNames: {38x1 cell}
        Indices: [132x72 double]
          Shape: [1x1 struct]
```

6 Use the function `maimage` to show pseudocolor images of the foreground and background. You can use the function `subplot` to put all the plots onto one figure.

```
figure
subplot(2,2,1);
maimage(wt,'F635 Median')
subplot(2,2,2);
maimage(wt,'F532 Median')
subplot(2,2,3);
maimage(wt,'B635 Median')
subplot(2,2,4);
maimage(wt,'B532 Median')
```

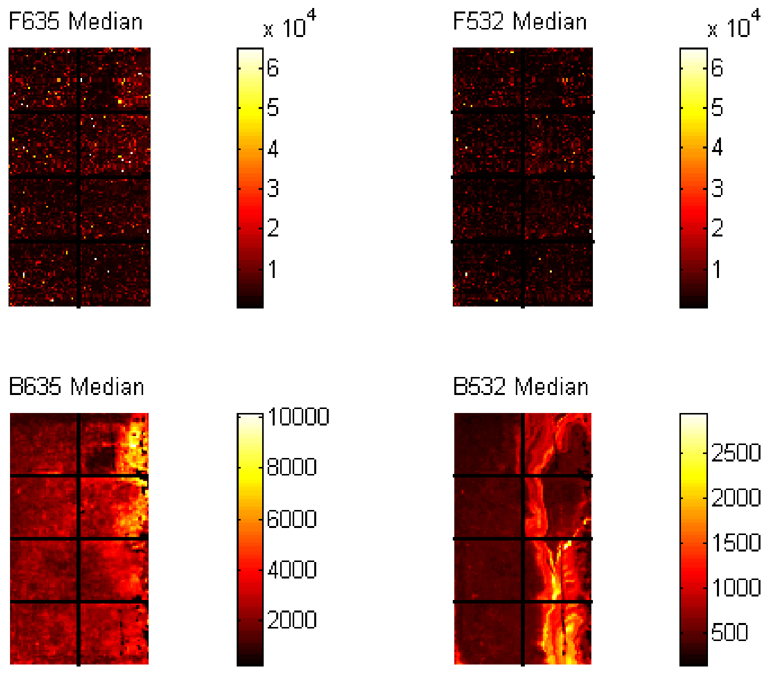
MATLAB plots the images.



7 If you look at the scale for the background images, you will notice that the background levels are much higher than those for the PD mouse and there appears to be something nonrandom affecting the background of the Cy3 channel of this slide. Changing the colormap can sometimes provide more insight into what is going on in pseudocolor plots. For more control over the color, try the `colormapeditor` function.

```
colormap hot
```

MATLAB plots the images.



8 The function `maimage` is a simple way to quickly create pseudocolor images of microarray data. However if you want more control over plotting, it is easy to create your own plots using the function `imagesc`.

First find the column number for the field of interest.

```
b532MedCol = find(strcmp(wt.ColumnNames,'B532 Median'))
```

MATLAB displays

```
b532MedCol =
    16
```

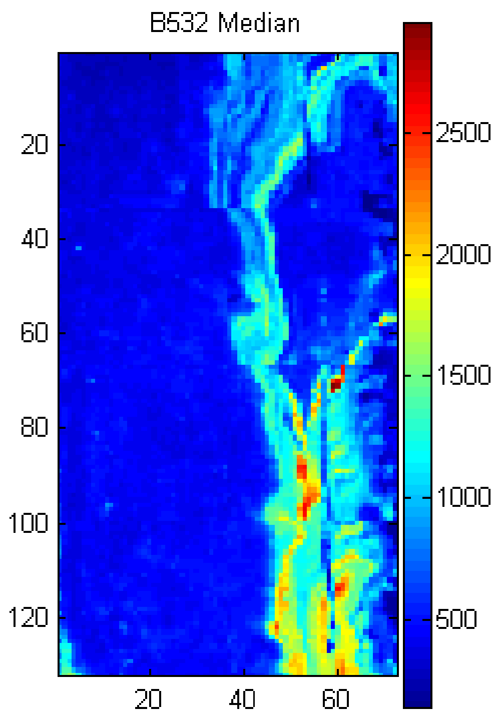
9 Extract that column from the field `Data`.

```
b532Data = wt.Data(:,b532MedCol);
```

10 Use the field Indices to index into the Data.

```
figure
subplot(1,2,1);
imagesc(b532Data(wt.Indices))
axis image
colorbar
title('B532 Median')
```

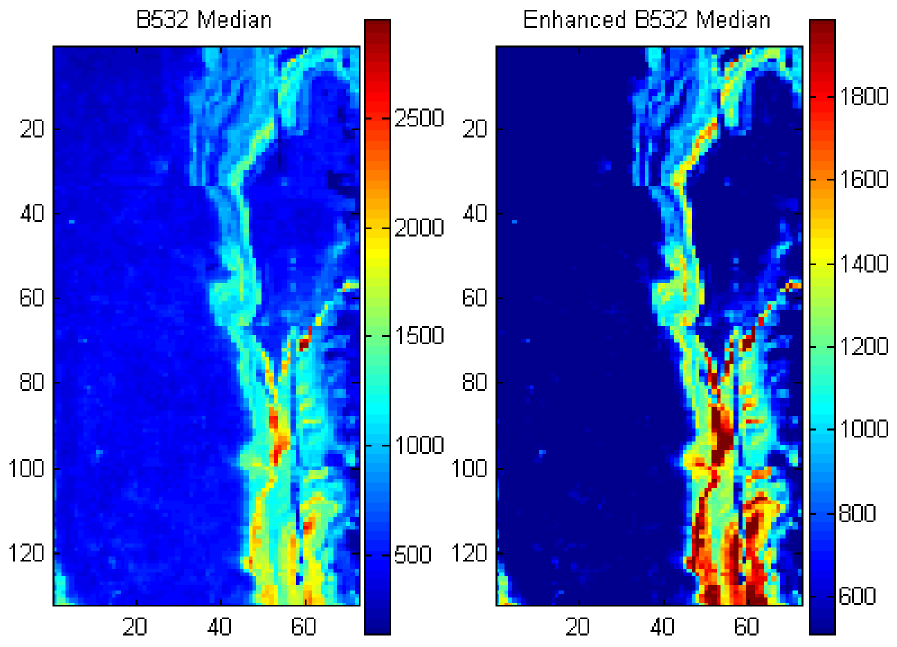
MATLAB plots the image.



11 Bound the intensities of the background plot to give more contrast in the image.

```
maskedData = b532Data;
maskedData(b532Data<500) = 500;
maskedData(b532Data>2000) = 2000;

subplot(1,2,2);
imagesc(maskedData(wt.Indices))
axis image
colorbar
title('Enhanced B532 Median')
```

MATLAB plots the images.



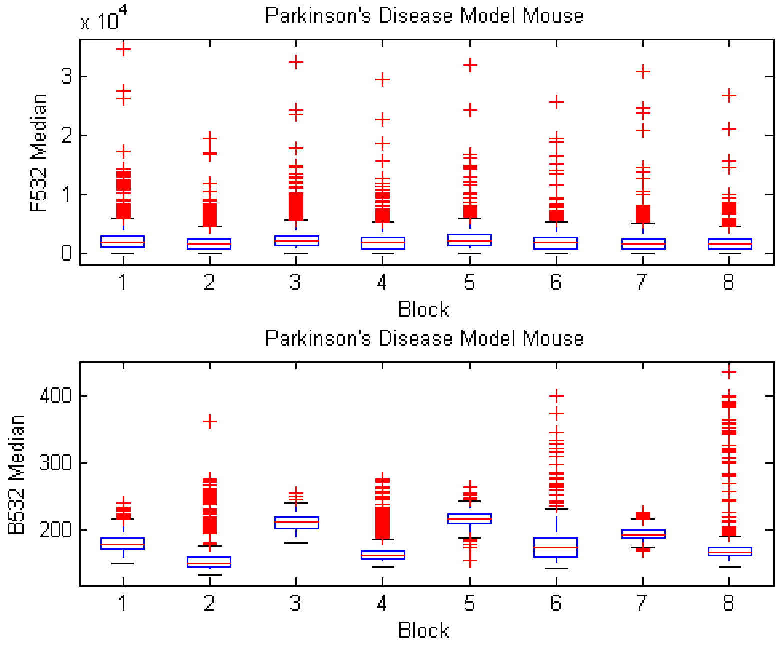
Statistics of the Microarrays

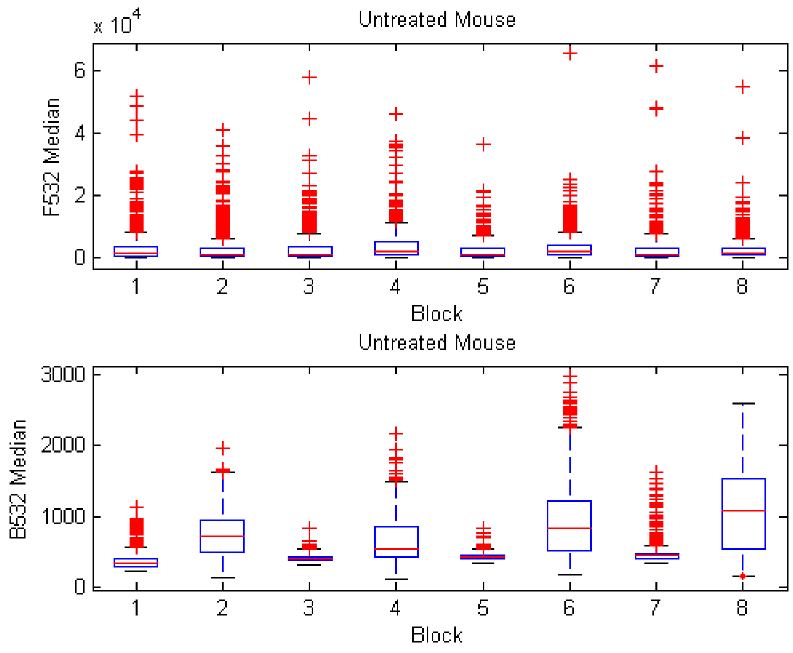
You can use the function maboxplot to look at the distribution of data in each of the blocks.

1 In the **MATLAB Command Window**, type

```
figure
subplot(2,1,1)
maboxplot(pd,'F532 Median','title','Parkinson''s Disease Model Mouse')
subplot(2,1,2)
maboxplot(pd,'B532 Median','title','Parkinson''s Disease Model Mouse')
figure
subplot(2,1,1)
maboxplot(wt,'F532 Median','title','Untreated Mouse')
subplot(2,1,2)
maboxplot(wt,'B532 Median','title','Untreated Mouse')
```

MATLAB plots the images.





2 Compare the plots.

From the box plots you can clearly see the spatial effects in the background intensities. Blocks numbers 1, 3, 5, and 7 are on the left side of the arrays, and numbers 2, 4, 6, and 8 are on the right side. The data must be normalized to remove this spatial bias.

Scatter Plots of Microarray Data

There are two columns in the microarray data structure labeled '`F635 Median - B635`' and '`F532 Median - B532`'. These columns are the differences between the median foreground and the median background for the 635 nm channel and 532 nm channel respectively. These give a measure of the actual expression levels, although since the data must first be normalized to remove spatial bias in the background, you should be careful about using these values without further normalization. However, in this example no normalization is performed.

1 Rather than working with data in a larger structure, it is often easier to extract the column numbers and data into separate variables.

```
cy5DataCol = find(strcmp(wt.ColumnNames,'F635 Median - B635'))
cy3DataCol = find(strcmp(wt.ColumnNames,'F532 Median - B532'))
cy5Data = pd.Data(:,cy5DataCol);
cy3Data = pd.Data(:,cy3DataCol);
```

MATLAB displays

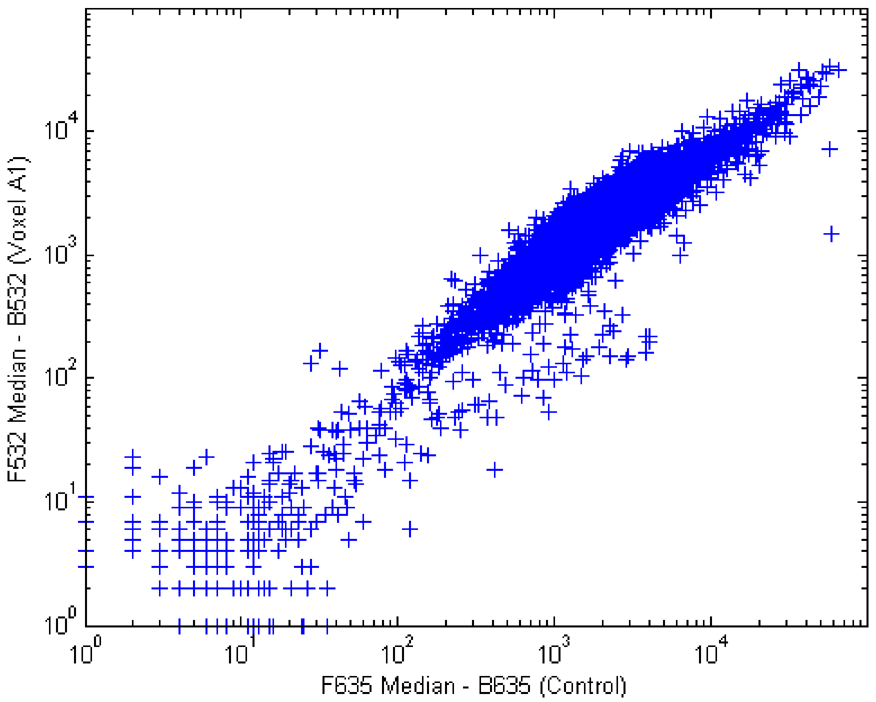
```
cy5DataCol =
    34

cy3DataCol =
    35
```

2 A simple way to compare the two channels is with a loglog plot. The function `maloglog` is used to do this. Points that are above the diagonal in this plot correspond to genes that have higher expression levels in the A1 voxel than in the brain as a whole.

```
figure
maloglog(cy5Data,cy3Data)
xlabel('F635 Median - B635 (Control)');
ylabel('F532 Median - B532 (Voxel A1)');
```

MATLAB displays the following messages and plots the images.

```
Warning: Zero values are ignored
(Type "warning off Bioinfo:MaloglogZeroValues" to suppress
 this warning.)
Warning: Negative values are ignored.
(Type "warning off Bioinfo:MaloglogNegativeValues" to suppress
 this warning.)
```



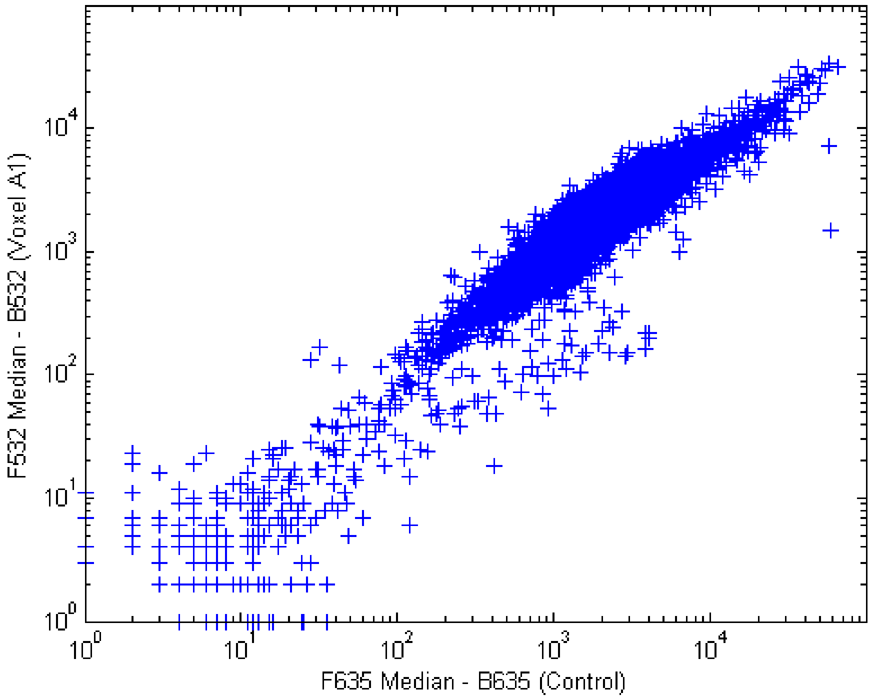
Notice that this function gives some warnings about negative and zero elements. This is because some of the values in the 'F635 Median - B635' and 'F532 Median - B532' columns are zero or even less than zero. Spots where this happened might be bad spots or spots that failed to hybridize. Points with positive, but very small, differences between foreground and background should also be considered to be bad spots.

3 Disable the display of warnings by using the warning command. Although warnings can be distracting, it is good practice to investigate why the warnings occurred rather than simply to ignore them. There might be some systematic reason why they are bad.

```
warnState = warning;          % First save the current warning
                                state.
                              % Now turn off the two warnings.
warning('off','Bioinfo:MaloglogZeroValues');
warning('off','Bioinfo:MaloglogNegativeValues');
figure
```

```
maloglog(cy5Data,cy3Data)      % Create the loglog plot
warning(warnState);            % Reset the warning state.
xlabel('F635 Median - B635 (Control)');
ylabel('F532 Median - B532 (Voxel A1)');
```

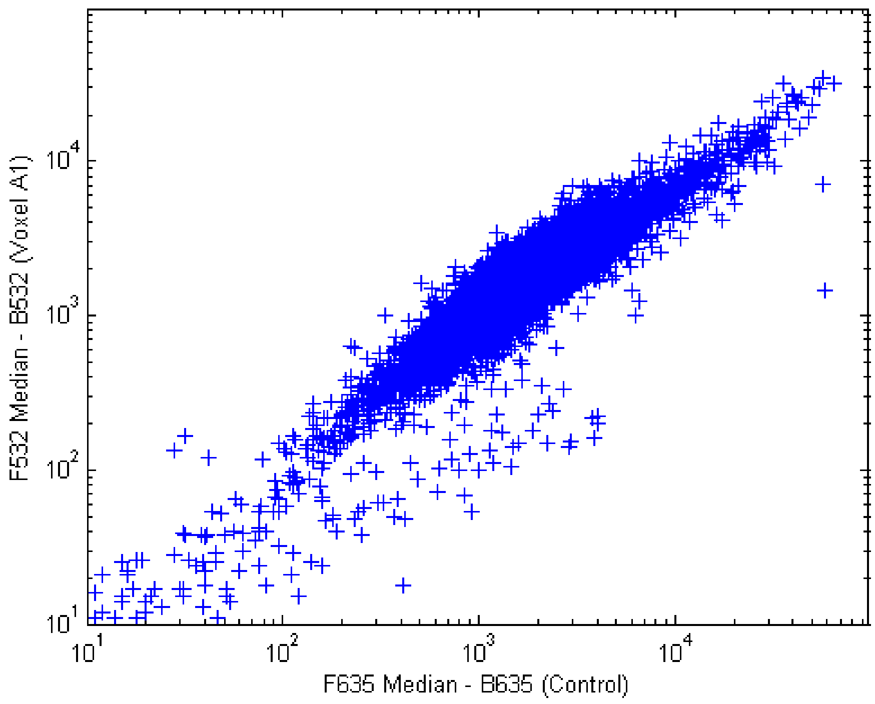
MATLAB plots the image.



4 An alternative to simply ignoring or disabling the warnings is to remove the bad spots from the data set. You can do this by finding points where either the red or green channel has values less than or equal to a threshold value. For example, use a threshold value of 10.

```
threshold = 10;
badPoints = (cy5Data <= threshold) | (cy3Data <= threshold);
```

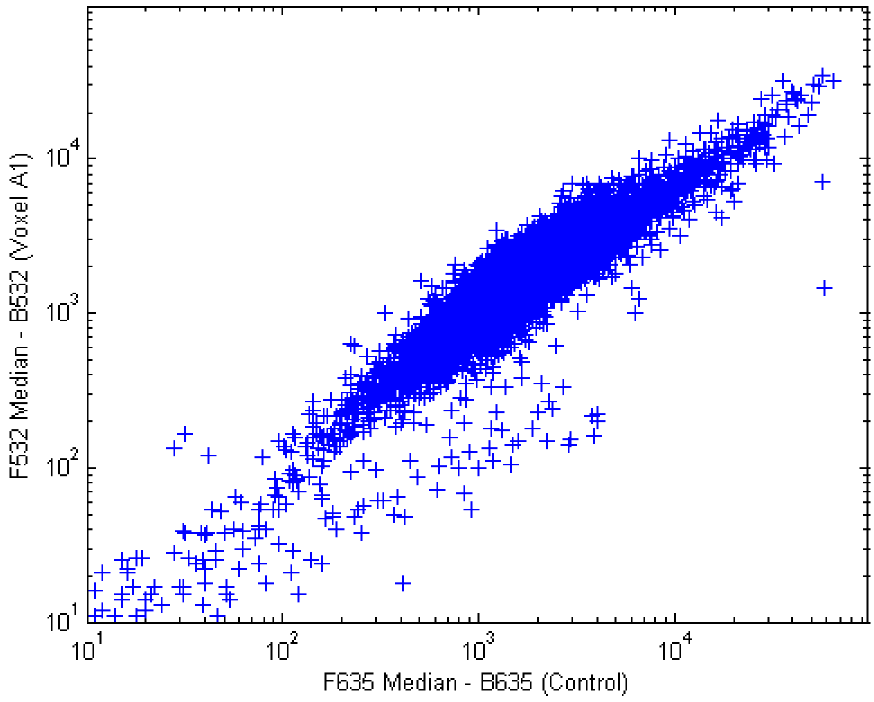
MATLAB plots the image.



5 You can then remove these points and redraw the loglog plot.

```
cy5Data(badPoints) = []; cy3Data(badPoints) = [];
figure
maloglog(cy5Data,cy3Data)
xlabel('F635 Median - B635 (Control)');
ylabel('F532 Median - B532 (Voxel A1)');
```

MATLAB plots the image.

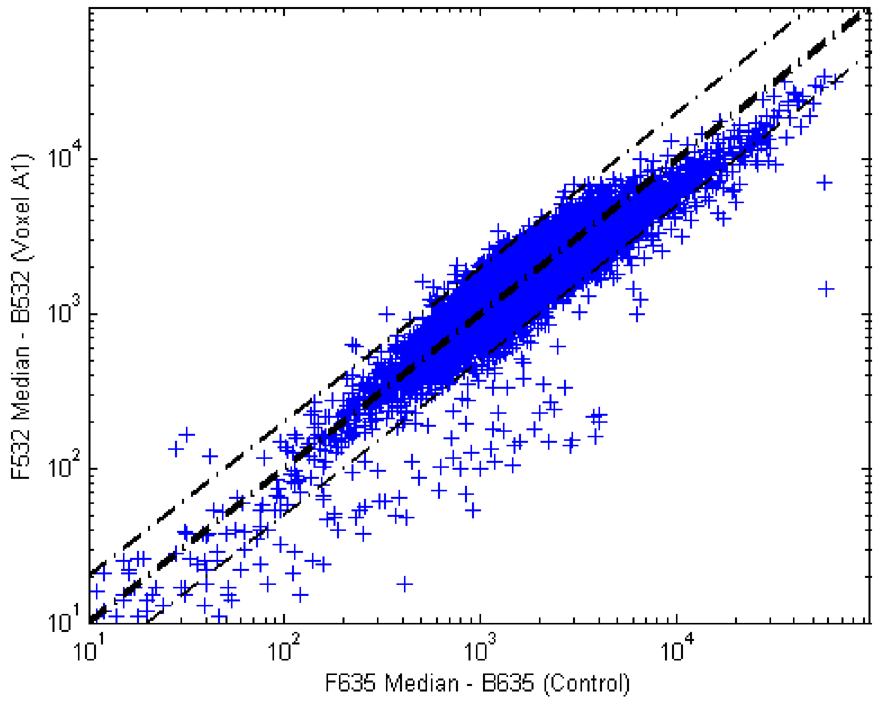


This plot shows the distribution of points but does not give any indication about which genes correspond to which points.

6 Add gene labels to the plot. Because some of the data points have been removed, the corresponding gene IDs must also be removed from the data set before you can use them. The simplest way to do that is `wt.IDs(~badPoints)`.

```
maloglog(cy5Data,cy3Data,'labels',wt.IDs(~badPoints),
         'factorlines',2)
xlabel('F635 Median - B635 (Control)');
ylabel('F532 Median - B532 (Voxel A1)');
```

MATLAB plots the image.



7 Try using the mouse to click some of the outlier points.

You will see the gene ID associated with the point. Most of the outliers are below the `y = x` line. In fact, most of the points are below this line. Ideally the points should be evenly distributed on either side of this line.

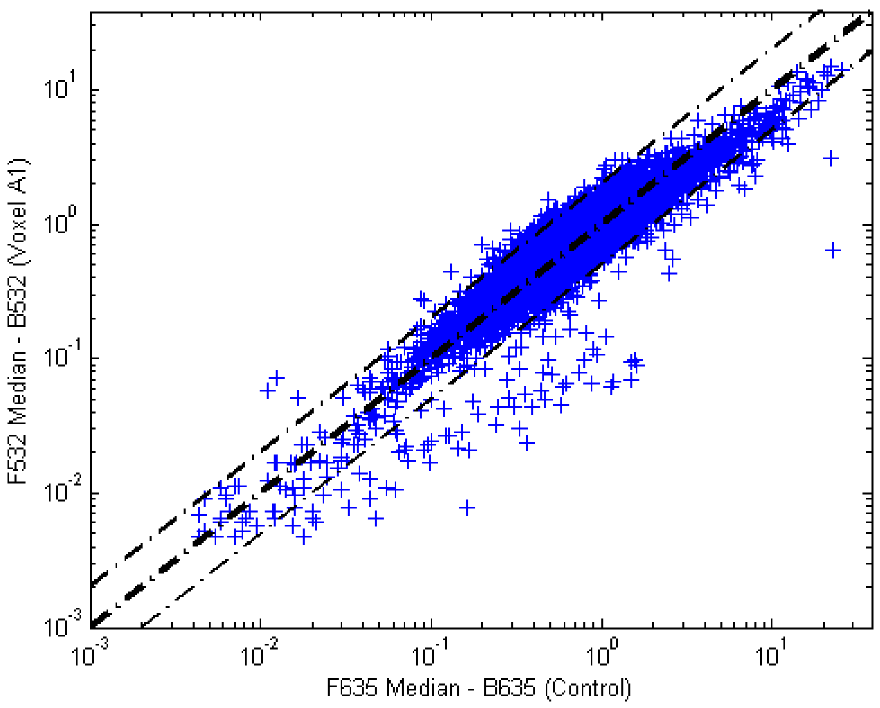
8 Normalize the points to evenly distribute them on either side of the line. Use the function `mameannorm` to perform global mean normalization.

```
normcy5 = mameannorm(cy5Data);
normcy3 = mameannorm(cy3Data);
```

If you plot the normalized data you will see that the points are more evenly distributed about the `y = x` line.

```
figure
maloglog(normcy5,normcy3,'labels',wt.IDs(~badPoints),
        'factorlines',2)
xlabel('F635 Median - B635 (Control)');
ylabel('F532 Median - B532 (Voxel A1)');
```

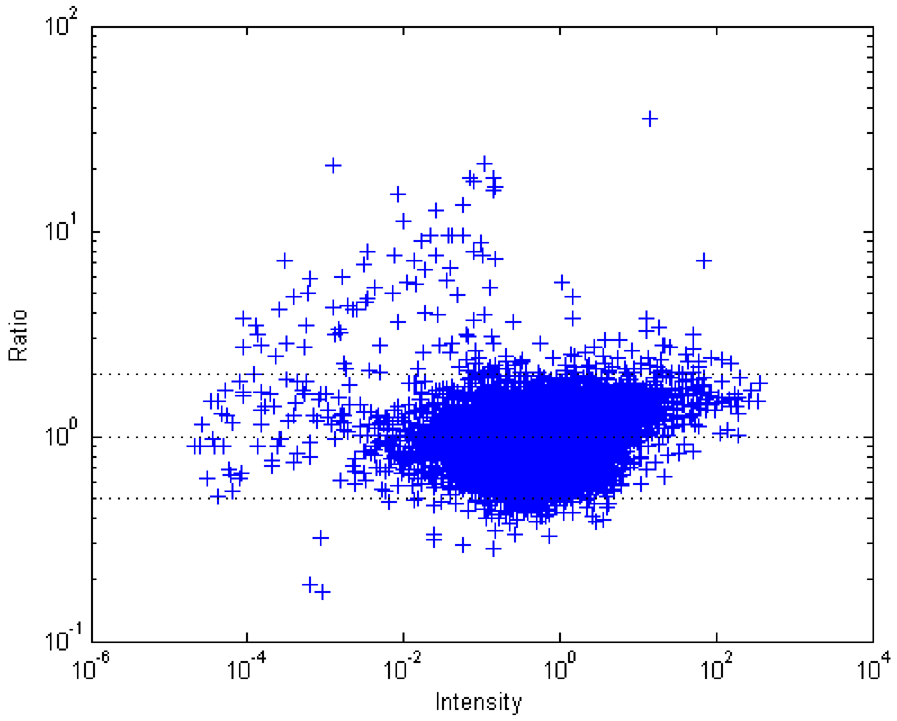
MATLAB plots the image.



9 The function `mairplot` is used to create an Intensity vs. Ratio plot for the normalized data. This function works in the same way as the function `maloglog`.

```
figure
mairplot(normcy5,normcy3,'labels',wt.IDs(~badPoints),
        'factorlines',2)
```

MATLAB plots the image.



10 You can click the points in this plot to see the name of the gene associated with the plot.

Example: Analyzing Gene Expression Profiles

This example demonstrates a number of ways to look for patterns in gene expression profiles.

Overview of the Yeast Example

The microarray data for this example is from DeRisi, JL, Iyer, VR, and Brown, PO.; "Exploring the metabolic and genetic control of gene expression on a genomic scale"; Science, 1997, Oct 24;278(5338):680-6, PMID: 9381177.

The authors 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. The full data set can be downloaded from the Gene Expression Omnibus Web site at

```
http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE28
```

Exploring the Data Set

The data for this procedure is available in the MAT-file `yeastdata.mat`. This file contains the VALUE data or LOG_RAT2N_MEAN, or log2 of ratio of CH2DN_MEAN and CH1DN_MEAN from the seven time steps in the experiment, the names of the genes, and an array of the times at which the expression levels were measured.

1 Load data into MATLAB.

```
load yeastdata.mat
```

2 Get the size of the data by typing

```
numel(genes)
```

MATLAB displays the number of genes in the data set. The MATLAB variable `genes` is a cell array of the gene names.

```
ans =
        6400
```

3 Access the entries using MATLAB cell array indexing.

```
genes{15}
```

MATLAB displays the 15th row of the variable `yeastvalues`, which contains expression levels for the open reading frame (ORF) YAL054C.

```
ans =
  YAL054C
```

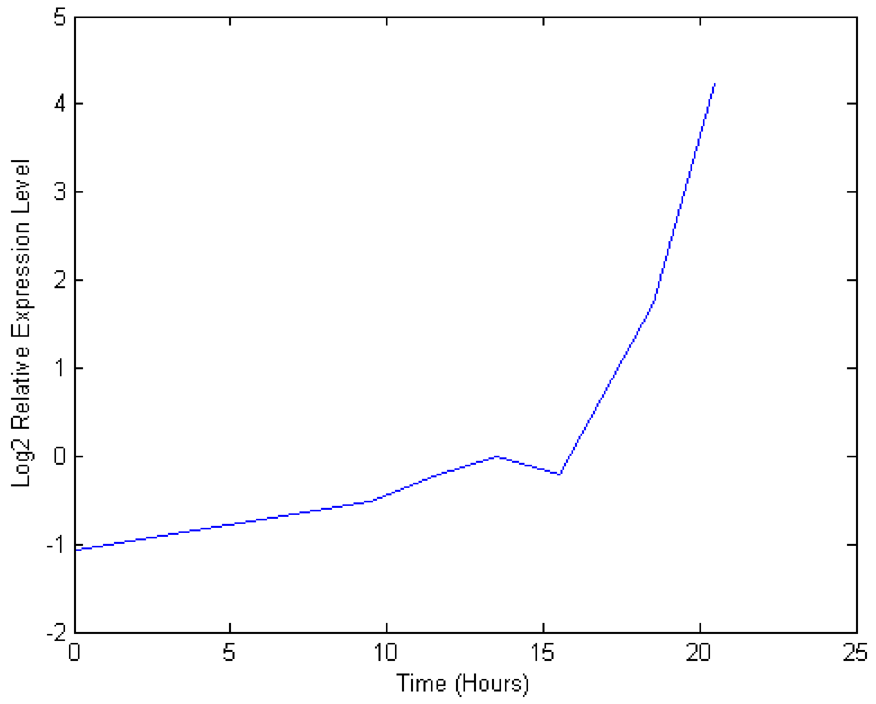
4 Use the function `web` to access information about this ORF in the Saccharomyces Genome Database (SGD).

```
url = sprintf(...
        'http://genome-www4.stanford.edu/cgi-bin/SGD/
         locus.pl?locus=%s',...
        genes{15});
web(url);
```

5 A simple plot can be used to show the expression profile for this ORF.

```
plot(times, yeastvalues(15,:))
xlabel('Time (Hours)');
ylabel('Log2 Relative Expression Level');
```

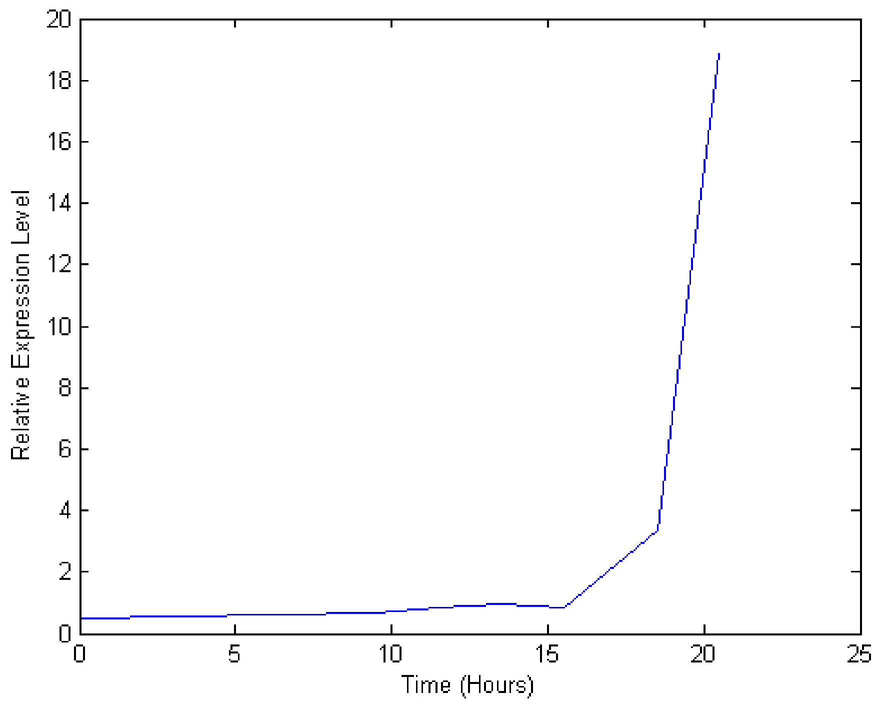
MATLAB plots the figure. The values are log2 ratios.



6 Plot the actual values.

```
plot(times, 2.^yeastvalues(15,:))
xlabel('Time (Hours)');
ylabel('Relative Expression Level');
```

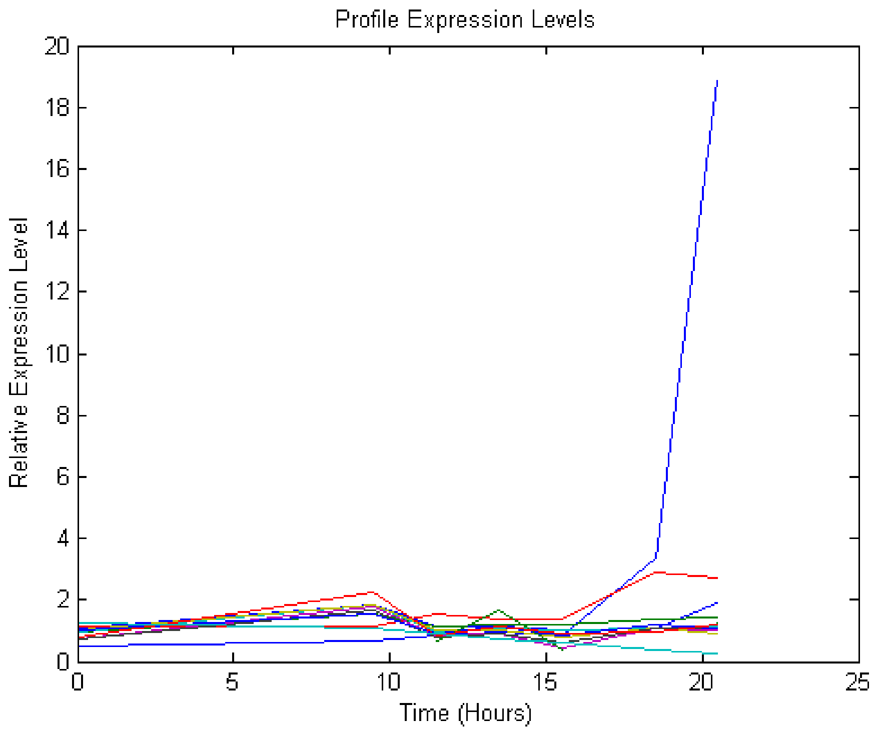
MATLAB plots the figure. The gene associated with this ORF, ACS1, appears to be strongly up-regulated during the diauxic shift.



7 Compare other genes by plotting multiple lines on the same figure.

```
hold on
plot(times, 2.^yeastvalues(16:26,:)')
xlabel('Time (Hours)');
ylabel('Relative Expression Level');
title('Profile Expression Levels');
```

MATLAB plots the image.



Filtering Genes

The data set is quite large and a lot of the information corresponds to genes that do not show any interesting changes during the experiment. To make it easier to find the interesting genes, reduce the size of the data set by removing genes with expression profiles that do not show anything of interest. There are 6400 expression profiles. You can use a number of techniques to reduce the number of expression profiles to some subset that contains the most significant genes.

1 If you look through the gene list you will see several spots marked as `'EMPTY'`. These are empty spots on the array, and while they might have data associated with them, for the purposes of this example, you can consider these points to be noise. These points can be found using the strcmp function and removed from the data set with indexing commands..

```
emptySpots = strcmp('EMPTY',genes);
yeastvalues(emptySpots,:) = [];
genes(emptySpots) = [];
numel(genes)
```

MATLAB displays

```
ans =
        6314
```

In the `yeastvalues` data you will also see several places where the expression level is marked as `NaN`. This indicates that no data was collected for this spot at the particular time step. One approach to dealing with these missing values would be to impute them using the mean or median of data for the particular gene over time. This example uses a less rigorous approach of simply throwing away the data for any genes where one or more expression levels were not measured.

2 Use function `isnan` to identify the genes with missing data and then use indexing commands to remove the genes.

```
nanIndices = any(isnan(yeastvalues),2);
yeastvalues(nanIndices,:) = [];
genes(nanIndices) = [];
numel(genes)
```

MATLAB displays

```
ans =
        6276
```

If you were to plot the expression profiles of all the remaining profiles, you would see that most profiles are flat and not significantly different from the others. This flat data is obviously of use as it indicates that the genes associated with these profiles are not significantly affected by the diauxic shift. However, in this example, you are interested in the genes with large changes in expression accompanying the diauxic shift. You can use filtering functions in the Bioinformatics Toolbox to remove genes with various types of profiles that do not provide useful information about genes affected by the metabolic change.

3 Use the function genevarfilter to filter out genes with small variance over time. The function returns a logical array of the same size as the variable genes with ones corresponding to rows of yeastvalues with variance greater than the 10th percentile and zeros corresponding to those below the threshold.

```
mask = genevarfilter(yeastvalues);
% Use the mask as an index into the values to remove the
% filtered genes.
yeastvalues = yeastvalues(mask,:);
genes = genes(mask);
numel(genes)
```

MATLAB displays

```
ans =
        5648
```

4 The function genelowvalfilter removes genes that have very low absolute expression values. Note that the gene filter functions can also automatically calculate the filtered data and names.

```
[mask, yeastvalues, genes] = genelowvalfilter(yeastvalues,genes,
                                               'absval',log2(4));
numel(genes)
```

MATLAB displays

```
ans =
   423
```

5 Use the function geneentropyfilter to remove genes whose profiles have low entropy:

```
[mask, yeastvalues, genes] = geneentropyfilter(yeastvalues,genes,...
                                                'prctile',15);
numel(genes)
```

MATLAB displays

```
ans =  310
```

Clustering Genes

Now that you have a manageable list of genes, you can look for relationships between the profiles using some different clustering techniques from the Statistics Toolbox.

1 For hierarchical clustering, the function `pdist` calculates the pairwise distances between profiles, and the function `linkage` creates the hierarchical cluster tree.

```
corrDist = pdist(yeastvalues, 'corr');
clusterTree = linkage(corrDist, 'average');
```

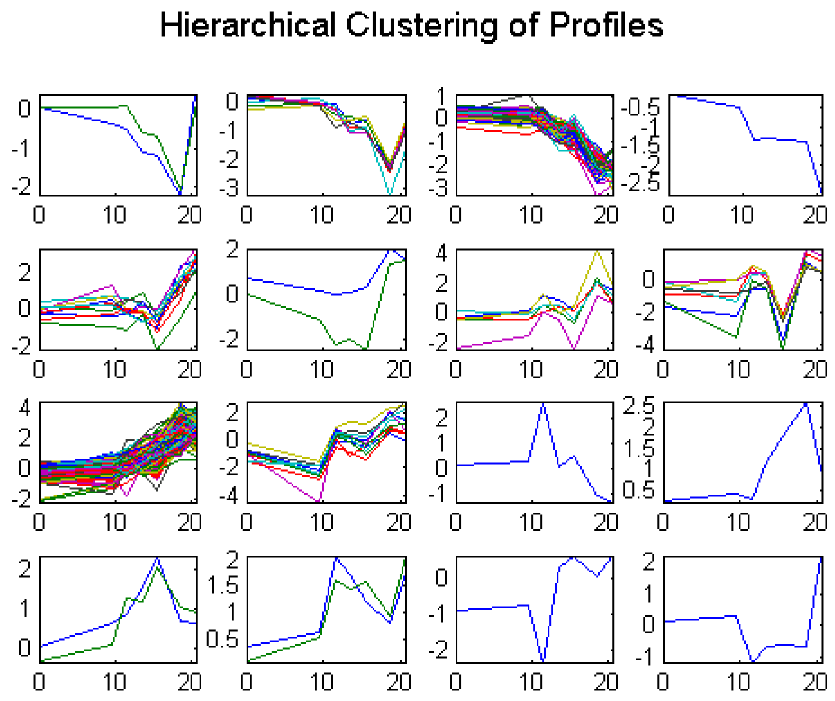
2 The function `cluster` calculates the clusters based on either a cutoff distance or a maximum number of clusters. In this case, the `'maxclust'` option is used to identify `16` distinct clusters.

```
clusters = cluster(clusterTree, 'maxclust', 16);
```

3 The profiles of the genes in these clusters can be plotted together using a simple loop and the function `subplot`.

```
figure
for c = 1:16
    subplot(4,4,c);
    plot(times,yeastvalues((clusters == c),:)');
    axis tight
end
suptitle('Hierarchical Clustering of Profiles');
```

MATLAB plots the images.

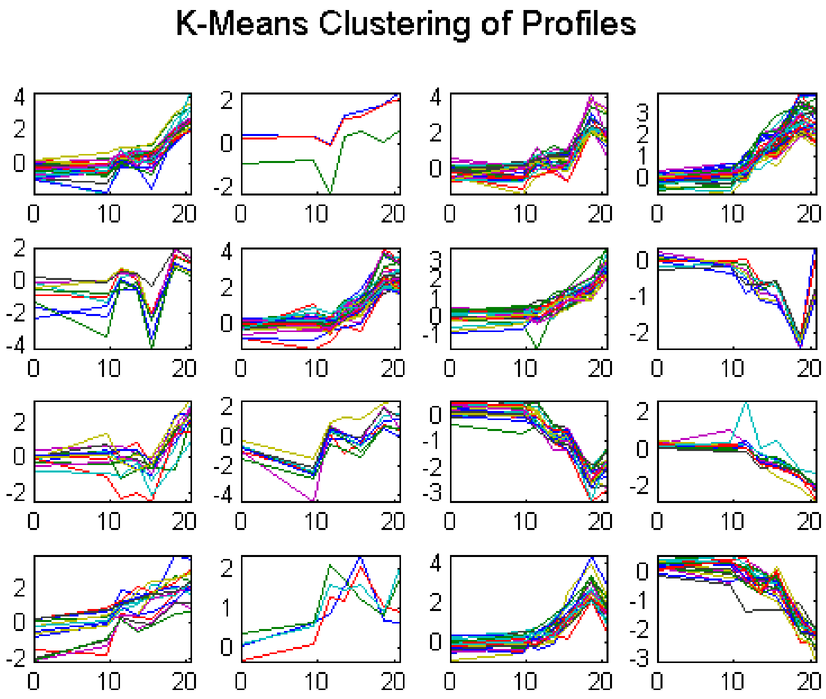


4 The Statistics Toolbox also has a K-means clustering function. Again, sixteen clusters are found, but because the algorithm is different these are not necessarily the same clusters as those found by hierarchical clustering.

```
[cidx, ctrs] = kmeans(yeastvalues, 16,
                      'dist','corr',
                      'rep',5,
                      'disp','final');
figure
for c = 1:16
    subplot(4,4,c);
    plot(times,yeastvalues((cidx == c),:)');
    axis tight
end
suptitle('K-Means Clustering of Profiles');
```

MATLAB displays

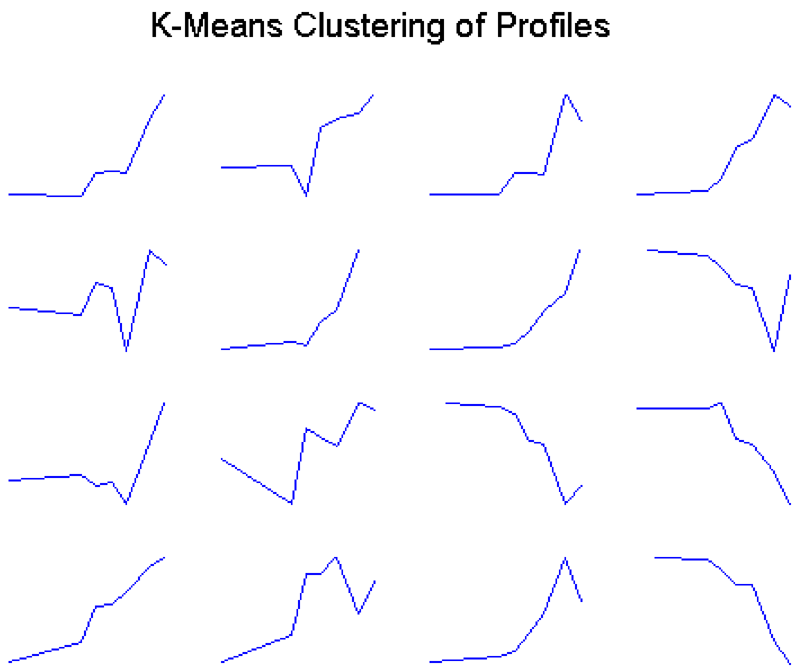
```
13 iterations, total sum of distances = 11.4042
14 iterations, total sum of distances = 8.62674
26 iterations, total sum of distances = 8.86066
22 iterations, total sum of distances = 9.77676
26 iterations, total sum of distances = 9.01035
```



5 Instead of plotting all of the profiles, you can plot just the centroids.

```
figure
for c = 1:16
    subplot(4,4,c);
    plot(times,ctrs(c,:)');
    axis tight
    axis off    % turn off the axis
end
suptitle('K-Means Clustering of Profiles');
```

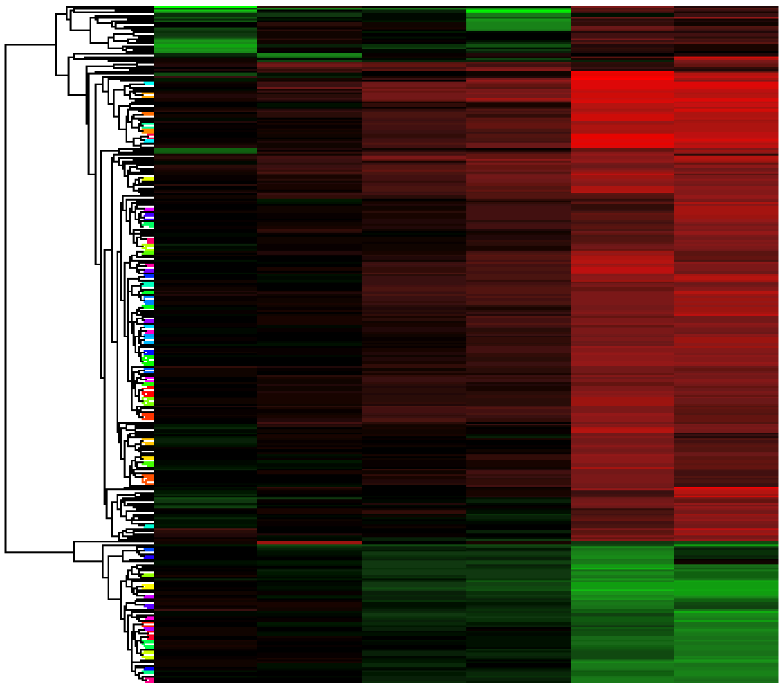
MATLAB plots the figure.



6 You can use the function `clustergram` to create a heat map and dendrogram from the output of the hierarchical clustering.

```
figure
clustergram(yeastvalues(:,2:end),'RowLabels',genes,...
                                  'ColumnLabels',times(2:end))
```

MATLAB plots the figure.



Principal Component Analysis

Principal-component analysis(PCA) is a useful technique you can use to reduce the dimensionality of large data sets, such as those from microarray analysis. PCA can also be used to find signals in noisy data.

1 You can use the The function `princomp` in the Statistics Toolbox to calculate the principal components of a data set.

```
[pc, zscores, pcvars] = princomp(yeastvalues)
```

MATLAB displays

```
pc =

  Columns 1 through 4
  -0.0245   -0.3033   -0.1710   -0.2831
   0.0186   -0.5309   -0.3843   -0.5419
   0.0713   -0.1970    0.2493    0.4042
   0.2254   -0.2941    0.1667    0.1705
   0.2950   -0.6422    0.1415    0.3358
   0.6596    0.1788    0.5155   -0.5032
   0.6490    0.2377   -0.6689    0.2601

 Columns 5 through 7

  -0.1155    0.4034    0.7887
  -0.2384   -0.2903   -0.3679
  -0.7452   -0.3657    0.2035
  -0.2385    0.7520   -0.4283
   0.5592   -0.2110    0.1032
  -0.0194   -0.0961    0.0667
  -0.0673   -0.0039    0.0521
```

2 You can use the function `cumsum` to see the cumulative sum of the variances.

```
cumsum(pcvars./sum(pcvars) * 100)
```

MATLAB displays

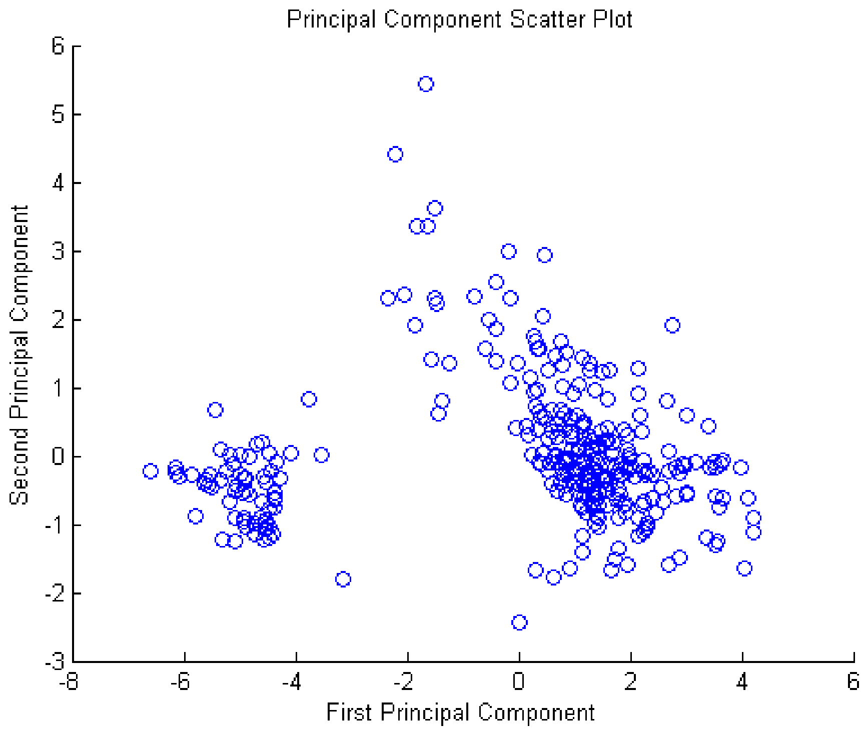
```
ans =
   78.3719
   89.2140
   93.4357
   96.0831
   98.3283
   99.3203
  100.0000
```

This shows that almost 90% of the variance is accounted for by the first two principal components.

3 A scatter plot of the scores of the first two principal components shows that there are two distinct regions. This is not unexpected, because the filtering process removed many of the genes with low variance or low information. These genes would have appeared in the middle of the scatter plot.

```
figure
scatter(zscores(:,1),zscores(:,2));
xlabel('First Principal Component');
ylabel('Second Principal Component');
title('Principal Component Scatter Plot');
```

MATLAB plots the figure.



4 The function gname from the Statistics Toolbox can be used to identify genes on a scatter plot. You can select as many points as you like on the scatter plot.

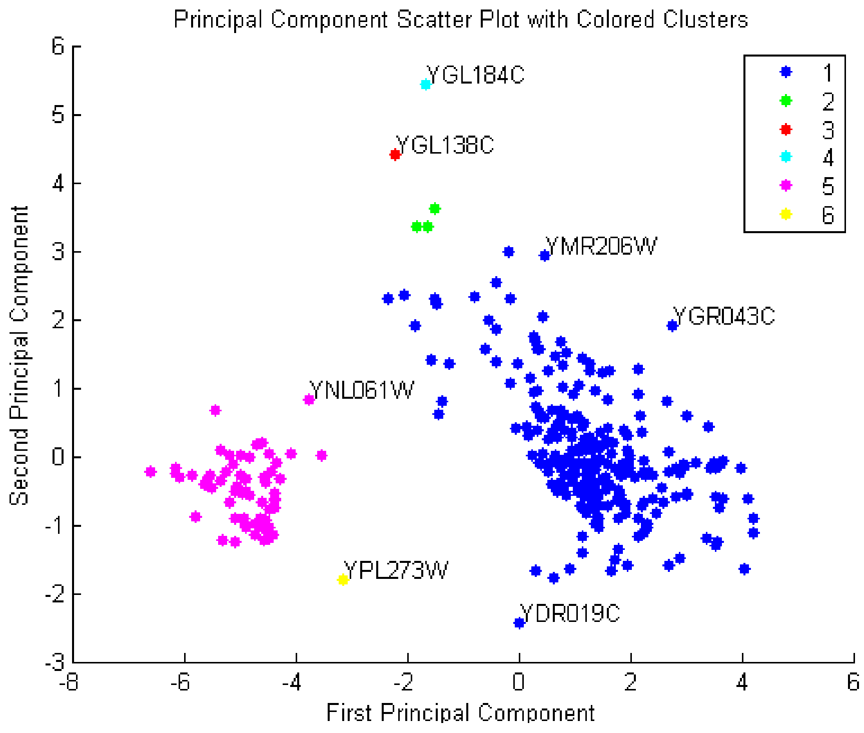
```
gname(genes);
```

When you have finished selecting points, press **Enter**.

5 An alternative way to create a scatter plot is with the function `gscatter` from the Statistics Toolbox. `gscatter` creates a grouped scatter plot where points from each group have a different color or marker. You can use `clusterdata`, or any other clustering function, to group the points.

```
figure
pcclusters = clusterdata(zscores(:,1:2),6);
gscatter(zscores(:,1),zscores(:,2),pcclusters)
xlabel('First Principal Component');
ylabel('Second Principal Component');
title('Principal Component Scatter Plot with Colored Clusters');
gname(genes)  % Press enter when you finish selecting genes.
```

MATLAB plots the figure.



Phylogenetic Analysis

Phylogenetic analysis is the process you use to determine the evolutionary relationships between organisms. The results of an analysis can be drawn in a hierarchical diagram called a cladogram or phylogram (phylogenetic tree). The branches in a tree are based on the hypothesized evolutionary relationships (phylogeny) between organisms. Each member in a branch, also known as a monophyletic group, is assumed to be descended from a common ancestor. Originally, phylogenetic trees were created using morphology, but now, determining evolutionary relationships includes matching patterns in nucleic acid and protein sequences.

Example: Building a Phylogenetic Tree

In this example, a phylogenetic tree is constructed from mitochondrial DNA (mtDNA) sequences for the family Hominidae. This family includes gorillas, chimpanzees, orangutans, and humans.

The following procedures demonstrate the phylogenetic analysis features in the Bioinformatics Toolbox. They are not intended to teach the process of phylogenetic analysis, but to show you how to use MathWorks products to create a phylogenetic tree from a set of nonaligned nucleotide sequences.

- "Overview for the Primate Example" on page 4-2 — Describes the biological background for this example.
- "Creating a Phylogenetic Tree for Five Species" on page 4-6 — Use the Jukes-Cantor method to calculate distances between sequences, and the Unweighted Pair Group Method Average (UPGMA) method for linking the tree nodes.
- "Creating a Phylogenetic Tree for Twelve Species" on page 4-8 — Add additional organisms to confirm the observed monophyletic groups.
- "Exploring the Phylogenetic Tree" on page 4-10 — Use the MATLAB command-line interface to programmatically determine characteristics in a phylogenetic tree.

For information on how to create a phylogenetic tree with multiply aligned sequences, see the function —`phytree`.

Overview for the Primate Example

The origin of modern humans is a heavily debated issue that scientists have recently tackled by using mitochondrial DNA (mtDNA) sequences. One hypothesis explains the limited genetic variation of human mtDNA in terms of a recent common genetic ancestry, implying that all modern population mtDNA originated from a single woman who lived in Africa less than 200,000 years ago.

Why use mitochondrial DNA sequences for phylogenetic study?

Mitochondrial DNA sequences, like the Y chromosome, do not recombine and are inherited from the maternal parent. This lack of recombination allows sequences to be traced through one genetic line and all polymorphisms assumed to be caused by mutations.

Mitochondrial DNA in mammals has a faster mutation rate than nuclear DNA sequences. This faster rate of mutation produces more variance between sequences and is an advantage when studying closely related species. The mitochondrial control region (Displacement or D-loop) is one of the fastest mutating sequence regions in animal DNA.

Neanderthal DNA

The ability to isolate mitochondrial DNA (mtDNA) from palaeontological samples has allowed genetic comparisons between extinct species and closely related nonextinct species. The reasons for isolating mtDNA instead of nuclear DNA in fossil samples have to do with the fact that

- mtDNA, because it is circular, is more stable and degrades slower then nuclear DNA.
- Each cell can contain a thousand copies of mtDNA and only a single copy of nuclear DNA.

While there is still controversy as to whether Neanderthals are direct ancestors of humans or evolved independently, the use of ancient genetic sequences in phylogenetic analysis adds an interesting dimension to the question of human ancestry.

References

Ovchinnikov, I., et al., 2000. "Molecular analysis of Neanderthal DNA from the northern Caucasus," Nature 404(6777), pp 490-493.

Sajantila, A., et al., 1995. "Genes and languages in Europe: an analysis of mitochondrial lineages," Genome Res. 5 (1), pp. 42-52 (1995).

Krings, M., et al., 1997. "Neanderthal DNA sequences and the origin of modern humans," Cell 90 (1), pp. 19-30.

Jensen-Seaman, M., and K. Kidd, 2001. "Mitochondrial DNA variation and biogeography of eastern gorillas," Mol. Ecol. 10(9), pp. 2241-2247.

Searching NCBI for Phylogenetic Data

The NCBI taxonomy Web site includes phylogenetic and taxonomic information from many sources. These sources include the published literature, Web databases, and taxonomy experts. And while the NCBI taxonomy database is not a phylogenetic or taxonomic authority, it can be useful as a gateway to the NCBI biological sequence databases.

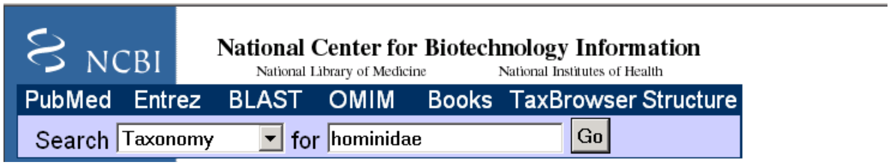
This procedure uses the family Hominidae (orangutans, chimpanzees, gorillas, and humans) as a taxonomy example for searching the NCBI Web site and locating mitochondrial D-loop sequences.

1 Use the MATLAB Help browser to search for data on the Web. In the MATLAB Command Window, type

```
web('http://www.ncbi.nlm.nih.gov')
```

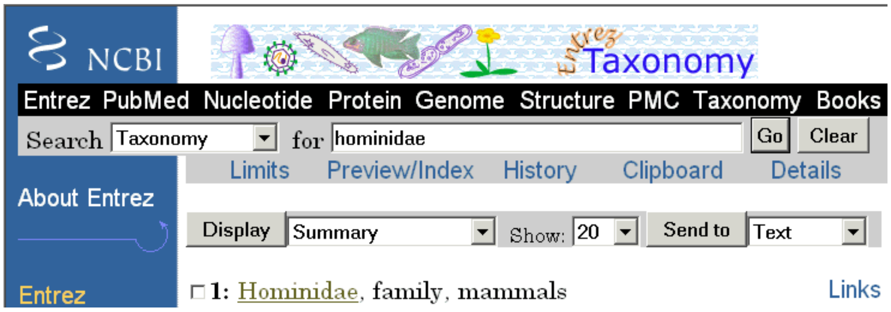
A separate browser window opens with the home page for the NCBI Web site.

2 Search the NCBI Web site for information. For example, to search for the human taxonomy, from the **Search** list, select `Taxonomy`, and in the **for** box, enter `hominidae`.


NCBI
National Center for Biotechnology Information
National Library of Medicine
National Institutes of Health
PubMed
Entrez
BLAST
OMIM
Books
TaxBrowser
Structure
Go

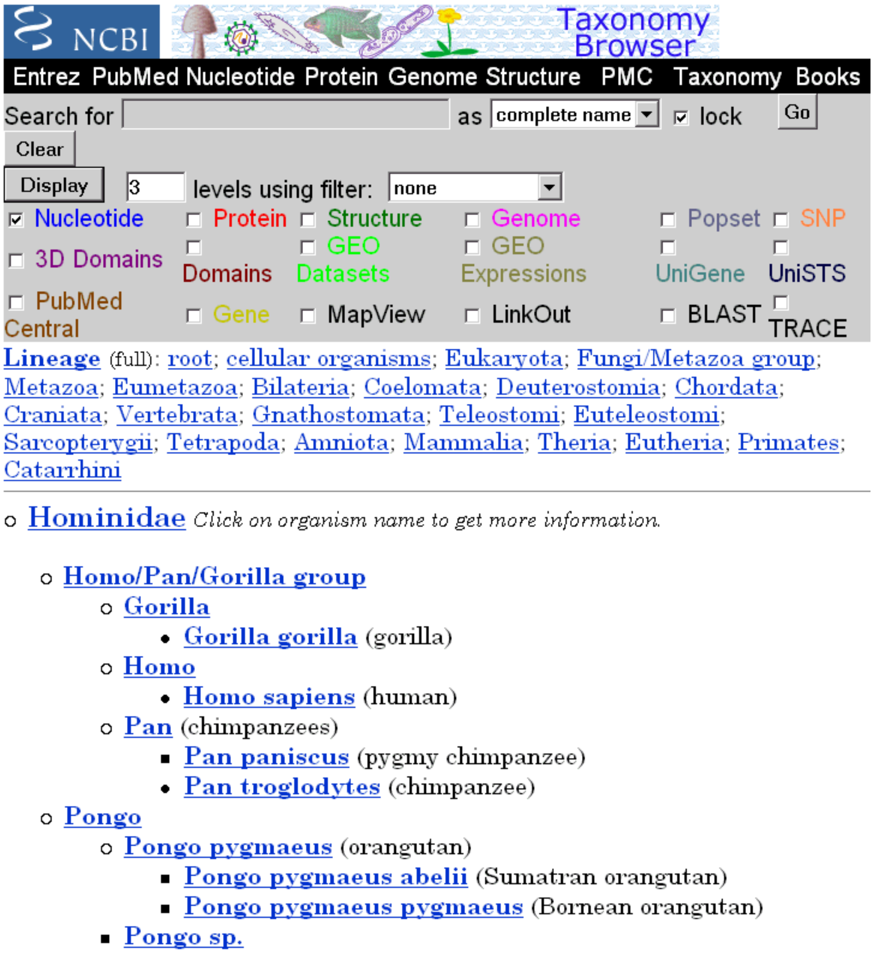
Search Taxonomy for hominidae

The NCBI Web search returns a list of links to relevant pages.


NCBI
Entrez Taxonomy
Entrez PubMed Nucleotide Protein Genome Structure PMC Taxonomy Books
Search Taxonomy for hominidae Go Clear
Limits Preview/Index History Clipboard Details
About Entrez
Display Summary Show: 20 Send to Text
Entrez
Links

☐ 1: Hominidae, family, mammals

3 Select the taxonomy link for the family Hominidae. A page with the taxonomy for the family is shown.



Lineage (full): root; cellular organisms; Eukaryota; Fungi/Metazoa group; Metazoa; Eumetazoa; Bilateria; Coelomata; Deuterostomia; Chordata; Craniata; Vertebrata; Gnathostomata; Teleostomi; Euteleostomi; Sarcopterygii; Tetrapoda; Amniota; Mammalia; Theria; Eutheria; Primates; Catarrhini

- Hominidae *Click on organism name to get more information.*
  - Homo/Pan/Gorilla group
    - Gorilla
      - Gorilla gorilla (gorilla)
    - Homo
      - Homo sapiens (human)
    - Pan (chimpanzees)
      - Pan paniscus (pygmy chimpanzee)
      - Pan troglodytes (chimpanzee)
  - Pongo
    - Pongo pygmaeus (orangutan)
      - Pongo pygmaeus abelii (Sumatran orangutan)
      - Pongo pygmaeus pygmaeus (Bornean orangutan)
    - Pongo sp.

Creating a Phylogenetic Tree for Five Species

Drawing a phylogenetic tree using sequence data is helpful when you are trying to visualize the evolutionary relationships between species. The sequences can be multiply aligned or a set of nonaligned sequences, you can select a method for calculating pairwise distances between sequences, and you can select a method for calculating the hierarchical clustering distances used to build a tree.

After locating the GenBank accession codes for the sequences you are interested in studying, you can create a phylogenetic tree with the data. For information on locating accession codes, see “Searching NCBI for Phylogenetic Data” on page 4-4.

1 Create a MATLAB structure with information about the sequences. This step uses the accession codes for the mitochondrial D-loop sequences isolated from different hominid species.

```
data = {'German_Neanderthal'      'AF011222';
        'Russian_Neanderthal'     'AF254446';
        'European_Human'          'X90314'  ;
        'Mountain_Gorilla_Rwanda' 'AF089820';
        'Chimp_Troglodytes'       'AF176766';
       };
```

2 Get sequence data from the GenBank database and copy into MATLAB.

```
for ind = 1:5
    seqs(ind).Header   = data{ind,1};
    seqs(ind).Sequence = getgenbank(data{ind,2},
                                    'sequenceonly', true);
end
```

3 Calculate pairwise distances and create a `phytree` object. For example, compute the pairwise distances using the Jukes-Cantor distance method and build a phylogenetic tree using the UPGMA linkage method. Since the sequences are not prealigned, `seqpdist` pairwise aligns them before computing the distances.

```
distances = seqpdist(seqs,'Method','Jukes-Cantor','Alphabet','DNA');
tree = seqlinkage(distances,'UPGMA',seqs)
```

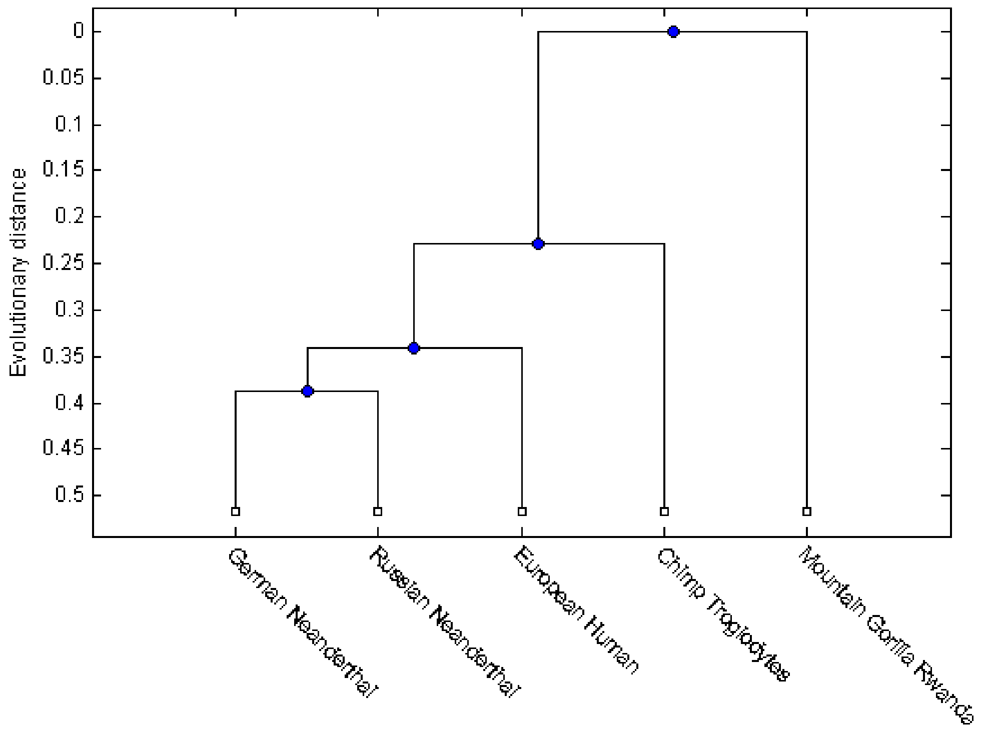
MATLAB displays information about the `phytree` object. The function `seqpdist` calculates the pairwise distances between pairs of sequences while the function `seqlinkage` uses the distances to build a hierarchical cluster tree. First, the most similar sequences are grouped together, and then sequences are added to the tree in decending order of similarity.

```
Phylogenetic tree object with 5 leaves (4 branches)
```

4 Draw a phylogenetic tree.

```
h = plot(tree,'orient','bottom');
ylabel('Evolutionary distance')
set(h.terminalNodeLabels,'Rotation',-45)
```

MATLAB draws a phylogenetic tree in a figure window. In the figure below, the hypothesized evolutionary relationships between the species. is shown by the location of species on the branches shows the The horizontal distances do not have any biological significance.



Creating a Phylogenetic Tree for Twelve Species

Plotting a simple phylogenetic tree for five species seems to indicate a number of monophyletic groups(see "Creating a Phylogenetic Tree for Five Species" on page 4-6). After a preliminary analysis with five species, you can add more species to your phylogenetic tree. Adding more species to the data set will help you to confirm the groups are valid.

1 Add more sequences to a MATLAB structure. For example, add mtDNA D-loop sequences for other hominid species.

```
data2 = {'Puti_Orangutan'          'AF451972';
         'Jari_Orangutan'          'AF451964';
         'Western_Lowland_Gorilla' 'AY079510';
         'Eastern_Lowland_Gorilla' 'AF050738';
         'Chimp_Schweinfurthii'    'AF176722';
         'Chimp_Vellerosus'        'AF315498';
         'Chimp_Verus'             'AF176731';
       };
```

2 Get additional sequence data from the GenBank database, and copy the data into the next indices of a MATALB structure.

```
for ind = 1:7
    seqs(ind+5).Header   = data2{ind,1};
    seqs(ind+5).Sequence = getgenbank(data2{ind,2},
                                  'sequenceonly', true);
end
```

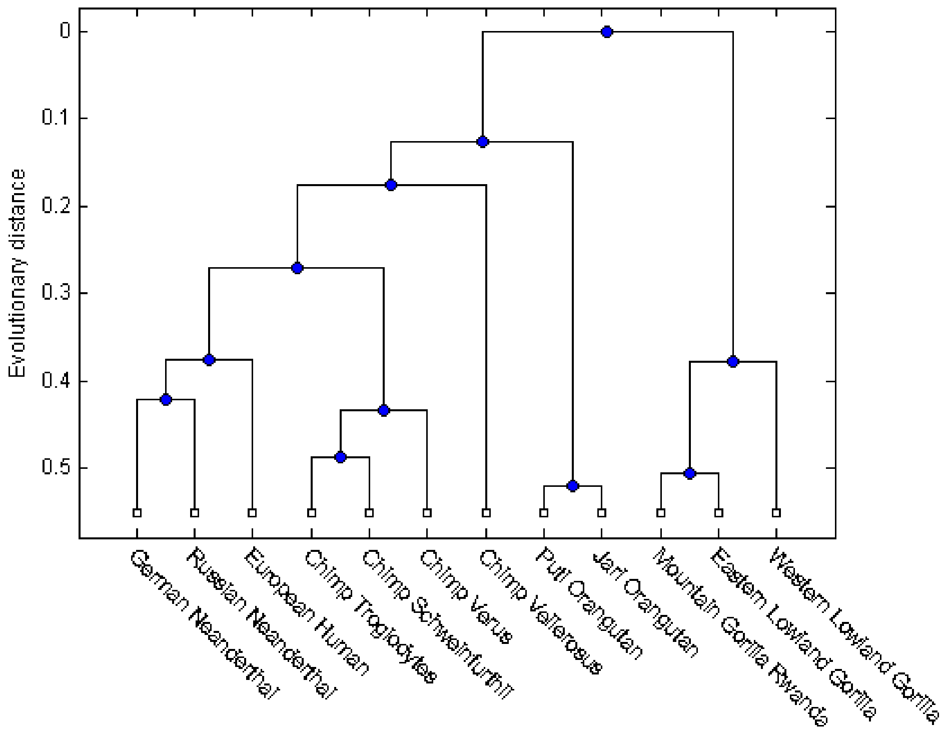
3 Calculate pairwise distances and the hierarchical linkage.

```
distances = seqpdist(seqs,'Method','Jukes-Cantor','Alpha','DNA');
tree = seqlinkage(distances,'UPGMA',seqs);
```

4 Draw a phylogenetic tree.

```
h = plot(tree,'orient','bottom');
ylabel('Evolutionary distance')
set(h.terminalNodeLabels,'Rotation',-45)
```

MATLAB draws a phylogenetic tree in a figure window. You can see four main clades for humans, gorillas, chimpanzee, and orangutans.



Exploring the Phylogenetic Tree

After you create a phylogenetic tree, you can explore the tree using the MATLAB command line or the `phytreetool` GUI. This procedure uses the tree created in "Creating a Phylogenetic Tree for Twelve Species" on page 4-8 as an example.

1 List the members of a tree.

```
names = get(tree,'LeafNames')
```

From the list, you can determine the indices for its members. For example, the European Human leaf is the third entry.

```
names =
    'German_Neanderthal'
    'Russian_Neanderthal'
    'European_Human'
    'Chimp_Troglodytes'
    'Chimp_Schweinfurthii'
    'Chimp_Verus'
    'Chimp_Vellerosus'
    'Puti_Orangutan'
    'Jari_Orangutan'
    'Mountain_Gorilla_Rwanda'
    'Eastern_Lowland_Gorilla'
    'Western_Lowland_Gorilla'
```

2 Find the closest species to a selected specie in a tree. For example, find the species closest to the European human.

```
[h_all,h_leaves] = select(tree,'reference',3,
                          'criteria','distance',
                          'threshold',0.6);
```

`h_all` is a list of indices for the nodes within a patristic distance of `0.6` to the European human leaf, while `h_leaves` is a list of indices for only the leaf nodes within the same patristic distance.

A patristic distance is the path length between species calculated from the hierarchical clustering distances. The path distance is not necessarily the biological distance.

3 List the names of the closest species.

```
subtree_names = names(h_leaves)
```

MATLAB prints a list of species with a patristic distance to the European human less than the specified distance. In this case, the patristic distance threshold is less than `0.6`.

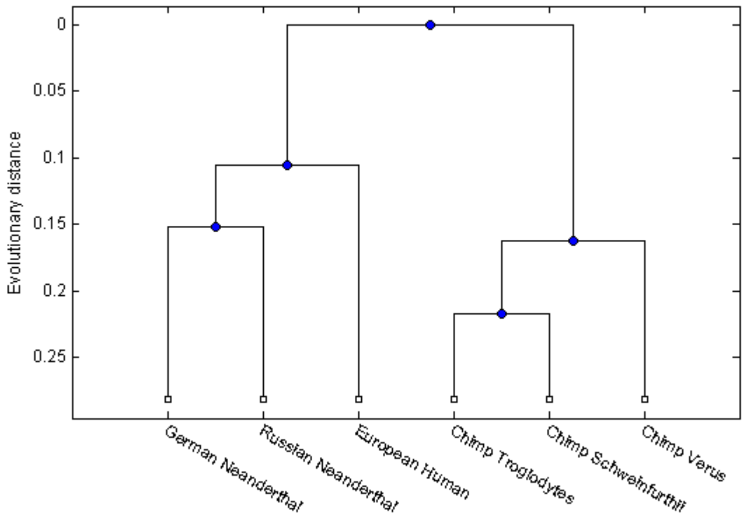
```
   subtree_names =

    'German_Neanderthal'
    'Russian_Neanderthal'
    'European_Human'
    'Chimp_Schweinfurthii'
    'Chimp_Verus'
    'Chimp_Troglodytes'
```

4 Extract a subtree from the whole tree by removing unwanted leaves. For example, prune the tree to species within 0.6 of the European human specie.

```
leaves_to_prune = ~h_leaves;
pruned_tree = prune(tree,leaves_to_prune)
h = plot(pruned_tree,'orient','bottom');
ylabel('Evolutionary distance')
set(h.terminalNodeLabels,'Rotation',-30)
```

MATLAB returns information about the new subtree and plots the pruned phylogenetic tree in a figure window.

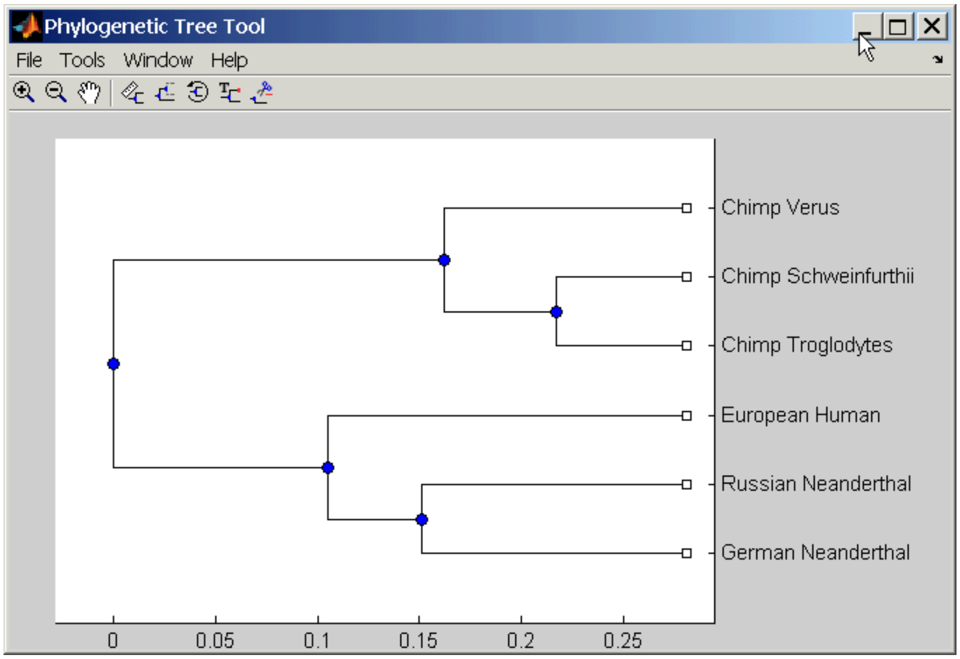
```
Phylogenetic tree object with 6 leaves (5 branches)
```



5 Explore, edit, and format a phylogenetic tree using an interactive GUI.

```
phytreetool(pruned_tree)
```

MATLAB opens the Phylogenetic Tree Tool window and draws the tree.



You can interactively change the appearance of the tree within the tool window. For information on using this GUI, see “Phylogenetic Tree Tool Reference” on page 4-14.

Phylogenetic Tree Tool Reference

The Phylogenetic Tree Tool is an interactive graphical user interface (GUI) that allows you to view, edit, format, and explore phylogenetic tree data. With this GUI you can prune, reorder, rename branches, and explore distances. You can also open or save Newick formatted files.

- "Opening the Phytreetool GUI" on page 4-14 — Draw a phylogenetic tree from data in a `phytree` object or a previously saved file.
- "File Menu" on page 4-16 — Open tree data from a Newick formatted file, copy data to a MATLAB figure window, another tool window, or the MATLAB workspace, and save tree data.
- "Tools Menu" on page 4-24 — Explore branch paths, rename and edit branch and leaf names, hide selected branches and leaves, and rotate branches.
- "Windows Menu" on page 4-32 — Switch to any open window.
- "Help Menu" on page 4-32 — Select quick links to the Bioinformatics Toolbox documentation for phylogenetic analysis functions, tutorials, and the `phytreetool` reference.

Opening the Phytreetool GUI

The Phylogenetic Tree Tool can read data from Newick and ClustalW tree formatted files.

This procedure uses the phylogenetic tree data stored in the file `pf00002.tree` as an example. The data was retrieved from the protein family (PFAM) Web database and saved to a file using the accession number `PF00002` and the function `gethmmtree`.

1 Create a `phytree` object. For example, to create a `phytree` object from tree data in the file `pf00002.tree`, type

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

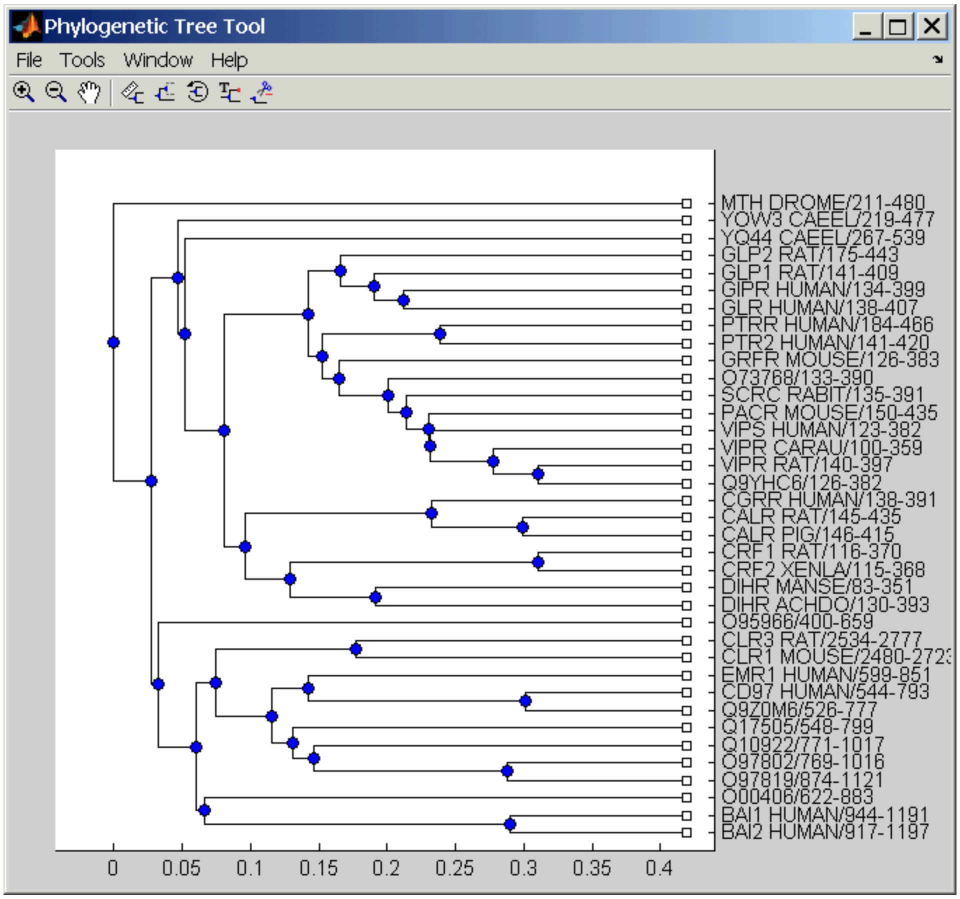
MATLAB creates a `phytree` object.

```
Phylogenetic tree object with 37 leaves (36 branches)
```

2 Open the Phylogenetic Tree Tool and draw a phylogenetic tree.

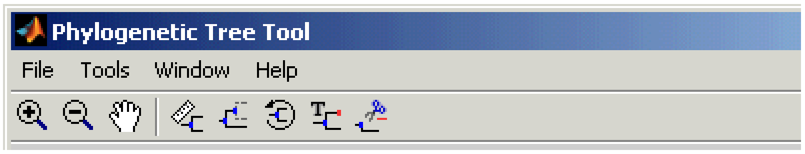
```
phytreetool(tr)
```

The Phylogenetic Tree Tool window opens.



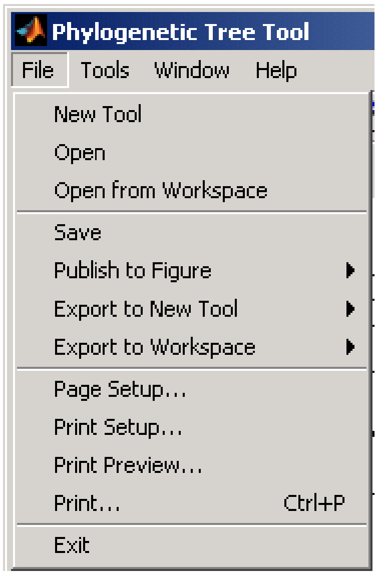
Alternatively, if you do not have to give the `phytreetool` function and argument, the **Select Phylogenetic Tree** dialog opens. Select a Newick formatted file and then click **Open**.

3 Select a command from the menu or toolbar.



File Menu

The **File** menu includes the standard commands for opening and closing a file, and it includes commands to use phytree object data from the MATLAB workspace.The File menu commands are shown below.



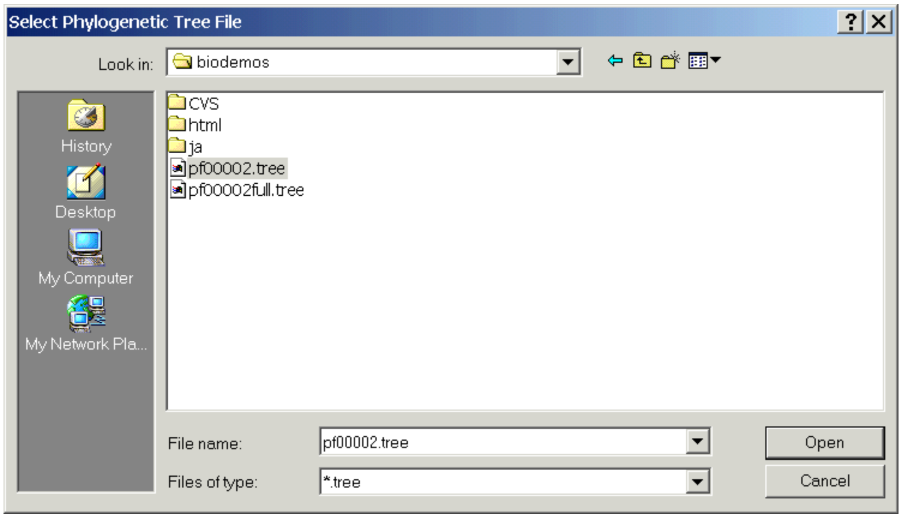
New Tool Command

Use the New Tool command to open tree data from a file into a second Phylogenetic Tree Tool window.

1 From the **File** menu, click **New Tool**.

   The **Select Phylogenetic Tree File** dialog opens.

2 Select a directory and select a file with the extension `.tree`, and then click **Open**. The Bioinformatics Toolbox uses the file extension `.tree` for Newick formatted files, but you can use any Newick formatted file with any extension.



MATLAB opens a second Phylogenetic Tree Tool window with tree data from the selected file.

Open Command

Use the **Open** command to read tree data from a Newick formatted file and display that data in a Phylogenetic Tree Tool.

1 From the **File** menu, click **Open**.

   The **Select Phylogenetic Tree File** dialog box opens.

2 Select a directory, select a Newick formatted file, and then click **Open**. The Bioinformatics Toolbox uses the file extension `.tree` for Newick formatted files, but you can use any Newick formatted file with any extension.

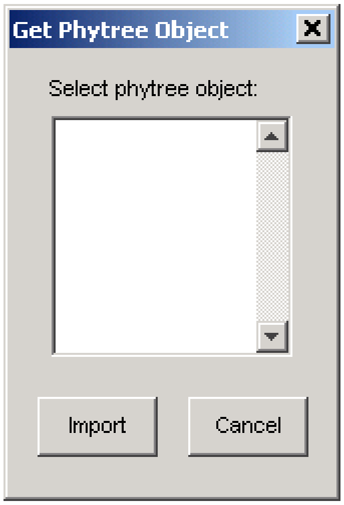
   MATLAB replaces the current tree data with data from the selected file.

Open from Workspace Command

Use the **Open from Workspace** command to read tree data from a `phytree` object in the MATLAB workspace and display that data in a Phylogenetic Tree Tool.

1 From the **File** menu, click **Open from Workspace**.

   The **Get Phytree Object** dialog box opens.



2 From the list, select a phytree object in the MATLAB workspace.

3 Click the **Import** button.

MATLAB replaces the current tree data in the Phylogenetic Tree Tool with data from the selected object.

Save Command

After you create a phytree object or prune a tree from existing data, you can save the resulting tree in a Newick formatted file. The sequence data used to create the phytree object is not saved with the tree.

1 From the **File** menu, click **Save**.

The **Save Phylogenetic tree as** dialog box opens.

2 In the **Filename** box, enter the name of a file. The Bioinformatics Toolbox uses the file extension .tree for Newick formatted files, but you can use file extension.

3 Click **Save**.

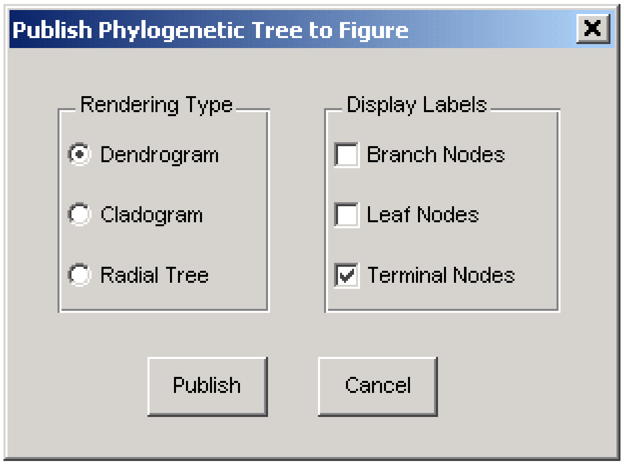
phytreetool saves tree data without the deleted branches, and it saves changes to branch and leaf names. Formatting changes such as branch rotations, collapsed branches, and zoom settings are not saved in the file.

Publish to Figure Command

After you have explored the relationships between branches and leaves in your tree, you can copy the tree to a MATLAB figure window. Using a figure window allows you to use all the MATLAB features for annotating, changing font characteristics, and getting your figure ready for publication. Also, from the figure window, you can save an image of the tree as it was displayed in the Phylogenetic Tree Tool window.

1 From the **File** menu, point to **Publish to Figure**, and then click either **With Hidden Nodes** or **Only Displayed**.

   The **Publish Phylogenetic Tree to Figure** dialog box opens.



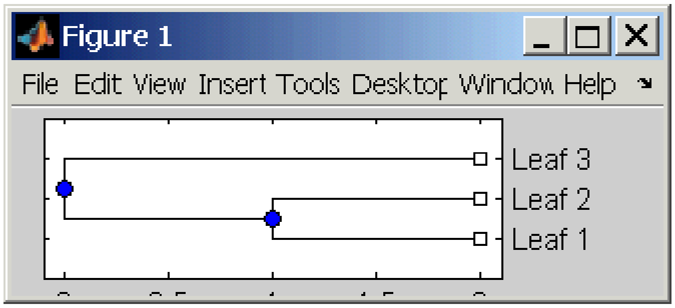
Publish Phylogenetic Tree to Figure

Rendering Type: [✓ Dendrogram] Cladogram Radial Tree

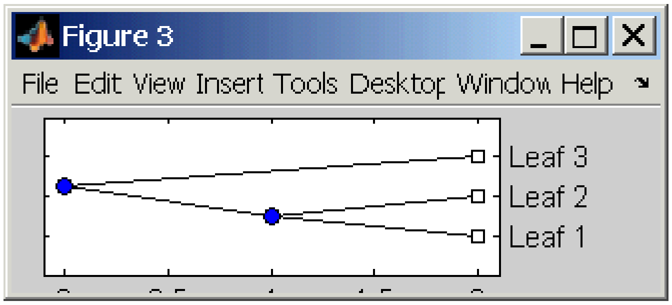
Display Labels: Branch Nodes, Leaf Nodes, [✓ Terminal Nodes]

2 Select one of the Rendering Types, and then select the **Display Labels** you want on your figure.

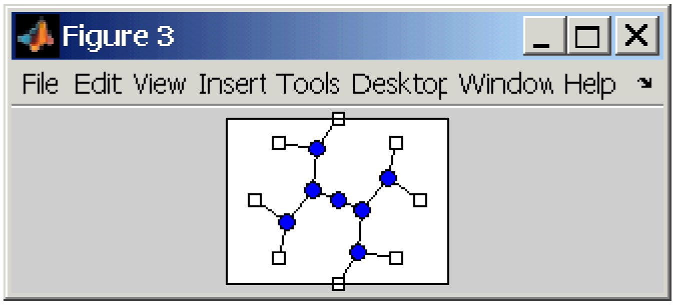
   - **Dendrogram** (square branches)



- **Cladogram** (angular branches)



- **Radial Tree**



3 Select the **Display Labels** you want on your figure. You can select from all to none of the options.

- **Branch Nodes** — Display branch node names on the figure.
- **Leaf Nodes** — Display leaf node names on the figure.

- **Terminal Nodes** — Display terminal node names on the right border.

4 Click the **Publish** button.

A new figure window opens with the characteristics you selected.

Export to New Tool Command

Because some of the Phylogenetic Tree Tool commands cannot be undone (for example, the Prune command), you might want to make a copy of your tree before trying a command. At other times, you might want to compare two views of the same tree, and copying a tree to a new tool window allows you to make changes to both tree views independently .

1 From the **File** menu, point to the **Export to New Tool** submenu, and then click either **With Hidden Nodes** or **Only Displayed**.

A new **Phylogenetic Tree Tool** window opens with a copy of the tree.

2 Use the new figure to continue your analysis.

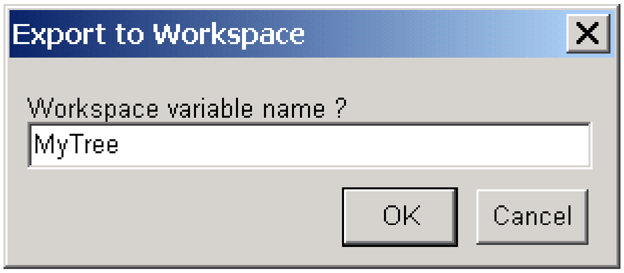
Export to Workspace Command

The Phylogenetic Tree Tool can open Newick formatted files with tree data. However, it does not create a phytree object in the MATLAB workspace. If you want to programmatically explore phylogenetic trees, you need to use the Export to Workspace command.

1 From the **File** menu, point to **Export to Workspace**, and then click either **With Hidden Nodes** or **Only Displayed**.

The **Export to Workspace** dialog box opens.

2 In the **MATLAB variable name** box, enter the name for your phylogenetic tree data.


Export to Workspace

Workspace variable name ?

MyTree

3 Click **OK**.

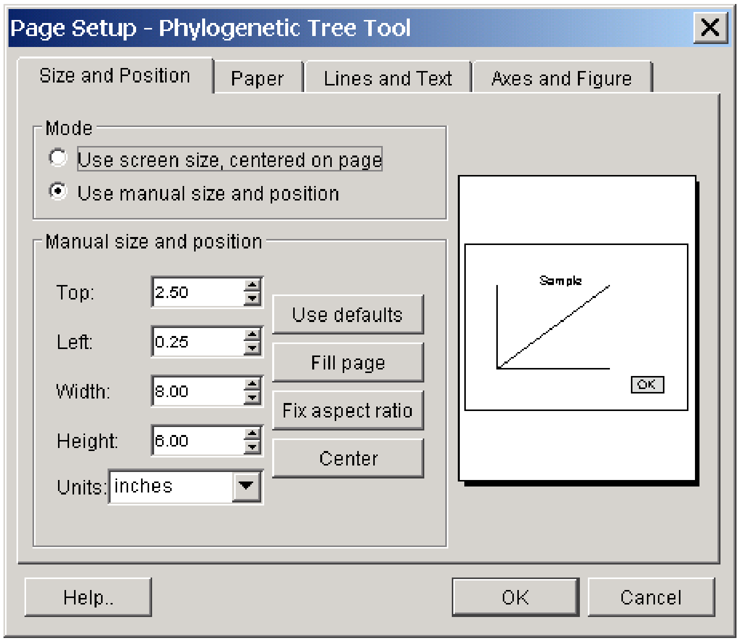
MATLAB creates an object in the MATLAB workspace with type `phytree`.

Page Setup Command

When you print from the Phylogenetic Tree Tool or a MATLAB figure window (with a tree published from the tool), you can specify setup options for printing a tree.

1 From the **File** menu, click **Page Setup**.

The **Page Setup - Phylogenetic Tree Tool** dialog box opens. This is the same dialog box MATLAB uses to select page formatting options.



Mode
( ) Use screen size, centered on page
(●) Use manual size and position

Manual size and position
Top: 2.50
Left: 0.25
Width: 8.00
Height: 6.00
Units: inches

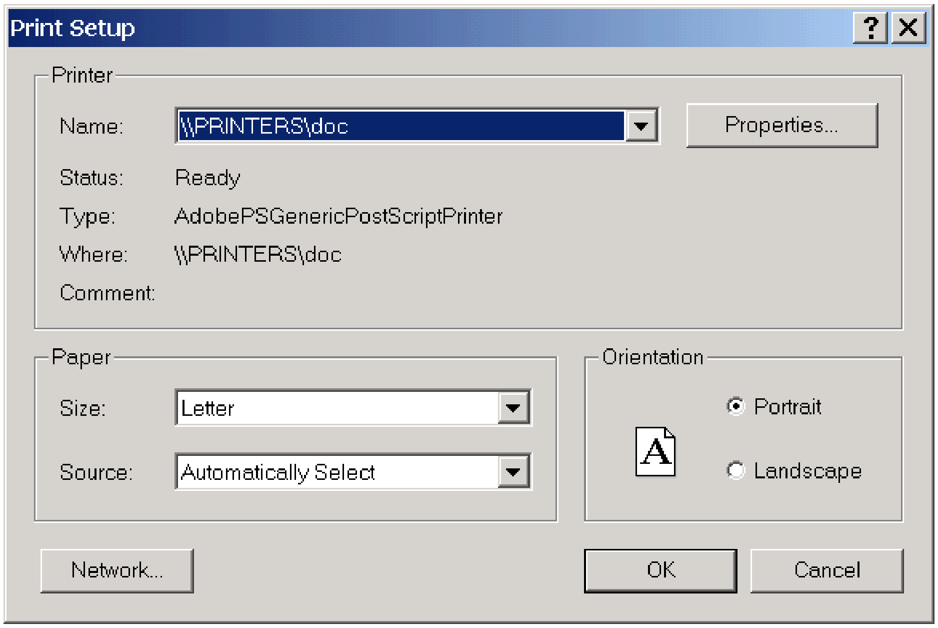
2 Select the page formatting options and values you want, and then click **OK**.

Print Setup Command

Use the Print Setup command with the Page Setup command to print a MATLAB figure window.

1 From the **File** menu, click **Print Setup**.

The Print Setup dialog box opens.


Print Setup

Printer

Name: \\PRINTERS\doc

Status: Ready

Type: AdobePSGenericPostScriptPrinter

Where: \\PRINTERS\doc

Comment:

Paper

Size: Letter

Source: Automatically Select

Orientation

[✓ Portrait] Landscape

2 Select the printer and options you want, and then click **OK**.

Print Preview Command

Use the **Print Preview** command to check the formatting options you selected with the **Page Setup** commend.

1 From the **File** menu, click **Print Preview**.

A window opens with a picture of your figure with the selected formatting options.

2 Click **Print** or **Close**.

Print

Use the **Print** command to make a copy of your phylogenetic tree after you use the **Page Setup** command to select formatting options.

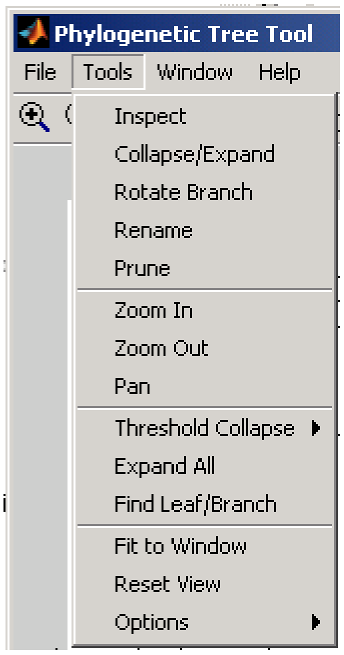
1 From the **File** menu, click **Print**.

   The **Print** dialog box opens.

2 From the **Name** list, select a printer, and then click **OK**.

Tools Menu

The **Tools** menu and toolbar are where you will find most of the commands specific to trees and phylogenetic analysis. Use these commands and modes to interactively edit and format your tree. The Tools menu commands are shown below.



Inspect Mode Command

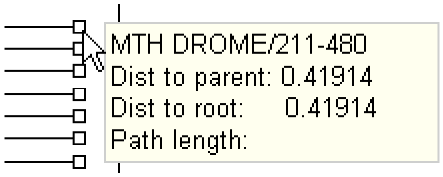
Use the inspect mode to compare path distances between sequences and to search for related sequences that might not be physically drawn close together.

1 From the **Tools** menu, click **Inspect**, or from the toolbar, click the Inspect Tool mode icon .

   The **Phylogenetic Tree Tool** is set to inspect mode.

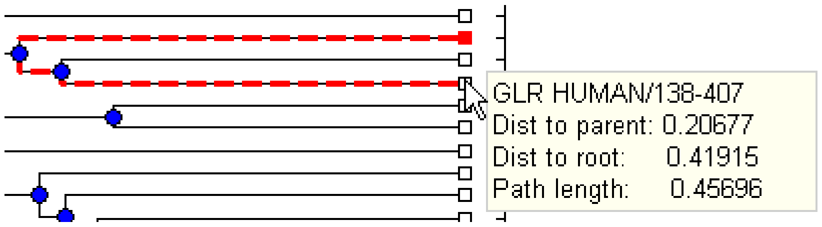
2 Point to a branch or leaf node.

A pop-up window opens with information about the patristic distances to parent and root nodes.


MTH DROME/211-480
Dist to parent: 0.41914
Dist to root: 0.41914
Path length:

3 Click a branch or leaf node, and then move your mouse over another leaf node.

The tool highlights the path between nodes and displays the path length in the pop-up window . The path length is the patristic distances calculated by `seqlinkage`.


GLR HUMAN/138-407
Dist to parent: 0.20677
Dist to root: 0.41915
Path length: 0.45696

Collapse/Expand Branch Mode Command

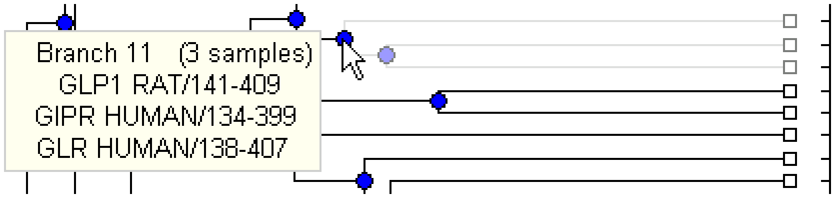
Some trees can have thousands of leaf and branch nodes. Displaying all the nodes can create a tree diagram that is unreadable. By collapsing some of the branches, you can better see the relationships between the remaining nodes.

1 From the **Tools** menu, click **Collapse/Expand**, or from the toolbar, click the Collapse/Expand node icon .

The **Phylogenetic Tree Tool** is set to collapse/expand mode.

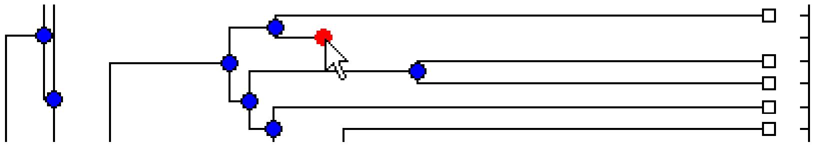
2 Point to a branch.

The selected paths to collapse (remove from view) are highlighted in gray.



3 Click the branch node.

The tool removes the display of branch and leaf nodes below the selected branch. The data is not removed.



4 To expand a branch, point to a collapsed branch and click.

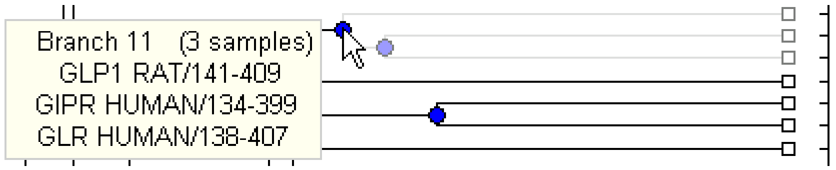
Rotate Branch Mode Command

A phylogenetic tree is initially created by pairing the two most similar sequences and then adding the remaining sequences in a decreasing order of similarity. You might want to rotate branches to emphasize the direction of evolution.

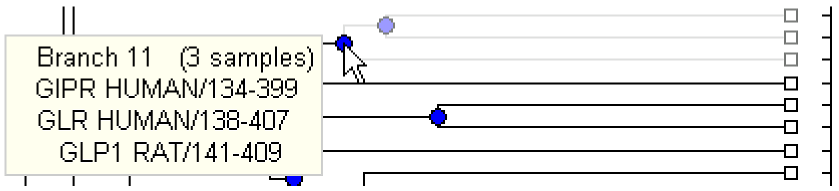
1 From the **Tools** menu, click **Rotate Branch**, or from the toolbar, click the Rotate Branch mode icon .

The **Phylogenetic Tree Tool** is set to rotate branch mode.

2 Point to a branch node.



3 Click the branch node.


Branch 11 (3 samples)
GIPR HUMAN/134-399
GLR HUMAN/138-407
GLP1 RAT/141-409

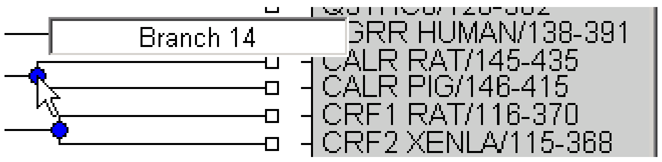
The branch and leaf nodes are rotated 180 degrees around the selected branch node.

Rename Leaf/Branch Mode Command

The Phylogenetic Tree Tool takes the node names from the `phytree` object and creates numbered branch names starting with `Branch 1`. You can edit and change or replace any of the leaf or branch names. Changes to branch and leaf names are saved when you use the **Save** command.

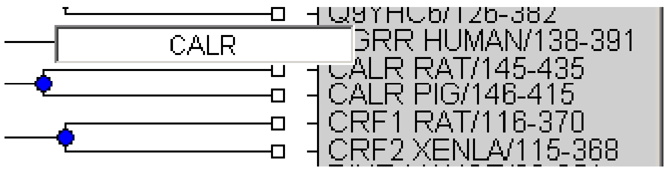
1 From the **Tools** menu, click **Rename**, or from the toolbar, click the Rename mode icon.

2 Click a branch or leaf node.


Branch 14
GRR HUMAN/138-391
CALR RAT/145-435
CALR PIG/146-415
CRF1 RAT/116-370
CRF2 XENLA/115-368

A text box opens with the current name of the node.

3 In the text box, edit or enter an new name.


CALR
Q8YHC6/126-382
GRR HUMAN/138-391
CALR RAT/145-435
CALR PIG/146-415
CRF1 RAT/116-370
CRF2 XENLA/115-368

4 To save your changes, click outside of text box.

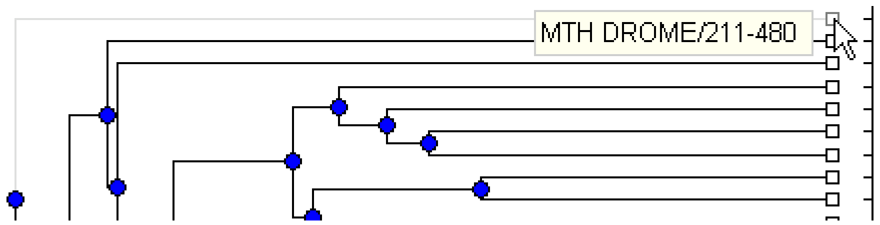
Prune (delete) Leaf/Branch Mode Command

Your tree might contain leaves that are far outside the phylogeny, or it might have duplicate leaves that you want to remove.

1 From the **Tools** menu, click **Prune**, or from the toolbar, click the prune icon .

   The **Phylogenetic Tree Tool** is set to rename mode.

2 Point to a branch or leaf node.



   For leaf node, the branch line connected to the leaf is highlighted in gray. For a branch nodes, the branch lines below the node are highlighted in light gray.

Note If you delete nodes (branches or leaves), you cannot undo the changes. The Phylogenetic Tree Tool does not have an Undo command.

3 Click the branch or leaf node.

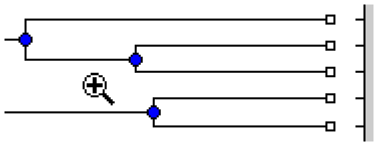
   The branch is removed from the figure and the other nodes are rearranged to balance the tree structure. The phylogeny is not recalculated.

Zoom In, Zoom Out, and Pan Commands

The Zoom and Pan commands are the standard controls with MATLAB figures for resizing and moving the screen.

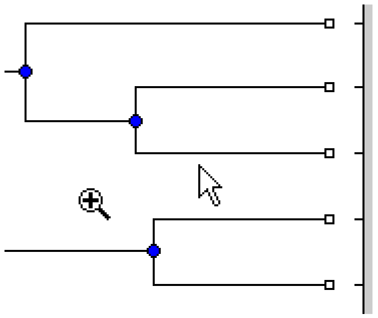
1 From the **Tools** menu, click **Zoom In**, or from the toolbar click the zoom in icon .

The tool activates zoom n mode and changes the cursor to a magnifying glass.



2 Place the cursor over the section of the tree diagram you want to enlarge and then click.

The tree diagram is enlarged to twice its size.



3 From the toolbar click the Pan icon .

4 Move the cursor over the tree diagram, left-click, and drag the diagram to the location you want to view.

Zoom In , Zoom Out , Pan

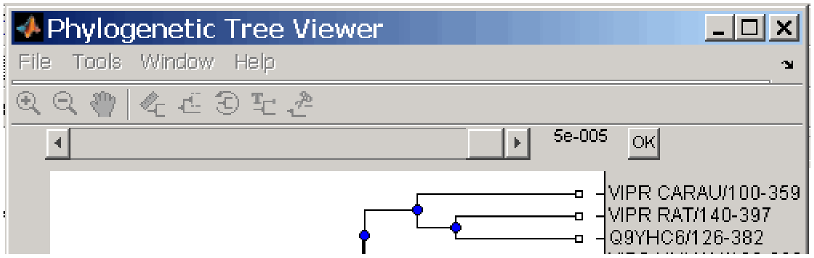
Threshold Collapse Command

Use the **Threshold Collapse** command to collapse the display of nodes using a distance criterion instead of interactively selecting nodes with the **Collapse/Expand** command. Branches with distances below the threshold are collapsed from the display.

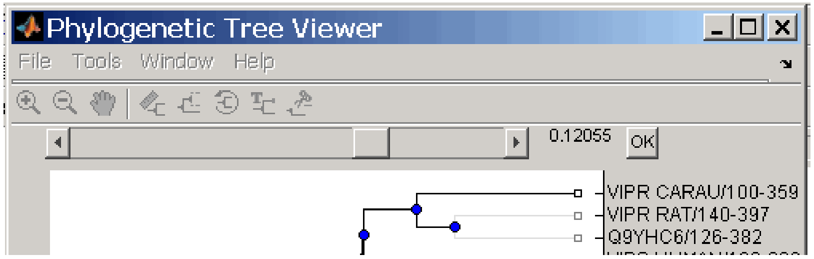
1 From the **Tools** menu, click **Threshold Collapse**, and select one of the following:

- **Distance to Leaves** — Sets the threshold starting from the right of the tree.
- **Distance to Root** — Sets the threshold starting from the root node at the left side of the tree.

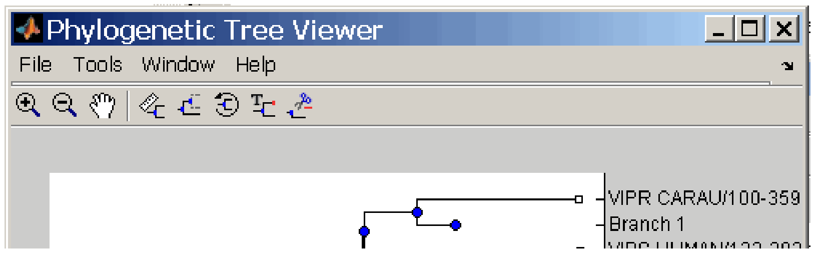
The collapse slider bar is displayed at the top of the diagram.



2 Click and drag the slider bar to the left to set the distance threshold.



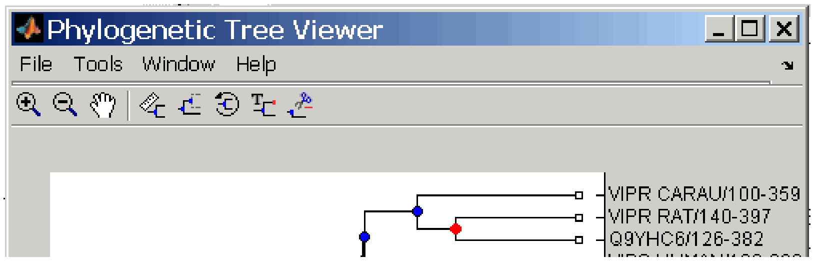
3 Click the **OK** button to the right of the slider. The nodes below the distance threshold are hidden.



Expand All Command

The data for branches and leaves you hide with the **Collapse/Expand** or **Threshold Collapse** commands is not removed from the tree. You can display the hidden data using these commands or display all hidden data with the **Expand All** command.

1 From the **Tool** menu, click **Expand All**. The hidden branches and leaves are displayed.

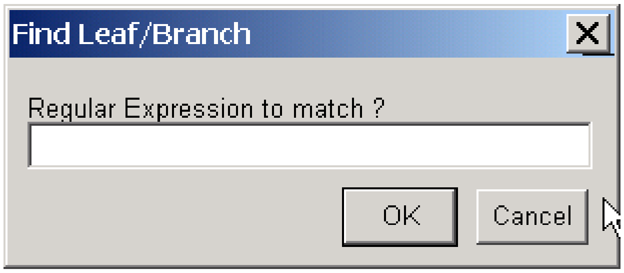


Find Leaf/Branch Command

Phylogenetic trees can have thousands of leaves and branches, and finding a specific node can be difficult. Use the Find command to locate a node using its name or part of its name.

1 From the **Tools** menu, click **Find Leaf/Branch**.

The Find Leaf/Branch dialog opens.



2 In the **Regular Expression to match** box, enter a name or partial name of a branch or leaf.

3 Click **OK**.

Fit to Window

After you hide nodes with the **Collapse/Expand** or **Threshold Collapse** commands, or delete nodes with the **Prune** command, there might be extra space in the tree diagram. Use the **Fit to Window** command to redraw the tree diagram to fill the entire figure window.

1 From the **Tools** menu, click **Fit to Window**.

Reset View Command

Use the Reset Window command to remove formatting changes such as rotations, collapsed branches, and zooms.

1 From the **Tools** menu, click **Reset Window**.

Options Submenu

Use the Options command to select the behavior for the zoom and pan modes.

- **Unconstrained Zoom** — Allow zooming in both horizontal and vertical directions.
- **Horizontal Zoom** — Restrict zoom to the horizontal direction.
- **Vertical Zoom** — Zoom only in the vertical direction (default).
- **Unconstrained Pan** — Allow panning in both horizontal and vertical directions.
- **Horizontal Pan** — Restrict panning to horizontal direction.
- **Vertical Pan** — Pan only in the vertical direction (default).

Windows Menu

The **Windows** menu is standard on MATLAB GUI and figure windows. Use this menu to select any opened window.

Help Menu

Use the **Help** menu to select quick links to the Bioinformatics Toolbox documentation for phylogenetic analysis functions, tutorials, and the `phytreetool` reference.

Functions – Categorical List

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

Data Formats and Databases

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

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

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

Sequence Conversion

Convert nucleotide and amino acid sequences.

aa2int	Convert an amino acid sequence from a letter to an integer representation
aa2nt	Convert an amino acid sequence to a nucleotide sequence
aminolookup	Display amino acid codes, integers, abbreviations, names, and codons
baselookup	Display nucleotide codes, integers, names, and abbreviations
dna2rna	Convert a DNA sequence to an RNA sequence
int2aa	Convert an amino acid sequence from an integer to a letter representation
int2nt	Convert a nucleotide sequence from an integer to a letter representation
nt2aa	Convert a sequence of nucleotides to a sequence of amino acids
nt2int	Convert a nucleotide sequence from a letter to an integer representation
rna2dna	Convert an RNA sequence of nucleotides to a DNA sequence
seq2regexp	Convert a sequence with ambiguous characters to a regular expression
seqcomplement	Calculate the complementary strand of a nucleotide sequence
seqrcomplement	Calculate the reverse complement of a nucleotide sequence
seqreverse	Reverse the letters or numbers in a nucleotide sequence

Sequence Statistics

List of sequence statistics functions

aacount	Count the amino acids in a sequence
aminolookup	Display amino acid codes, integers, abbreviations, names, and codons
basecount	Count the number of nucleotides in a sequence
baselookup	Display nucleotide codes, integers, names, and abbreviations
codoncount	Count the number of codons in a nucleotide sequence
dimercount	Count the number of dimers in a sequence
nmercount	Count the number of n-mers in a nucleotide or amino acid sequence
ntdensity	Plot the density of nucleotides along a sequence
seqshowwords	Graphically display the words in a sequence
seqwordcount	Count the number of occurrences of a word in a sequence

Sequence Utilities

List of sequence utilities functions

aminolookup	Display amino acid codes, integers, abbreviations, names, and codons
baselookup	Display nucleotide codes, integers, names, and abbreviations
blastncbi	Generate a remote BLAST request
geneticcode	Return nucleotide codon to amino acid mapping
joinseq	Join two sequences to produce the shortest supersequence
palindromes	Find palindromes in a sequence
randseq	Generate a random sequence from a finite alphabet
restrict	Split a sequence at a specified restriction site
revgeneticcode	Get the reverse mapping for a genetic code
seqdisp	Format long sequence output for easy viewing
seqmatch	Find matches for every string in a library
seqshoworfs	Graphically display the open reading frames in a sequence

Pairwise Sequence Alignment

List of pairwise sequence alignment functions

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

Protein Analysis

List of protein analysis functions

aacount	Count the amino acids in a sequence
aminolookup	Display amino acid codes, integers, abbreviations, names, and codons
atomiccomp	Calculate the atomic composition of a protein
cleave	Cleave a protein with an enzyme
isoelectric	Estimate the isoelectric point for an amino acid sequence
molweight	Calculate the molecular weight of an amino acid sequence
pdbdistplot	Visualize the intermolecular distances in a PDB file
proteinplot	Display property values for amino acid sequences
ramachandran	Draw a Ramachandran plot for PDB data

Trace Tools

List of functions for analysis of nucleotide traces

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

Profile Hidden Markov Models

List of Hidden Markov Model functions

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

Microarray File Formats

List of microarray file format functions

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

Microarray Visualization

List of microarray visualization functions

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

Microarray Normalization and Filtering

List of microarray normalization and filtering functions

exprprofrange	Calculate the range of gene expression profiles
exprprofvar	Calculate the variance of gene expression profiles
geneentropyfilter	Remove genes with low entropy expression values
genelowvalfilter	Remove gene profiles with low absolute values
generangefilter	Remove gene profiles with small profile ranges
genevarfilter	Filter genes with small profile variance
malowess	Smooth microarray data using the Lowess method
mamadnorm	Normalize microarray data by median absolute deviation (MAD)
mameannorm	Normalize microarray data using the global mean

Scoring Matrices

List of scoring matrices

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

Phylogenetic Tree Tools

List of functions for phylogenetic tree analysis.

gethmmtree	Get phylogenetic tree data from PFAM database
phytreeread	Read phylogenetic tree files
phytreetool	View, edit, and explore phylogenetic tree data
phytreewrite	Write a phylogenetic tree object to a Newick formatted file
seqlinkage	Construct a phylogenetic tree from pairwise distances
seqpdist	Calculate the pairwise distance between biological sequences

Phylogenetic Tree Methods

List of methods for the phytree object

get (phytree)	Get information about a phylogenetic tree object
getbyname (phytree)	Select branches and leaves by name from a phytree object
pdist (phytree)	Calculate the pairwise patristic distances in a phytree object
phytree	Object constructor for a phylogenetic tree object
plot (phytree)	Draw a phylogenetic tree
prune	Remove branch nodes from a phylogenetic tree
select	Select tree branches and leaves in a phytree object

Tutorials, Demos, and Examples

Sequence analysis

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

Hidden Markov Model profiles

- hmmprofdemo — HMM profile alignment tutorial example

Microarray analysis

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

Phylogenetic Analysis

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

External software interface

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

External web database interface

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

Functions — Alphabetical List

aa2int

Purpose

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

Syntax

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

Arguments

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

Description

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

Examples

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

```
SeqInt = aa2int('MATLAB')

SeqInt =
  13    1    17    11    1    21
```

Convert a random amino acid sequence of letters to integers.

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

SeqChar =
   dwcztecakfuecvifchds

SeqInt = aa2int(SeqChar)

SeqInt =
  Columns 1 through 13
```

```
    4   18    5   22   17    7    5    1   12   14    0    7    5
  Columns 14 through 20
   20   10   14    5    9    4   16
```

See Also Bioinformatics Toolbox functions `aminolookup`, `int2aa`, `int2nt`, `nt2int`

Purpose

Convert an amino acid sequence to a nucleotide sequence

Syntax

```
SeqNT = aa2nt(SeqAA, 'PropertyName', PropertyValue)

aa2nt(..., 'GeneticCode', GeneticCodeValue)
aa2nt(..., 'Alphabet' AlphabetValue)
```

Arguments

`SeqAA`	Amino acid sequence. Enter a character string or a vector of integers from the table . Examples: 'ARN' or [1 2 3]
GeneticCodeValue	Property to select a genetic code. Enter a code number or code name from the table Genetic Code below. If you use a code name, you can truncate the name to the first two characters of the name.
AlphabetValue	Property to select a nucleotide alphabet. Enter either 'DNA' or 'RNA'. The default value is 'DNA', which uses the symbols A, C, T, G. The value 'RNA' uses the symbols A, C, U, G.

Genetic Code

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

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

Description

`SeqNT = aa2nt(SeqAA, '`*`PropertyName`*`', `*`PropertyValue`*`)` converts an amino acid sequence to a nucleotide sequence using the standard genetic code. In general, the mapping from an amino acid to a nucleotide codon is not a one-to-one mapping. For amino acids with more then one possible nucleotide codon, this function selects randomly a codon corresponding to that particular amino acid.

For the ambiguous characters `B` and `Z`, one of the amino acids corresponding to the letter is selected randomly, and then a codon sequence is selected randomly. For the ambiguous character `X`, a codon sequence is selected randomly from all possibilities.

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

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

Standard Genetic Code

Amino Acid		Amino Acid	
Alanine	A—GCT, GCC, GCA, GCG	Phenylalanine	F—TTT, TTC
Arginine	R—CGT, CGC, CGA, CGG, AGA, AGG	Proline	P—CCT, CCC, CCA, CCG
Asparagine	N—ATT, AAC	Serine	S—TCT, TCC, TCA,TCG, AGT, AGC
Aspartic acid (Aspartate)	D—GAT, GAC	Threonine	T—ACT, ACC, ACA, ACG
Cysteine	C—TGT, TGC	Tryptophan	W—TGG
Glutamine	Q—CAA, CAG	Tyrosine	Y—TAT, TAC
Glutamic acid (Glutamate)	E—GAA, GAG	Valine	V—GTT, GTC, GTA, GTG
Glycine	G—GGT, GGC, GGA, GGG	Aspartic acid or Asparagine	B—random codon from D and N
Histidine	H—CAT, CAC	Glutamic acid or Glutamine	Z—random codon from E and Q
Isoleucine	I—ATT, ATC, ATA	Unknown or any amino acid	X—random codon
Leucine	L—TTA, TTG, CTT, CTC, CTA, CTG	Translation stop	*—TAA, TAG, TGA

Amino Acid		Amino Acid	
Lysine	K—AAA, AAG	Gap of indeterminate length	- to ---
Methionine	M—ATG	Any character or any symbol not in table	?—???

Examples

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

```
aa2nt('MATLAB')

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

Use the Vertebrate Mitochondrial genetic code.

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

ans =
ATGGCAACTCTAGCGCCT
```

Use the genetic code for the Echinoderm Mitochondrial RNA alphabet.

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

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

Convert a sequence with the ambiguous amino acid characters B.

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

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

See Also

Bioinformatics Toolbox functions `aminolookup`, `baselookup`, `geneticcode`, `nt2aa` , `revgeneticcode`

Purpose

Count the amino acids in a sequence

Syntax

```
Amino = aacount(SeqAA, 'PropertyName', PropertyValue)

aacount(...,'Chart', ChartValue)
aacount(...,'Others', OthersValue)
```

Arguments

`SeqAA`	Amino acid sequence. Enter a character string or vector of integers from the table . Examples: `'ARN'` or `[1 2 3]`. You can also enter a structure with the field `Sequence`.
ChartValue	Property to select a type of plot. Enter either `'pie'` or `'bar'`.
OthersValue	Property to control the counting of ambiguous characters individually. Enter either `'full'` or `'bundle'`. The default value is `'bundle'`.

Description

`Amino = aacount(SeqAA, '`*`PropertyName`*`', `*`PropertyValue`*`)` counts the type and number of amino acid in an amino acid sequence and returns the counts in a 1-by-1 structure (`Amino`) with fields for the standard 20 amino acids (`A C D E F G H K L M N P Q R S T U V W Y`).

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

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

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

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

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

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

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

Examples

Count the amino acids in the string 'MATLAB'.

```
AA  = aacount('MATLAB')

Warning: Symbols other than the standard 20 amino acids appear
in the sequence.
AA =
     A: 2
     R: 0
     N: 0
     D: 0
     C: 0
     Q: 0
     E: 0
     G: 0
     H: 0
     I: 0
     L: 1
     K: 0
     M: 1
     F: 0
     P: 0
     S: 0
     T: 1
     W: 0
     Y: 0
     V: 0
Others: 1

AA.A
ans =
       2
```

See Also Bioinformatics Toolbox functions `basecount`, `codoncount`, `dimercount`

affyread

Purpose

Read microarray data from an Affymetrix GeneChip file

Syntax

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

Arguments

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

Description

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

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

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

GeneChip and Affymetrix are registered trademarks of Affymetrix, Inc.

See Also

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

Purpose

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

Syntax

```
aminolookup
aminolookup(SeqAA)

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

Arguments

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

Description

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

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

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

```
aminolookup('abc')

ans=
AlaAsxCys
```

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

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

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

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

Examples

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

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

ans =
P  Pro
```

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

```
aminolookup('MWKQAEDIRDIYDF')

ans =
MetTrpLysGlnAlaGluAspIleArgAspIleTyrAspPhe
```

Convert a three-letter amino acid sequence to a single-letter sequence.`aminolookup('Me`

```
ans =
MWKQAEDIRDIYDF
```

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

```
aminolookup('integer', 1)

ans =
A  Ala Alanine
```

See Also

Bioinformatics Toolbox functions aa2int, aacount, geneticcode, int2aa, nt2aa, revgeneticcode

atomiccomp

Purpose

Calculate the atomic composition of a protein

Syntax

```
Atoms = atomiccomp(SeqAA)
```

Arguments

`SeqAA`	Amino acid sequence. Enter a character string or vector of integers from the table . You can also enter a structure with the field `Sequence`.

Description

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

Examples

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

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

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

mwcchu.C

ans=
      526
```

See Also

Bioinformatics Toolbox functions `aacount`, `molweight`

Purpose

Count the number of nucleotides in a sequence

Syntax

```
Bases = basecount(SeqNT, 'PropertyName', PropertyValue)

basecount(..., 'Chart', ChartValue)
basecount(..., 'Others', OthersValue)
```

Arguments

`SeqNT`	Nucleotide sequence. Enter a character string with the letters `A`, `T`, `U`, `C`, and `G`. The count for `U` characters is included with the count for `T` characters. . You can also enter a structure with the field `Sequence`.
ChartValue	Property to select a type of plot. Enter either `'pie'` or `'bar'`.
OthersValue	Property to control counting ambiguous characters individually. Enter either `full'` or `'bundle'`. The default value is `'bundle'`.

Description

`Bases = basecount(SeqNT, '`*PropertyName*`', `*PropertyValue*`)` counts the number of bases in a nucleotide sequence and returns the base counts in a `1`-by-`1` structure with the fields `A`, `C`, `G`, `T`.

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

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

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

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

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

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

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

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

Examples

Count the number of bases in a DNA sequence.

```
Bases = basecount('TAGCTGGCCAAGCGAGCTTG')

Bases =
    A: 4
    C: 5
    G: 7
    T: 4

Bases.A

ans =
    4
```

Count the bases in a DNA sequence with ambiguous characters.

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

ans =
```

```
   A: 4
   C: 5
   G: 6
   T: 2
   R: 0
   Y: 0
   K: 0
   M: 0
   S: 0
   W: 0
   B: 1
   D: 1
   H: 0
   V: 0
   N: 0
Gaps: 0
```

See Also

Bioinformatics Toolbox functionsaacount, baselookup, codoncount, dimercount, nmercount, ntdensity

baselookup

Purpose

Display nucleotide codes, integers, names, and abbreviations

Syntax

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

Arguments

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

Nucleotide Lookup Table

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

Code	Integer	Base Name	Meaning	Complement
N,X	15	A**N**y nucleotide	G|A|T|C	N
-	16	Gap of indeterminate length	Gap	-

Description

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

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

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

displays the corresponding letter code, meaning, and nucleotide name.

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

Examples

```
baselookup('COMPLEMENT', 'TAGCTGRCCAAGGCCAAGCGAGCTTN')

baselookup('name','cytosine')
```

See Also

Bioinformatics Toolbox functions aminolookup, basecount, codoncount, dimercount, geneticcode, nt2aa, nt2int, revgeneticcode

Purpose

Generate a remote BLAST request

Syntax

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

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

Arguments

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

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

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

Description

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

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

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

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

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

Information for all of the optional parameters can be found at http://www.ncbi.nlm.nih.gov/blast/html/blastcgihelp.html

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

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

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

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

blastncbi(... , 'Expect', *ExpectValue*) provides a statistical significance threshold for matches against database sequences. You can learn more about the statistics of local sequence comparison at

http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html#head2

blastncbi(..., 'Word', *WordValue*) selects a word size for amino acid sequences.

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

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

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

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

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

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

Examples

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

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

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

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

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

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

See Also

Bioinformatics function `blastread`,

blastread

Purpose

Read an BLAST report from a file

Syntax

`Data = blastread(`*`File`*`)`

Arguments

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

Description

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

`Data = blastread(`*`File`*`)` reads a BLAST report from an NCBI formatted file (*File*) and returns a data structure (Data) containing fields corresponding to the BLAST keywords.

`Data` contains the following fields

```
RID
Algorithm
Query
Database
Hit.Name
Hit.Length
Hit.HSP.Score
Hit.HSP.Expect
Hit.HSP.Identities
Hit.HSP.Positives  (peptide sequences)
Hit.HSP.Gaps
Hit.HSP.Frame      (translated searches)
Hit.HSP.Strand     (nucleotide sequences)
```

blastread parses the basic BLAST reports BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX.

For more information about reading and interpreting BLAST reports, see

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

Examples

```
% Create a BLAST request with a GenPept accession number.
RID = blastncbi('AAA59174', 'blastp', 'expect', 1e-10)
%
% Then pass the RID to getblast to download the report and save
% it to a text file.
getblast(RID, 'ToFile' ,'AAA59174_BLAST.rpt')

% Using the saved file, read the results into a MATLAB structure.
results = blastread('AAA59174_BLAST.rpt')
```

See Also

Bioinformatics functions blastncbi, getblast

Purpose

Return a BLOSUM scoring matrix

Syntax

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

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

Arguments

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

Description

`Matrix = blosum(Identity, '`*`PropertyName`*`', `*`PropertyValue`*`)` returns a BLOSUM (**Blo**cks **Sub**stitution **M**atrix) with a specified percent identity. The default ordering of the output includes the extended characters B, Z, X, and *.

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

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

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

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

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

Examples

Return a BLOSUM matrix with a value of 50.

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

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

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

See Also

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

cleave

Purpose

Cleave a protein with an enzyme

Syntax

```
cleave(SeqAA, PeptidePattern, Position,
        'PropertyName', PropertyValue)

cleave(... 'PartialDigest', PartialDigestValue)
```

Arguments

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

Description

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

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

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

Protease	Peptide Pattern	Position
Trypsin	`[KR][^P]`	1
Chymotrypsin	`[WYF][^P]`	1
Glutamine C	`[ED][^P]`	1
Lysine C	`[K][^P]`	1
Aspartic acid N	`D`	1

Examples

```
S = getgenpept('AAA59174')
% Trypsin cleaves after K or R when the next residue is not P
parts = cleave(S.Sequence,'[KR][^P]',1);
```

See Also

Bioinformatics Toolbox functions `restrict`, `seqshowwords`

Purpose

Create a dendrogram and heat map on the same figure

Syntax

```
clustergram(Data, 'PropertyName', PropertyValue)

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

Arguments

`Data`	Matrix where each row corresponds to a gene. The first column is the names of the genes and each additional column is the result from an experiment.
RowLabelsValue	Property to label the rows in `Data.ColLabels` Enter a cell array of text strings.
ColumnLabelsValue	Property to label the columns in `Data`. Enter a cell array of text strings.
PdistValue	Property to pass arguments to the function `pdist`.
LinkageValue	Property to pass arguments to the function `linkage`.
DendrogramValue	Property to pass arguments to the function `dendrogram`.

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

Description

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

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

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

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

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

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

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

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

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

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

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

Examples

```
load filteredyeastdata;
clustergram(yeastvalues);

% Add some labels.
clustergram(yeastvalues,'ROWLABELS',genes,'COLUMNLABELS',times);
```

```
% Change the clustering parameters.
clustergram(yeastvalues,'PDIST','euclidean','LINKAGE','complete');

% Change the dendrogram color parameter.
clustergram(yeastvalues,'ROWLABELS',genes,'DENDROGRAM',{'color',5});
```

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

codoncount

Purpose

Count the number of codons in a nucleotide sequence

Syntax

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

codoncount(..., 'Frame', FrameValue)
codoncount(..., 'Reverse', ReverseValue)
codoncount(..., 'Figure', FigureValue)
```

Arguments

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

Description

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

- For sequences that have codons with the character `U`, the `U` characters are added to codons with `T` characters.
- If the sequence contains ambiguous nucleotide characters (`R Y K M S W B D H V N`) , or gaps indicated with a hyphen (-), this function creates a field `Others` and displays a warning message.

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

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

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

`[Codons, CodonArray] = codoncount(SeqNT)` returns a 4x4x4 array with the raw count data for each codon. The three dimensions correspond to the three positions in the codon. For example, the element `(2,3,4)` of the array gives the number of `CGT` codons where `A <=> 1`, `C <=> 2`, `G <=> 3`, and `T <=> 4`.

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

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

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

Examples

Count the number of standard codons in a nucleotide sequence.

```
codons = codoncount('AAACGTTA')

codons =
      AAA: 1  ATC: 0  CGG: 0  GCT: 0  TCA: 0
      AAC: 0  ATG: 0  CGT: 1  GGA: 0  TCC: 0
      AAG: 0  ATT: 0  CTA: 0  GGC: 0  TCG: 0
      AAT: 0  CAA: 0  CTC: 0  GGG: 0  TCT: 0
      ACA: 0  CAC: 0  CTG: 0  GGT: 0  TGA: 0
      ACC: 0  CAG: 0  CTT: 0  GTA: 0  TGC: 0
      ACG: 0  CAT: 0  GAA: 0  GTC: 0  TGG: 0
      ACT: 0  CCA: 0  GAC: 0  GTG: 0  TGT: 0
      AGA: 0  CCC: 0  GAG: 0  GTT: 0  TTA: 0
      AGC: 0  CCG: 0  GAT: 0  TAA: 0  TTC: 0
      AGG: 0  CCT: 0  GCA: 0  TAC: 0  TTG: 0
      AGT: 0  CGA: 0  GCC: 0  TAG: 0  TTT: 0
```

```
ATA: 0   CGC: 0   GCG: 0   TAT: 0
```

Count the codons in the second frame for the reverse complement of a sequence.

```
r2codons = codoncount('AAACGTTA', 'Frame',2,...
                                  'Reverse',true);
```

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

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

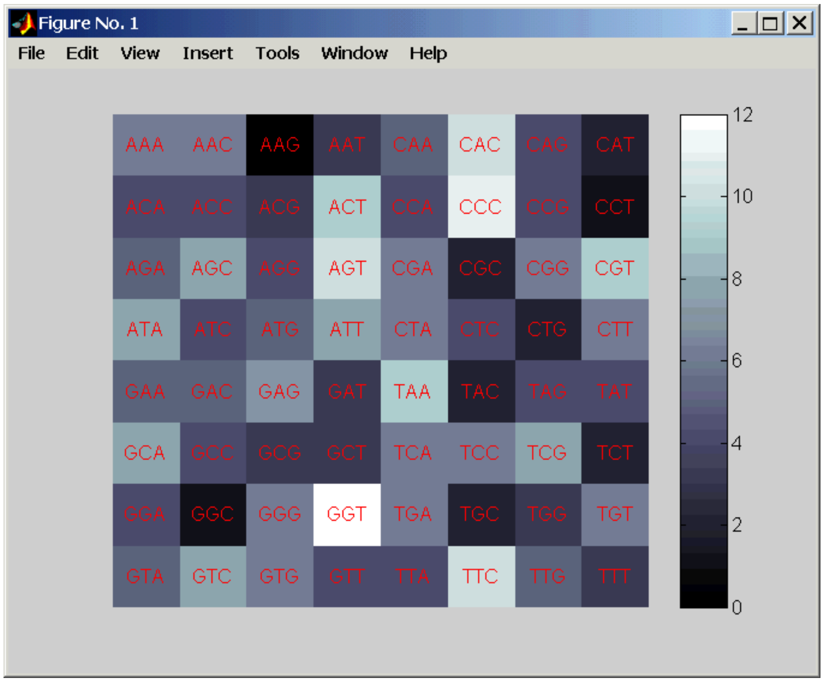


Figure No. 1
File
Edit
View
Insert
Tools
Window
Help
AAA AAC AAG AAT CAA CAC CAG CAT
ACA ACC ACG ACT CCA CCC CCG CCT
AGA AGC AGG AGT CGA CGC CGG CGT
ATA ATC ATG ATT CTA CTC CTG CTT
GAA GAC GAG GAT TAA TAC TAG TAT
GCA GCC GCG GCT TCA TCC TCG TCT
GGA GGC GGG GGT TGA TGC TGG TGT
GTA GTC GTG GTT TTA TTC TTG TTT
12
10
8
6
4
2
0

See Also

Bioinformatics Toolbox functionsaacount, basecount, dimercount, baselookup, nmercount, nmercount, seqcomplement, seqshoworfs, seqwordcount

dayhoff

Purpose

Return a Dayhoff scoring matrix

Syntax

```
ScoringMatrix = dayhoff
```

Description

`PAM250` type scoring matrix. Order of amino acids in the matrix is `A R N D C Q E G H I L K M F P S T W Y V B Z X *`.

See Also

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

Purpose

Count the number of dimers in a sequence

Syntax

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

dimercount(..., 'Chart', ChartStyle)
```

Arguments

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

Description

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

- For sequences that have dimers with the character `U`, the `U` characters are added to dimers with `T` characters.
- If the sequence contains ambiguous nucleotide characters (`R Y K M S W B D H V N`) , or gaps indicated with a hyphen (-), this function creates a field `Others` and displays a warning message.

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

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

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

[Dimers, Percent] = dimercount(SeqNT) returns a 4-by-4 matrix with the relative proportions of the dimers in SeqNT. The rows correspond to A, C, G, and T in the first element of the dimer, and the columns correspond to A, C, G, and T in the second element.

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

Examples

Count the number of dimers in a nucleotide sequence.

```
dimercount('TAGCTGGCCAAGCGAGCTTG')

ans =
    AA: 1
    AC: 0
    AG: 3
    AT: 0
    CA: 1
    CC: 1
    CG: 1
    CT: 2
    GA: 1
    GC: 4
    GG: 1
    GT: 0
    TA: 1
    TC: 0
    TG: 2
    TT: 1
```

See Also

Bioinformatics Toolbox functions aacount, basecount, baselookup, codoncount, nmercount

Purpose

Convert a DNA sequence to an RNA sequence

Syntax

`SeqRNA = dna2rna(SeqDNA)`

Arguments

`SeqDNA`	DNA sequence. Enter either a character string with the characters `A`, `T`, `G`, `C`, and ambiguous characters `R`, `Y`, `K`, `M`, `S`, `W`, `B`, `D`, `H`, `V`, `N`, or a vector of integers from the table Mapping Nucleotide Letters to Integers on page 6-143. You can also enter a structure with the field `Sequence`.
`SeqRNA`	RNA sequence.

Description

`SeqRNA = dna2rna(SeqDNA)` converts a DNA sequence to an RNA sequence by converting any thymine nucleotides (`T`) in the DNA sequence to uracil (`U`). The RNA sequence is returned in the same format as the DNA sequence. For example, if `SeqDNA` is a vector of integers, then so is `SeqRNA`.

Examples

Convert a DNA sequence to an RNA sequence.

```
rna = dna2rna('ACGATGAGTCATGCTT')

rna =
ACGAUGAGUCAUGCUU
```

See Also

Bioinformatics Toolbox function `rna2dna`

MATLAB functions `regexp`, `strrep`

emblread

Purpose

Read data from an EMBL file

Syntax

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

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

Arguments

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

Description

`EMBLData = emblread('`*File*`', '`*PropertyName*`',` *PropertyValue*`)` reads data from an EMBL formatted file (*File*) and creates a MATLAB structure (`EMBLData`) with fields corresponding to the EMBL two-character line type code. Each line type code is stored as a separate element in the structure.

`EMBLData` for the `137.0` version contains the following fields:

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

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

`Seq = emblread('`*`File`*`', 'SequenceOnly', `*`SequenceOnlyValue`*`)`, when `SequenceOnly` is `true`, reads only the sequence information .

Examples

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

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

See Also

Bioinformatics Toolbox functions `getembl`, `fastaread`, `genbankread`, `genpeptread`, `pirread`, `pdbread`

exprprofrange

Purpose

Calculate the range of gene expression profiles

Syntax

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

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

Arguments

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

Description

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

`[Range, LogRange] = exprprofrange(Data)` returns the log range, that is, `log(max(prof))- log(min(prof))`, of each expression profile. If you do not specify output arguments, `exprprofrange` displays a histogram bar plot of the range.

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

Examples

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

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

See Also

Bioinformatics Toolbox function `generangefilter`

Purpose

Calculate the variance of gene expression profiles

Syntax

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

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

Arguments

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

Description

`exprprofvar(Data, 'PropertyName', PropertyValue)` calculates the variance of each expression profile in a dataset (`Data`). If you do not specify output arguments, this function displays a histogram bar plot of the range.

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

Examples

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

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

See Also

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

fastaread

Purpose

Read data from a FASTA formatted file

Syntax

```
FASTAData = fastaread('File')
[Header, Sequence] = fastaread('File')
```

Arguments

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

Description

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

```
Header
Sequence
```

A file with a FASTA format begins with a right angle bracket (>) and a single line description. Following this description is the sequence as a series of lines with fewer than `80` characters. Sequences are expected to use the standard IUB/IUPAC amino acid and nucleotide letter codes.

For a list of codes, see `aminolookup` and `baselookup`.

`FASTAData = fastaread('`*File*`')` reads a file with a FASTA format and returns the data in a structure. `FASTAData.Header` is the header information, while `FASTAData.Sequence` is the sequence stored as a string of letters.

`[Header, Sequence] = fastaread('`*File*`')` reads data from a file into separate variables. If the file contains more than one sequence, then `header` and `sequence` are cell arrays of header and sequence information.

Examples

Get a FASTA formatted sequence from GenBank, save it, and then read the FASTA file into the MATLAB workspace as a structure.

```
s= fastaread('p53nt.txt')

s =
    Header: [1x94 char]
    Sequence: [1x2629 char]
```

See Also

Bioinformatics Toolbox function `aminolookup`, `baselookup`, `fastawrite`

fastawrite

Purpose

Write to a file using a FASTA format

Syntax

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

Arguments

File	Enter either a filename or a path and filename supported by your operating system. (ASCII text file).
`Data`	Enter a character string with a FASTA format, a sequence object, a structure containing the fields `Sequence` and `Header`, or a GenBank/GenPept structure.
`Header`	Information about the sequence.
`Sequence`	Nucleotide or amino acid sequence using the standard IUB/IUPAC codes. For a list of valid characters, see and Mapping Nucleotide Letters to Integers on page 6-143.

Description

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

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

Examples

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

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

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

%extract the coding region.
```

```
codingSeq = seq.Sequence(start:stop)

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

Save multiple sequences.

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

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

>First sequence
ACACAGGAAA

>Second sequence
ACGTCAGGTC
```

See Also

Bioinformatics Toolbox function `fastaread`

Purpose

Read microarray data from a GenePix array list file

Syntax

`GALData = galread('File')`

Arguments

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

Description

`galread` reads data from a GenePix formatted file into a MATLAB structure.

`GALData = galread('File')` reads in a GenePix Array List formatted file (*File*) and creates a structure (`GALData`) containing the following fields:

```
Header
BlockData
IDs
Names
```

The field `BlockData` is an `N`-by-`3` array. The columns of this array are the block data, the column data, and the row data respectively. For more information on the GAL format, see

```
http://www.axon.com/GN_GenePix_File_Formats.html#gal
```

For a list of supported file format versions, see

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

GenePix is a registered trademark of Axon Instruments, Inc.

See Also

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

Purpose

Read data from a GenBank file

Syntax

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

Arguments

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

Discussion

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

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

`GenBankData` contains the following fields:

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

Examples

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

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

See Also

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

Purpose

Remove genes with low entropy expression values

Syntax

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

geneentropyfilter(..., 'Prctile', PrctileValue)
```

Arguments

`Data`	Matrix where each row corresponds to the experimental results for one gene. Each column is the results for all genes from one experiment.
`Names`	Cell array with the same number of rows as `Data`. Each row contains the name or ID of the gene in the data set.
PrctileValue	Property to specify a percentile below which gene data is removed. Enter a value from `0` to `100`.

Description

`Mask = geneentropyfilter(Data, 'PropertyName', PropertyValue)` identifies gene expression profiles in `Data` with entropy values less than the 10th percentile.

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

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

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

geneentropyfilter(..., 'Prctile', *PrctileValue*) removes from Data gene expression profiles with entropy values less than the percentile Prctile.

Examples

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

See Also

Bioinformatics Toolbox functions exprprofrange, exprprofvar, genelowvalfilter, generangefilter

Purpose

Remove gene profiles with low absolute values

Syntax

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

genelowvalfilter(..., 'Prctile', PrctileValue)
genelowvalfilter(..., 'AbsValue', AbsValueValue)
genelowvalfilter(..., 'AnyVal', AnyValValue)
```

Arguments

`Data`	Matrix where each row corresponds to the experimental results for one gene. Each column is the results for all genes from one experiment.
`Names`	Cell array with the same number of rows as `Data`. Each row contains the name or ID of the gene in the data set.
PrctileValue	Property to specify a percentile below which gene expression profiles are removed. Enter a value from `0` to `100`.
AbsValueValue	Property to specify an absolute value below which gene expression profiles are removed.
AnyValValue	Property to select the minimum or maximum absolute value for comparison with `AbsValue`. If `AnyValValue` is `true`, selects the minimum absolute value. If `AnyVal` is `false`, selects the maximum absolute value. The default value is `false`.

Description

Gene expression profile experiments have data where the absolute values are very low. The quality of this type of data is often bad due to large quantization errors or simply poor spot hybridization.

`Mask = genelowvalfilter(Data, 'PropertyName', PropertyValue)` identifies gene expression profiles in `Data` with all absolute values less than the 10th percentile.

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

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

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

`genelowvalfilter(..., 'Prctile', PrctileValue)` removes from `Data` gene expression profiles with all absolute values less than the percentile `Prctile`.

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

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

Examples

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

See Also

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

Purpose

Remove gene profiles with small profile ranges

Syntax

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

generangefilter(..., 'Prctile', PrctileValue)
generangefilter(..., 'AbsValue', AbsValueValue)
generangefilter(..., 'LOGPrctile', LOGPrctileValue)
generangefilter(..., 'LOGValue', LOGValueValue)
```

Arguments

`Data`	Matrix where each row corresponds to the experimental results for one gene. Each column is the results for all genes from one experiment.
`Names`	Cell array with the same number of rows as `Data`. Each row contains the name or ID of the gene in the data set.
PrctileValue	Property to specify a percentile below which gene expression profiles are removed. Enter a value from 0 to 100.
AbsValueValue	Property to specify an absolute value below which gene expression profiles are removed.
LOGPrctileValue	Property to specify the LOG of a percentile.
LOGValueValue	Property to specify the LOG of an absolute value.

Description

`Mask = generangefilter(Data, '`*`PropertyName`*`', `*`PropertyValue`*`)` calculates the range for each gene expression profile in `Data`, and then identifies the expression profiles with ranges less than the 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 then the threshold have a value of 1, and those with a range less then the threshold are 0.

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

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

generangefilter(..., 'Prctile', *PrctileValue*) removes from Data gene expression profiles with ranges less than the percentile Prctile.

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

generangefilter(..., 'LOGPrctile', *LOGPrctileValue*) filters genes with profile ranges in the lowest LOGPrctile percent of the log range.

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

Examples

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

See Also

Bioinformatics Toolbox functions exprprofrange, geneentropyfilter, genelowvalfilter, genevarfilter

Purpose Return nucleotide codon to amino acid mapping

Syntax

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

Arguments

GeneticCode	Enter a code number or code name from the table Genetic Code below. If you use a code name, you can truncate the name to the first two characters of the name.

Genetic Code

Code Number	Code Name
1	`Standard`
2	`Vertebrate Mitochondrial`
3	`Yeast Mitochondrial`
4	`Mold, Protozoan, Coelenterate Mitochondrial, and Mycoplasma/Spiroplasma`
5	`Invertebrate Mitochondrial`
6	`Ciliate, Dasycladacean, and Hexamita Nuclear`
9	`Echinoderm Mitochondrial`
10	`Euplotid Nuclear`
11	`Bacterial and Plant Plastid`
12	`Alternative Yeast Nuclear`
13	`Ascidian Mitochondrial`

Code Number	Code Name
14	Flatworm Mitochondrial
15	Blepharisma Nuclear
16	Chlorophycean Mitochondrial
21	Trematode Mitochondrial
22	Scenedesmus Obliquus Mitochondrial
23	Thraustochytrium Mitochondrial

Description

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

geneticcode(*GeneticCode*) returns a structure of the mapping for alternate genetic codes, where GeneticCode is either the transl_table (code) number from the NCBI Genetics web page (http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c) or one of the supported names in the genetic code table above.

Examples

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

```
wormcode = geneticcode('Flatworm Mitochondrial');
```

See Also

Bioinformatics Toolbox functions aa2nt, baselookup, nt2aa, revgeneticcode, seqshoworfs

Purpose

Filter genes with small profile variance

Syntax

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

genevarfilter(..., 'Prctile', PrctileValue)
genevarfilter(..., 'AbsValue', AbsValueValue)
```

Arguments

`Data`	Matrix where each row corresponds to a gene. The first column is the name of the genes, and each additional column is the results from an experiment.
`Names`	Cell array with the same number of rows as `Data`. Each row contains the name or ID of the gene in the data set.
PrctileValue	Property to specify a percentile below which gene expression profiles are removed. Enter a value from `0` to `100`
AbsValueValue	Property to specify an absolute value below which gene expression profiles are removed.

Description

Gene profiling experiments have genes which exhibit little variation in the profile and are generally not of interest in the experiment. Removing (filtering) these genes from the data is a commonly done.

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

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

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

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

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

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

Examples

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

See Also

Bioinformatics Toolbox functions exprprofrange, exprprofvar, generangefilter

Purpose

Read data from a GenPept file

Syntax

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

Arguments

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

Description

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

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

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

```
LocusName
LocusSequenceLength
LocusMoleculeType
LocusGenBankDivision
LocusModificationDate
Definition
Accession
PID
Version
GI
```

```
DBSource
Keywords
Source
SourceDatabase
SourceOrganism
Reference.Number
Reference.Authors
Reference.Title
Reference.Journal
Reference.MedLine
Reference.PubMed
Reference.Remark
Comment
Features
Weight
Length
Sequence
```

Examples

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

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

See Also

Bioinformatics Toolbox functions `fastaread`, `genbankread`, `getgenpept`, `pdbread`, `pirread`

Purpose

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

Syntax

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

Arguments

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

Description

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

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

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

Examples

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

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

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

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

See Also

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

get (phytree)

Purpose

Get information about a phylogenetic tree object

Syntax

```
[Value1,Value2, ...] = GET(Tree, 'Name1', 'Name2', ...)
```

Arguments

Tree	Phytree object created with the function `phytree`.
Name	Property name for a `phytree` object.

Description

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

The valid choices for `'Name'` are

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

Examples

```
tr = phytreeread('pf00002.tree')
protein_names = get(tr,'LeafNames')
```

See Also

Bioinformatics Toolbox functions `phytree`, `phytreeread`, and phytree object method `select`

Purpose

Get BLAST report from NCBI web site

Syntax

```
Data = getblast(RID)

getblast(..., 'Descriptions', DescriptionsValue)
getblast(..., 'Alignments', AlignmentsValue)
getblast(..., 'ToFile', ToFileValue)
getblast(..., 'FileFormat', FileFormatValue)
```

Arguments

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

Description

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

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

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

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

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

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

For more information about reading and interpreting BLAST reports, see

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

Examples

```
Run a BLAST search with an NCBI accession number.
RID = blastncbi('AAA59174','blastp','expect',1e-10)

% Then pass the RID to GETBLAST to parse the report, load it into
% a MATLAB structure, and save a copy as a text file.
 report = getblast(RID,'TOFILE','Report.txt')
```

See Also

Bioinformatics Toolbox functions `blastncbi`, `blastread`

getbyname (phytree)

Purpose

Select branches and leaves by name from a phytree object

Syntax

`S = getbyname(`*`Tree`*`, `*`Expression`*`)`

Arguments

Tree	Phytree object created with the function `phytree`.
Expression	Regular expression.

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. When `Expression` is a cell array of strings, `getbyname` returns a matrix where each column corresponds to a query in `Expression`

For information about the symbols that you can use in a matching regular expression, see the MATLAB function `regexp`.

Examples

```
% Load a phylogenetic tree created from a protein family:
tr = phytreeread('pf00002.tree');

% Select all the 'mouse' and 'human' proteins:
sel = getbyname(tr,{'mouse','human'});
view(tr,any(sel,2));
```

See Also

The MATLAB function `regexp`

Purpose

Retrieve sequence information from the EMBL database

Syntax

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

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

Arguments

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

Description

`getembl` retrieves information from the European Molecular Biology Laboratory (EMBL) database for nucleotide sequences. This database is maintained by the European Bioinformatics Institute (EBI). For more details about the EMBL-Bank database, see

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

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

```
Comments
Identification
Accession
SequenceVersion
DateCreated
```

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

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

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

Examples

Retrieve data for the rat liver apolipoprotein A-I.

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

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

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

Retrieve only the sequence for the rat liver apolipoprotein.

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

See Also

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

getgenbank

Purpose

Retrieve sequence information from the GenBank database

Syntax

```
Data = getgenbank('AccessionNumber',
                  'PropertyName',PropertyValue)

getgenbank(..., 'ToFile', ToFileValue)
getgenbank(..., 'FileFormat', FileFormatValue)
getgenbank(..., 'SequenceOnly', SequenceOnlyValue)
```

Arguments

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

Description

`getgenbank` retrieves nucleotide and amino acid sequence information from the GenBank database. This database is maintained by the National Center for Biotechnology Information (NCBI). For more details about the GenBank database, see

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

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

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

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

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

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

Examples

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

```
S = getgenbank('M10051')

S =

                LocusName: 'HUMINSR'
      LocusSequenceLength: '4723'
     LocusNumberofStrands: ''
            LocusTopology: 'linear'
        LocusMoleculeType: 'mRNA'
     LocusGenBankDivision: 'PRI'
    LocusModificationDate: '06-JAN-1995'
               Definition: 'Human insulin receptor mRNA, complete cds.
                Accession: 'M10051'
                  Version: 'M10051.1'
                       GI: '186439'
                 Keywords: 'insulin receptor; tyrosine kinase.'
```

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

See Also Bioinformatics Toolbox functions `genbankread`, `getembl`, `getgenpept`, `getpdb`, `getpir`

Purpose

Retrieve sequence information from the GenPept database

Syntax

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

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

Arguments

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

Description

`getgenpept` retrieves a protein (amino acid) sequence and sequence information from the database `GenPept`. This database is a translation of the nucleotide sequences in GenBank and is maintained by the National Center for Biotechnology Information (NCBI).

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

For more details about the GenBank database, see

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

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

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

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

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

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

Examples

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

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

See Also

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

Purpose

Get Gene Expression Omnibus (GEO) data

Syntax

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

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

Arguments

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

Description

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

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

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

For more information, see

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

Examples

```
geoStruct = getgeodata('GSM1768')
```

See Also

Bioinformatics Toolbox functions `geosoftread`, `getgenbank`, `getgenpept`

Purpose

Retrieve multiple aligned sequences from the PFAM database

Syntax

‘

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

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

Arguments

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

Description

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

```
Header
Sequence
```

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

`gethmmalignment(..., 'Type',` *TypeValue*`)` returns only the alignments used to generate the HMM model if `Type='seed'`, and if `Type='full'`, returns all alignments that fit the model. Default is `'full'`.

Examples

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

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

or

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

See Also

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

Purpose

Retrieve profile hidden Markov models from the PFAM database

Syntax

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

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

Arguments

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

Description

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

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

```
InsertX
DeleteX
FlankingInsertX
```

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

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

Examples

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

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

or

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

See Also

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

Purpose

Get phylogenetic tree data from PFAM database

Syntax

```
Tree = gethmmtree(AccessionNumber)

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

Arguments

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

Description

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

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

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

Examples

Retrieve a phylogenetic tree built from the multiple aligned sequences used to train the HMM profile model for global alignment. The PFAM accession number `PF00002` is for the 7-transmembrane receptor protein in the secretin family.

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

See Also Bioinformatics Toolbox functions, `fastaread`, `gethmmprof`, `pfamhmmread`

Purpose

Retrieve protein structure information from the PDB database

Syntax

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

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

Arguments

PDBid	Unique identifier for a protein structure record. Each structure in the PDB is represented by a 4-character alphanumeric identifier. For example, `4hhb` is the identification code for hemoglobin.
ToFileValue	Property to specify the location and filename for saving data. Enter either a filename or a path and filename supported by your system (ASCII text file).
MirrorSiteValue	Property to select Web site. Enter either `http://rutgers.rcsb.org/pdb` to use the Rutgers University Web site, or enter `http://nist.rcsb.org/pdb` for the National Institute of Standards and Technology site.

Description

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

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

```
Header
Title
```

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

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

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

Examples

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

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

See Also

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

Purpose

Retrieve sequence data from the PIR-PSD database

Syntax

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

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

Arguments

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

Description

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

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

```
Sequence
```

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

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

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

Examples

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

```
pirdata = getpir('cchu')

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

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

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

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

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

See Also

Bioinformatics Toolbox functions genpeptread, getgenpept, getpdb, pdbread, pirread

Purpose Return a Gonnet scoring matrix

Syntax `gonnet`

Description PAM 250 matrix recommended by Gonnet, Cohen & Benner in Science, June 5, 1992. Values are rounded to the nearest integer for the following amino acid order:

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

Gaston. H. Gonnet, Mark A. Cohen, and Steven A. Benner; “Exhaustive matching of the entire protein sequence database” in Science; 256:1443-1445; June 1992.

See Also Bioinformatics Toolbox functions `dayhoff`, `pam`

gprread

Purpose

Read microarray data from a GenePix Results (GPR) file

Syntax

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

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

Arguments

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

Description

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

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

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

The field Indices of the structure contains MATLAB indices that can be used for plotting heat maps of the data.

For more details on the GPR format, see

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

For a list of supported file format versions, see

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

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

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

GenePix is a registered trademark of Axon Instruments, Inc.

Examples

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

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

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

See Also

Bioinformatics Toolbox functions galread, maimage, sptread

hmmprofalign

Purpose

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

Syntax

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

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

Arguments

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

Description

`Alignment = hmmprofalign(Model, Seq, 'PropertyName', PropertyValue)` returns the score for the optimal alignment of the query amino acid or nucleotide sequence (`Seq`) to the profile hidden Markov model (`Model`). Scores are computed using log-odd ratios for emission probabilities and log probabilities for state transitions.

`[Alignment, Score] = hmmprofalign(Model, Seq)` returns a string showing the optimal profile alignment.

Uppercase letters and dashes correspond to MATCH and DELETE states respectively (the combined count is equal to the number of states in the model). Lowercase letters are emitted by the INSERT states. For more information about the HMM profile, see `hmmprofstruct`.

`[Score, Alignment, Prointer] = hmmprofalign(Model, Seq)` returns a vector of the same length as the profile model with indices pointing to the respective symbols of the query sequence. Null pointers (`NaN`) mean that such states did not emit a symbol in the aligned sequence because they represent model jumps from the BEGIN state of a MATCH state, model jumps from the from a MATCH state to the END state, or because the alignment passed through DELETE states.

`hmmprofalign(..., 'ShowScore', ShowScoreValue)`when `ShowScore` is `true`, displays the scoring space and the winning path .

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

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

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

Note Multiple hit alignment is not unsupported in this implementation. All the `Model.LoopX` probabilities are ignored.

Examples

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

See Also

Bioinformatics Toolbox functions `gethmmprof`, `hmmprofestimate`, `hmmprofgenerate`, `hmmprofmerge`, `hmmprofstruct`, `pfamhmmread`, `showhmmprof`

Purpose

Estimate profile HMM parameters using pseudocounts

Syntax

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

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

Arguments

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

hmmprofestimate

Description

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

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

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

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

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

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

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

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

See Also

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

hmmprofgenerate

Purpose

Generate a random sequence drawn from the profile HMM

Syntax

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

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

Arguments

`Model`	Hidden Markov model created with the function hmmprofstruc.
AlignValue	Property to control using upper case letters for matches and lower case letters for inserted letters. Enter either `true` or `false`. The default value is `false`.
FlanksValue	Property to control including the symbols generated by the FLANKING INSERT states in the output sequence. Enter either `true` or `false`. The default values is `false`.
SignatureValue	Property to control returning the most likely path and symbols. Enter either `true` or `false`. Default value is `false`.

Description

`Seq = hmmprofgenerate(Model, '`*`PropertyName`*`', `*`PropertyValue`*`)` returns a string (`Seq`) showing a sequence of amino acids or nucleotides drawn from the profile (Model). The length, alphabet, and probabilities of the `Model` are stored in a structure. For move information about this structure, see `hmmprofstruct`).

`[Sequence, Profptr] = hmmprofgenerage(Model)` returns a vector of the same length as the profile model pointing to the respective states in the output sequence. Null pointers (0) mean that such states do not exist in the output sequence, either because they are never touched (i.e.

jumps from the BEGIN state to MATCH states or from MATCH states to the END state), or because DELETE states are not in the output sequence (not aligned output; see below).

`hmmprofgenerate(..., 'Align', `*`AlignValue`*`)` if `Align` is `true`, the output sequence is aligned to the model as follows: uppercase letters and dashes correspond to MATCH and DELETE states respectively (the combined count is equal to the number of states in the model). Lowercase letters are emitted by the INSERT or FLANKING INSERT states. If `Align` is `false`, the output is a sequence of uppercase symbols. The default value is `true`.

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

`hmmprofgenerate(..., 'Signature', `*`SignatureValue`*`)` if `Signature` is `true`, returns the most likely path and symbols. The default value is `false`.

Examples

```
load('hmm_model_examples','model_7tm_2') % load a model example
rand_sequence = hmmprofgenerate(model_7tm_2)
```

See Also

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

hmmprofmerge

Purpose

Concatenate the prealigned strings of several sequences to a profile HMM

Syntax

```
A = hmmprofmerge(Sequences)
hmmprofmerge(Sequences, Names)
hmmprofmerge(Sequences, Names, Scores)
```

Arguments

`Sequences`	Array of sequences. `Sequences` can also be a structured array with the aligned sequences in a field `Aligned` or `Sequences`, and the optional names in a field `Header` or `Name`.
`Names`	
`Scores`	Pairwise alignment scores from the function hmmprofalign. Enter a vector of values with the same length as the number of sequences in `Sequences`.

Description

`hmmprofmerge(Sequences)` displays a set of prealigned sequences to a HMM model profile. The output is aligned corresponding to the HMM states.

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

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

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

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

Examples

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

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

See Also

Bioinformatics Toolbox functions hmmprofalign, hmmprofstruct

hmmprofstruct

Purpose

Create a profile HMM structure

Syntax

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

Arguments

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

Description

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

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

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

HMM Profile Structure Format

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

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

`MatchEmission`	Symbol emission probabilities in the MATCH states Size is [`ModelLength` x `AlphaLength`]. Note: `sum(S.MatchEmission,2) = [1;1;1; ... ;1]` Default is `1/AlphaLength`.
`InsertEmission`	Symbol emission probabilities in the INSERT state. Size is [`ModelLength` x `AlphaLength`]. Note: `sum(S.InsertEmission,2) = [1;1;1; ... ;1]` Default is `1/AlphaLength`.
`NullEmission`	Symbol emission probabilities in the MATCH and INSERT states for the NULL model. The NULL model is used to compute the log-odds ratio at every state and avoid overflow when the probabilities are propagated through the model. Size is [`1` x `AlphaLength`]. Note: `sum(S.NullEmission) = 1` Default is `1/AlphaLength`.

`BeginX`	BEGIN state transition probabilities
	Format is
	`[B->D1 B->M1 B->M2 B->M3 .... B->Mend]`
	Notes:
	`sum(S.BeginX) = 1`
	For fragment profiles
	`sum(S.BeginX(3:end)) = 0`
	Default is `[0.01 0.99 0 0 ...  0]`.

`MatchX`	MATCH state transition probabilities
	Format is
	`[M1->M2 M2->M3 ... M[end-1]->Mend;`
	`M1->I1 M2->I2 ... M[end-1]->I[end-1];`
	`M1->D2 M2->D3 ... M[end-1]->Dend;`
	`M1->E  M2->E  ... M[end-1]->E  ]`
	Notes:
	`sum(S.MatchX) = [ 1 1 ... 1 ]`
	For fragment profiles
	`sum(S.MatchX(4,:)) = 0`
	Default is `repmat([0.998 0.001 0.001 0],profLength-1,1)`.

`InsertX`	INSERT state transition probabilities
	
	Format is
	
	`[I1->M2 I2->M3 ... I[end-1]->Mend;`
	`[I1->I1 I2->I2 ... I[end-1]->I[end-1] ]`
	
	Note:
	
	`sum(S.InsertX) = [ 1 1 ... 1 ]`
	
	Default is `repmat([0.5 0.5],profLength-1,1).`

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

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

`LoopX`	Loop states transition probabilities used for multiple hits alignment. The format is
	`[E->C  J->B ;`
	`E->J  J->J ]`
	Note: `sum(S.LoopX) = [1 1]`
	Default is `[0.5 0.01; 0.5 0.99]`

`NullX`	Null transition probabilities used to provide scores with log-odds values also for state transitions. The format is `[G->F ; G->G]` Note: `sum(S.NullX) = 1` Default is `[0.01; 0.99]`

Annotation fields (optional)

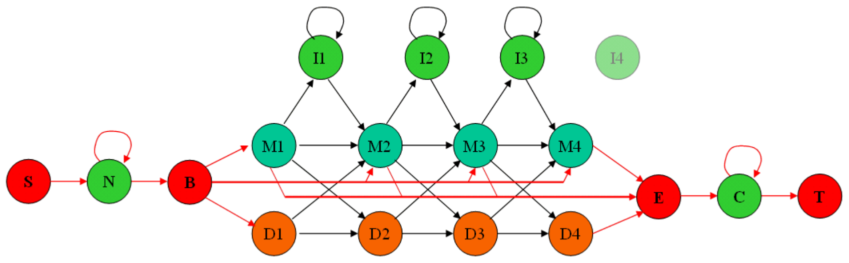
Name	**Model Name**
`IDNumber`	Identification Number
`Description`	Short description of the model

A profile Markov model is a common statistical tool for modeling structured sequences composed of symbols . These symbols include randomness in both the output (emission of symbols) and the state transitions of the process. Markov models are generally represented by state diagrams.

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

- Match state means that the target sequence is aligned to the profile at the specific location,
- Delete state represents a gap or symbol absence in the target sequence (also know as a silent state because it does not emit any symbol),
- Insert state represents the excess of one or more symbols in the target sequence that are not included in the profile.

Flanking states (`S`, `N`, `B`, `E`, `C`, `T`) are used for proper modeling of the ends of the sequence, either for global, local or fragment alignment of the profile. `S`, `N`, `E`, and `T` are silent while `N` and `C` are used to insert symbols at the flanks.



Examples

```
hmmprofstruct(100,'Alphabet','AA')
```

See Also

Bioinformatics Toolbox functions `gethmmprof`, `hmmprofalign`, `hmmprofestimate`, `hmmprofgenerate`, `hmmprofmerge`, `pfamhmmread`, `showhmmprof`

imageneread

Purpose

Read microarray data from an ImaGene Results file

Syntax

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

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

Arguments

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

Description

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

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

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

The field Indices of the structure contains MATLAB indices that you can use for plotting heat maps of the data with the functions image or imagesc.

For more details on the ImaGene format and example data, see the ImaGene User Manual. .

ImaGene is a registered trademark of BioDiscovery, Inc.

Examples

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

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

See Also

The Bioinformatics Toolbox functions gprread, maboxplot, maimage, sptread

Purpose

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

Syntax

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

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

Arguments

`SeqInt`	Amino acid sequence represented with integers. Enter a vector of integers from the table Mapping Amino Acid Integers to Letters below. The array does not have to be of type integer, but it does have to contain only integer numbers. Integers are arbitrarily assigned to IUB/IUPAC letters.
CaseValue	Property to select the case of the returned character string. Enter either `'upper'` or `'lower'`. Default is `'upper'`.

Mapping Amino Acid Integers to Letters

Amino Acid	Code	Amino Acid	Code	Amino Acid	
Alanine	A—1	Isoleucine	I—10	Tyrosine	Y—19
Arginine	R—2	Leucine	L—11	Valine	V—20
Asparagine	N—3	Lysine	K—12	Aspartic acid or Asparagine	B—21
Aspartic acid (aspartate)	D—4	Methionine	M—13	Glutamic acid or Glutamine	Z—22
Cystine	C—5	Phenylalanine	F—14	Any amino acid	X—23

Amino Acid	Code	Amino Acid	Code	Amino Acid	
Glutamine	Q—6	Proline	P—15	Translation stop	*—24
Glutamic acid (glutamate)	E—7	Serine	S—16	Gap of indeterminate length	- —25
Glycine	G—8	Threonine	T—17	Unknown or any integer not in table	?—0
Histidine	H—9	Tryptophan	W—18		

Description

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

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

Examples

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

s =
MATLAB
```

See Also

Bioinformatics Toolbox functions `aminolookup`, `aa2int`, `int2nt`, `nt2int`

Purpose Convert a nucleotide sequence from an integer to a letter representation

Syntax

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

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

Arguments

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

Mapping Nucleotide Integers to Letters

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

Description

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

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

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

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

Examples

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

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

s =
   ACTGCTAGC
```

Define a symbol for unknown numbers 16 and greater.

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

s =
ACT#CT#GC
```

See Also

Bioinformatics Toolbox function `aa2int`, `baselookup`, `int2aa`, `nt2int`

Purpose

Estimate the isoelectric point for an amino acid sequence

Syntax

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

isoelectric(..., 'PKVals', PKValsValue)
isoelectric(..., 'Charge', ChargeValue)
isoelectric(..., 'Chart', ChartValue)
```

Arguments

SeqAA	Amino acid sequence. Enter a character string or a vector of integers from the table . Examples: 'ARN' or `[1 2 3]`.
PKValsValue	Property to provide alternative pK values.
ChargeValue	Property to select a specific pH for estimating charge. Enter a number between `0` and `14`. The default value is `7.2`.
ChartValue	Property to control plotting a graph of charge versus pH. Enter `true` or `false`.

Description

`isoelectric` estimates the isoelectric point (the pH at which the protein has a net charge of zero) for an amino acid sequence and it estimates the charge for a given pH (default is pH 7.2). The estimates skewed by the underlying assumptions that all amino acids are fully exposed to the solvent, that neighboring peptides have no influence on the pK of any given amino acid, and that the constitutive amino acids, as well as the N- and C-termini, are unmodified. Cysteine residues participating in disulfide bridges also affect the true pI and are not considered here.

By default, isoelectric uses the EMBOSS amino acid pK table, or you can substitute other values using the property PKVals.

- If the sequence contains ambiguous amino acid characters (b z * –), isoelectric ignores the characters and displays a warning message.

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

- If the sequence contains undefined amino acid characters (i j o) , isoelectric ignores the characters and displays a warning message.

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

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

isoelectric(..., 'PKVals', *PKValsValue*) uses the alternative pK table stored in the text file *PKValValues*. For an example of a pK text file, see the file Emboss.pK.

```
N_term 8.6
K 10.8
R 12.5
H 6.5
D 3.9
E 4.1
C 8.5
Y 10.1
C_term 3.6
```

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

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

Example

```
% Get a sequence from PDB and estimate the isoelectric point.
pdbSeq = getpdb('1CIV', 'SequenceOnly', true)
% then estimate its isoelectric point
isoelectric(pdbSeq)

% plot the charge against the pH for a short polypeptide sequence
isoelectric('PQGGGGWGQPHGGGWGQPHGGGGWGQGGSHSQG', 'CHART', true)

% Get the Rh blood group D antigen from NCBI and calculates
% its charge at pH 7.3 (typical blood pH)
gpSeq = getgenpept('AAB39602')
[pI Charge] = isoelectric(gpSeq, 'Charge', 7.38)
```

See Also

Bioinformatics functions `aacount`, `molweight`

joinseq

Purpose

Join two sequences to produce the shortest supersequence

Syntax

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

Arguments

`SeqNT1, SeqNT2`	Nucleotide sequences.

Description

`joinseq(SeqNT1, SeqNT2)` creates a new sequence that is the shortest supersequence of `Seq1` and `Seq2`. If there is no overlap between the sequences, then `SeqNT2` is concatenated to the end of `SeqNT1`. If the length of the overlap is the same at both ends of the sequence, then the overlap at the end of `SeqNT1` and the start of `SeqNT2` is used to join the sequences.

If `SeqNT1` is a subsequence of `SeqNT2`, then `SeqNT2` is returned as the shortest supersequence and vice versa.

Examples

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

joined =
    ACGTAAATGCA
```

See Also

MATLAB functions `cat`, `paren`, `strcat`, `strfind`

Purpose

Display a box plot for microarray data

Syntax

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

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

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

Description

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

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

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

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

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

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

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

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

`H = maboxplot(...)` returns the handle of the box plot axes.

`[H, HLines] = maboxplot(...)` returns the handles of the lines used to separate the different blocks in the image.

Examples

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

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

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

See Also

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

Statistics Toolbox function `boxplot`

Purpose

Display a spatial image for microarray data

Syntax

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

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

Description

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

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

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

`maimage(..., 'HandleGraphicsPropertyName' PropertyValue)` allows you to pass optional Handle Graphics property `name`/`property` value pairs to the function. For example, a name/value pair for color could be `maimage(..., 'color' 'r')`.

`H = maimage(...)` returns the handle of the image.

`[H, HLines] = maimage(...)` returns the handles of the lines used to separate the different blocks in the image.

Examples

```
madata = gprread('mouse_a1wt.gpr');
maimage(madata,'F635 Median');

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

See Also

Bioinformatics Toolbox functions `mairplot`, `maloglog`

mairplot

Purpose

Display intensity versus ratio scatter plot for microarray signals

Syntax

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

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

Arguments

X, Y	
FactorLinesValue	Property to specify a factor of change.
TitleValue	Property to specify a title for the plot.
LabelsValue	Property to specify labels for the plot.
HandleGraphicsValue	Property to pass optional property name/value pairs from Handle Graphics.

Description

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

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

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

`mairplot(..., 'Labels', LabelsValue)` allows you to specify a cell array of labels for the data. If labels are defined, then clicking a point on the plot shows the label corresponding to that point.

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

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

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

Examples

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

See Also

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

maloglog

Purpose

Create a loglog plot of microarray data

Syntax

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

maloglog(..., 'FactorLines', FactorLinesValue)
maloglog(..., 'Title', TitleValue)
maloglog(..., 'Labels', LablesValues)
maloglog(..., HandleGraphics name/value)
H = maloglog(...)
```

Description

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

`maloglog(..., 'FactorLines', N)` adds lines showing a factor of *N* change.

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

`maloglog(..., 'Labels', LabelsValues)` allows you to specify a cell array of labels for the data. If `Labels` is defined, then clicking a point on the plot shows the label corresponding to that point.

`maloglog(..., HandleGraphics name/value)` allows you to pass optional Handle Graphics property name/property value pairs to the function.

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

Examples

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

See Also

Bioinformatics Toolbox functions `maboxplot`, `mairplot`

Purpose

Smooth microarray data using the Lowess method

Syntax

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

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

Description

`YSmooth = malowess(X, Y, 'PropertyName', PropertyValue)` smooths scatter data `X`, `Y` using the Lowess smoothing method. The default window size is 10% of the length of `X`.

`malowess(..., 'Order', OrderValue)` allows you to choose the order of the algorithm. This can be `1` (linear fit) or `2` (quadratic fit). The default order is `1`. Note that the MATLAB Curve Fitting Toolbox refers to Lowess smoothing of order `2` as Loess smoothing.

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

`malowess(..., 'Span', SpanValue)` allows you to modify the window size for the smoothing function. If `Span` is less than `1`, the window size is taken to be a fraction of the number of points in the data. If `Span` is greater than `1`, the window is of size `Span`. The default value is `0.05`,which corresponds to a window size equal to 5% of the number of points in `X`.

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
cy3data = maStruct.Data(:,4);
cy5data = maStruct.Data(:,13);
[x,y] = mairplot(cy3data, cy5data);
drawnow
ysmooth = malowess(x,y);
hold on;
plot(x,ysmooth,'rx');
ynorm = y - ysmooth;
```

See Also Bioinformatics Toolbox functions `mairplot`, `maloglog`, `mamadnorm`, `mameannorm`

Purpose

Normalize microarray data by median absolute deviation (MAD)

Syntax

```
XNorm = mamadnorm(X, 'PropertyName', PropertyValue)
[XNorm, MAD] = mamadnorm(X)

mamadnorm(..., 'Global', GlobalValue)
```

Description

`XNorm = mamadnorm(X, 'PropertyName', PropertyValue)` divides the values in each column of `X` by the MAD of the column.

`[XNorm, MAD] = mamadnorm(X)` returns the median absolute deviation.

`mamadnorm(..., 'Global', GlobalValue)` if `Global` is `true`, divides the values in the data set by the global MAD, as opposed to the MAD of each column of the data.

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
Red = maStruct.Data(:,4);
Green = maStruct.Data(:,13);
maloglog(Red,Green,'factorlines',true)
figure
normRed = mamadnorm(Red);
normGreen = mamadnorm(Green);
maloglog(normRed,normGreen,'title','Normalized',...
         'factorlines',true)
```

See Also

Bioinformatics Toolbox functions `malowess`, `mameannorm`

mameannorm

Purpose

Normalize microarray data using the global mean

Syntax

```
XNorm = mameannorm(X, 'PropertyName', PropertyValue)
[XNorm, ColMean] = mameannorm(X)

mameannorm(..., 'Prctile', PrctileValue)
mameannorm(..., 'Global', GlobalValue)
```

Description

`XNorm = mameannorm(X, 'PropertyName', PropertyValue)` divides the values in each column of `X` by the mean column intensity.

`[XNorm, ColMean] = mameannorm(X)` returns the column means used to scale the data.

`mameannorm(..., 'Prctile', PrctileValue)` scales the mean of the percentile `Prctile` for the data. This is useful to prevent large outliers from skewing the normalization.

`mameannorm(..., 'Global', GlobalValue)` if `Global` is `true`, divides the values in the data set by the global mean of the data, as opposed to the mean of each column of the data.

Examples

```
maStruct = gprread('mouse_a1wt.gpr');
Red = maStruct.Data(:,4);
Green = maStruct.Data(:,13);
maloglog(Red,Green,'factorlines',true)
figure
normRed = mameannorm(Red);
normGreen = mameannorm(Green);
maloglog(normRed,normGreen,'title','Normalized',...
         'factorlines',true)
```

See Also

Bioinformatics Toolbox functions `malowess`, `mamadnorm`

Purpose

Creates a Principal Component plot of expression profile data

Syntax

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

Arguments

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

Description

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

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

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

Examples

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

See Also

Bioinformatics Toolbox function `clustergram`

Statistical Toolbox function `princomp`

Purpose

Calculate the molecular weight of an amino acid sequence

Syntax

`molweight(SeqAA)`

Arguments

`SeqAA`	Amino acid sequence. Enter a character string or a vector of integers from the table . Examples: `'ARN'`, `[1 2 3]`. You can also enter a structure with the field `Sequence`.

Description

`molweight(SeqAA)` calculates the molecular weight for the amino acid sequence `SeqAA`.

Examples

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

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


mwcchu =
  1.1749e+004
```

See Also

Bioinformatics Toolbox functions `aacount`, `atomiccomp`

Purpose

Read a multiple sequence alignment file

Syntax

```
S = multialingread(File)
[Headers, Sequences] = multialignread(File)
```

Arguments

File	Multiple sequence alignment file (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text of a multiple sequence alignment file. You can read common multiple alignment file types, such as ClustalW (`.aln`) and GCG (`.msf`)

Description

`S = multialingread(`*File*`)` reads a multiple sequence alignment file. The file contains multiple sequence lines that start with a sequence header followed by an optional number (not used by `multialignread`) and a section of the sequence. The multiple sequences are broken into blocks with the same number of blocks for every sequence. (for an example, type `open aagag.aln`). The output S is a structure array where `S.Header` contains the header information and `S.Sequence` contains the amino acid or nucleotide sequences.

`[Headers, Sequences] = multialignread(`*File*`)` reads the file into separate variables `Headers` and `Sequences`.

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
```

Create a phylogenetic tree with multiply aligned sequences.

```
Sequences = multialignread('aagag.aln')
distances = seqpdist(Sequences)
tree = seqlinkage(distances)
phytreetool(tree)
```

See Also

Bioinformatics Toolbox function `fastaread`, `gethmmalignment`

Purpose Count the number of n-mers in a nucleotide or amino acid sequence

Syntax `nmercount(Seq, Length)`

Arguments

Seq	Nucleotide or amino acid sequence. Enter a character string or a structure with the field `Sequence`.
Length	Length of n-mer to count. Enter an integer.

Description `nmercount(Seq, Length)` counts the number of n-mers or patterns of a specific length in a sequence.

Examples Count the number of n-mers in an amino acid sequence and display the first six rows in the cell array.

```
S = getgenpept('AAA59174','SequenceOnly',true)
nmers = nmercount(S,4);
nmers(1:6,:)

ans =
    'apes'    [2]
    'dfrd'    [2]
    'eslk'    [2]
    'frdl'    [2]
    'gnys'    [2]
    'lkel'    [2]
```

See Also Bioinformatics Toolbox functions `basecount`, `codoncount`, `dimercount`

nt2aa

Purpose

Convert a sequence of nucleotides to a sequence of amino acids

Syntax

```
SeqAA = nt2aa(SeqNT, 'PropertyName', PropertyValue)

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

Arguments

`SeqNT`	DNA nucleotide sequence. Enter a character string with only the characters `A`, `T`, `C`, and `G`. You cannot use the character `U`, ambiguous characters, or a hyphen. You can also enter a structure with the field `Sequence`.
FrameValue	Property to select a frame. Enter `1`, `2`, `3`, or `'ALL'`. The default value is `1`.
GeneticCodeValue	Property to select a genetic code. Enter a code number or code name from the table Genetic Code on page 6-140. If you use a code name, you can truncate the name to the first two characters of the name.
AlternativeValue	Property to control the use of alternative codons. Enter either `true` or `false`. The default value is `true`.

Genetic Code

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

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

Description

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

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

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

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

If this option is set to `false`, then alternative start codons at the start of a sequence are translated to their corresponding amino acids for the genetic code that you use, which might not necessarily be methionine. For example, in the human mitochondrial genetic code, `AUA` and `AUU` are known to be alternative start codons.

For more details of alternative start codons, see

```
www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=t#SG1
```

Examples

Convert the gene ND1 on the human mitochondria genome.

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

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

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

See Also

Bioinformatics Toolbox functions `aa2nt`, `baselookup`, `geneticcode`, `revgeneticcode`

Purpose

Convert a nucleotide sequence from a letter to an integer representation

Syntax

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

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

Arguments

`SeqNT`	Nucleotide sequence represented with letters. Enter a character string from the table Mapping Nucleotide Letters to Integers below. Integers are arbitrarily assigned to IUB/IUPAC letters. If the property `ACGTOnly` is `true`, you can only enter the characters `A`, `C`, `T`, `G`, and `U`.
UnknownValue	Property to select the integer for unknown characters. Enter an integer. Maximum value is `255`. Default value is `0`.
ACGTOnlyValue	Property to control the use of ambiguous nucleotides. Enter either `true` or `false`. Default value is `false`.

Mapping Nucleotide Letters to Integers

Base	**Code**	**Base**	**Code**	**Base**	**Code**
Adenosine	A—1	A, G (purine)	R—6	T, G, C	R—12
Cytidine	C—2	T, C (pyrimidine)	Y—7	A, T, G	Y—13
Guanine	G—3	G, T (keto)	K—8	A, T, C	K—14

Base	Code	Base	Code	Base	Code
Thymidine	T—4	A, C (amino)	M—9	A, G, C	V—15
Uridine	U—4	G, C (strong)	S—10	Gap of indeterminate length	- —16
A, T, G, C (any)	N—5	A, T (weak)	W—11	Unknown (default)	*—0

Description

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

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

`nt2int(SeqNT,'ACGTOnly', ACGTONlyValue)` if `ACGTOnly` is `true`, the ambiguous nucleotide characters (N, R, Y, K, M, S, W, B, D, H, and V) are represented by the unknown nucleotide number.

Examples

Convert a nucleotide sequence with letters to integers.

```
s = nt2int('ACTGCTAGC')

s =
     1     2     4     3     2     4     1     3     2
```

See Also

Bioinformatics Toolbox function `aa2int`, `baselookup`, `int2aa`, `int2nt`

Purpose

Plot the density of nucleotides along a sequence

Syntax

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

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

Description

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

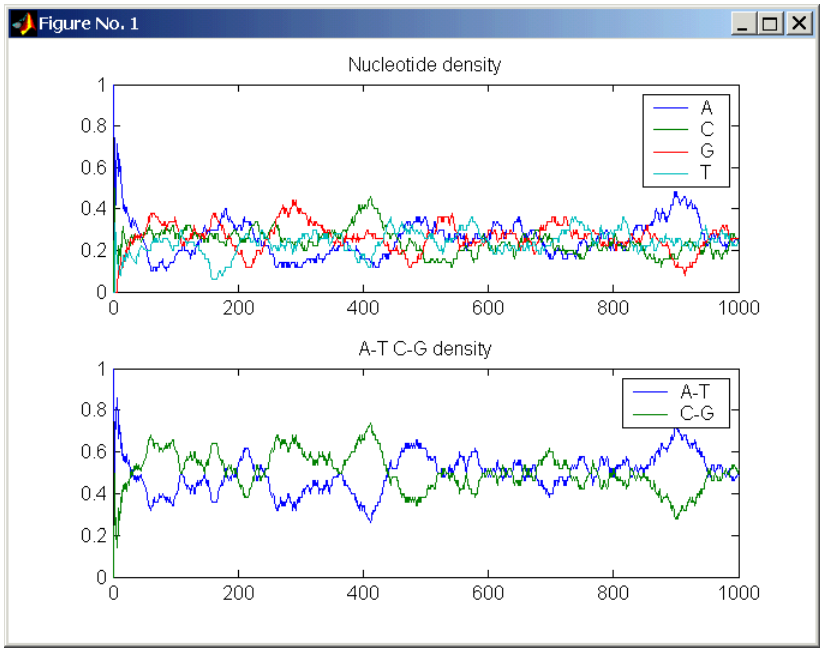
`Denstity = ntdensity(SeqNT, '`*`PropertyName`*`', `*`PropertyValue`*`)` returns a MATLAB structure with the density of nucleotides `A`, `C`, `G`, and `T`.

`ntdensity(..., 'Window', `*`WindowValue`*`)` uses a window of length `Window` for the density calculation. The default value is `length(SeqNT)/20`.

`[Density, HighCG] = ntdensity(..., 'CGThreshold',` *`CGThresholdValue`*`)` returns indices for regions where the `CG` content of `SeqNT` is greater than `CGThreshold`. The default value for `CGThreshold` is `5`.

Examples

```
s = randseq(1000, 'alphabet', 'dna');
ndensity(s)
```



See Also

Bioinformatics Toolbox functions `basecount`, `codoncount`, `dimercount`

MATLAB function `filter`

Purpose

Return a NUC44 scoring matrix for nucleotide sequences

Syntax

```
ScoringMatrix = nuc44
```

Description

The `nuc44` scoring matrix uses ambiguous nucleotide codes and probabilities rounded to the nearest integer.

Scale = `0.277316`

Expected score = `-1.7495024`, Entropy = `0.5164710` bits

Lowest score = `-4`, Highest score = `5`

Order: `A C G T R Y K M S W B D H V N`

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

nwalign

Purpose

Globally align two sequences using the Needleman-Wunsch algorithm

Syntax

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

nwalign(...,'ScoringMatrix', ScoringMatrixValue)
nwalign(...,'GapOpen', GapOpenValue)
nwalign(...,'ExtendGap', ExtendGapValue)
nwalign(...,'Alphabet', AlphabetVlaue)
```

Arguments

`Seq1, Seq2`	Nucleotide or amino acid sequence. Enter a character string or a structure with the field `Sequence`.
ScoringMatrixValue	Enter the name of a scoring matrix. Values are 'PAM40', 'PAM250', DAYHOFF, GONNET, 'BLOSUM30' increasing by 5 to 'BLOSUM90', 'BLOSUM62', or 'BLOSUM100'. The default value when `AlphabetValue = 'aa'` is 'BLOSUM50', while the default value when `AlphabetValue = 'nt'` is `nuc44`.
GapOpenValue	Property to specify the penalty for opening a gap. The default value is `8`.
ExtendGapValue	Property to specify the penalty for extending a gap. If `ExtendGap` is not specified, then the default value is equal to `GapOpen`.
AlphabetValue	Property to select the type of sequence. Value is either'AA' or 'NT'. The default value is 'AA'.

Description

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

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

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

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

`nwalign(..., 'Alphabet',` *AlphabetValue*`)` specifies amino acid or nucleotide sequences.

Examples

Globally align two amino acid sequences.

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

Score =
    7.3333

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

Select scoring matrix and gap penalty.

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

Score =
    2.3333
Alignment =

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

See Also

Bioinformatics Toolbox functions `blosum`, `dayhoff`, `gonnet`, `nt2aa`, `showalignment`, `swalign`

Purpose

Find palindromes in a sequence

Syntax

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

palindromes(..., 'Length', LengthValue)
palindromes(..., 'Complement', ComplementValue)
```

Description

`[Position, Length] = palindromes(SeqNT,` *'PropertyName'*, *PropertyValue*`)` finds all palindromes in sequence `SeqNT` with a length greater than or equal to `6`, and returns the starting indices, `Position`, and the lengths of the palindromes, `Length`.

`[Position, Length, Pal] = palindromes(SeqNT)` also returns a cell array `Pal` of the palindromes.

`palindromes(..., 'Length',`*LengthValue*`)` finds all palindromes longer than or equal to `Length`. The default value is `6`.

`palindromes(..., 'Complement',` *ComplementValue*`)` finds complementary palindromes if `Complement` is `true`, that is, where the elements match their complementary pairs `A`-`T`(or `U`) and `C`-`G` instead of an exact nucleotide match.

Examples

```
[p,l,s] = palindromes('GCTAGTAACGTATATATAAT')

p =
    11
    12
l =
     7
     7
s =
    'TATATAT'
    'ATATATA'
```

```
[pc,lc,sc] = palindromes('GCTAGTAACGTATATATAAT',...
                         'Complement',true);
```

Find the palindromes in a random nucleotide sequence.

```
a = randseq(100)

a =
TAGCTTCATCGTTGACTTCTACTAA
AAGCAAGCTCCTGAGTAGCTGGCCA
AGCGAGCTTGCTTGTGCCCGGCTGC
GGCGGTTGTATCCTGAATACGCCAT

[pos,len,pal]=palindromes(a)

pos =
    74
len =
     6
pal =
    'GCGGCG'
```

See Also

Bioinformatics Toolbox functions `seqrcomplement`, `seqshowwords`

MATLAB functions `regexp`, `strfind`

Purpose

Return a PAM scoring matrix

Syntax

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

ScoringMatrix = pam(..., 'Extended', ExtendedValue)
ScoringMatrix = pam(..., 'Order', 'OrderString')
```

Arguments

`N`	Enter values `10:10:500`. The default ordering of the output is `A R N D C Q E G H I L K M F P S T W Y V B Z X *`. Entering a larger value for N to allow sequence alignments with larger evolutionary distances.
ExtendedValue	Property to add ambiguous characters to the scoring matrix. Enter either `true` or `false`. Default is `false`.
OrderString	Property to control the order of amino acids in the scoring matrix. Enter a string with at least the 20 standard amino acids.

Description

`ScoringMatrix = pam(N, 'PropertyName',` *PropertyValue*`)` returns a PAM scoring matrix for amino acid sequences.

`[ScoringMatrix, MatrixInfo] = pam(N)` returns a structure with information about the PAM matrix. The fields in the structure are `Name`, `Scale`, `Entropy`, `Expected`, and `Order`.

`B = pam(..., 'Extended', '`*ExtendedValue*`')` if `Extended` is `true`, returns a scoring matrix with the 20 amino acid characters, the ambiguous characters, and stop character (`B`, `Z`, `X`, `*`), . If `Extended` is `false`, only the standard 20 amino acids are included in the matrix.

`B = pam(..., 'Order', '`*OrderString*`')` returns a PAM matrix ordered by the amino acid sequence in `Order`. If `Order` does not contain the extended characters B, Z, X, and *, then these characters are not returned.

PAM50 substitution matrix in 1/2 bit units, Expected score = -3.70, Entropy = 2.00 bits, Lowest score = -13, Highest score = 13.

PAM250 substitution matrix in 1/3 bit units, Expected score = -0.844, Entropy = 0.354 bits, Lowest score = -8, Highest score = 17.

Examples

Get the PAM matrix with N = 50.

```
PAM50 = pam(50)

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

See Also

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

Purpose

Visualize the intermolecular distances in a PDB file

Syntax

```
pdbdistplot('PDBid')
pdbdistplot('PDBid', Distance)
```

Arguments

PDBid	Unique identifier for a protein structure record. Each structure in the PDB is represented by a 4-character alphanumeric identifier. For example, `4hhb` is the identification code for hemoglobin.
`Distance`	Threshold distance in Angstroms shown on a spy plot. Default value is 7.

Description

`pdbdistplot` displays the distances between atoms and amino acids in a PDB structure.

`pdbdistplot('`*PDBid*`')` retrieves the entry `PDBid` from the Protein Data Bank (PDB) database and creates a heat map showing interatom distances and a spy plot showing the residues where the minimum distances apart are less than 7 Angstroms. `PDBid` can also be the name of a variable or a file containing a PDB MATLAB structure.

`pdbdistplot('PDBid', Distance)` specifies the threshold distance shown on a spy plot.

Examples

Show spy plot at 7 Angstroms of the protein cytochrome C from albacore tuna.

```
pdbdistplot('5CYT');
```

Now take a look at 10 Angstroms.

```
pdbdistplot('5CYT',10);
```

See Also

Bioinformatics Toolbox functions `getpdb`, `pdbread`

Purpose

Read data from a Protein Data Bank (PDB) file

Syntax

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

Arguments

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

Description

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

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

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

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

For more information about the PDB format, see

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

Examples

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

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

See Also

Bioinformatics Toolbox functions genpeptread, getgenpept, getpdb, pirread

pdist (phytree)

Purpose

Calculate the pairwise patristic distances in a phytree object

Syntax

```
D = pdist(Tree)
D = pdist(..., 'Nodes', NodeValue)
D = pdist(... ,'Squareform', SquareformValue)
[D,C] = pdist(Tree)
```

Arguments

Tree	Phylogenetic tree object created with the function `phytree`.
NodeValue	Property to select the nodes. `Enter either` `'leaves'` (default) or 'all'.
SquareformValue	Property to control creating a square matrix.

Description

`D = pdist(`*`Tree`*`)` returns a vector (`D`) containing the patristic distances between all pairs of leaf nodes in a `phygtree` object (`Tree`). The patristic path distances are computed by following paths through the branches of the tree and adding the patristic branch distances originally created with seqlinkage.

The output vector `D` is arranged in the order `((2,1),(3,1),...,(M,1),(3,2),...(M,3),.....(M,M-1))` (the lower left triangle of the full `M`-by-`M` distance matrix). To get the distance between the `I`th and `J`th nodes (`I > J`), use the formula `D((J-1)*(M-J/2)+I-J)`. `M` is the number of leaves).

`D = 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`).

`D = 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.

`[D,C] = pdist(`*`Tree`*`)` returns in C the index of the closest common parent nodes for every possible pair of query nodes.

Examples

```
% get the tree distances between pairs of leaves
tr = phytreeread('pf00002.tree')
dist = pdist(tr,'nodes','leaves','squareform',true)
```

See Also

Bioinformatics Toolbox function `seqpdist`, `seqlinkage` and the `phytree` object methods `phytree`, `phytreetool`

pfamhmmread

Purpose

Read data from a PFAM-HMM file

Syntax

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

Arguments

File	PFAM-HMM formatted file. Enter a filename, a path and filename, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text of a PFAM-HMM file.

Description

`pfamhmmread` reads data from a PFAM-HHM formatted file (file saved with the function `gethmmprof`) and creates a MATLAB structure.

`Data = pfamhmmread('`*`File`*`')` reads from *File* a Hidden Markov Model described by the PFAM format, and converts it to the MATLAB structure `Data`, containing fields corresponding to annotations and parameters of the model. For more information about the model structure format, see `hmmprofstruct`. *File* can also be a URL or a MATLAB cell array that contains the text of a PFAM formatted file.

`pfammread` is based on the HMMER 2.0 file formats.

Examples

```
pfamhmmread('pf00002.ls')

site='http://www.sanger.ac.uk/';
pfamhmmread([site 'cgi-bin/Pfam/download_hmm.pl?id=7tm_2'])
```

See Also

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

Purpose

Object constructor for a phylogenetic tree object

Syntax

```
Tree = phytree(B)
Tree = phytree(B, D)
Tree = phytree(B, C)
Tree = phytree(BC)
Tree = phytree(..., N)
```

Arguments

`B`	Numeric array of size `[NUMBRANCHES X 2]` in which every row represents a branch of the tree. It contains two pointers to the branch or leaf nodes.
`C`	Column vector with distances for every branch.
`D`	Column vector with distances from every node to their parent branch.
`BC`	Combined matrix with pointers to branch or leaves, and distances of branches.
`N`	Cell array with the names of leafs and branches.

Description

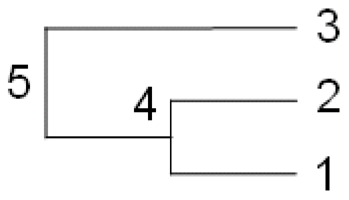
`Tree = phythree(B)` creates an ultrametric phylogenetic tree object.

`B` is a numeric array of size `[NUMBRANCHES X 2]` in which every row represents a branch of the tree and it contains two pointers to the branch or leave nodes which are its children.

Leaf nodes are numbered from `1` to `NUMLEAVES` and branch nodes are numbered from `NUMLEAVES + 1` to `NUMLEAVES + NUMBRANCHES`. Note that because only binary trees are allowed, `NUMLEAVES = NUMBRANCHES + 1`.

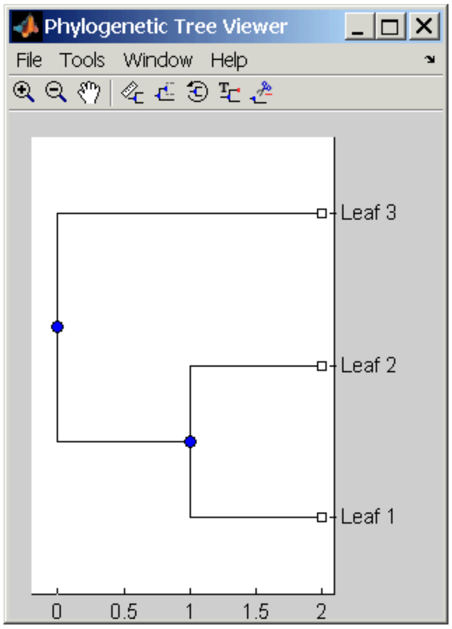
Branches are defined in chronological order (for example, `B(i,:) > NUMLEAVES + i`). As a consequence, the first row can only have pointers to leaves, and the last row must represent the root branch. Parent-child distances are set to `1`, unless the child is a leaf and to satisfy the ultrametric condition of the tree its distance is increased.

Given a tree with 3 leafs and 2 branches as an example.



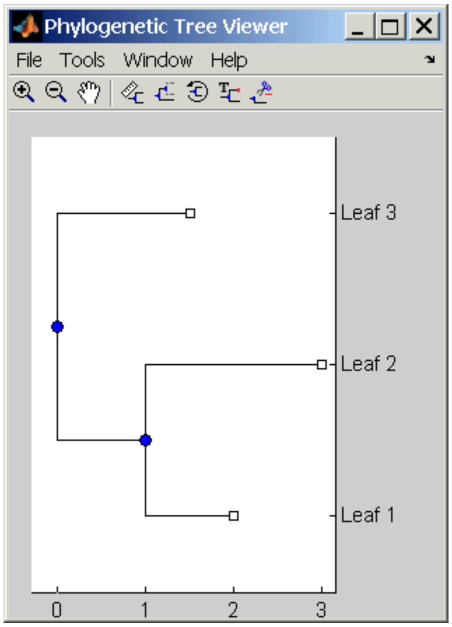
In the MATLAB Command window, type

```
B = [1 2 ; 3 4]
tree = phytree(B)
view(tree)
```



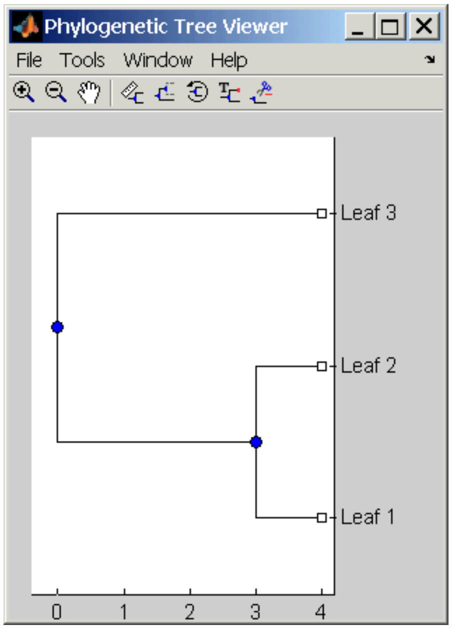
`Tree = phytree(B, D)` creates an additive phylogenetic tree object with branch distances defined by `D`. `D` is a numeric array of size `[NUMNODES X 1]` with the distances of every child node (leaf or branch) to its parent branch equal to `NUMNODES = NUMLEAVES + NUMBRANCHES`. The last distance in `D` is the distance of the root node and is meaningless.

```
  b = [1 2 ; 3 4 ]: d = [1 2 1.5 1 0]
  view(phytree(b,d))
```



`Tree = phytree(B, C)` creates an ultrametric phylogenetic tree object with branch distances defined by `C`. C is a numeric array of size `[NUMBRANCHES X 1]` with the coordinates of every branch node. In ultrametric trees all the leaves are at the same location (for example, same distance to the root).

```
  b = [1 2 ; 3 4]; c = [1 4]'
  view(phytree(b,c))
```



`Tree = phytree(BC)` creates an ultrametric phylogenetic binary tree object with branch pointers in `BC(:,[1 2])` and branch coordinates in `BC(:,3)`. Same as `phytree(B,C)`.

`Tree = phytree(..., N)` specifies the names for the leaves and/or the branches. `N` is a cell of strings. If `NUMEL(N)==NUMLEAVES`, then the names are assigned chronologically to the leaves. If `NUMEL(N)==NUMBRANCHES`, the names are assigned to the branch nodes. If `NUMEL(N)==NUMLEAVES + NUMBRANCHES`, all the nodes are named. Unassigned names default to `'Leaf #'` and/or `'Branch #'` as required.

`Tree = phytree` creates an empty phylogenetic tree object.

Examples

Create phylogenetic tree for a set of multiply aligned sequences.

```
Sequences = multialignread('aagag.aln')
distances = seqpdist(Sequences)
tree = seqlinkage(distances)
```

```
phytreetool(tree)
```

See Also

Bioinformatics Toolbox functions phytreeread, phytreetool, phytreewrite, seqlinkage, seqpdist, and the phytree object methods get (phytree), select

phytreeread

Purpose

Read phylogenetic tree files

Syntax

`Tree = phytreeread(`*`File`*`)`

Arguments

File	Newick formatted tree files (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text for a file.
Tree	`phytree` object created with the function `phytree`.

Description

`Tree = phytreeread(Filename)` reads a Newick formatted tree file and returns a phytree object in the MATLAB workspace with data from the file.

The NEWICK tree format can be found at

```
http://evolution.genetics.washington.edu/
       phylip/newicktree.html
```

Note This implementation only allows binary trees. Non-binary trees are translated into a binary tree with extra branches of length 0.

Examples

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

See Also

Bioinformatics Toolbox functions `gethmmtree`, `phytreetool`, `phytreewrite` and the `phytree` object method `phytree`

Purpose

View, edit, and explore phylogenetic tree data

Syntax

```
phytreetool(Tree)
phytreetool(File)
```

Arguments

Tree	`Phytree` object created with the function `phytree` or `phytreeread`.
File	Newick or ClustalW tree formatted file (ASCII text file) with phylogenetic tree data. Enter a filename, a path and filename, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text for a Newick file.

Description

`phytreetool` is an interactive GUI that allows you to view, edit, and explore phylogenetic tree data. This GUI allows branch pruning, reordering, renaming, and distance exploring. It can also open or save Newick formatted files.

`phytreetool(`*Tree*`)` loads data from a `phytree` object in the MATLAB workspace into the GUI.

`phytreetool(`*File*`)` loads data from a Newick formatted file into the GUI.

Examples

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

See Also

Bioinformatics Toolbox functions `phytreeread`, `phytreewrite` and the phytree object methods `phytree`, `plot (phytree)`, `view (phytree)`

phytreewrite

Purpose

Write a phylogenetic tree object to a Newick formatted file

Syntax

```
phytreewrite('File', Tree)
phytreewrite(Tree)
```

Arguments

File	Newick formatted file. Enter either a filename or a path and filename supported by your operating system (ASCII text file).
Tree	Phylogenetic tree object. `Tree` must be an object created with either the function `phytree` or imported using the function `phytreeread`.

Description

`phytreewrite('`*File*`', `*Tree*`)` copies the contents of a `phytree` object from the MATLAB workspace to a file. Data in the file uses the Newick format for describing trees.

The NEWICK tree format can be found at

```
http://evolution.genetics.washington.edu/
       phylip/newicktree.html
```

`phytreewrite(`*Tree*`)` opens the **Save Phylogenetic tree as** dialog box for you to enter or select a filename.

Examples

Read tree data from a Newick formatted file.

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

Remove all the 'mouse' proteins

```
ind = getbyname(tr,'mouse');
tr = prune(tr,ind);
view(tr)
```

Write pruned tree data to a file.

```
phytreewrite('newtree.tree', tr)
```

See Also

Bioinformatics Toolbox functions phytreeread, phytreetool, seqlinkage, and the phytree object methods phytree,

Purpose

Read data from a PIR file

Syntax

```
PIRData = pirread('File')
pirread('String')
```

Arguments

File	Protein Information Resource (PIR-PSD) formatted file (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text for a PIR-PSD file.
String	Character string with PIR data.

Description

`PIRData = pirread('`*File*`')` reads data from a Protein Information Resource (PIR-PSD) formatted file *File* and creates a MATLAB structure `PIRData` with the following fields:

```
Entry
EntryType
Title
Organism
Date
Accessions
Reference
Genetics
Classification
Keywords
Feature
Summary
Sequence: [1x105 char]
```

`pirread('`*String*`')` attempts to retrieve PIR data from the string `String`.

For more information on the PIR-PSD database, see

```
http://pir.georgetown.edu
```

Examples

Get protein information for cytochrome C from the PIR-PSD database, save the information in the file `cchu.txt`, and then read the information back into MATLAB.

```
getpir('cchu', 'ToFile', 'cchu.txt')
pirdata = pirread('cchu.txt')
```

See Also

Bioinformatics Toolbox functions `genpeptread`, `getpir`, `pdbread`

plot (phytree)

Purpose

Draw a phylogenetic tree

Syntax

```
plot(Tree)
plot(Tree, ActiveBranches)

plot(..., 'Type', TypeValue)
plot(..., 'Orientation', OrientationValue)
plot(..., 'BranchLabels', BranchLabelsValue)
plot(..., 'LeafLabels', LeafLabelsValue)
plot(..., 'TerminalLabels', TerminalLabelsValue)
```

Arguments

`Tree`	`phytree` object created with the function `phytree`
`ActiveBranches`	Branches veiwable in the figure window.
TypeValue	Property to select a method for drawing a phylogenetic tree. Enter `'phylogram'` , `'cladogram'`, or `'radial'`. The default value is `'phylogram'`.
OrientationValue	Property to orient a phylogram or cladogram tree. Enter `'top'`, `'bottom'`, `'left'`, or `'right'`. The default value is `'left'`.
BranchLabelsValue	Property to control displaying branch labels. Enter either `true` or `false`. The default value is `false`.
LeafLabelsValue	Property to control displaying leaf labels. Enter either `true` or `false`. The default value is `false`.
TerminalLabels	Property to control displaying terminal labels. Enter either `true` or `false`. The default value is `false`.

Description

`plot(Tree)` draws a phylogenetic tree object into a MATLAB figure as a phylogram. The significant distances between branches and nodes are in the horizontal direction. Vertical distances have no significance and are selected only for display purposes. Handles to graph elements are stored in the figure field `UserData` so that you can easily modify graphic properties.

`plot(Tree, ActiveBranches)` hides the nonactive branches and all of their descendants. `ActiveBranches` is a logical array of size `numBranches x 1` indicating the active branches.

`plot(..., 'Type',` *TypeValue*`)` selects a method for drawing a phylogenetic tree.

`plot(...,'Orientation',` *OrientationValue*`)` orients a phylogenetic tree within a figure window. The `Orientation` property is valid only for phylogram and cladogram trees.

`plot(...,'BranchLabels',` *BranchLabelsValue*`)` hides or displays branch labels placed next to the branch node.

`plot(...,'LeafLabels',` *LeafLabelsValue*`)` hides or displays leaf labels placed next to the leaf nodes.

`plot(...,'TerminalLabels', TerminalLabelsValue)` hides or displays terminal labels. Terminal labels are placed over the axis tick labels and ignored when `Type= 'radial'`.

`H = plot(...)` returns a structure with handles to the graph elements.

Examples

```
tr = phytreeread('pf00002.tree')
plot(tr,'Type','radial')
```

Graph element properties can be modified as follows:

```
h=get(gcf,'UserData')
set(h.branchNodeLabels,'FontSize',6,'Color',[.5 .5 .5])
```

See Also

Bioinformatics Toolbox functions `phytreeread`, `phytreetool`, `seqlinkage` phytree object methods phytree, view (phytree)

Purpose

Display property values for amino acid sequences

Syntax

```
proteinplot(SeqAA)
```

Arguments

`SeqAA`	Amino acid sequence or a structure with a field `Sequence` containing an amino acid sequence.

Description

`proteinplot` is a tool for analyzing a single amino acid sequence. You can use the results from `proteinplot` to compare the properties of several amino acid sequences. It displays smoothed line plots of various properties such as the hydrophobicity of the amino acids in the sequence.

Importing sequences into proteinplot

1 In the **MATLAB Command Window**, type

```
proteinplot(Seq_AA)
```

The `proteinplot` interface opens and the sequence `Seq_AA` is shown in the **Sequence** text box.

2 Alternatively, type or paste an amino acid sequence into the **Sequence** text box.

You can or 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.

The File menu allows you to Import a sequence, save the plot that you have created to a `FIG` file, you can export the data values in the figure to a workspace variable or to a MAT file, you can export the figure to a normal figure window for customizing, and you can print the figure.

The Edit menu allows you to create a new property, to reset the property values to the default values, and to modify the smoothing parameters with the Configuration Values menu item.

The View menu allows you to turn the toolbar on and off, and to add a legend to the plot.

The Tools menu allows you to zoom in and zoom out of the plot, to view Data Statistics such as mean, minimum and maximum values of the plot, and to normalize the values of the plot from `0` to `1`.

The Help menu allows you to view this document and to see the references for the sequence properties built into `proteinplot`

See Also

Bioinformatics Toolbox functions `aacount`, `atomiccomp`, `molweight`

MATLAB function `plotyy`

Purpose

Remove branch nodes from a phylogenetic tree

Syntax

```
T2 = prune(T1, Nodes)
T2 = prune(T1, Nodes, 'exclusive')
```

Arguments

`T1`	Phylogenetic tree object. See `phytree`.
`Nodes`	Nodes to remove from tree.
`exclusive`	Property to control the method of pruning.

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, 'exclusive')` removes only the descendants of the nodes listed in the vector `Nodes`. Nodes that do not have a predecessor become leaves in the list `Nodes`. In this case, pruning is the process of reducing a tree by turning some branch nodes into leaf nodes, and removing the leaf nodes under the original branch.

Examples

Load a phylogenetic tree created from a protein family

```
tr = phytreeread('pf00002.tree');
view(tr)
                 % To :
```

Remove all the 'mouse' proteins use

```
ind = getbyname(tr,'mouse');
tr = prune(tr,ind);
```

```
view(tr)
```

Remove potential outliers in the tree

```
[sel,sel_leaves] = select(tr,'criteria','distance',...
                             'threshold',.3,...
                             'reference','leaves',...
                             'exclude','leaves',...
                             'propagate','toleaves');
tr = prune(tr,~sel_leaves)
view(tr)
```

See Also

Bioinformatics Toolbox function `phytree`

ramachandran

Purpose

Draw a Ramachandran plot for PDB data

Syntax

```
ramachandran('PDBid')
ramachandran('File')
ramachandran(PDBData)
Angles = ramachandran(...)
[Angles, Handle] = ramachandran(...)
```

Arguments

PDBid	Unique identifier for a protein structure record. Each structure in the PDB is represented by a 4-character alphanumeric identifier. For example, `4hhb` is the identification code for hemoglobin.
File	Protein Data Bank (PDB) formatted file (ASCII text file). Enter a filename, a path and filename, or a URL pointing to a file. *File* can also be a MATLAB character array that contains the text for a PDB file.
`PDBData`	MATLAB structure with PDB formatted data.

Description

`ramachandran` generates a plot of the torsion angle PHI (torsion angle between the `'C-N-CA-C'` atoms) and the torsion angle PSI (torsion angle between the `'N-CA-C-N'` atoms) of the protein sequence.

`ramachandran(PDBid)` generates the Ramachandran plot for the protein with PDB code `ID`.

`ramachandran('`*File*`')` generates the Ramachandran plot for protein stored in the PDB file *File*.

`ramachandran(PDBData)` generates the Ramachandran plot for the protein stored in the structure `PDBData`, where `PDBData` is a MATLAB structure obtained by using `pdbread` or `getpdb`.

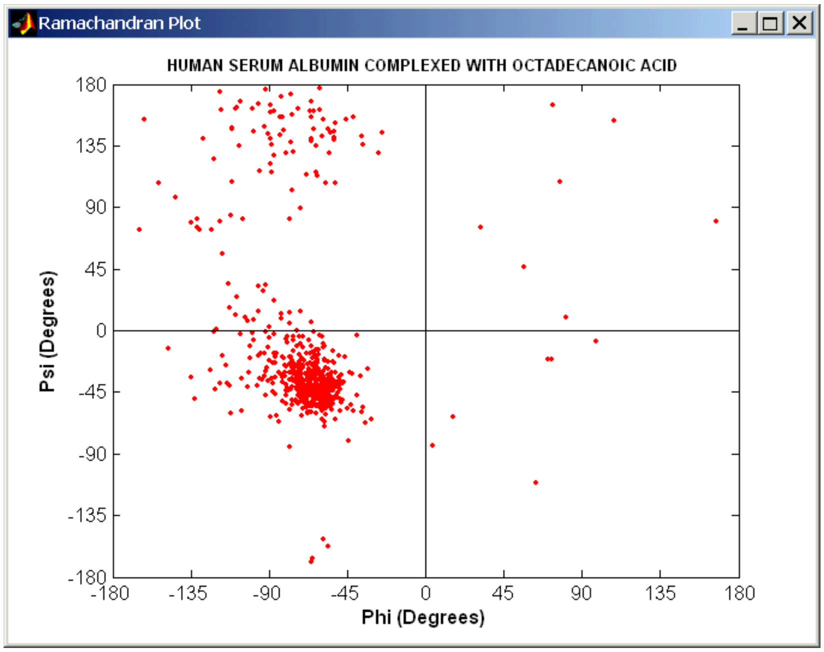
`Angles = ramachandran(...)` returns an array of the torsion angles PHI, PSI, and OMEGA for the residue sequence.

`[Angles, Handle] = ramachandran(...)` returns a handle to the plot.

Examples

Generate the Ramachandran plot for the human serum albumin complexed with octadecanoic acid.

```
ramachandran('1E7I')
```



See Also

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

Statistics Toolbox function `hmmgenerate`

Purpose

Generate a random sequence from a finite alphabet

Syntax

```
Seq = randseq(Length, 'PropertyName', PropertyValue)

randseq(..., 'Alphabet', AlphabetValue)
randseq(..., 'Weights', WeightsValue)
randseq(..., 'FromStructure', FromStructureValue)
randseq(..., 'Case',CaseValue)
randseq(..., 'DataType', DataTypeValue)
```

Arguments

`Length`	
AlphabetValue	Property to select the alphabet for the sequence. Enter `'dna'`, `'rna'`, or `'amino'`. The default value is `'dna'`.
WeightsValue	Property to specify a weighted random sequence.
FromStructureValue	Property to specify a weighted random sequence using output structures from the functions `basecount`, `dimercount`, `codoncount`, or `aacount`.
CaseValue	Property to select the case of letters in a sequence when `Alphabet` is `'char'`. Values are`'upper'` or `'lower'`. The default value is `'upper'`.
DataTypeValue	Property to select the data type for a sequence. Values are `'char'` for letter sequences, and `'uint8'` or `'double'` for numeric sequences. Creates a sequence as an array of *DataType*. The default data type is `'char'`.

Description

`randseq(...,'Alphabet',` *`AlphabetValue`*`)` generates a sequence from a specific alphabet.

`randseq(..., 'Weights',` *`WeightsValue`*`)` creates a weighted random sequence where the ith letter of the sequence alphabet is selected with weight `W(i)`. The weight vector is usually a probability vector or a frequency count vector. Note that the `i`th element of the nucleotide alphabet is given by `int2nt(i)`, and the `i`th element of the amino acid alphabet is given by `int2aa(i)`.

`randseq(..., 'FromStructure',` *`FromStructureValue`*`)` creates a weighted random sequence with weights given by the output structure from `basecount`, `dimercount`, `codoncount`, or `aacount`.

`randseq(..., 'Case',` *`CaseValue`*`)` specifies the case for a letter sequence.

`randseq(...,'DataType',` *`DataTypeValue`*`)` specifies the data type for the sequence array.

Examples

Generate a random DNA sequence.

```
randseq(20)

ans =
TAGCTGGCCAAGCGAGCTTG
```

Generate a random RNA sequence.

```
randseq(20,'alphabet','rna')

ans =
GCUGCGGCGGUUGUAUCCUG
```

Generate a random protein sequence.

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

ans =
DYKMCLYEFGMFGHFTGHKK
```

See Also MATLAB functions `rand`, `randperm`, `permute`, `datatypes`

Purpose

Display a red and green colormap

Syntax

`redgreencmap(Length)`

Arguments

`Length`	Length of the colormap. Enter either `256` or `64`. The default value is the length of the colormap of the current figure.

Description

`redgreencmap(Length)` returns an `M`-by-`3` matrix containing a red and green colormap. Low values are bright green, values in the center of the map are black, and high values are red.

`redgreencmap`, by itself, is the same length as the current colormap.

Examples

Reset the color map of the current figure.

```
pd =gprread('mouse_a1pd.gpr')
maimage(pd,'F635 Median')
colormap(redgreencmap)
```

See Also

Bioinformatics Toolbox function `clustergram`

MATLAB functions `colormap`, `colormapeditor`, `jet`

restrict

Purpose Split a sequence at a specified restriction site

Syntax

```
restrict(SeqNT, Enzyme, 'PropertyName', PropertyValue)
restrict(SeqNT, Pattern, Position)
restrict(..., 'PartialDigest', PartialDigestValue)
```

Arguments

`SeqNT`	Nucleotide sequence. Enter either a character string with the characters `A`, `T`, `G`, `C`, and ambiguous characters `R`, `Y`, `K`, `M`, `S`, `W`, `B`, `D`, `H`, `V`, `N`, or a vector of integers. You can also enter a structure with the field `Sequence`.
`Enzyme`	Enter the name of a restriction enzyme from REBASE.
`Pattern`	Enter a short nucleotide pattern. `Pattern` can be a regular expression.
`Position`	Defines the position on `Pattern` where the sequence is cut. `Position=0` corresponds to the 5' end of the `Pattern`.
PartialDigestValue	Property to specify a probability for partial digestion. Enter a value from `0` to `1`.

Description

`restrict(SeqNT, Enzyme)` cuts a sequence at restriction sites defined by a restriction enzyme in REBASE. The return values are stored in a cell array of sequences.

REBASE, the restriction enzyme database, is a collection of information about restriction enzymes and related proteins. Search REBASE for the name of a restriction enzyme at

```
http://rebase.neb.com/rebase/rebase.html
```

For more information on REBASE, go to

```
http://rebase.neb.com/rebase/rebase.html
```

`restrict(SeqNT, Pattern, Position)` cuts a sequence at restriction sites specified by a nucleotide pattern.

`restrict(..., 'PartialDigest',` *PartialDigestValue*`)` simulates a partial digest where each restriction site in the sequence has a probability `PartilDigest` of being cut.

Examples

Use the recognition pattern (sequence) `GCGC` with the point of cleavage at position 3 to cleave a nucleotide sequence.

```
Seq = 'AGAGGGGTACGCGCTCTGAAAAGCGGGAACCTCGTGGCGCTTTATTAA';
partsP = restrict(Seq,'GCGC',3);


partsP =
    'AGAGGGGTACGCG'
    'CTCTGAAAAGCGGGAACCTCGTGGCG'
    'CTTTATTAA'
```

Use the restriction enzyme `HspAI` (recognition sequence `GCGC` with the point of cleavage at position 1) to cleave a nucleotide sequence.

```
partsE = restrict(Seq,'HspAI')

partsE =
    'AGAGGGGTACG'
    'CGCTCTGAAAAGCGGGAACCTCGTGG'
    'CGCTTTATTAA'
```

See Also

Bioinformatics Toolbox function `seqshowwords`

MATLAB function `regexp`

Purpose

Get the reverse mapping for a genetic code

Syntax

```
map = revgeneticcode
revgeneticcode(GeneticCode,
               'PropertyName', PropertyValue)

revgeneticcode(..., 'Alphabet' AlphabetValue)
revgeneticcode(..., 'ThreeLetterCodes', CodesValue)
```

Arguments

GeneticCode	Enter a code number or code name from the table Genetic Code on page 6-189. If you use a code name, you can truncate the name to the first two characters of the name.
AlphabetValue	Property to select the nucleotide alphabet. Enter either `'dna'` or `'rna'`. The default value is `'dna'`.
CodesValue	Property to select one- or three-letter amino acid codes. Enter `true` for three-letter code or `false` for one-letter code.

Genetic Code

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

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

Description

`revgeneticcode` returns a structure containing reverse mappings for the genetic code.

`map = revgeneticcode` returns a structure containing the reverse mapping for the standard genetic code.

`revgeneticcode(`*GeneticCode*`)` returns a structure of the inverse mapping for alternate genetic codes.

`revgeneticcode(..., 'Alphabet' A`*lphabetValue*`)` defines the nucleotide alphabet to use in the map.

`revgeneticcode(..., 'ThreeLetterCodes', CodesValue)` returns the mapping structure with three-letter amino acid codes as field names instead of the default single-letter codes if `ThreeLetterCodes` is `true`.

Examples

```
moldcode = revgeneticcode(4,'Alphabet','rna');
wormcode = revgeneticcode('Flatworm Mitochondrial',...
                          'ThreeLetterCode',true);

map = revgeneticcode

map =
      Name: 'Standard'
         A: {'GCT'  'GCC'  'GCA'  'GCG'}
         R: {'CGT'  'CGC'  'CGA'  'CGG'  'AGA'  'AGG'}
         N: {'AAT'  'AAC'}
         D: {'GAT'  'GAC'}
         C: {'TGT'  'TGC'}
         Q: {'CAA'  'CAG'}
         E: {'GAA'  'GAG'}
         G: {'GGT'  'GGC'  'GGA'  'GGG'}
         H: {'CAT'  'CAC'}
         I: {'ATT'  'ATC'  'ATA'}
         L: {'TTA'  'TTG'  'CTT'  'CTC'  'CTA'  'CTG'}
         K: {'AAA'  'AAG'}
         M: {'ATG'}
         F: {'TTT'  'TTC'}
         P: {'CCT'  'CCC'  'CCA'  'CCG'}
         S: {'TCT'  'TCC'  'TCA'  'TCG'  'AGT'  'AGC'}
         T: {'ACT'  'ACC'  'ACA'  'ACG'}
         W: {'TGG'}
         Y: {'TAT'  'TAC'}
         V: {'GTT'  'GTC'  'GTA'  'GTG'}
    Starts: {'TAA'  'TAG'  'TGA'}
```

See Also

Bioinformatics Toolbox functions `aa2nt`, `baselookup`, `geneticcode`, `nt2aa`

rna2dna

Purpose

Convert an RNA sequence of nucleotides to a DNA sequence

Syntax

```
SeqDNA = rna2dna(SeqRNA)
```

Arguments

`SeqRNA`	Nucleotide sequence for RNA. Enter a character string with the characters `A`, `C`, `U`, `G`, and the ambiguous nucleotide bases `N`, `R`, `Y`, `K`, `M`, `S`, `W`, `B`, `D`, `H`, and `V`.

Description

`SeqDNA = rna2dna(SeqRNA)` converts any uracil nucleotides in an RNA sequence into thymine (`U-->T`), and returns in the same format as DNA. For example, if the RNA sequence is an integer sequence then so is `SeqRNA`.

Examples

```
rna2dna('ACGAUGAGUCAUGCUU')

ans =
ACGATGAGTCATGCTT
```

See Also

Bioinformatics Toolbox function `dna2rna`

MATLAB functions `strrep`, `regexp`

Purpose Read trace data from a SCF file

Syntax

```
[Sample, Probability, Comments] = scfread('File')
[A,C,T,G, ProbA, ProbC, ProbG, ProbT, Comments]
    = scfread ('File')
```

Arguments

File	SCF formatted file. Enter a filename or a path and filename.

Description

`scfread` reads data from a SCF formatted file into a MATLAB structure.

`[Sample, Probability, Comments] = scfread('`*File*`')` reads an SCF formatted file and returns the sample data in the structure `Sample`, with fields `A`, `C`, `T`, `G`, probability data in the structure `Probability`, and comment information from the file in `Comments`.

`[A,C,T,G, ProbA, ProbC, ProbG, ProbT, Comments] = scfread ('`*File*`')` reads an SCF formatted file and returns the sample data and probabilities for nucleotides in separate variables.

SCF files store data from DNA sequencing instruments. Each file includes sample data, sequence information, and the relative probabilities of each of the four bases. For more information on SCF files, see

```
http://www.mrc-lmb.cam.ac.uk/pubseq/manual/formats_unix_2.html
```

Examples

Examples of SCF files can be found at

```
ftp://ftp.ncbi.nih.gov/pub/TraceDB/example/
```

Unzip the file `bcm-example.tgz` with SCF files to your MATLAB working directory.

```
[Sample, Probability, Comments] = scfread('HCIUP1D61207.scf')

Sample =
    A: [10827x1 double]
    C: [10827x1 double]
    G: [10827x1 double]
    T: [10827x1 double]

Probability =
    prob_A: [742x1 double]
    prob_C: [742x1 double]
    prob_G: [742x1 double]
    prob_T: [742x1 double]

Comments =

SIGN=A=121,C=103,G=119,T=82
SPAC= 16.25
PRIM=0
MACH=Arkansas_SN312
DYEP=DT3700POP5{BD}v2.mob
NAME=HCIUP1D61207
LANE=6
GELN=
PROC=
RTRK=
CONV=phred version=0.990722.h
COMM=
SRCE=ABI 373A or 377
```

See Also

Bioinformatics Toolbox functions genbankread, traceplot

Purpose

Select tree branches and leaves in a phytree object

Syntax

```
S = select(T)
S = select(T, N)
[S, Selleaves, Selbranches] = select(...)

S = select(..., 'Reference', ReferenceValue)
S = select(..., 'Criteria', CriteriaValue)
S = select(..., 'Threshold', ThresholdValue)
S = select(..., 'Exclude', ExcludeValue)
S = select(..., 'Propagate', PropagateValue)
```

Arguments

Tree	Phylogenetic tree created with the function `phytree`.
`N`	Number of closest nodes to the root node.
ReferenceValue	Property to select a reference point for measuring distance.
CriteriaValue	Property to select a criteria for measuring distance.
ThresholdValue	Property to select a distance value. Nodes with distances below this value are selected.
ExcludeValue	Property to remove (exclude) branch or leaf nodes from the output. Enter 'none', 'branchs', or 'leaves'. The default value is 'none'.
PropagateValue	Property to select propagating nodes toward the leaves or the root.

Description

`S = select(Tree, N)` returns a logical vector (`S`) of size `[NumNodes x 1]` indicating the `N` closest nodes to the root node of a `phytree` object (`Tree`) where `NumNodes = NumLeaves + NumBranches`. The first criterion `select` uses is branch levels, then patristic distance (also known as tree distance). By default, `select` uses `inf` as the value of `N`, and `select(`*`Tree`*`)` returns a vector with values of `true`.

`S = select(..., 'Reference',` *`ReferenceValue`*`)` changes the reference point(s) to measure the closeness. `Reference` can be the root (default) or leaves. When using leaves, a node can have multiple distances to its descendant leaves (nonultrametric tree). If this the case, `select` considers the minimum distance to any descendant leaf.

`S = select(..., 'Criteria',` *`CriteriaValue`*`)` changes the criteria `select` uses to measure closeness. If `C = 'levels'` (default), the first criterion is branch levels and then patristic distance. If `C = 'distance'`, the first criterion is patristic distance and then branch levels.

`S = select(..., 'Threshold',` *`ThresholdValue`*`)` selects all the nodes where closeness is less than or equal to the threshold value `V`. Notice that you can also use either of the properties `'criteria'` or `'reference'`, if `N` is not specified, then `N = infF`; otherwise you can limit the number of selected nodes by `N`.

`S = select(..., 'Exclude',` *`ExcludeValue`*`)` sets a postfilter that excludes all the branch nodes from `S` when `E='branches'` or all the leaf nodes when `E='leaves'`. The default is `'none'`.

`S = select(..., 'Propagate',` *`PropagateValue`*`)` activates a postfunctionality that propagates the selected nodes to the leaves when `P=='toleaves'` or toward the root finding a common ancestor when `P == 'toroot'`. The default value is `'none'`. `P` may also be `'both'`. The `'Propagate'` property acts after the `'Exclude'` property.

`[S, Selleaves, Selbranches] = select(...)` returns two additional logical vectors, one for the selected leaves and one for the selected branches.

Examples

```
% Load a phylogenetic tree created from a protein family:
tr = phytreeread('pf00002.tree');

% To find close products for a given protein (e.g. vips_human):
ind = getbyname(tr,'vips_human');
[sel,sel_leaves] = select(tr,'criteria','distance',...
                          'threshold',0.6,'reference',ind);
view(tr,sel_leaves)

% To find potential outliers in the tree, use
[sel,sel_leaves] = select(tr,'criteria','distance',...
                             'threshold',.3,...
                             'reference','leaves',...
                             'exclude','leaves',...
                             'propagate','toleaves');
view(tr,~sel_leaves)
```

See Also

The Bioinformatics Toolbox functions phytree, phytreetool

phytree object methods pdist, get.

seq2regexp

Purpose Convert a sequence with ambiguous characters to a regular expression

Syntax `seq2regexp(Seq)`

Arguments

`Seq`	Nucleotide or amino acid sequence.

Nucleotide Conversions

Nucleotide Letter	Nucleotide	Nucleotide Letter	Nucleotide
`A—A`	Adenosine	`S—[GC]`	(Strong)
`C—C`	Cytosine	`W—[AT]`	(Weak)
`G—G`	Guanine	`B—[GTC]`	
`T—T`	Thymidine	`D—[GAT]`	
`U—U`	Uridine	`H—[ACT]`	
`R—[GA]`	(Purine)	`V—[GCA]`	
`Y—[TC]`	(Pyrimidine)	`N—[AGCT]`	Any nucleotide
`K—[GT]`	(Keto)	`- — -`	Gap of indeterminate length
`M—[AC]`	(Amino)	`?—?`	Unknown

Amino Acid Conversion

Amino Acid Letter	Description
B—[DN]	Aspartic acid or asparagine
Z—[EQ]	Glutamic acid or glutamine
X—[ARNDCQEGHILKMFPSTWYV]	Any amino acid

Description

`seq2regexp(Seq)` converts ambiguous nucleotide or amino acid symbols in a sequence into a regular expression format using IUB/IUPAC codes.

Examples

Convert a nucleotide sequence into a regular expression.

```
r = seq2regexp('ACWTMAN')

r =
AC[AT]T[AC]A[AGCT]
```

See Also

Bioinformatics Toolbox functions `restrict`, `seqwordcount`

MATLAB functions `regexp`, `regexpi`

seqcomplement

Purpose

Calculate the complementary strand of a nucleotide sequence

Syntax

```
SeqC = seqcomplement(SeqNT)
```

Arguments

SeqNT	Enter either a character string with the characters A, T (U), G, C, and ambiguous characters R, Y, K, M, S, W, B, D, H, V, N, or a vector of integers. You can also enter a structure with the field Sequence.

Description

SeqC = seqcomplement(SeqNT) calculates the complementary strand (A-->T, C-->G, G-->C, T-->A) of a DNA sequence and returns a sequence in the same format as SeqNT. For example, if SeqNT is an integer sequence then so is SeqC.

Examples

Return the complement of a DNA nucleotide sequence.

```
s = 'ATCG';
seqcomplement(s)

ans =
TAGC
```

See Also

Bioinformatics Toolbox functions seqrcomplement, seqreverse

Purpose

Format long sequence output for easy viewing

Syntax

```
seqdisp(Seq)

seqdisp(..., 'Row', RowValue)
seqdisp(..., 'Column', ColumnValue)
seqdisp(..., 'HiddenNumbers', HiddenNumber)
```

Arguments

Seq	Nucleotide or amino acid sequence of characters. Enter a character array, a FASTA file name, a MATLAB structure with fields from GenBank or GenPept. Multiple sequences are allowed.FASTA files can have the file extensions `fa`, `fasta`, `fas`, `fsa`, and `fst`.
RowValue	Property to select the length of each row. Enter an integer. The default length is `60`.
ColumnValue	Property to select the column width. Enter an integer. The default column width is `10`.
HiddenNumber	Property to control displaying numbers at the start of each row. Enter `true` to hide numbers.

Description

`seqdisp(Seq)` prints a sequence (`Seq`) in rows with a default row length of 60 and a default column width of `10`.

`seqdisp(..., 'Row',` *RowValue*`)` defines the length of each row for the displayed sequence.

seqdisp(..., ’Column’, *ColumnValue*) defines the column width of data for the displayed sequence.

seqdisp(..., ’ShowNumbers’, *ShowNumbers*), when `ShowNumbers` is `false`, turns the position numbers at the start of each row off. The default is '`true`'.

Examples

```
% Read in sequence information from a GenBank file,
% then display it in rows of 50 with column widths of 10.
M10051 = genbankread('HGENBANKM10051.GBK')
seqdisp(M10051, 'row', 50)
```

Create and save a FASTA file with two sequences, and then display it with seqdisp.

```
hdr = ['Sequnece A'; 'Sequence B'];
seq = ['TAGCTGRCCAAGGCCAAGCGAGCT';'ATCGACYGGTTCCGGTTCGCTCGA']
fastawrite('local.fa', hdr,seq);
seqdisp('local.fa','ShowNumbers', false')


ans =

>Sequnece A
 1  TAGCTGRCCA AGGCCAAGCG AGCTTN

>Sequence B
 1  ATCGACYGGT TCCGGTTCGC TCGAAN
```

See Also

Bioinformatics Toolbox function getgenbank

Purpose

Create a dot plot of two sequences

Syntax

```
seqdotplot(Seq1,Seq2)
seqdotplot(Seq1,Seq2, Window, Number)
```

Arguments

`Seq1, Seq2`	Nucleotide or amino acid sequences. Enter two character strings. Do not enter a vector of integers. You can also enter a structure with the field `Sequence`.
`Window`	Enter an integer for the size of a window.
`Number`	Enter an integer for the number of characters within the window that match.

Description

`seqdotplot (Seq1, Seq2)` plots a figure that visualizes the match between two sequences.

`seqdotplot(Seq1,Seq2, Window, Number)` plots sequence matches when there are at least *`Number`* matches in a window of size *`Window`*.

When plotting nucleotide sequences, start with a `Window` of `11` and `Number` of `7`.

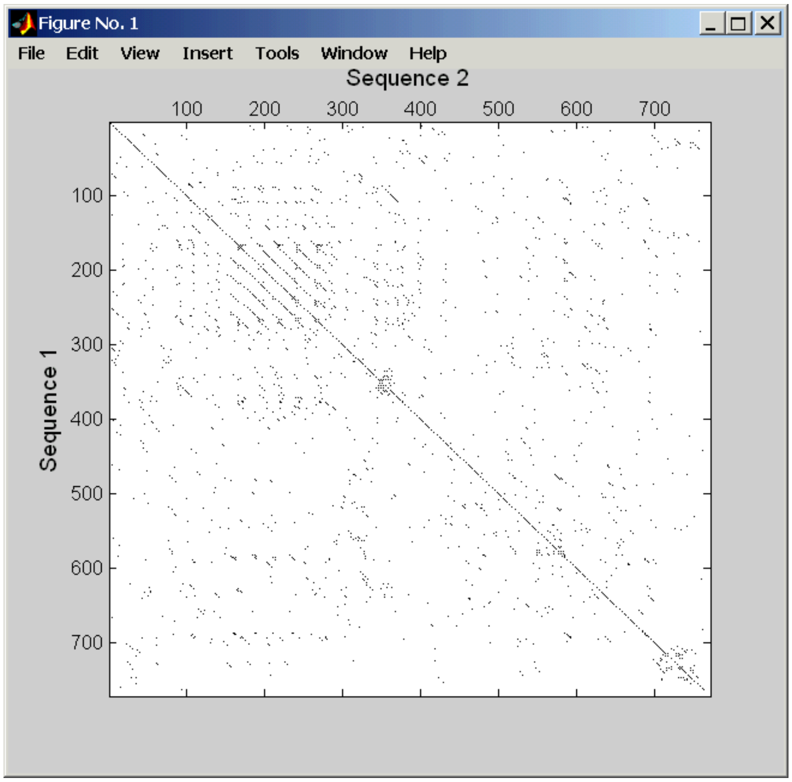
`Matches = seqdotplot(...)` returns the number of dots in the dot plot matrix.

[Matches, Matrix] = seqdotplot(...) = returns the dotplot as a sparse matrix.

Examples

This example shows the similarities between the prion protein (PrP) nucleotide sequences of two ruminants, the moufflon and the golden takin.

```
moufflon = getgenbank('AB060288','Sequence',true);
takin = getgenbank('AB060290','Sequence',true);
seqdotplot(moufflon,takin,11,7)
```



```
Matches = seqdotplot(moufflon,takin,11,7)
Matches =
        5552

[Matches, Matrix] = seqdotplot(moufflon,takin,11,7)
```

See Also

Bioinformatics Toolbox functions `hmmprofalign`, `nwalign`, `swalign`

Purpose

Construct a phylogenetic tree from pairwise distances

Syntax

```
Tree = seqlinkage(Dist)
Tree = seqlinkage(Dist, Method)
Tree = seqlinkage(Dist, Method, Names)
```

Arguments

`Dist`	Pairwise distances generated from the function `seqpdist`.
`Method`	Property to select a distance method. Enter a method from the table below.
`Names`	Property to use alternative labels for leaf nodes. Enter a vector of structures, with the fields `'Header'` or `'Name'`, or a cell array of strings. In both cases the number of elements you provide must comply with the number of samples used to generate the pairwise distances in `Dist`.

Description

`Tree = seqlinkage(Dist)` returns a phylogenetic tree object from the pairwise distances (`Dist`) between the species or products. `Dist` is a matrix (or vector) such as is generated by the function `seqpdist`.

`Tree = seqlinkage(Dist, Method)` creates a phylogenetic tree object using a specified patristic distance method. The available methods are

`'single'`	Nearest distance (single linkage method)
`'complete'`	Furthest distance (complete linkage method)
`'average'` (default)	Unweighted Pair Group Method Average (UPGMA, group average).
`'weighted'`	Weighted Pair Group Method Average (WPGMA)

'centroid'	Unweighted Pair Group Method Centroid (UPGMC)
'median'	Weighted Pair Group Method Centroid (WPGMC)

Tree = seqlinkage(Dist, Method, Names) passes a list of names to label the leaf nodes (for example, species or products) in a phylogenetic tree object.

Examples

```
% Load a multiple alignment of amino acids:
seqs = fastaread('pf00002.fa');
% Measure the 'Jukes-Cantor' pairwise distances:
dist = seqpdist(seqs,'method','jukes-cantor',...
                              'indels','pair');
% Build the phylogenetic tree with the single linkage
% method and pass the names of the sequences:
tree = seqlinkage(dist,'single',seqs)
view(tree)
```

See Also

The Bioinformatics Toolbox functions phytree, phytreewrite, seqpdist

phytree object methods plot and view

Purpose

Find matches for every string in a library

Syntax

`Index = seqmatch(Strings, Library)`

Description

`Index = seqmatch(Strings, Library)` looks through the elements of `Library` to find strings that begin with every string in `Strings`. `Index` contains the index to the first occurrence for every string in the query. `Strings` and `Library` must be cell arrays of strings.

Examples

```
lib = {'VIPS_HUMAN', 'SCCR_RABIT', 'CALR_PIG' ,'VIPR_RAT', 'PACR_MOUSE'};
query = {'CALR','VIP'};
h = seqmatch(query,lib);
lib(h)
```

See Also

MATLAB functions strmatch, regexpi

seqpdist

Purpose

Calculate the pairwise distance between biological sequences

Syntax

```
D = seqpdist(Seqs, 'PropertyName', PropertyValue)

seqpdist(..., 'Method', MethodValue)
seqpdist(..., 'Indels', IndelsValue)
seqpdist(..., 'Optargs', OptargsValue)
seqpdist(..., 'PairwiseAlignment',PairwiseAlignmentValue)
seqpdist(..., 'Squareform', SquareformValue)
seqpdist(..., 'Alphabet', AlphabetValue)

seqpdist(..., 'ScoringMatrix', ScoringMatrixValue)
seqpdist(..., 'Scale', ScaleValue
seqpdist(..., 'GapOpen', GapOpenValue)
seqpdist(..., 'ExtendGap', ExtendGapValue)
```

Arguments

`Seqs`	Cell array with nucleotide or amino acid sequences.
MethodValue	Property to select the method for calculating pariwise distances.
IndelsValue	Property to indicate how to treat gaps.
OptargsValue	Property to pass required arguments by the distance method selected with the property `Method`
PairwiseAlignmentValue	Property to force pariwise alignment.
SquareFormValue	Property to control formatting the output as a square or triangular matrix.
AlphabetValue	Property to select an alphabet. Enter either `'NT'` for nucleotides or `'AA'` for amino acids.
ScoringMatrixValue	Property to select a scoring matrix for pariwise alignment.

ScaleValue	Property to select a scale factor for the scoring matrix.
GapOpenValue	Property to select a gap penalty.
ExtendGapValue	Property to select a penalty for extending a gap.

Description

`D = seqpdist(Seqs, '`*`PropertyName`*`', `*`PropertyValue`*`)` returns a vector `D` containing biological distances between each pair of sequences stored in the `M` elements of the cell `Seqs`.

`D` is an `(M*(M-1)/2)`-by-1 vector corresponding to the `M*(M-1)/2` pairs of sequences in `Seqs`. The output `D` is arranged in the order `((2,1),(3,1),..., (M,1),(3,2),...(M,2),.....(M,M-1))`. This is the lower left triangle of the full `M`-by-`M` distance matrix. To get the distance between the `I`th and the `J`th sequences for `I > J`, use the formula `D((J-1)*(M-J/2)+I-J)`. `Seqs` can also be a vector of structures with the field `Sequence` or a matrix of `chars`.

`seqpdist(..., 'Method', `*`MethodValue`*`)` selects the method `seqpdist` uses to compute the distances between every pair of sequences.

Distances defined for both nucleotides and amino acids:

'p-distance'	Proportion of sites at which the two sequences are different. p —> 1 for poorly related and p —> 0 for similar sequences.
'Jukes-Cantor' (default)	Maximum likelihood estimate of the number of substitutions between two sequences. For NT d = -3/4 log(1p * 4/3) AA d = -19/20 log(1p * 20/19)
'alignment-score'	Distance (d) between two sequences (1 and 2) is computed from the pairwise alignment score (s) as follows: `d(1,2) = (1-s(1,2)/s(1,1)) * (1-s(1,2)/s(2,2))` This option does not imply that prealigned input sequences will be realigned, it only scores them. Use with care; this distance method does not comply with the ultrametric condition. In the rare case where s(x,y)>s(x,x), then d(x,y)=0.

Distances defined only for nucleotides and no scoring of gaps:

'Tajima-Nei'	Maximum likelihood estimate considering the background nucleotide frequencies. It can be computed from the input sequences or given by setting 'OPTARGS' to [gA gC gG gT].
'Kimura'	Considers separately the transitional and transversion nucleotide substitution.

'Tamura'	Considers separately the transitional and transversion nucleotide substitution and the GC content. GC content can be computed from the input sequences or given by setting 'OPTARGS'.
'Hasegawa'	Considers separately the transitional and transversional nucleotide substitution and the background nucleotide frequencies. Background frequencies can be computed from the input sequences or given by setting 'OPTARGS' to [gA gC gG gT].
'Nei-Tamura'	Considers separately the transitional substitution between purines, the transitional substitution between pyramidines and the transversional substitution and the background nucleotide frequencies. Background frequencies can be computed from the input sequences or given by setting 'OPTARGS' to [gA gC gG gT].

Distances defined only for amino acids and no scoring of gaps:

'Poisson'	Asumes that the number of amino acid substitutions at each site has a Poisson distribution.
'Gamma'	Assumes that the number of amino acid substitutions at each site has a Gamma distribution with parameter 'a'. 'a' can be set by 'OPTARGS'. The default value is 2.

A user defined distance function can also be specified using @, for example, @distfun, the distance function must be of the form:

```
function D = distfun(S1, S2, OPTARGS)
```

Taking as arguments two same-length sequences (NT or AA) plus zero or more additional problem-dependent arguments in `OPTARGS`, and returning a scalar that represents the distance between `S1` and `S2`.

`seqpdist(..., 'Indels', `*`IndelsValue`*`)` indicates how to treat sites with gaps. Options are

- `'score'` (default) — Scores these sites either as a point mutation or with the alignment parameters depending on the method selected.
- 'pairwise-del' — For every pairwise comparison it ignores the sites with gaps.
- 'complete-del' — Ignores all the columns in the multiple alignment that contain a gap, this option is available only if a multiple alignment was provided at the input Seqs.

`seqpdist(..., 'Optargs', `*`OptargsValue`*`)` some distance methods require or accept optional arguments. Use a cell array to pass more than one input argument (for example, The nucleotide frequencies in the Tajima-Nei distance function can be specified instead of computing them from the input sequences).

`seqpdist(..., 'PairwiseAlignment', `*`PairwiseAlignmentValue`*`)`, when `PairwiseAlignment` is `true`, ignores multiple alignment of the input sequences (if any) and forces a pairwise alignment of input sequences. If the input sequences are not prealigned, this flag is set automatically. Pairwise alignment can be slow for a large number of sequences. The default value is `false`.

`seqpdist(..., 'Squareform', `*`SquareformValue`*`)`, when `SquareForm` is `true`, converts the output into a square formatted matrix so the `D(I,J)` denotes the distance between the `I`th and `J`th sequences. The output matrix is symmetric and has a zero diagonal. Setting the property

Squareform to true is the same as using the function squareform in the Statistical Toolbox.

seqpdist(..., 'Alphabet', *AlphabetValue*) specifies whether the sequences are amino acids ('AA') or nucleotides ('NT'). The default value is 'AA'.

The remaining input properties are analogous to the function nwalign and are used when the property PairwiseAlignment = true or the property Method = 'alignment-score'. For more information about these properties, see nwalign.

seqpdist(..., 'ScoringMatrix', *ScoringMatrixValue*) specifies the scoring matrix to be used for the alignment. The default value is BLOSUM50 for AA and NUC44 for NT.

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

seqpdist(..., GapOpen', *GapOpenValue*) specifies the penalty for opening a gap in the alignment. The default gap open penalty is 8.

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

Examples

```
% Load a multiple alignment of amino acids:
seqs = fastaread('pf00002.fa');

% For every possible pair of sequences in the multiple
% alignment removes sites with gaps and scores with the
% substitution matrix PAM250:

dist = seqpdist(seqs,'method','alignment-score',...
                'indels','pairwise-delete',...
                'scoringmatrix','pam250')

% To force the realignment of every pair of sequences
% ignoring the provided multiple alignment:
```

```
dist = seqpdist(seqs,'method','alignment-score',...
                'indels','pairwise-delete',...
                'scoringmatrix','pam250',...
                'pairwisealignment',true)

% To measure the 'Jukes-Cantor' pairwise distances after
% realigning every pair of sequences, counting the gaps as
% point mutations:

dist = seqpdist(seqs,'method','jukes-cantor',...
                              'indels','score',...
                              'scoringmatrix','pam250',...
                              'pairwisealignment',true)
```

See Also

Bioinformatics Toolbox functions `fastaread`, `seqlinkage`

`phytree` methods `phytree`, `pdist (phytree)`

Statistical Toolbox functions pdist, squareform

Purpose

Calculate the reverse complement of a nucleotide sequence

Syntax

```
SeqRC = seqrcomplement(SeqNT)
```

Arguments

`SeqNT`	Nucleotide sequence. Enter either a character string with the characters `A`, `T` (`U`), `G`, `C`, and ambiguous characters `R`, `Y`, `K`, `M`, `S`, `W`, `B`, `D`, `H`, `V`, `N`, or a vector of integers. You can also enter a structure with the field `Sequence`.

Description

`seqrcomplement` calculates the reverse complementary strand of a DNA sequence.

`SeqRC = seqrcomplement(SeqNT)` calculates the reverse complementary strand 3' --> 5' (A-->T, C-->G, G-->C, T-->A) for a DNA sequence and returns a sequence in the same format as `SeqNT`. For example, if `SeqNT` is an integer sequence then so is `SeqRC`.

Examples

Reverse a DNA nucleotide sequence and then return its complement.

```
s = 'ATCG'
seqrcomplement(s)

ans =
CGAT
```

See Also

Bioinformatics Toolbox functions `codoncount`, `palindromes` `seqcomplement`, `seqreverse`

seqreverse

Purpose Reverse the letters or numbers in a nucleotide sequence

Syntax `SeqR = seqreverse(SeqNT)`

Arguments

`SeqNT`	Enter a nucleotide sequence. Enter either a character string with the characters `A`, `T` (`U`), `G`, `C`, and ambiguous characters `R`, `Y`, `K`, `M`, `S`, `W`, `B`, `D`, `H`, `V`, `N`, or a vector of integers. You can also enter a structure with the field `Sequence`.
`SeqR`	Returns a sequence in the same format as the nucleotide sequence. For example, if `SeqNT` is an integer sequence, then so is `SeqR`.

Description `seqreverse` calculates the reverse strand of a DNA or RNA sequence.

`SeqR = seqreverse(SeqNT)` calculates the reverse strand 3' --> 5' of the nucleotide sequence.

Examples Reverse a nucleotide sequence.

```
s = 'ATCG'
seqreverse(s)

ans =
GCTA
```

See Also Bioinformatics Toolbox functions `seqcomplement`, `seqrcomplement`

MATLAB function `fliplr`

Purpose

Graphically display the open reading frames in a sequence

Syntax

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

seqshoworfs(..., 'Frames', FramesValue)
seqshoworfs(..., 'GeneticCode', GeneticCodeValue)
seqshoworfs(..., 'MinimumLength', MinimumLengthValue)
seqshoworfs(..., 'AlternativeStartCodons', StartCodonsValue)
seqshoworfs(..., 'Color', ColorValue)
seqshoworfs(..., 'Columns', ColumnsValue)
```

Arguments

`SeqNT`	Nucleotide sequence. Enter either a character string with the characters `A`, `T` (`U`), `G`, `C`, and ambiguous characters `R`, `Y`, `K`, `M`, `S`, `W`, `B`, `D`, `H`, `V`, `N`, or a vector of integers. You can also enter a structure with the field `Sequence`.
FramesValue	Property to select the frame. Enter `1`, `2`, `3`, `-1`, `-2`, `-3`, enter a vector with integers, or `'all'`. The default value is the vector `[1 2 3]`. Frames `-1`, `-2`, and `-3` correspond to the first, second, and third reading frames for the reverse complement.
GeneticCodeValue	Genetic code name. Enter a code number or a code name from the table `geneticcode`.
MinimumLengthValue	Property to set the minimum number of codons in an ORF.
StartCodonsValue	Property to control using alternative start codons. Enter either `true` or `false`. The default value is `false`.

ColorValue	Property to select the color for highlighting the reading frame. Enter either a 1-by-3 RGB vector specifying the intensity (`0 to 255`) of the red, green, and blue components of the color, or a character from the following list: `'b'`—blue, `'g'`—green, `'r'`—red, `'c'`—cyan, `'m'`—magenta, or `'y'`—yellow. To specify different colors for the three reading frames, use a 1-by-3 cell array of color values. If you are displaying reverse complement reading frames, then COLOR should be a 1-by-6 cell array of color values.
ColumnsValue	Property to specify the number of columns in the output.

Description

`seqshoworfs` identifies and highlights all open reading frames using the standard or an alternative genetic code.

`seqshoworfs(SeqNT)` displays the sequence with all open reading frames highlighted, and it returns a structure of start and stop positions for each ORF in each reading frame. The standard genetic code is used with start codon `'AUG'` and stop codons `'UAA'`, `'UAG'`, and `'UGA'`.

`seqshoworfs(..., 'Frames',` *FramesValue*`)` specifies the reading frames to display. The default is to display the first, second, and third reading frames with ORFs highlighted in each frame.

`seqshoworfs(..., 'GeneticCode',` *GeneticCodeValue*`)` specifies the genetic code to use for finding open reading frames.

`seqshoworfs(..., 'MinimumLength',` *MinimumLengthValue*`)` sets the minimum number of codons for an ORF to be considered valid. The default value is `10`.

`seqshoworfs(..., 'AlternativeStartCodons',` *StartCodonsValue*`)` uses alternative start codons if `AlternativeStartCodons` is set to `true`. For example, in the human mitochondrial genetic code, `AUA` and `AUU` are known to be alternative start codons. For more details of alternative start codons, see

```
http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/
wprintgc.cgi?mode=t#SG1
```

`seqshoworfs(..., 'Color', ColorValue)` selects the color used to highlight the open reading frames in the output display. The default color scheme is blue for the first reading frame, red for the second, and green for the third frame.

`seqshoworfs(..., 'Columns', ColumnsValue)` specifies how many columns per line to use in the output. The default value is `64`.

Examples

Look for the open reading frames in a random nucleotide sequence.

```
s = randseq(200,'alphabet', 'dna');
seqshoworfs(s);
```

Identify the open reading frames in a GenBank sequence.

```
HLA_DQB1 = getgenbank('NM_002123');
seqshoworfs(HLA_DQB1.Sequence);
```

See Also

Bioinformatics Toolbox functions `codoncount`, `geneticcode`, `seqdisp`,`seqshowwords`, `seqwordcount`

MATLAB function `regexp`

seqshowwords

Purpose

Graphically display the words in a sequence

Syntax

```
seqshowwords(Seq, Word, 'PropertyName', PropertyValue)

seqshowwords(...,'Color', ColorValue)
seqshowwords(...,'Columns', ColumnsValue)
```

Arguments

Seq	Enter either a nucleotide or amino acid sequence. You can also enter a structure with the field `Sequence`.
Word	Enter a short character sequence.
ColorValue	Property to select the color for highlighted characters. Enter a 1-by-3 RGB vector specifying the intensity (0255) of the red, green, and blue components, or enter a character from the following list: `'b'`– blue, `'g'`– green, `'r'`– red, `'c'`– cyan, `'m'`– magenta, or `'y'`– yellow. The default color is red `'r'`.
ColumnsValue	Property to specify the number of characters in a line. Default value is 64.

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(...,'Color',` *ColorValue*`)` selects the color used to highlight the words in the output display.

`seqshowwords(...,'Columns',` *ColumnsValue*`)` specifies how many columns per line to use in the output.

Examples

If `word` contains nucleotide or amino acid symbols that represent multiple possible symbols (ambiguous characters), then `seqshowwords` shows all matches. For example, the symbol `R` represents either `G` or `A` (purines). For another example, if word equals 'ART', then seqshowwords counts occurrences of both 'AAT' and 'AGT'. 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`, `restrict`, `seqdisp`, `seqshoworfs`

MATLAB functions `findstr`, `regexp`

Purpose

Count the number of occurrences of a word in a sequence

Syntax

```
seqwordcount(Seq, Word)
```

Arguments

`Seq`	Enter a nucleotide or amino acid sequence of characters. You can also enter a structure with the field `Sequence`.
`Word`	Enter a short sequence of characters.

Description

`seqwordcount(Seq, Word)` counts the number of times that a word appears in a sequence, and then returns the number of occurrences of that word.

If `Word` contains nucleotide or amino acid symbols that represent multiple possible symbols (ambiguous characters), then `seqwordcount` counts all matches. For example, the symbol `R` represents either `G` or `A` (purines). For another example, if `word` equals `'ART'`, then `seqwordcount` counts occurrences of both `'AAT'` and `'AGT'`.

Examples

`seqwordcount` does not count overlapping patterns multiple times. In the following example, `seqwordcount` reports three matches. `TATATATA` is counted as two distinct matches, not three overlapping occurrences.

```
seqwordcount('GCTATAACGTATATATAT','TATA')

ans =
     3
```

The following example reports two matches (`'TAGT'` and `'TAAT'`). `B` is the ambiguous code for `G`, `T`, or `C`, while `R` is an ambiguous code for `G` and `A`.

```
seqwordcount('GCTAGTAACGTATATATAAT','BART')

ans =
     2
```

See Also

Bioinformatics Toolbox functions `codoncount`, `seqshoworfs`, `seqshowwords`

MATLAB functions `seq2regexp`, `strfind`

Purpose

Display a sequence alignment with color

Syntax

```
showalignment(Alignment, 'PropertyName', PropertyValue)

showalignment(..., 'StartPointers', StartPointersValue)
showalignment(..., 'MatchColor', MatchColorValue)
showalignment(..., 'SimilarColor' SimilarColorValue)
showalignment(..., 'Columns', ColumnsValue)
```

Arguments

`Alignment`	Enter the output from either the function `swalign` or `nwalign`.
SimilarColorValue	Property to specify the starting indices of the aligned sequences. `StartPointers` is the two element vector returned as the third output of the function `swalign`.
MatchColorValue	Property to select the color to highlight matching characters. Enter a `1`-by-`N` RGB vector specifying the intensity (`0` to `255`) of the red, green, and blue components, or enter a character from the following list: `'b'`– blue, `'g'`– green, `'r'`– red, `'c'`– cyan, `'m'`– magenta, or `'y'`– yellow. The default color is red, `'r'`.
SimilarColorValue	Property to select the color to highlight similar characters. Enter a `1`-by-`3` RGB vector or color character. The default color is magenta.
ColumnsValue	Property to specify the number of characters in a line. Enter the number of characters to display in one row. The default value is `64`.

Description

`showalignment(Alignment, 'PropertyName', PropertyValue)` displays an alignment string with matches and similar residues highlighted with color.

`showalignment(..., 'StartPointers', StartPointersValue)` specifies the starting indices in the original sequences of a local alignment.

`showalignment(..., 'MatchColor', MatchColorValue)` selects the color to highlight the matches in the output display. The default color is red. For example, to use cyan, enter `'c'` or `[0 255 255]`.

`showalignment(..., 'SimilarColor' SimilarColorValue)` selects the color to highlight similar residues that are not exact matches. The default color is magenta.

`showalignment(..., 'Columns', ColumnsValue)` specifies how many columns per line to use in the output, and labels the start of each row with the sequence positions.

Examples

Enter two amino acid sequences and show their alignment.

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

```
Identities = 6/11 (55%), Positives = 7/11 (64%)
VSPAGMASGYD
: | | || ||
I-P-GKAS-YD
```

See Also

Bioinformatics Toolbox functions `nwalign`, `swalign`

Purpose

Plot an HMM profile

Syntax

```
showhmmprof(Model, 'PropertyName', PropertyValue)

showhmmprof(..., 'Scale', ScaleValue)
```

Arguments

`Model`	Hidden Markov model created with the functions `gethmmprof` and `pfamhmmread` functions.
ScaleValue	Enter one of the following values: `'logprob'` — Log probabilities `'prob'` — Probabilities `'logodds'` — Log-odd ratios

Description

`showhmmprof(Model)` plots a profile hidden Markov model described by the structure `Model`.

`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 `DomainValue` is `'logprob'`.

Examples

```
load('hmm_model_examples','model_7tm_2') % load a model example
showhmmprof(model_7tm_2,'Scale','logodds')
```

See Also

Bioinformatics Toolbox functions `gethmmprof`, `hmmprofalign`, `hmmprofestimate`, `hmmprofgenerate`, `hmmprofstruct`, `pfamhmmread`

sptread

Purpose Read data from a SPOT file

Syntax

```
SPOTData = sptread('File',
                   'PropertyName', PropertyValue)

sptread(..., 'CleanColNames, 'CleanColNamesValues')
```

Arguments

File	SPOT formatted file (ASCII text file). Enter a filename, a path and filename, or URL pointing to a file. *File* can also be a MATLAB character array that contains the text for a SPOT file.
CleanColNamesValue	Property to control using valid MATLAB variable names.

Description

`SPOTData = sptread('`*File*`')` reads a SPOT formatted file and creates a MATLAB structure `SPOTData` containing the following fields:

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

`sptread(..., 'CleanColNames,` *CleanColNamesValue*`)` The column names in the SPOT file contain periods and some characters that cannot be used in MATLAB variable names. If you plan to use the column names as variable names in a function, use this option with `CleanColNames` set to `true` and the function will return the field `ColumnNames` with valid variable names.

The Indices field of the structure includes the MATLAB indices that you can use for plotting heat maps of the data.

Examples

```
% Read in a sample SPOT file and plot the median foreground
% intensity for the 635 nm channel.
spotStruct = sptread('spotdata.txt')
maimage(spotStruct,'Rmedian');

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

rmedCol = find(strcmp(spotStruct.ColumnNames,'Rmedian'));
Rmedian = spotStruct.Data(:,rmedCol);
imagesc(Rmedian(spotStruct.Indices));
colormap bone
colorbar
```

See Also

Bioinformatics Toolbox functions gprread, maimage

Purpose

Locally align two sequences using the Smith-Waterman algorithm

Syntax

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

swalign(..., 'Alphabet', AlphabetValue)
swalign(..., 'ScoringMatrix', ScoringMatrixValue)
swalign(..., 'Scale', ScaleValue)
swalign(..., 'GapOpen', GapOpenValue)
swalign(..., 'ExtendGap', ExtendGapValue)
```

Arguments

`Seq1, Seq2`	Nucleotide or amino acid sequences. Enter a character string or vector of integers. You can also enter a structure with the field `Sequence`.
AlphabetValue	Property to select an amino acid or nucleotide sequences. Enter either `'AA'` or `'NT'`. The default value is `'AA'`.
ScoringMatrixValue	Enter the name of a scoring matrix. Values are `'PAM40'`, `'PAM250'`, `DAYHOFF`, `GONNET`, `'BLOSUM30'` increasing by `5` to `'BLOSUM90'`, or `'BLOSUM62'`, or `'BLOSUM100'`. The default value when `AlphabetValue = 'aa'` is `'BLOSUM50'`, while the default value when `AlphabeValue = 'nt'` is `nuc44`.
ScaleValue	Property to specify the scale factor for a scoring matrix.
GapOpenValue	Enter an integer for the gap penalty. Default value is `8`.
ExtendGapValue	Enter an integer for the extended gap penalty. The default value equals the `GapOpen` value.

`Score`	Returns the alignment score. Units for `Score` are `bits`.
`Alignment`	Returns a 3-by-n character array showing the two sequences and the alignment between them.
`Start`	Position where the alignment begins in each sequence.

Description

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

`[Score, Alignment, Start] = swalign(Seq1, Seq2)` returns a 2-by-1 vector with the starting point indices where the alignment begins for each sequence.

`swalign(...,'Alphabet',` *AlphabetValue*`)` specifies whether the sequences are amino acids ('AA') or nucleotides ('NT'). The default value is 'AA'.

`swalign(..., 'ScoringMatrix',` *ScoringMatrixValue*`)` specifies the scoring matrix to use for the alignment. The default is '`blosum50`' for `Alphabet` = 'AA' or 'NUC44' for `Alphabet` = `NT`.

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

`swalign(..., 'GapOpen',` *GapOpenValue*`)` specifies the penalty for opening a gap in the alignment. The default gap open penalty is `8`.

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

Examples

Return the score in bits and the local alignment using the default `ScoringMatrix` ('BLOSUM50') and default values for the `GapOpen` and `ExtendGap` values.

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

Score =
     8.6667

Alignment =
PAGMASGYD
| | || ||
P-GKAS-YD
```

Align two amino sequences using a specified scoring matrix ('pam250') and a gap open penalty of 5.

```
[Score, Alignment] = swalign('HEAGAWGHEE','PAWHEAE',...
                             'ScoringMatrix', 'pam250',...
                             'GapOpen',5)

Score =
     8
Alignment =
GAWGHE
:|| ||
PAW-HE
```

Align two amino sequences and return the `Score` in nat units (nats).

```
[Score, Alignment] = swalign('HEAGAWGHEE','PAWHEAE',...
                             'Scale',log(2))

Score =
    6.4694
Alignment =
AWGHE
|| ||
   AW-HE
```

See Also

Bioinformatics Toolbox functions `blosum`, `dayhoff`, `gonnet`, `nt2aa`, `nwalign`, `showalignment`

Purpose

Draw nucleotide trace plots

Syntax

```
traceplot(TraceStructure)
traceplot(A, C, G, T)
h = traceplot()
```

Description

`traceplot(`*`TraceStructure`*`)` creates a trace plot from data in a structure with fields `A`, `C`, `G`, `T`.

`traceplot(A, C, G, T)` creates a trace plot from data in vectors `A`, `C`, `G`, `T`.

`h = traceplot()` returns a structure with the handles of the lines corresponding to `A`, `C`, `G`, `T`.

Examples

```
tstruct = scfread('sample.scf');
traceplot(tstruct)
```

See Also

Bioinformatics Toolbox function `scfread`

Purpose

View a phylogenetic tree in the `phytreetool` window.

Syntax

```
view(Tree)
view(Tree, IntNodes)
```

Arguments

`Tree`	`phytree` object created with `phytree`.
`IntNodes`	Nodes form 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.

Examples

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

See Also

Bioinformatics Toolbox functions `phytreeread`, `phytreetool`, `seqlinkage`

`phytree` object methods `phytree`, `plot (phytree)`

Examples

Sequence Analysis

Microarray Analysis

Phylogenetic Analysis

“Example: Building a Phylogenetic Tree” on page 4-2

Index