Bioinformatics Toolbox

For Use with MATLAB®

- Computation
- Visualization
- Programming

User's Guide

Version 1

The MathWorks

Printing History: September 2003 Online only New for Version 1.0 (Release 13SP1+)

Contents

Introduction

This chapter is an overview of the functions and features for 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?

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

You can use the basic bioinformatics 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
- Reading, normalizing, and visualizing microarray data

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 scientists will need.

Computation — 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.

Visualization — Visualize sequence alignments, gene expression data, and protein structure analyses.

Expected User

The Bioinformatics Toolbox is for computational biologists 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 bioinformatics 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 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 1–4 — List of MathWorks products you need to purchase with the Bioinformatics Toolbox
- "Additional Software" on page 1–4 — 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 1.0 of the Bioinformatics Toolbox requires MATLAB Version 6.5 on the Release 13SP1 CD.
Statistics Toolbox	Provides basic statistics and probability functions that the Bioinformatics Toolbox functions use. Version 1.0 of the Bioinformatics Toolbox requires the Statistics Toolbox Version 4.1 on the Release 13SP1 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, you can 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	Neural networks are used to solve problems where algorithms are not available. 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 COM Builder	Create COM objects to use with any COM-based programming environment.
MATLAB Excel Builder	Create Excel add-in functions from MATLAB functions to use with Excel spreadsheets.

Features and Functions

The Bioinformatics Toolbox includes many features to help you with genome and proteome analysis. Most functions are implemented in the open MATLAB language. This open environment lets you explore and customize the existing toolbox algorithms or develop your own.

- "Data Formats and Databases" on page 1–6 — Access online databases, read and write to files with standard genome and proteome formats such as FASTA and PDB.
- "Sequence Alignments" on page 1–7 — Compare nucleotide or amino acid sequences using pairwise and multiple sequence alignment functions.
- "Sequence Utilities and Statistics" on page 1–8 — Manipulate sequences and determine physical, chemical, and biological characteristics.
- "Microarray Analysis" on page 1–8 — Read, filter, normalize, and visualize microarray data.
- "Protein Structure Analysis" on page 1–9 — Determine protein characteristics and simulate enzyme cleavage reactions.
- "Prototype and Development Environment" on page 1–9 — Create new algorithms, try new ideas, and compare alternatives.
- "Share Algorithms and Deploy Applications" on page 1–10 — Create GUIs and stand-alone applications.

Data Formats and Databases

The Bioinformatics Toolbox supports access to many 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 format readers.

Web-based databases — You can directly access public databases on the Web and read 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`).

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`
- 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.

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`).

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

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`).

Additional functions in MATLAB efficiently handle string operations with regular expressions (, `seq2regexp`), and look for possible cleavage sites in DNA/RNA sequences by searching for palindromes (`palindromes`).

Sequence statistics — You can determine various statistics about the sequences (`aacount`, `basecount`, `codoncount`, `dimercount`, `nmercount`, `ntdensity`), search for specific patterns within a sequence (`seqshowwords`, `seqwordcount`), or search for open reading frames (`showhmmprof`). You can manipulate your sequence by performing an in-silico digestion with restriction endonucleases (`restrict`) and proteases (`cleave`). In addition, you can create random sequences for test cases (`randseq`).

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.

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

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.

Prototype and Development Environment

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

- **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

Share Algorithms and Deploy Applications

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.

"Sequence Statistics Example" on page 2–2	Starting with a DNA sequence, calculate statistics for the nucleotide content.
"Sequence Alignment Example" on page 2–16	Starting with a DNA sequence for a human gene, locate and verify a corresponding gene in a model organism.

Sequence Statistics Example

After a piece of DNA is sequenced, 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.
- "Nucleotide Composition" on page 2–5 — Determine the monomers and dimers, and then visualize data in graphs and bar plots.
- "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.

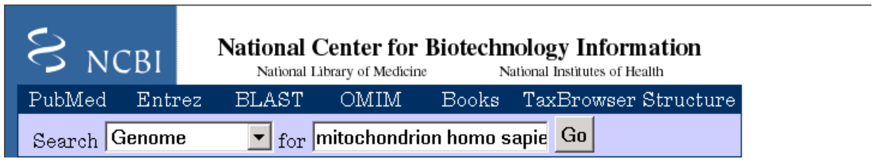
First research information about the human mitochondria and find the nucleotide sequence for the genome. Next, look at the nucleotide content for the entire sequence. And finally, determine open reading frames and extract specific gene sequences.

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

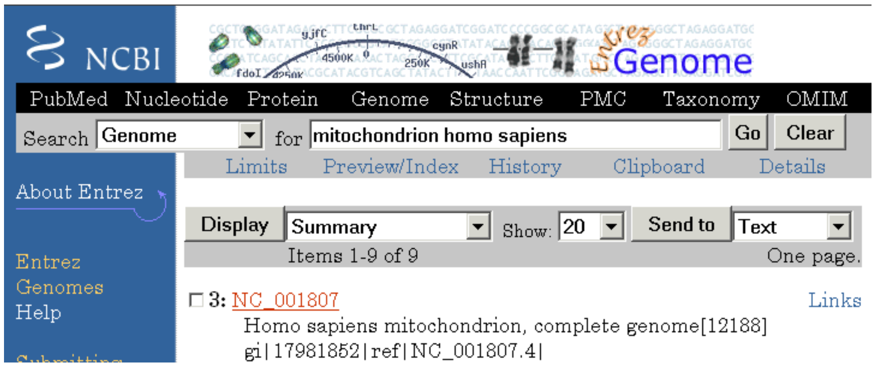
```
web('http://www.ncbi.nlm.nih.gov/')
```

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 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.

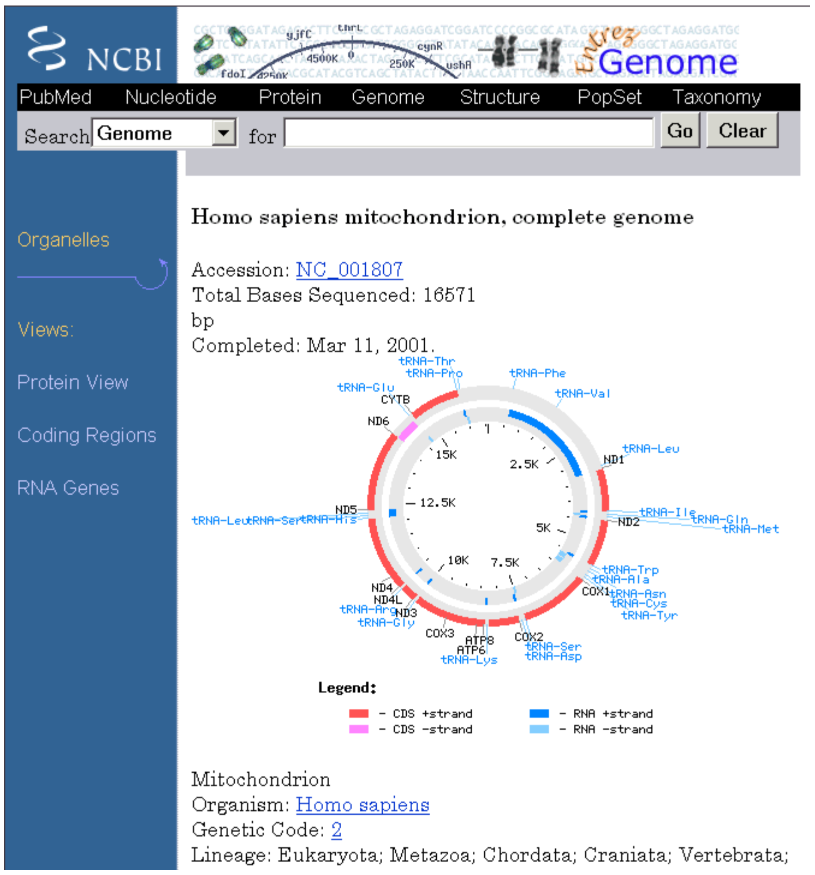


Items 1-9 of 9 One page.

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.

Legend:

🟥 - CDS +strand
🟪 - CDS -strand
🟦 - RNA +strand
🟦 - RNA -strand

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
```

Nucleotide Composition

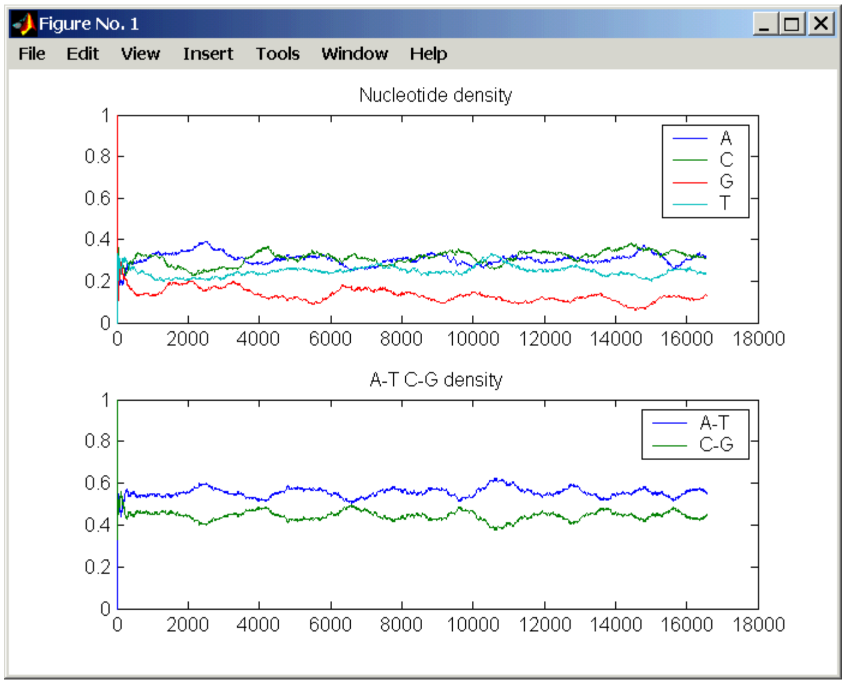
Sections of a DNA sequence with a high percent of A+T nucleotides usually indicates intergenic parts of the sequence, while low A+T and higher G+C nucleotide percentages indicate possible genes. Many times high CG dinucleotide content is located before a gene.

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))
```

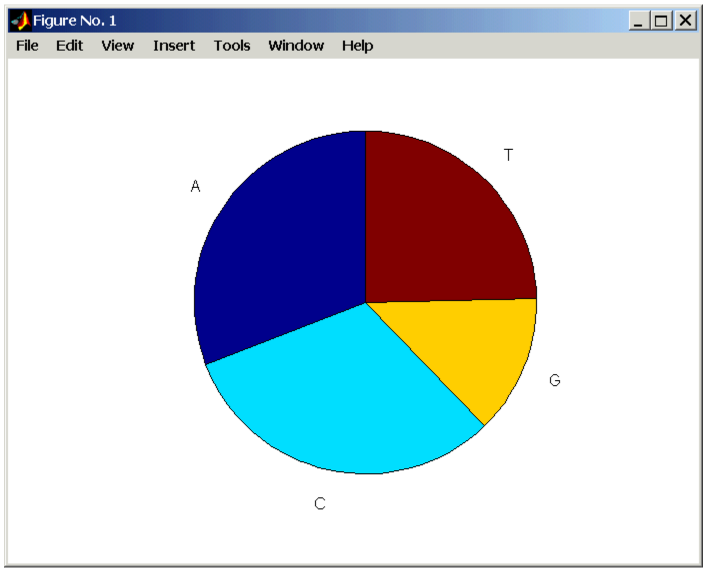
As expected, the nucleotide counts on the reverse complement strand are complementary to the 5'-3' strand.

```
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');
```

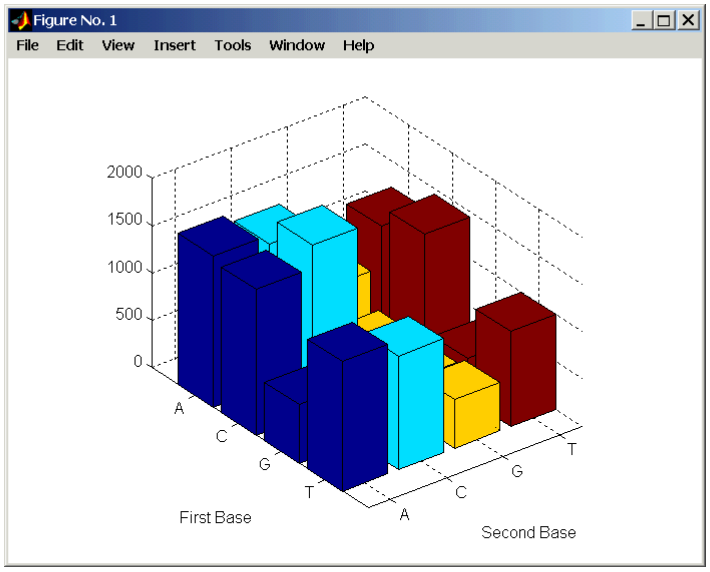
MATLAB draws a pie chart in a figure window.



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.



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.

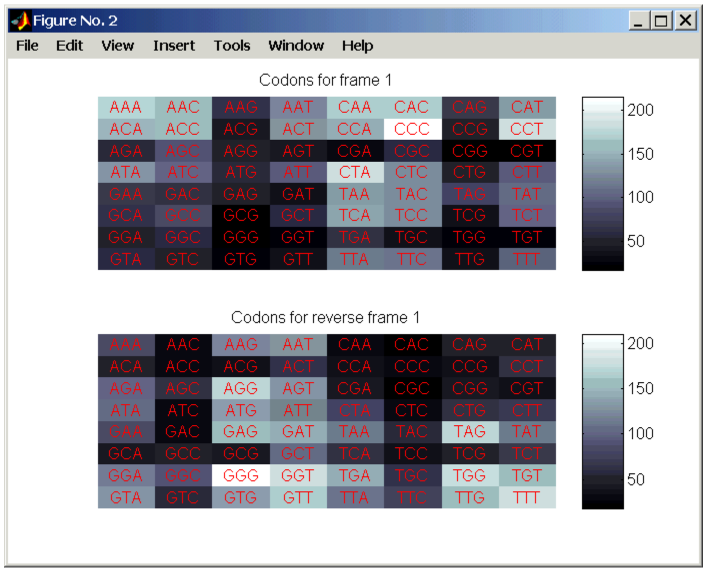


Figure No. 2
File Edit View Insert Tools Window Help
Codons for frame 1
AAA AAC AAG AAT CAA CAC CAG CAT
ACA ACC ACG ACT CCA CCC CCG CCT
AGA AGC AGG AGT CGA CGC CGG CGT
ATA ATC ATG ATT CTA CTC CTG CTT
GAA GAC GAG GAT TAA TAC TAG TAT
GCA GCC GCG GCT TCA TCC TCG TCT
GGA GGC GGG GGT TGA TGC TGG TGT
GTA GTC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

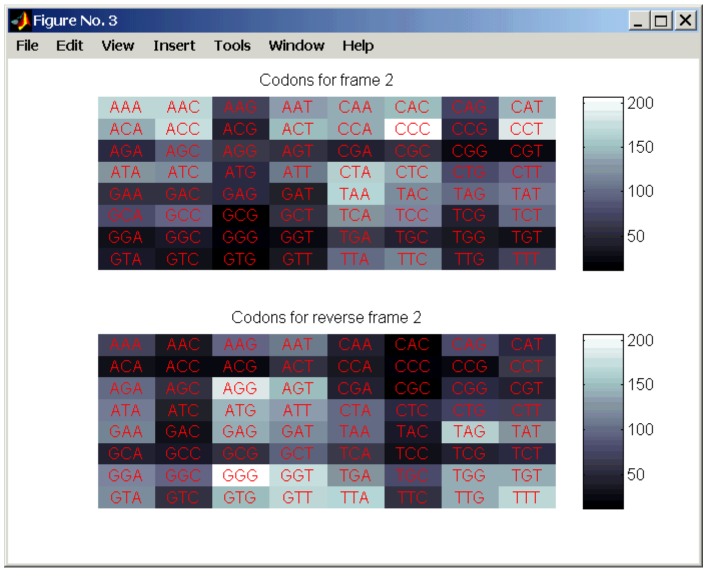
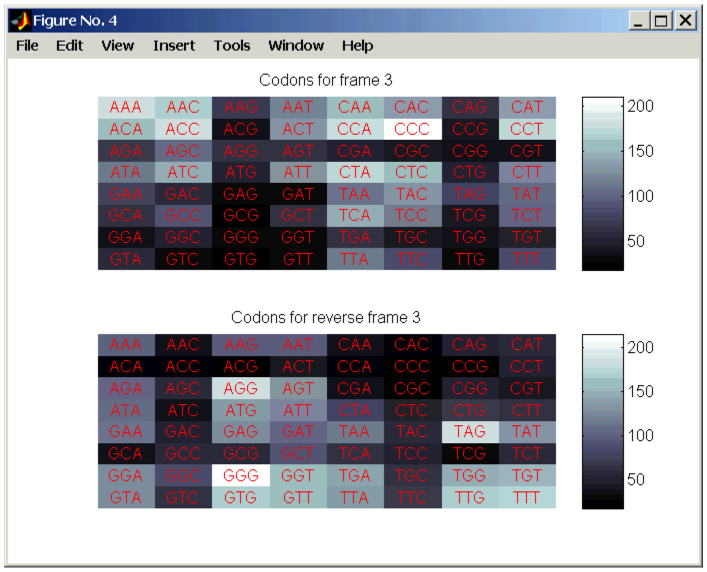


Figure No. 3
File Edit View Insert Tools Window Help
Codons for frame 2
AAA AAC AAG AAT CAA CAC CAG CAT
ACA ACC ACG ACT CCA CCC CCG CCT
AGA AGC AGG AGT CGA CGC CGG CGT
ATA ATC ATG ATT CTA CTC CTG CTT
GAA GAC GAG GAT TAA TAC TAG TAT
GCA GCC GCG GCT TCA TCC TCG TCT
GGA GGC GGG GGT TGA TGC TGG TGT
GTA GTC GTG GTT TTA TTC TTG TTT
200
150
100
50
Codons for reverse frame 2
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 5–9.

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
actcatcacagcgctaagctcgcactgattttttacctgagtaggcctag
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('letter',nt2aa('ATA'))
aminolookup('letter',nt2aa('CTA'))
aminolookup('letter',nt2aa('ACC'))
aminolookup('letter',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.

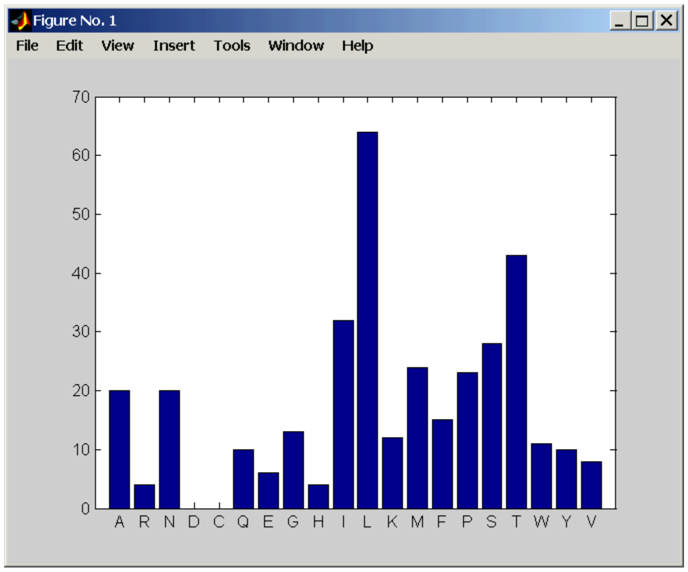
```
ND2protein = getgenpept('NP_536844','sequenceonly',true)
```

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.

Sequence Alignment Example

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 2–16 — Use the MATLAB Help browser to search the Web for information.
- "Getting Sequence Information from a Public Database" on page 2–18 — 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 2–20' — 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 2–22 — Convert a sequence from nucleotides to amino acids and identify the open reading frames.
- "Comparing Amino Acid Sequences" on page 2–25 — 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.

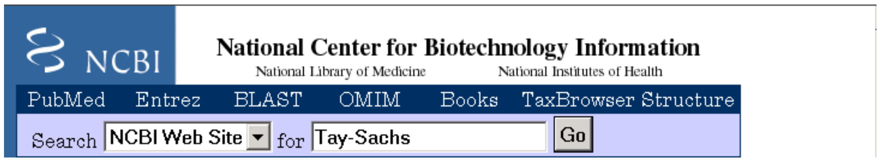
First, to research information about Tay-Sachs and the enzyme that is associated with this disease, then find the nucleotide sequence for the human gene that codes for the enzyme, and finally find a corresponding gene in another organism to use as a model for study.

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

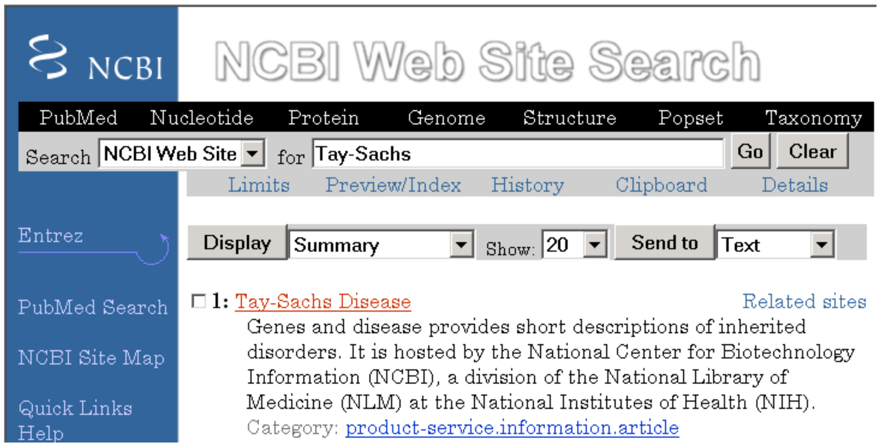
```
web('http://www.ncbi.nlm.nih.gov/')
```

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.



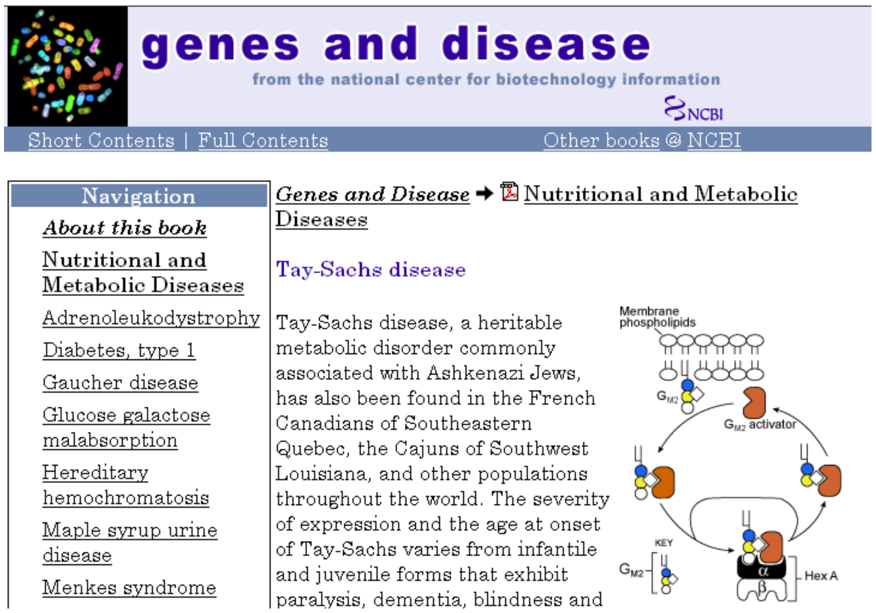
1: Tay-Sachs Disease

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.

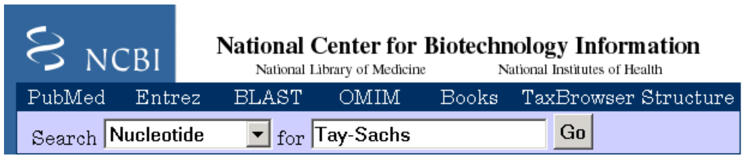
After you locate a sequence, you need to move the sequence data into the MATLAB workspace.

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

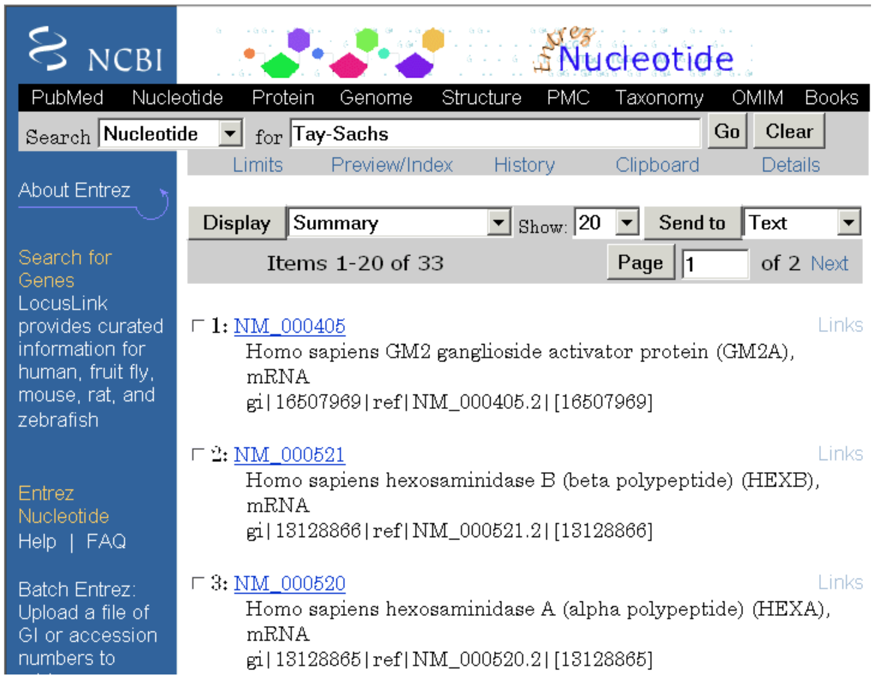
```
web('http://www.ncbi.nlm.nih.gov/')
```

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`.



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

Items 1-20 of 33

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

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

3: NM_000520
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

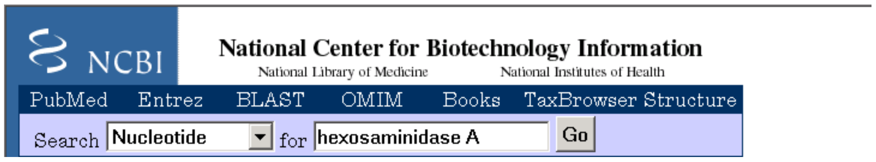
The sequence and function of many genes is conserved during the evolution of species through homologous genes. Homologous genes are genes that have a common ancestor and similar sequences. One goal of searching a public database is to find similar genes. If you are able to locate a sequence in a database 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.

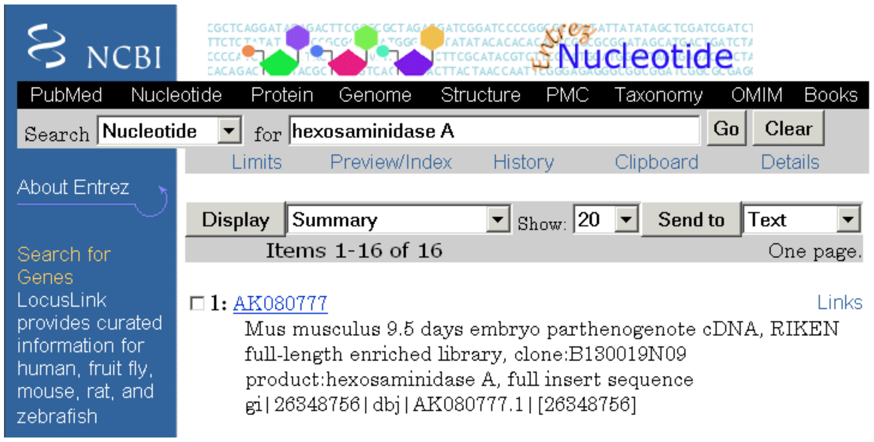
1 Open the MATLAB Help browser to the NCBI Web site. In the **MATLAB Command** window, type

```
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`.


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

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



1: AK080777
Mus musculus 9.5 days embryo parthenogenote cDNA, RIKEN full-length enriched library, clone:B130019N09 product:hexosaminidase A, full insert sequence
gi|26348756|dbj|AK080777.1|[26348756]

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.

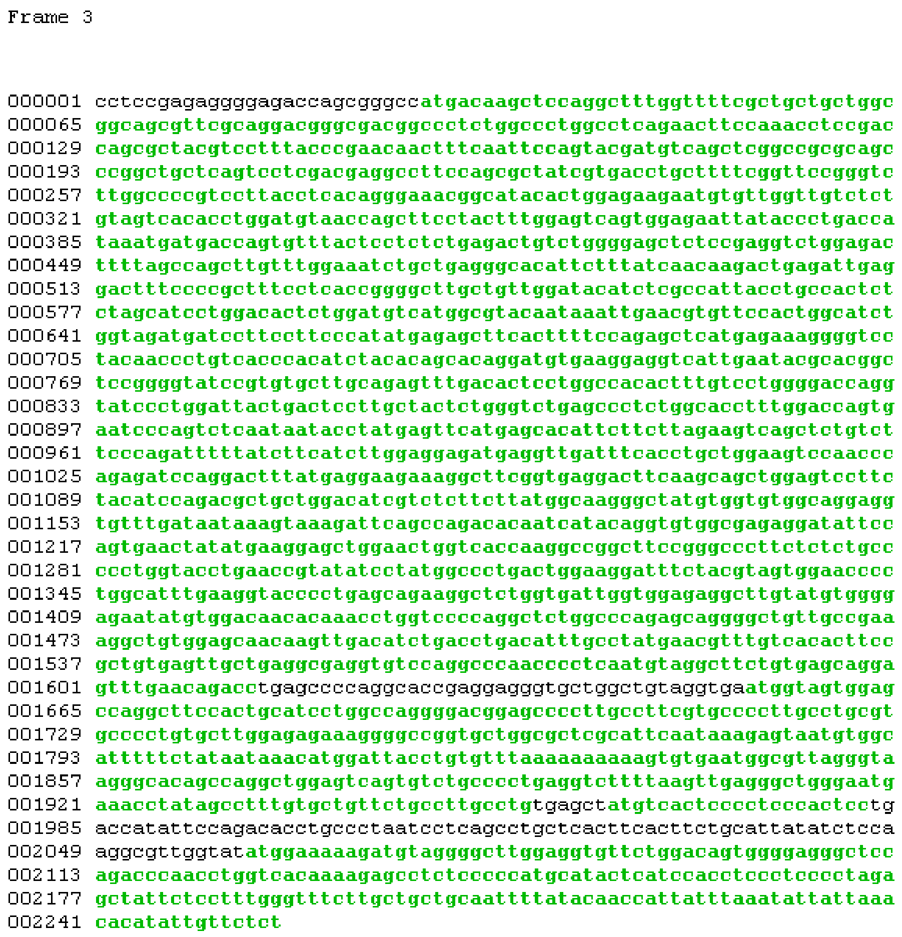
2 Look for open reading frames in the human gene. For example, for the human gene HEXA, type

```
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 tcccagatttttatcttcatcttggaggagatgaggttgatttcacctgctggaagtccaaccc
001025 agagatccaggactttatgaggaagaaaggcttcggtgaggacttcaagcagctggagtccttc
001089 tacatccagacgctgctggacatcgtctcttcttatggcaagggctatgtggtgtggcaggagg
001153 tgtttgataataaagtaaagattcagccagacacaatcatacaggtgtggcgagaggatattcc
001217 agtgaactatatgaaggagctggaactggtcaccaaggccggcttccgggcccttctctctgcc
001281 ccctggtacctgaaccgtatatcctatggccctgactggaaggatttctacgtagtggaacccc
001345 tggcatttgaaggtaccccctgagcagaaggctctggtgattggtggagaggcttgtatgtgggg
001409 agaatatgtggacaacacaaacctggtccccaggctctggcccagagcaggggctgttgccgaa
001473 aggctgtggagcaacaagttgacatctgacctgacatttgcctatgaacgtttgtcacacttcc
001537 gctgtgagttgctgaggcgaggtgtccaggcccaacccctcaatgtaggcttctgtgagcagga
001601 gtttgaacagacctgagccccaggcaccgaggagggtgctggctgtaggtgaatggtagtggag
001665 ccaggcttccactgcatcctggccaggggacggagccccttgccttcgtgccccttgcctgcgt
001729 gcccctgtgcttggagagaaaggggccggtgctggcgctcgcattcaataaagagtaatgtggc
001793 atttttctataataaacatggattacctgtgtttaaaaaaaaaagtgtgaatggcgttagggta
001857 agggcacagccaggctggagtcagtgtctgcccctgaggtcttttaagttgagggctgggaatg
001921 aaacctatagcctttgtgctgttctgccttgcctgtgagctatgtcactcccctcccactcctg
001985 accatattccagacacctgccctaatcctcagcctgctcacttcacttctgcattatatctcca
002049 aggcgttggtatatggaaaaagatgtaggggcttggaggtgttctggacagtggggagggctcc
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 ctccgtcgtcacagctgaatgtaatgaatttcctaatttggagtcggtagaaaattacaccta
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)
```

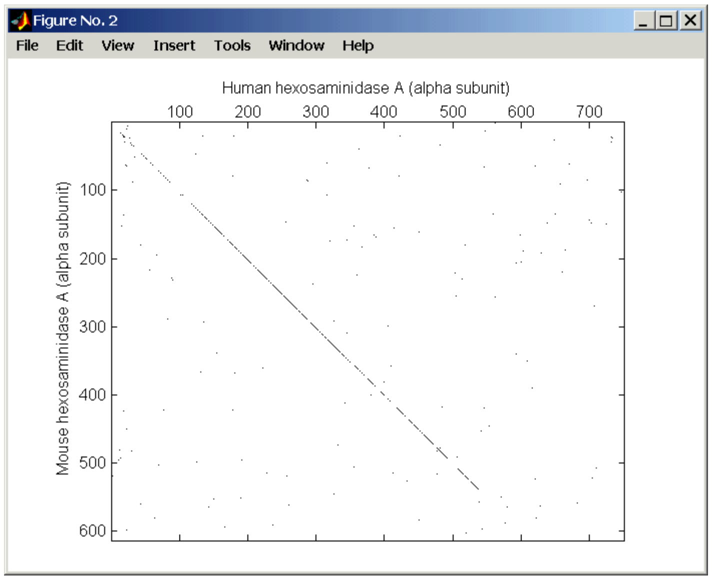
Remember that the human HEXA gene was on the third reading frame, so you need to indicate which frame to use.

```
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.



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

These gene expression profiles from microarray data can be used for

- Researching the function of cells and tissue types
- Comparing the differences between healthy and diseased tissue
- Observing changes with the application of drugs for drug discovery and monitoring treatments

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.

"Mouse Demo: An Example of Visualizing Microarray Data" on page 3–2	Create figures to visualize microarray data and get the data ready for analysis
"Yeast Demo: An Example of Gene Expression Profile Analysis" on page 3–25	Analyze microarray data for patterns and plot the results

Mouse Demo: An Example of Visualizing Microarray Data

This example looks at 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).

Defining the Example

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

```
http://www.pharmacology.ucla.edu/smithlab/genome_multiplex/
```

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 “Defining the 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 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 R†'
        '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

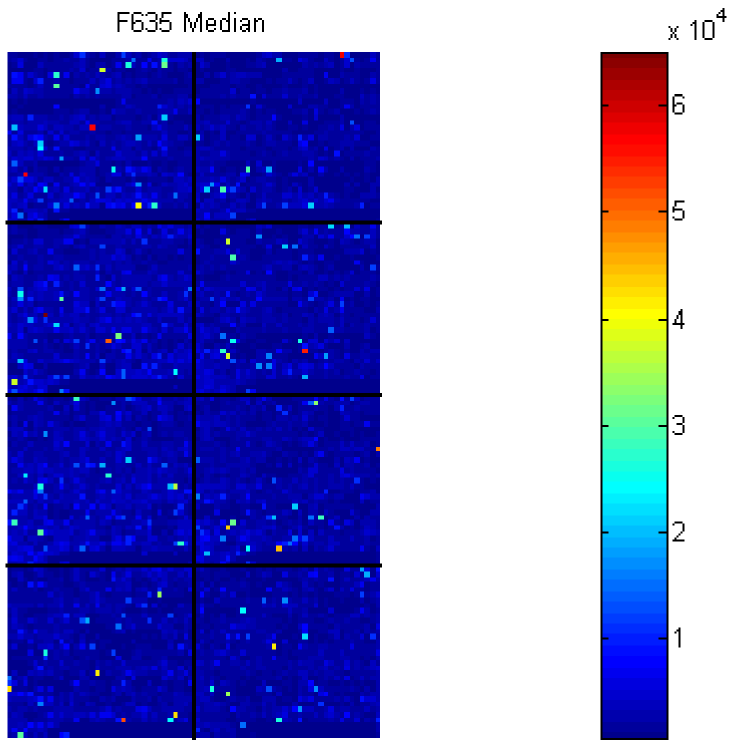
The function `maimage` can take a microarray data structure and create a pseudocolor image of the data arranged in the same order as the spots on the array. In other words, `maimage` plots a spatial plot of the microarray.

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')
   ```

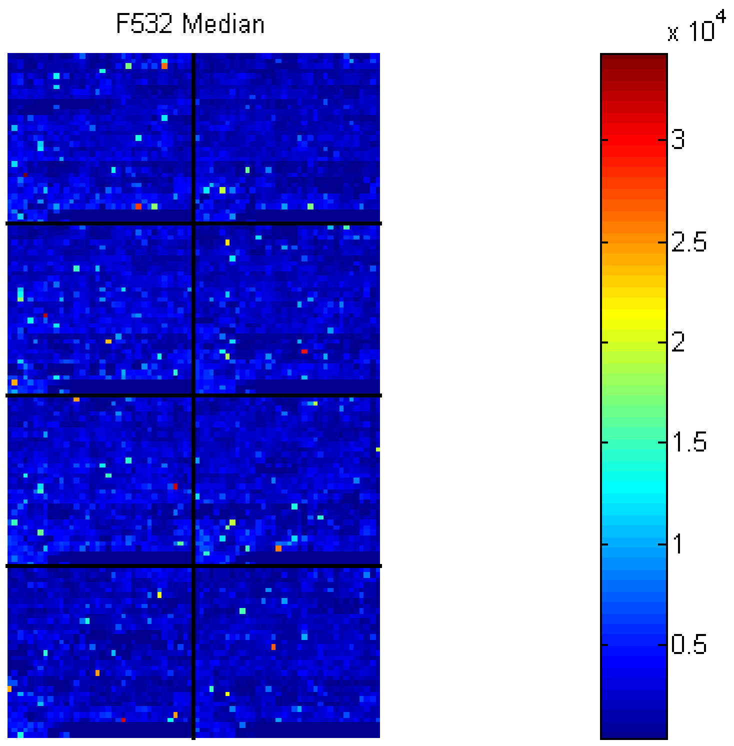
MATLAB plots an image showing the median pixel values for the foreground of the red (Cy5) channel.



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')
```

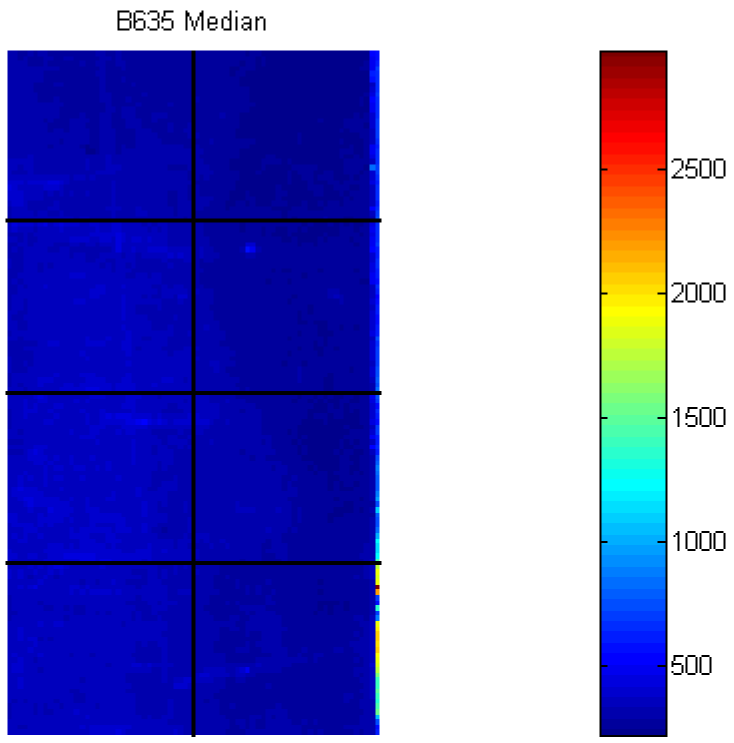
MATLAB plots an image showing the median pixel values of the foreground of the green (Cy3) channel.



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')
```

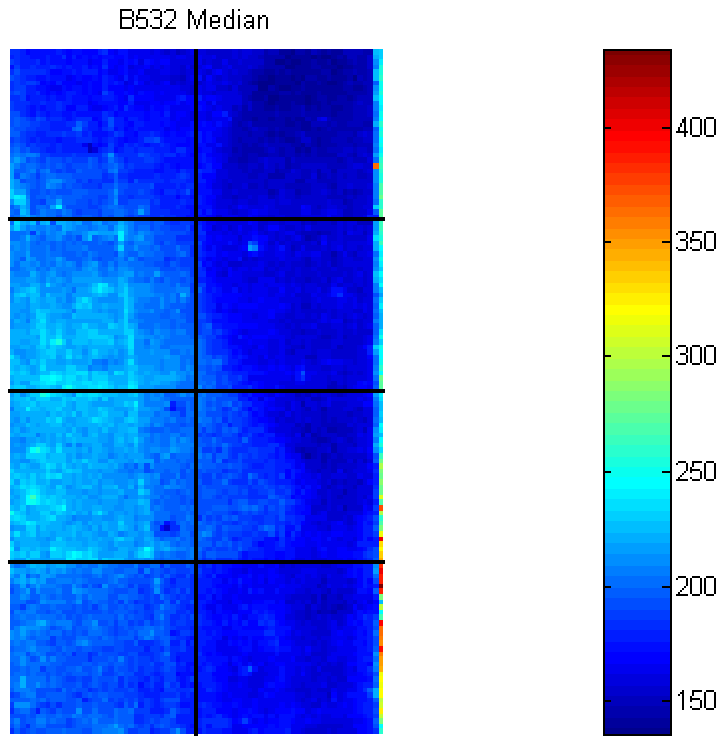
MATLAB plots an image for the background of the red channel. Notice the very high background levels down the right side of the array.



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')
```

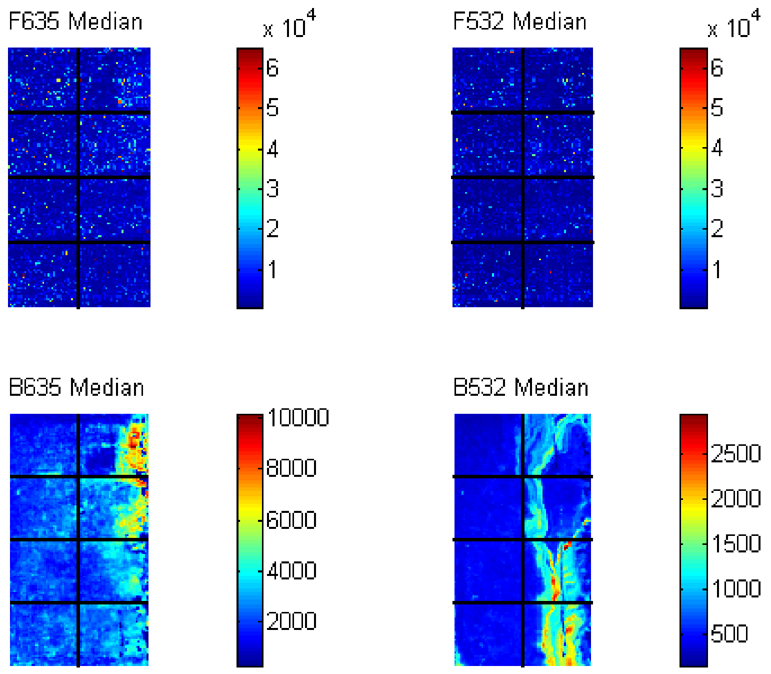
MATLAB creates a structure and displays information about the structure.

```
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')
```

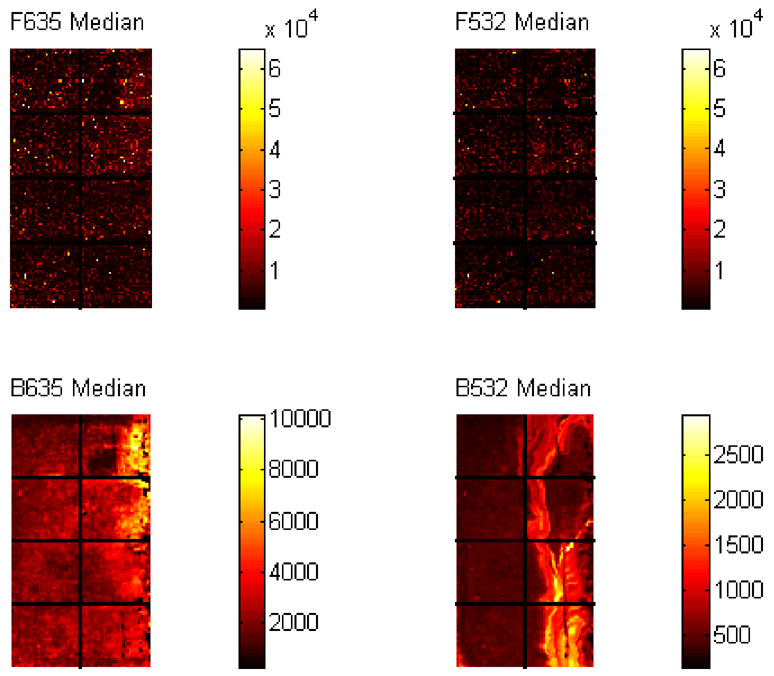
MATLAB plots the images.



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
```

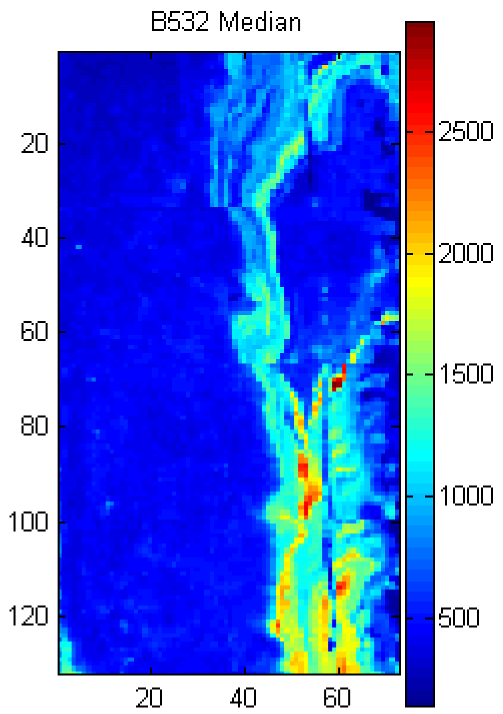
9 Extract that column from the field `Data`.

```
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')
```

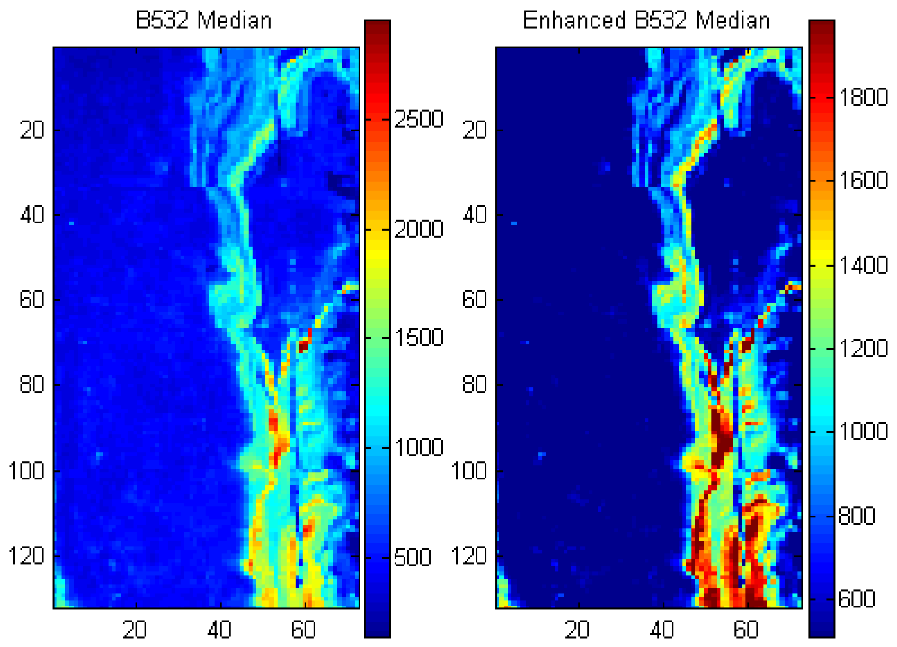
MATLAB plots the image.



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.



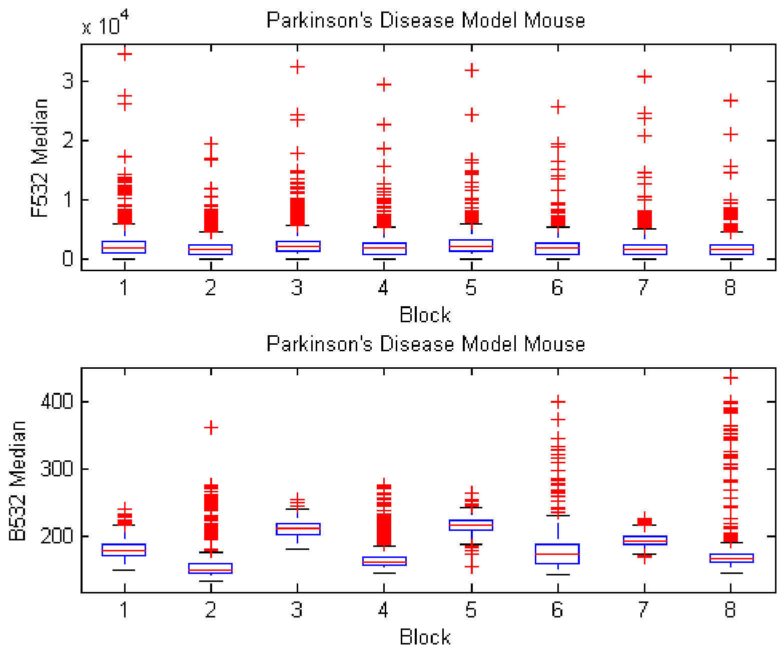
Statistics of the Microarrays

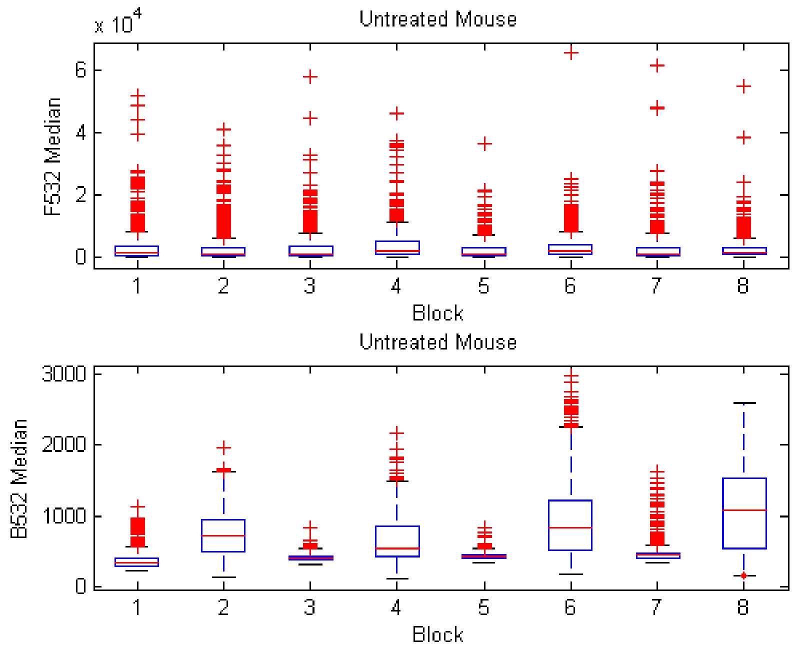
You can use the function `maboxplot` to look at the distribution of data in each of the blocks.

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 work with the data in the larger structure, it is often easier to extract the column numbers and then the 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

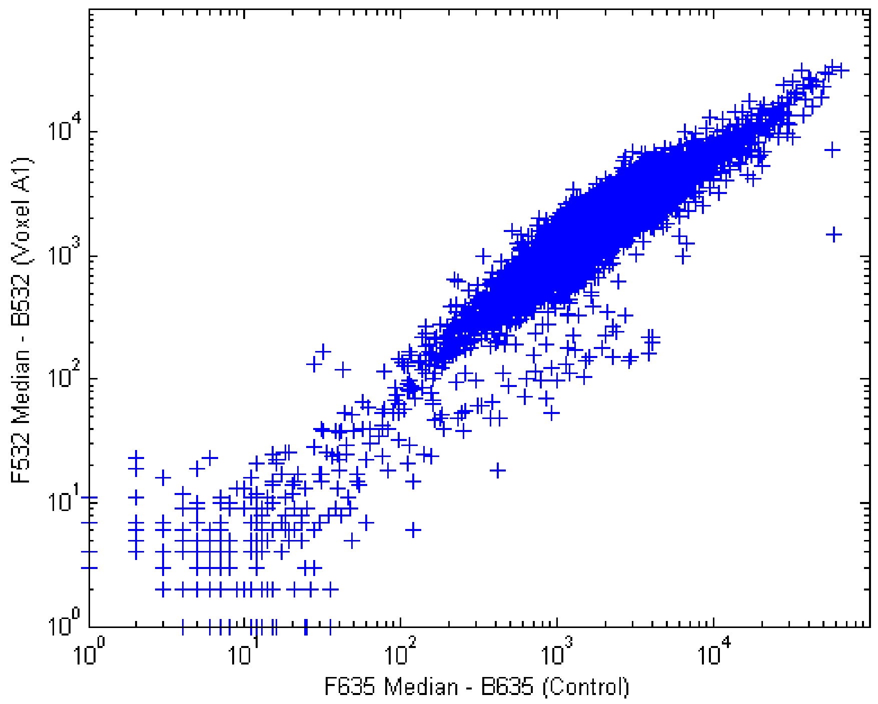
```
cy5DataCol =
    34


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.)
```



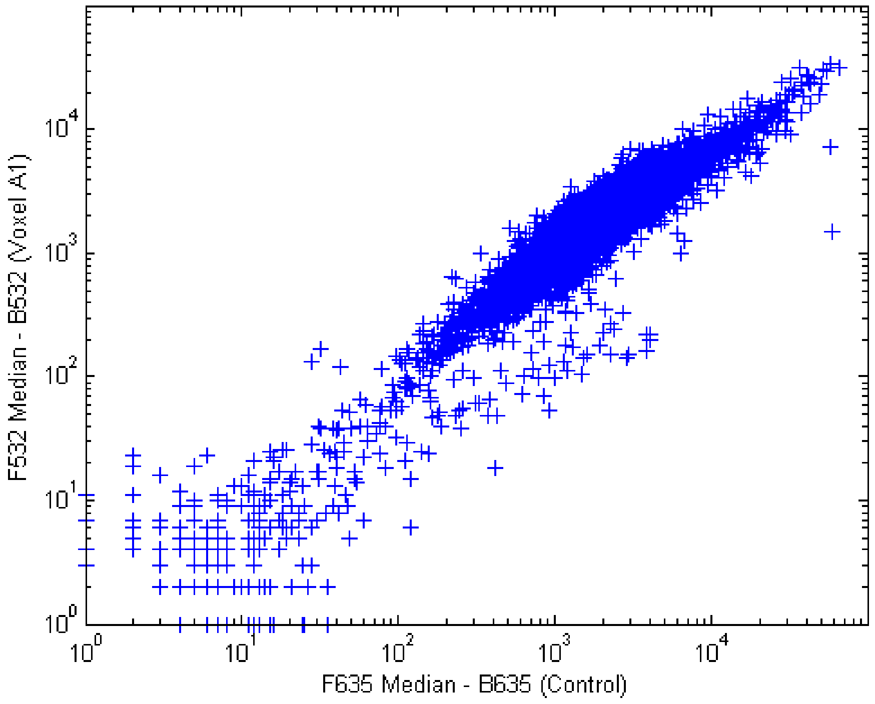
Notice that this function gives some warnings about negative and zero elements. This is because some of the values in the `'F635 Median - B635'` and `'F532 Median - B532'` columns are zero or even less than zero. Spots where this happened might be bad spots or spots that failed to hybridize. Points with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

3 These warnings can be disabled using the `warning` command, although it is good practice to investigate why warnings occur, rather than simply to ignore them, as 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)');
```

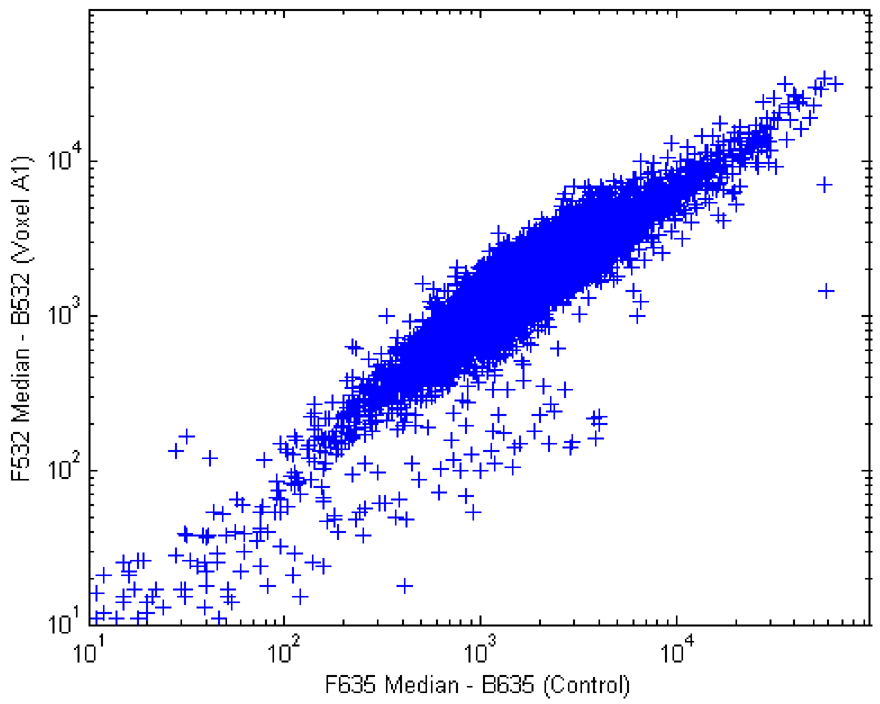
MATLAB plots the image.



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);
```

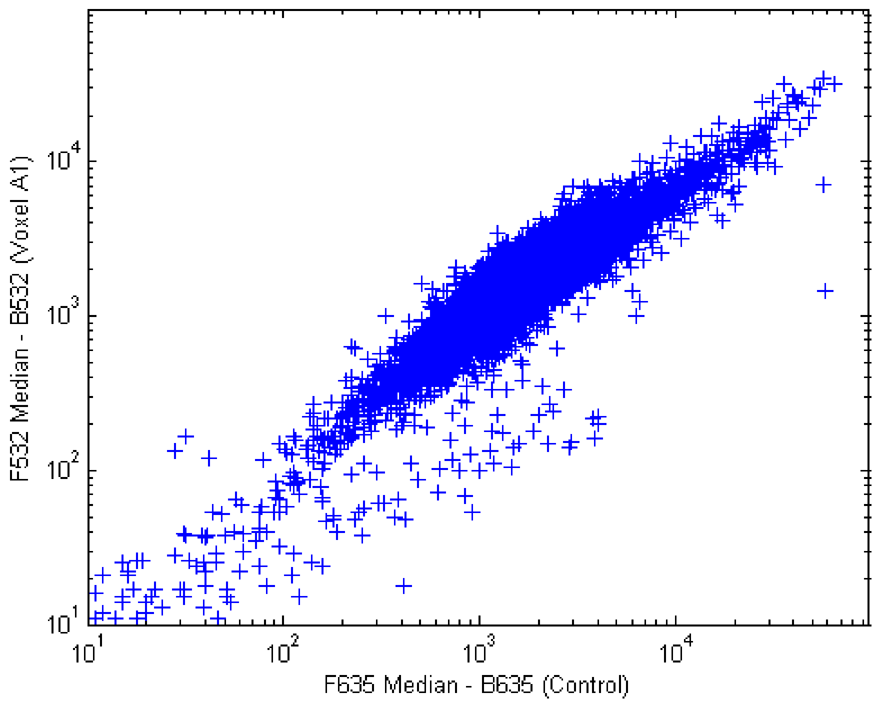
MATLAB plots the image.



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.

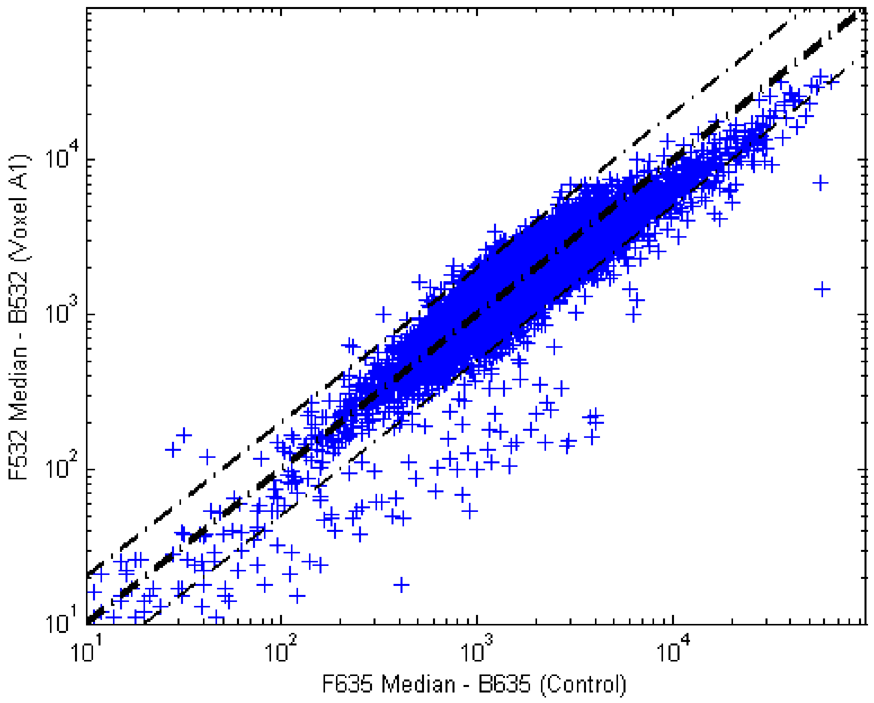


This plot shows the distribution of points but does not give any indication about which genes correspond to which points.

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.

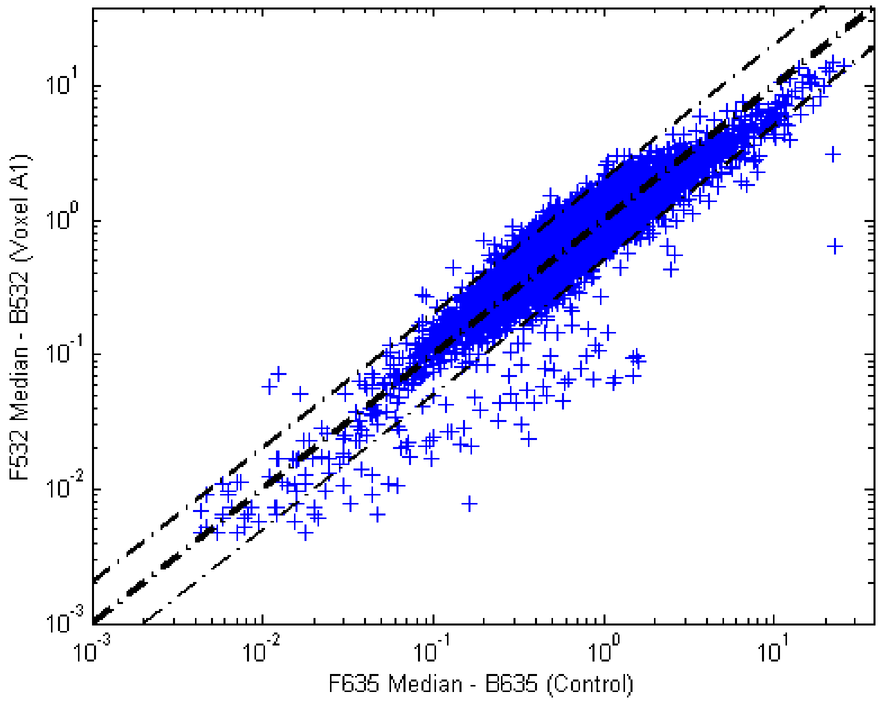
8 Normalize the points to evenly distribute them on either side of the line. Use the function `mameannorm` to perform global mean normalization.

```
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)');
```

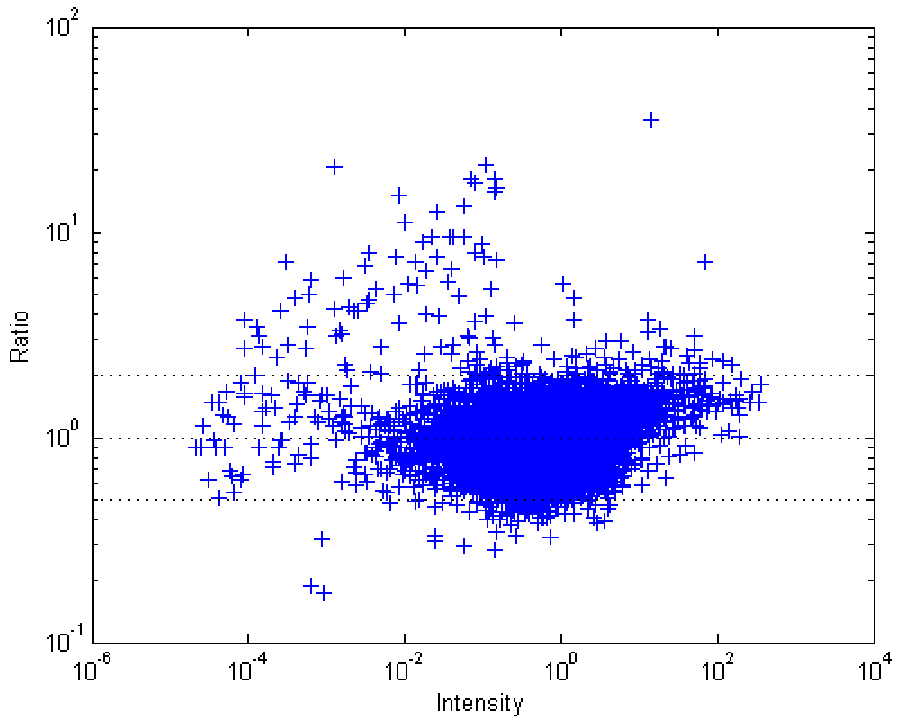
MATLAB plots the image.



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.

Yeast Demo: An Example of Gene Expression Profile Analysis

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

Exploring the Data Set

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
```

The data is also 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 `15`th row of the variable `yeastvalues`, which contains expression levels for the open reading frame (ORF) YAL054C.

```
ans =
  YAL054C
```

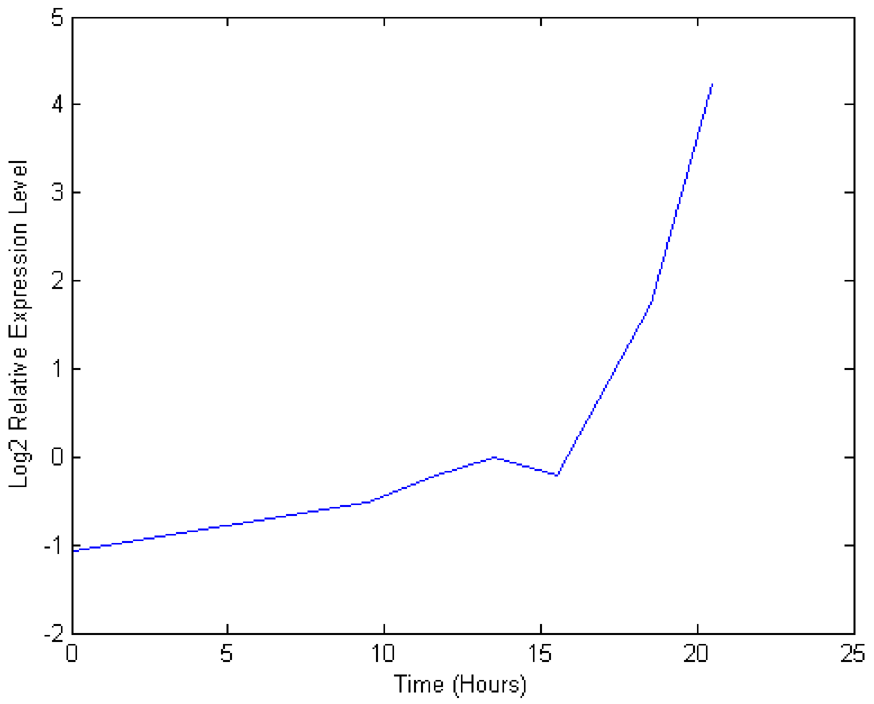
4 Use the function `web` to access information about this ORF in the Saccharomyces Genome Database (SGD).

```
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');
```

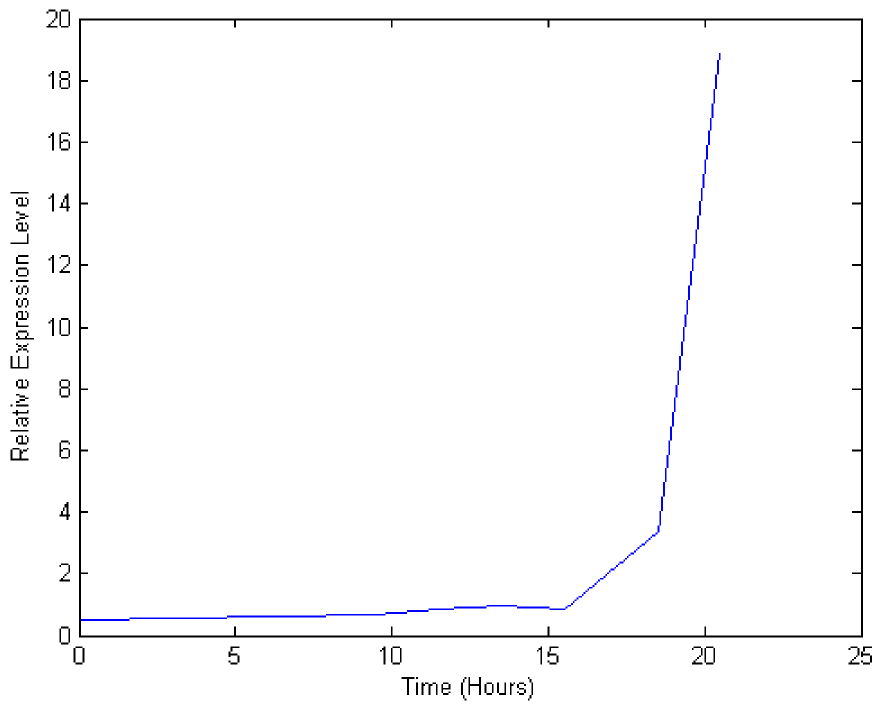
MATLAB plots the figure. The values are log2 ratios.



6 Plot the actual values.

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

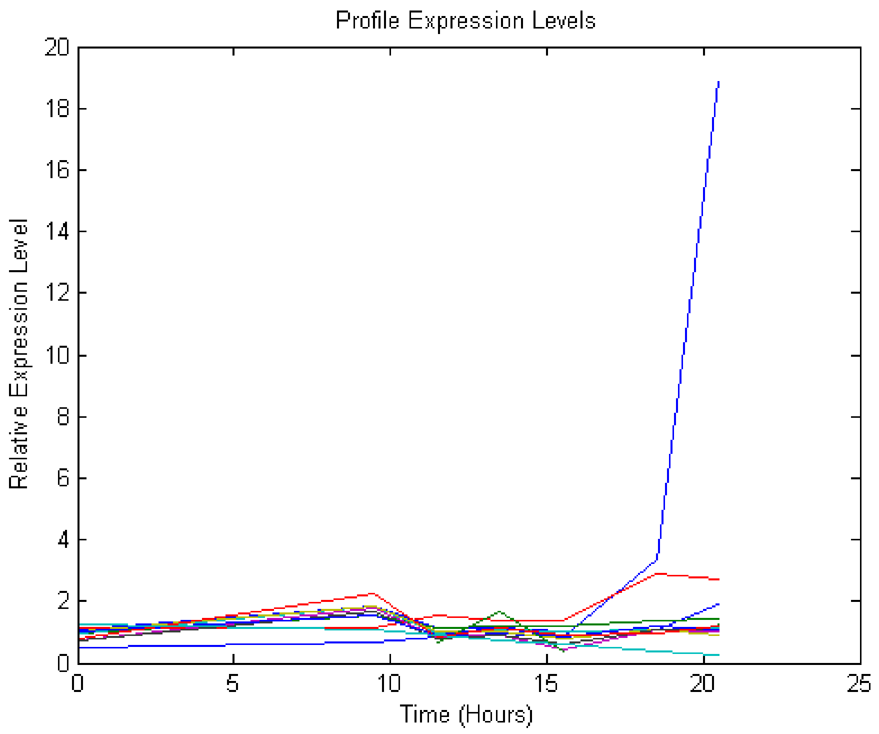
MATLAB plots the figure. The gene associated with this ORF, ACS1, appears to be strongly up-regulated during the diauxic shift.



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. You can find these points using the

function `strmatch` and remove the points from the data set with indexing functions.

```
emptySpots = strmatch('EMPTY',genes,'exact');
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 `10`th 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');
```

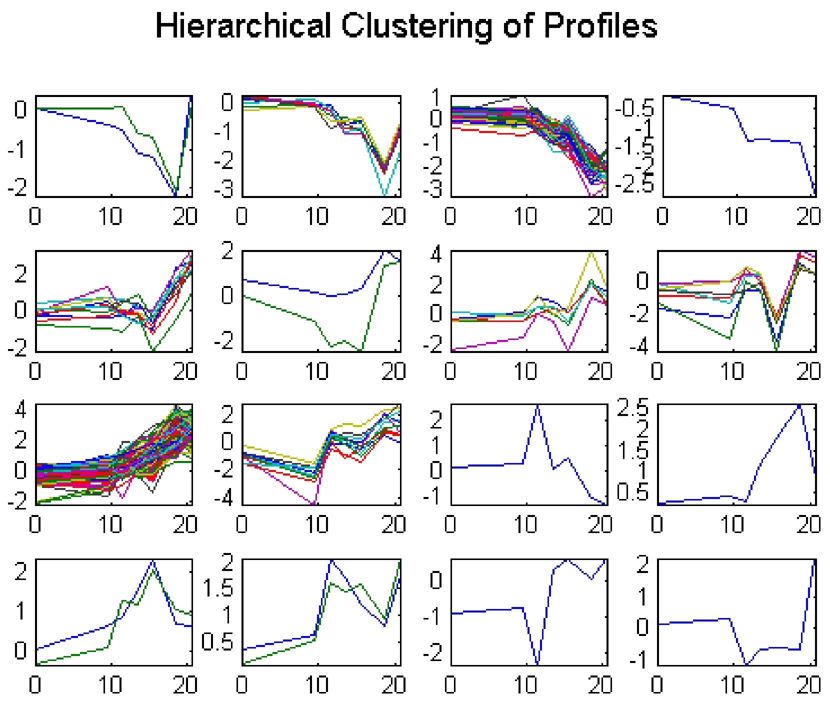
2 The function `cluster` calculates the clusters based on either a cutoff distance or a maximum number of clusters. In this case, the `'maxclust'` option is used to identify `16` distinct clusters.

```
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.

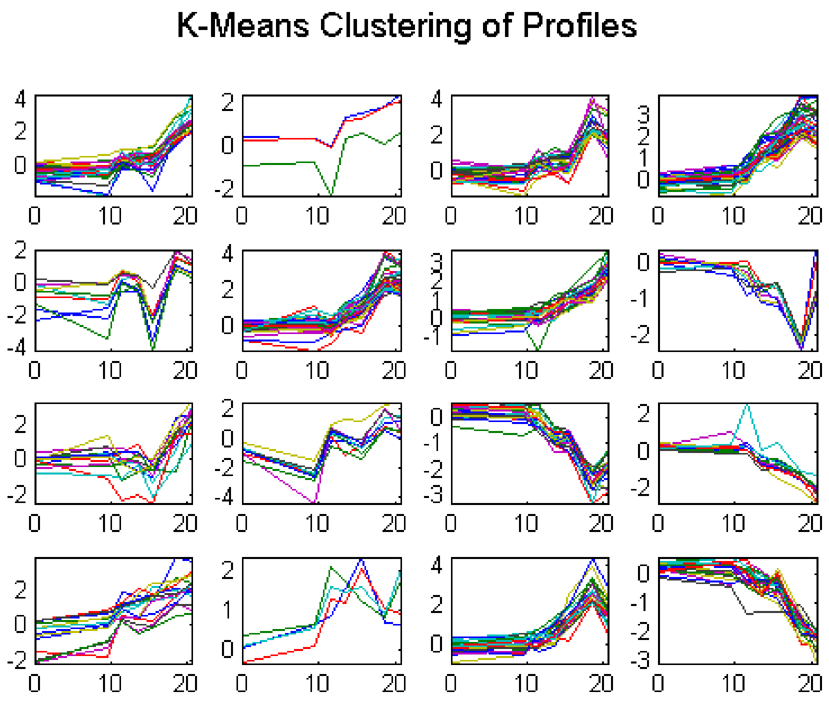


4 The Statistics Toolbox also has a K-means clustering function. Again, sixteen clusters are found, but because the algorithm is different these are not necessarily the same clusters as those found by hierarchical clustering.

```
[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

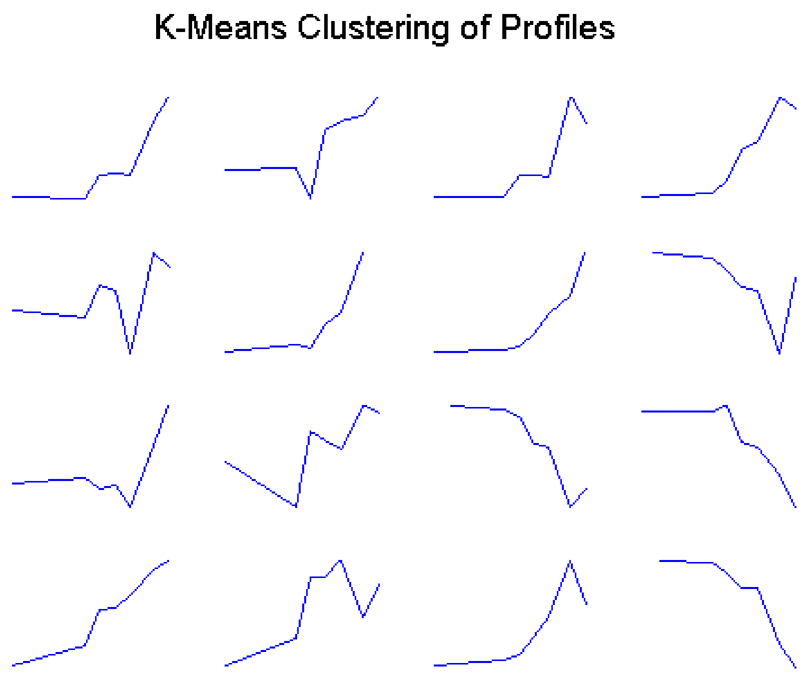
```
13 iterations, total sum of distances = 11.4042
14 iterations, total sum of distances = 8.62674
26 iterations, total sum of distances = 8.86066
22 iterations, total sum of distances = 9.77676
26 iterations, total sum of distances = 9.01035
```



5 Instead of plotting all 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');
```

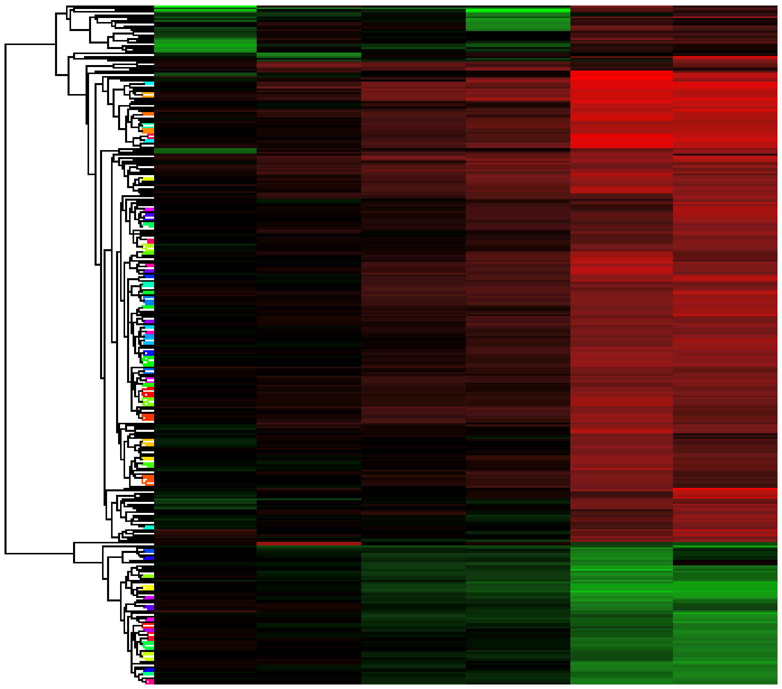
MATLAB plots the figure.



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

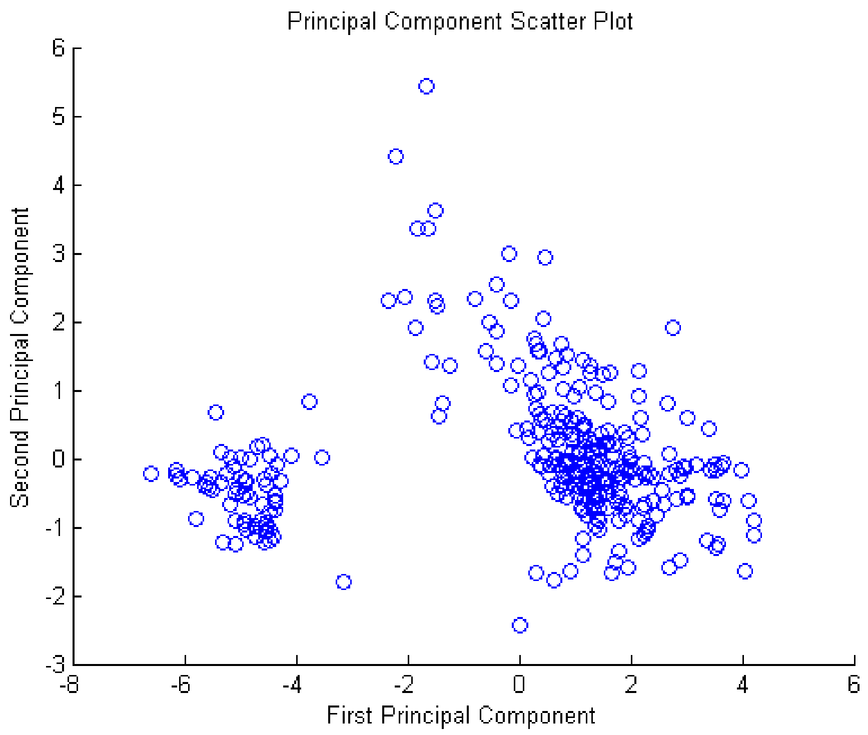
```
ans =
   78.3719
   89.2140
   93.4357
   96.0831
   98.3283
   99.3203
  100.0000
```

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.

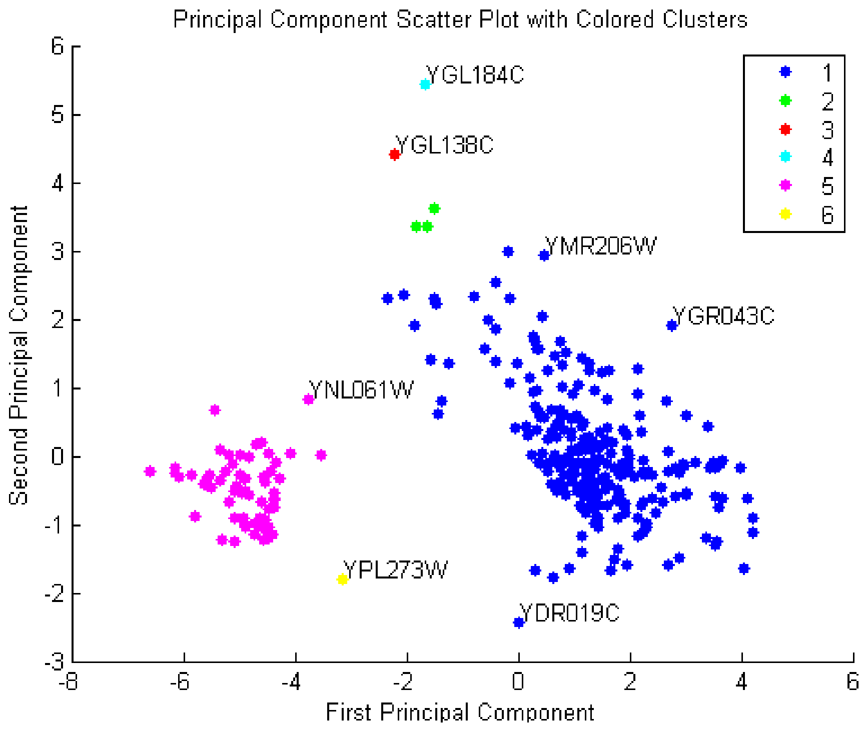
```
gname(genes);
```

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.



Functions – By Category

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 Internet data bases into MATLAB, and read and write to files within MATLAB using specific data formats.

emblread	Read data from an EMBL file
fastaread	Read data from a FASTA formatted file
fastawrite	Write to a file using a FASTA format
genbankread	Read data from a GenBank file
genpeptread	Read data from a GenPept file
getembl	Retrieve sequence information from the EMBL database
getgenbank	Retrieve sequence information from the GenBank database
getgenpept	Retrieve sequence information from the GenPept database
getpdb	Retrieve protein structure information from the PDB database
getpir	Retrieve sequence data from the PIR-PSD database
pdbread	Read data from a Protein Data Bank (PDB) file
pirread	Read data from a PIR file
scfread	Read trace data from a SCF 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
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
basecount	Count the number of nucleotide bases in a sequence
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

geneticcode	Mapping for the genetic code
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
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

Multiple Sequence Alignment

List of multiple sequence alignment 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

Scoring Matrices for Sequence Alignment

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

Microarray File Formats

List of microarray file format functions

affyread	Read data from Affymetrix GeneChip formatted files
galread	Read array list data from a GenePix file
geosoftread	Read data from a Gene Expression Omnibus (GEO) SOFT file
getgeodata	Get Gene Expression Omnibus (GEO) data
gprread	Read data results from a GenePix Results (GPR) 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
redgreencmap	Display a red and green colormap

Microarray Normalization and Filtering

List of microarray normalization and filtering functions

exprprofrange	Calculate the range of expression profiles
exprprofvar	Calculate the variance of 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

Protein Analysis

List of protein analysis functions

aacount	Count the amino acids in a sequence
aminolookup	Display amino acid codes, names, and abbreviations
atomiccomp	Calculate the atomic composition of a protein
cleave	Cleave a protein with an enzyme
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

Tutorials, Demos, and Examples

`aligndemo`	Basic sequence alignment tutorial
`alignsigdemo`	Advanced sequence alignment tutorial
`alignscoringdemo`	Tutorial showing the use of scoring matrices
`seqstatsdemo`	Sequence statistics tutorial example
`hmmprofdemo`	HMM profile alignment tutorial example
`mousedemo`	Microarray normalization and visualization example
`yeastdemo`	Microarray data analysis example

Functions — Alphabetical List

pdbread
pfamhmmread
pirread
proteinplot
ramachandran
randseq
redgreencmap
restrict
revgeneticcode
rna2dna
scfread
seq2regexp
seqcomplement
seqdotplot
seqrcomplement
seqreverse
seqshoworfs
seqshowwords
seqwordcount
showalignment
showhmmprof
sptread
swalign
traceplot

aa2int

Purpose

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

Syntax

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

Arguments

`SeqChar`	Amino acid sequence. Enter a string with characters from the table Mapping Amino Acid Letter to Numbers below.
`SeqInt`	Amino acid sequence represented with numbers.

Mapping Amino Acid Letters to Numbers

Amino Acid		**Amino Acid**	
Alanine	`A 1`	Phenylalanine	`F 14`
Arginine	`R 2`	Proline	`P 15`
Asparagine	`N 3`	Serine	`S—16`
Aspartic acid (Aspartate)	`D 4`	Threonine	`T—17`
Cysteine	`C 5`	Tryptophan	`W 18`
Glutamine	`Q 6`	Tyrosine	`Y 19`
Glutamic acid (Glutamate)	`E 7`	Valine	`V 20`
Glycine	`G 8`	Aspartic acid or Asparagine	`B 21`
Histidine	`H 9`	Glutamic acid or glutamine	`Z 22`

Amino Acid		Amino Acid	
Isoleucine	`I 10`	Unknown or any amino acid	`X 23`
Leucine	`L 11`	Translation stop	`* 24`
Lysine	`K 12`	Gap of indeterminate length	`- 25`
Methionine	`M 13`	Any character or symbol not in table	`? 0`

Description

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

Examples

Convert a character string 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 Standard Genetic Code below. Examples: `'ARN'` or `[1 2 3]`
`GeneticCodeValue`	Property to select a genetic code. Enter a code number or code name from the table . 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`.

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`

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

Genetic Code

Code Number		Code Number	
1	`Standard`	12	`Alternative Yeast Nuclear`
2	`Vertebrate Mitochondrial`	13	`Ascidian Mitochondrial`
3	`Yeast Mitochondrial`	14	`Flatworm Mitochondrial`
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 amino acid to nucleotide codon is not a one-to-one mapping, so the function chooses a random codon corresponding to a 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.

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

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

Examples

Convert the 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 one of the ambiguous amino acid characters `B`, `Z`, or `X`.

```
aa2nt('abcd')

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

See Also

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

Purpose

Count the amino acids in a sequence

Syntax

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

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

Arguments

`SeqAA`	Amino acid sequence. Enter a character string or a vector of integers from the table Mapping Amino Acid Letters to Numbers on page 5–4. Examples: `'ARN'` or `[1 2 3]`
`ChartStyle`	Property to select the type of plot. Enter `'pie'` or `'bar'`.
`OthersValue`	Property to control the counting of ambiguous characters individually. Enter either `'full'` or `'bundle'`. Default value is `'bundle'`.

Description

`aacount(SeqAA, 'PropertyName', PropertyValue)` counts the number of each amino acid in an amino acid sequence and returns these numbers in a `1`-by-`1` structure 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 characters other than the `20` standard amino acids are present in the sequence, a warning is given and the field `Others` is added to the structure.

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

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

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

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

- If `Others` equals `'full'`, ambiguous characters are listed separately, asterisks are counted in a new field, `Stop`, and hyphens are counted in a new field, `Gap`.

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

`aacount(...,'Others', OthersValue)` If `Others` equals `'full'`, counts all the standard ambiguous amino acid symbols individually instead of bundling them all together into the `Others` field of the output structure.

Examples

Count the amino acids in the string `'MATLAB'`.

```
aacount('MATLAB')

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

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

Purpose

Read data from Affymetrix GeneChip formatted files

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.
`AFFYData`	MATLAB structure.

Description

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

`AFFYData = affyread(File,LibraryDir)` allows you to specify the directory where the library files (`CDF`) are stored.

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

GeneChip and Affymetrix are registered trademarks of Affymetrix, Inc.

See Also

Bioinformatics Toolbox functions `gprread`, `sptread`

aminolookup

Purpose

Display amino acid codes, names, and abbreviations

Syntax

```
aminolookup
aminolookup('Letter', LetterValue)
aminolookup('Code', CodeValue)
aminolookup('Name', NameValue)
aminolookup(SeqAA)
```

Arguments

`LetterValue`	Amino acid single-letter code. Enter a single character from the Amino Acid Lookup Table below.
`CodeValue`	Amino acid three-letter code. Enter one of the three-letter codes from the Amino Acid Lookup Table below.
`NameValue`	Amino acid name. Enter one of the amino acid names from the Amino Acid Lookup Table below.
`SeqAA`	Amino acid sequence. Enter a character string of single letters or a sequence of three-letter amino acid codes from the Amino Acid Lookup Table below.

Amino Acid Lookup Table

Amino Acid	Letter Code	Three-Letter Code
Alanine	`A`	`Ala`
Arginine	`R`	`Arg`
Asparagine	`N`	`Asn`
Aspartic acid (Aspartate)	`D`	`Asp`
Cystine	`C`	`Cys`

Amino Acid	Letter Code	Three-Letter Code
Glutamine	Q	Gln
Glutamic acid (Glutamate)	E	Glu
Glycine	G	Gly
Histidine	H	His
Isoleucine	I	Ile
Leucine	L	Leu
Lysine	K	Lys
Methionine	M	Met
Phenylalanine	F	Phe
Proline	P	Pro
Serine	S	Ser
Threonine	T	Thr
Tryptophan	W	Trp
Tyrosine	Y	Tyr
Valine	V	Val
Aspartic acid or Asparagine	B	Asp or Asn
Glutamic acid or Glutamine	Z	Glu or Gln
ANY	X	ANY
Gap of unknown length	-	GAP
Terminator (translation stop)	*	END
Unknown amino acid	?	???

Description

`aminolookup('Letter', LetterValue)` displays the corresponding amino acid three-letter code and name. If you enter one of the ambiguous characters `B`, `Z`, `X`, this function displays the possible amino acids in square brackets.

```
aminolookup('abc')

ans =
Ala[Asp or Asn]Cys
```

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

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

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

The function `aminolookup` displays amino acid codes, names, and abbreviations from the Amino Acid Lookup Table above. Any character not in the table gives an error.

Examples

Display the letter and three-letter code 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('MetTrpLysGlnAlaGluAspIleArgAspIleTyrAspPhe')

ans =
MWKQAEDIRDIYDF
```

See Also

Bioinformatics Toolbox function `geneticcode`

Purpose

Calculate the atomic composition of a protein

Syntax

`atomiccomp(SeqAA)`

Arguments

`SeqAA`	Amino acid sequence. Enter a character string or vector of integers from the table Mapping Amino Acid Letters to Numbers on page 5–4.

Description

`atomiccomp(SeqAA)` counts the type and number of atoms in an amino acid sequence.

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
```

See Also

Bioinformatics Toolbox functions `aacount`, `molweight`

Purpose

Count the number of nucleotide bases in a sequence

Syntax

```
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`. For sequences with the letter `U`, the number of `U` bases is included with the count of `T` bases.
`ChartValue`	Property to select the type of plot. Enter either `'pie'` or `'bar'`.
`OthersValue`	Property to control counting ambiguous amino acids individually. Enter `'full'`. Default value is `'bundle'`.

Description

`basecount(SeqNT, '`*`PropertyName`*`',` *`PropertyValue`*`)` counts the number of each base in a nucleotide sequence and returns these numbers in a `1`-by-`1` structure with the fields `A`, `C`, `G`, `T`. If other characters are present in the sequence, then a warning is given and the field `Others` is added to the structure.

- For sequences with the character `U`, the count of `U` bases is added to the count of `T` bases.

- If the sequence contains nucleotides (bases) with ambiguous characters (`R`, `Y`, `K`, `M`, `S`, `W`, `B`, `D`, `H`, `V`, `N`), or gaps indicated with a hyphen (`-`), a field `Others` is created and a warning message is displayed.

  ```
  Warning: Symbols other than A, C, T, and G appear in the
  sequence
  ```

- If the sequence contains any characters other than `A`, `C`, `G`, `T`, `U`, `R`, `Y`, `K`, `M`, `S`, `W`, `B`, `D`, `H`, `V`, `N`, and `-`, the characters are ignored and a warning message is displayed.

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

- If `OthersValue` is `'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. Valid chart styles are `'Pie'` and `'Bar'`.

`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 functions `aacount`, `codoncount`, `dimercount`

Purpose

Return a BLOSUM scoring matrix

Syntax

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

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

Arguments

`N`	Enter 30 to 90 in increments of 5, 62, or 100.
`ExtendedValue`	Property to control the listing of extended amino acid codes. Enter either `true` or `false`. Default value for `Extended` is `true`.
`OrderString`	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(N, '`*`PropertyName`*`', `*`PropertyValue`*`)` returns a BLOSUM matrix. 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`*`)` when `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', `*`OrderString`*`)` returns the BLOSUM matrix ordered by the amino acid sequence *`OrderString`*.

`[B, MatrixInfo] = blosum(N)` 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 `dayhoff`, `pam`, `gonnet`

Purpose

Cleave a protein with an enzyme

Syntax

```
cleave(SeqAA, PeptidePattern, Position)

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

Arguments

`SeqAA`	Amino acid sequence. Enter a character string or a vector of integers from the table Mapping Amino Acid Letters to Numbers on page 5–4. Examples: `'ARN'` or `[1 2 3]`.
`PeptidePattern`	Enter a short amino acid sequence. *`PeptidePattern`* can be 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`	Probability that a cleavage site will be cleaved. Enter a value from `0` to `1`. 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 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)
```

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.
`RowLabels`	Property to label the rows in `Data.ColLabels` Enter a cell array of text strings.
`ColumnLabels`	Property to label the columns in `Data`. Enter a cell array of text strings.
`PdistArgs`	Property to pass arguments to the function `pdist`.
`LinkageArgs`	Property to pass arguments to the function `linkage`.
`DendrogramArgs`	Property to pass arguments to the function `dendrogram`.

Description

`clustergram(Data, 'PropertyName', PropertyValue)` creates a dendrogram and heat map from `Data` using hierarchical clustering with correlation as the distance metric and average linkage used to generate the hierarchical tree.

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

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

`clustergram(...,'Pdist',` *`PdistValue`*`)` allows you to set the distance metric used by the function `pdist`. This function calculates the pairwise distances between observations. If the distance metric requires extra arguments then these should be passed as a cell array. For example, to use the Minkowski distance with exponent `P` you would use `{'minkowski', P}`. See the help for `pdist` for more details of the available options. The default distance metric for `clustergram` is `'correlation'`.

`clustergram(..., 'Linkage',` *`LinkageValue`*`)` allows you to select the linkage method used by the function `linkage`. This function creates the hierarchical cluster tree. For more details of the available options, see the help for `linkage`. The default linkage method used by `clustergram` is `'average'`.

`clustergram(..., 'Dendrogram',` *`DendrogramValue`*`)` allows you to pass arguments to the function `dendrogram`. This function creates the dendrogram. `Dendrogram` should be a cell arrays of parameter/value pairs that can be passed to `dendrogram`. For more details of the available options, see the help for `dendrogram` .

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

Examples

```
Data = sort(rand(20,4));
% Add some labels.
RowLabels = {'a','b','c','d','e','f','g','h','i','j',...
             'k','l','m','n','o','p','q','r','s','t'};
ColLabels = {'one','two','three','four'};
clustergram(Data,'RowLabels',RowLabels,...
                 'ColumnLabels',ColLabels);
% Change the clustering parameters.
clustergram(Data,'Pdist','euclidean',...
                 'Linkage','complete');
```

```
% Change the dendrogram color parameter.
clustergram(Data,'RowLabels',RowLabels,...
                 'Dendrogram',{'color',5});
```

See Also

Purpose

Count the number of codons in a nucleotide sequence

Syntax

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

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

Arguments

`SeqNT`	Nucleotide sequence. Enter a character string or vector of integers.
`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`. The default value is `false`.
`FigureValue`	Property to control plotting a heat map. Enter either `true` or `false`. Default value is `false`.

Description

`codoncount` counts the number of each codon in the sequence and displays a formatted table of the result.

`codoncount(SeqNT, 'PropertyName',PropertyValue)` returns the codon counts in a structure with the fields `AAA`, `AAC`, `AAG`, ..., `TTG`, `TTT`. If characters other than `A`, `C`, `G`, and `T` are in the sequence, then a warning is given and the field `Others` is added to the structure, containing the number of undetermined codons found in the sequence. Undetermined codons include ambiguous characters and undefined characters using the IUB/IUPAC code.

```
Warning: Symbols other than A, C, G and T appear in the sequence
```

[Codons, CodonArray] = codoncount(SeqNT) returns a 4x4x4 array of 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*) counts the codons for the reverse complement of the sequence if Reverse is true

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

Examples

Count the number of standard codons from the following sequence:

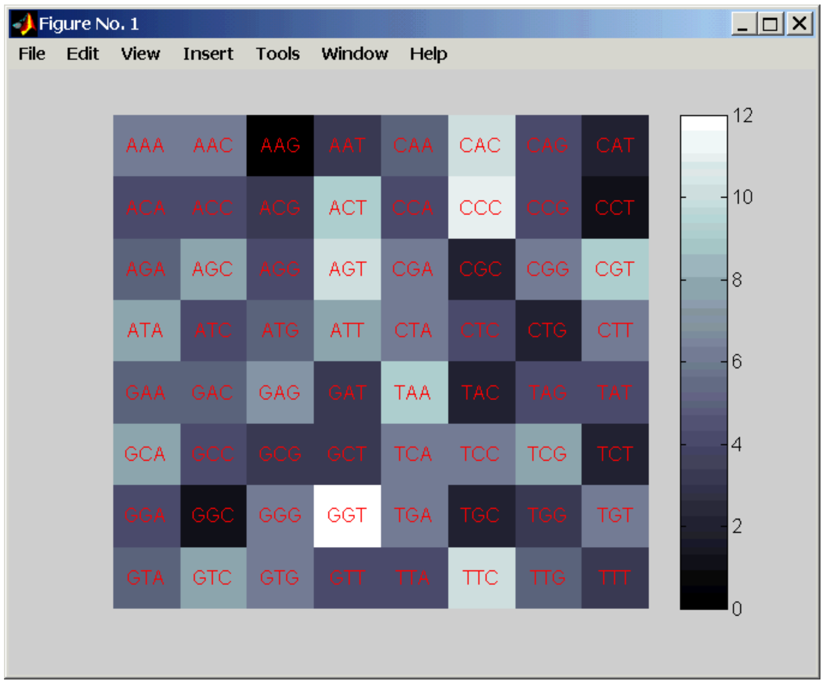
```
codons = codoncount('AAACGTTA')

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);
```



See Also

Bioinformatics Toolbox functions `basecount`, `dimercount`, `nmercount`, `seqrcomplement`

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

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

Description

`dimercount(SeqNT, 'PropertyName', PropertyValue)` counts the number of nucleotide dimers in a sequence and returns these numbers in a structure with the fields `AA`, `AC`, `AG`, `AT`, `CA`, `CC`, `CG`, `CT`, `GA`, `GC`, `GG`, `GT`, `TA`, `TC`, `TG`, `TT`. If other characters are present in the sequence, then a warning is given and the field `Others` is added to the structure.

- If the sequence contains nucleotides (bases) with ambiguous characters or undefined characters using the IUB/IUPAC code, a field `Others` is created and a warning message is displayed.

  ```
  Warning: Symbols other than A, C, G, and T appear in
  the sequence
  ```

- For sequences that include the character `U`, the `U`s are added to the count of dimers with the character `T`.

`[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 `basecount`, `codoncount`

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.
`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`*`')` 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 of 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`*`)` reads only the sequence information if `SequenceOnly` is `true`.

Examples

```
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 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.

Description

`exprprofrange(Data, 'PropertyName', PropertyValue)` calculates the range of each expression profile in the 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)` displays a histogram of the range data if `ShowHist` is `true`.

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 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.

Description

`exprprofvar(Data, 'PropertyName', PropertyValue)` calculates the variance of each expression profile in the dataset `Data`. If you do not specify output arguments, `exprprofvar` displays a histogram bar plot of the range.

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

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. 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. FASTA files are ASCII text files.
`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.

`FASTAData = fastaread('File')` reads a file with FASTA format and returns the data in the file as 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 `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 using a FASTA format or a structure containing the fields Sequence and Header.
Header	Information about the sequence.
Sequence	Nucleotide or amino acid sequence using the standard IUB/IUPAC codes. For a list of valid characters, see Mapping Amino Acid Letters to Numbers on page 5–4 and Mapping Nucleotide Characters to Integers on page 5–118.

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);
```

See Also Bioinformatics Toolbox function `fastaread`

Purpose

Read array list data from a GenePix file

Syntax

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

Arguments

File	GenePix Array List formatted file (file extension `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
```

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 `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 `getgenbank`, `getgenpept`, `genpeptread`

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 the filtered data matrix `FData`. `FData` can also be created using `FData = Data(find(I),:)`.

`[Mask, FData,FNames] = geneentropyfilter(Data, Names)` also returns the 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. Default value is `false`.

Description

Gene expression profile experiments have data where the absolute values are very low. This quality of this type of data is often bad due to large quantization errors or simply poor spot hybridization.

`Mask = genelowvalfilter(Data, '`*`PropertyName`*`', `*`PropertyValue`*`)` identifies gene expression profiles in `Data` with all absolute values less than the `10`th percentile.

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

`[Mask, FData] = genelowvalfilter(Data)` returns the filtered data matrix `FData`. `FData` can also be created using `FData = Data(find(I),:)`.

`[Mask, FData,FNames] = genelowvalfilter(Data, Names)` also returns the 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 set to `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)` calculates the range for each gene expression profile in `Data`, and then identifies the expression profiles with ranges less than the `10`th percentile.

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

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

`[Maks, FData, FNames] = generangefilter(Data, Names)` also returns the 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
[fyeastvalues, fgenes] = generangefilter(yeastvalues,genes);
```

See Also

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

Purpose

Mapping for the genetic code

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`

Code Number	Code Name
13	`Ascidian Mitochondrial`
14	`Flatworm Mitochondrial`
15	`Blepharisma Nuclear`
16	`Chlorophycean Mitochondrial`
21	`Trematode Mitochondrial`
22	`Scenedesmus Obliquus Mitochondrial`
23	`Thraustochytrium Mitochondrial`

Description

`geneticcode` returns a structure containing mappings for the genetic code.

`Map = geneticcode` returns a structure containing mappings 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 `nt2aa`, `revgeneticcode`

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 common task.

`Mask = genevarfilter(Data)` calculates the variance for each gene expression profile in `Data` and then identifies the expression profiles with a variance less than the `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)` also returns the 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.

`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 `genbankread`, `getgenbank`, `getgenpept`

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 into a MATLAB structure with the following fields:

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

`GEOSOFTData = geosoftread('File')` reads in Gene Expression Omnibus (GEO) SOFT formatted data from *File* and creates a structure `GEOSOFTData`, containing fields corresponding to the GenBank keywords. Each separate entry listed in *File* is stored as a separate element of the structure.

File can also be a URL or a MATLAB character array that contains the text of a GenBank formatted file.

Examples

Get a file from GEO and save it to a file.

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

In subsequent MATLAB sessions you can use `geosoftread` to access the 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`

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 `getgenbank`, `getgenpept`, `getpdb`

Purpose

Retrieve sequence information from the GenBank database

Syntax

```
Data = getgenbank('AccessionNumber',
                         'PropertyName',PropertyValue)

getgenbank(..., 'ToFile', ToFileValue)
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).
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 the following fields:

```
LocusName
LocusSequenceLength
LocusNumberofStrands
```

```
LocusTopology
LocusMoleculeType
LocusGenBankDivision
LocusModificationDate
Definition
Accession
Version
GI
Keywords
Segment
Source
SourceOrganism
Reference
Comment
Features
BaseCount
Sequence
```

`getgenbank(..., 'ToFile', ToFileValue)` returns a MATLAB structure containing information about the sequence and 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 GenBank formatted file back into MATLAB using the function `genbankread`.

`getgenbank(..., 'SequenceOnly', SequenceOnlyValue)` returns only the sequence information without the metadata if `SequenceOnly` is `true`.

Examples

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

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

See Also

Bioinformatics Toolbox functions `genbankread`, `getembl`, `getgenpept`, `getpdb`

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).
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). 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 the following fields:

```
LocusName
LocusSequenceLength
```

```
LocusNumberofStrands
LocusTopology
LocusMoleculeType
LocusGenBankDivision
LocusModificationDate
Definition
Accession
Version
GI
DBSource
Keywords
Source
SourceOrganism
Reference
Comment
Features
Sequence
```

`getgenpept(..., 'ToFile', ToFileValue)` returns a MATLAB structure containing information about the sequence and 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(..., 'SequenceOnly', SequenceOnlyValue)` returns only the sequence information without the metadata if `SequenceOnly` is `true`.

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`

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, secretin family (`PFAMKey = PF00002`).

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

or

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

See Also

Bioinformatics Toolbox function `fastaread`

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, secretin family. (`PFAM key = PF00002`)

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

or

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

See Also

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

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`, `pdbread`

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  *.
```

See Also Bioinformatics Toolbox functions `dayhoff`, `pam`

Purpose

Read data results from a GenePix Results (GPR) file

Syntax

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

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

Arguments

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

Description

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

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

`gprread(..., 'CleanColName', `*`CleanColNameValue`*`)` By default, `CleanColName` is `false`, and the column names in the `GPR` file contain spaces 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 then use this option with CleanColName set to true, and the function will return ColNames that are valid 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
```

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
% background intensity for the 635nm channel.
gprStruct = gprread('Demo.gpr')
maimage(gprStruct,'B635 Median');

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

b635Col = find(strcmp(gprStruct.ColumnNames,'B635 Median'));
B635Median = gprStruct.Data(:,b635Col);
 imagesc(B635Median(gprStruct.Indices));
colormap bone
colorbar
```

See Also

Bioinformatics Toolbox functions galread, maimage, sptread

Purpose

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

Syntax

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

hmmprofalign(..., 'ShowScore', ShowScoreValue)
hmmprofalign(..., 'Flanks', FlanksValue)
```

Description

`hmmprofalign(Model, Seq)` returns a string showing the profile alignment of amino acid (or nucleotide) sequence `Seq` to the hidden Markov model `Model`. 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`.

`[Alignment, Score] = hmmprofalign(Model, Seq)` returns the score for the optimal alignment. Scores are computed using log-odd ratios for emission probabilities and log probabilities for state transitions.

`hmmprofalign(..., 'ShowScore', ShowScoreValue)` displays the scoring space and the winning path if `ShowScore` is `true`.

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

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 `hmmprofestimate`, `hmmprofmerge`, `hmmprofstruct`, `showhmmprof`

Purpose

Estimate profile HMM parameters using pseudocounts

Syntax

```
hmmprofestimate(Model, MultipleAlignment)

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

Description

`hmmprofestimate(Model, MultipleAlignment)` 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 = value` when estimating the symbol emission probabilities. Default value is `20`.

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

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

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

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

See Also

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

Purpose

Generate a random sequence drawn from the profile HMM

Syntax

```
Sequence = hmmprofgenerate(Model)
[Sequence, Profptr] = hmmprofgenerage(Model)

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

Description

`Sequence = hmmprofgenerate(Model)` returns a string 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 (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 plain sequence of uppercase symbols. The default value is `true`.

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

`hmmprofgenerate(..., 'Signature', `*`SignatureValue`*`)` if `Signature` is `true`, returns the most likely path and symbols. The default value is `false`.

Examples

```
load('hmm_model_examples','model_7tm_2') % load a model example
rand_sequence = hmmprofgenerate(model_7tm_2)
```

See Also

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

Purpose

Concatenate the prealigned strings of several sequences to a profile HMM

Syntax

```
hmmprofmerge(Sequences)
hmmprofmerge(Sequences, Names)
hmmprofmerge(Sequences, Names, Scores)
```

Description

`hmmprofmerge(Sequences)` displays a set of prealigned `Sequences` to aHMM model profile. The output is aligned corresponding to the HMM states.

Match states: Uppercase letters

Insert states: Lowercase letters or dots (*)

Delete states: Dashes

(*) Dots are added accordingly to contain inserts in other sequences. The input sequences must have the same number of profile states (i.e. the combined count of uppercase 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`.

`A = hmmprofmerge( ...)` stores the output in `A`.

Examples

```
load('hmm_model_examples','model_7tm_2')  %load model
load('hmm_model_examples','sequences')  %load sequences
A=[];
for ind =1:length(sequences)
    [s,a]=hmmprofalign(model_7tm_2,sequences(ind).Sequence);
    A=strvcat(A,a);
end
hmmprofmerge(A,strvcat(sequences(:).Header))
```

See Also

Bioinformatics Toolbox functions `hmmprofalign`, `hmmprofstruct`

Purpose Create a profile HMM structure

Syntax

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

Description

`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 default values.

`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.

`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
	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.
	Format is
	`[N->B  C->T ;`
	`[N->N  C->C ]`
	Notes:
	`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]`.

Annotation fields (optional)

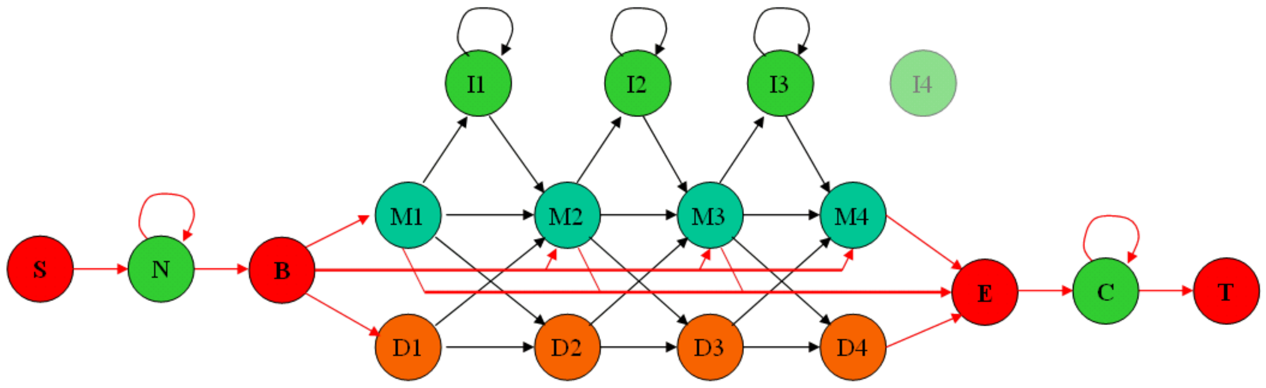
Name	Model Name
`IDNumber`	Identification Number
`Description`	Short description of the model

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 `hmmprofalign`, `hmmprofestimate`, `showhmmprof`

int2aa

Purpose

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

Syntax

`int2aa(SeqAA, 'PropertyName', PropertyValue)`

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

Arguments

`SeqAA`	Amino acid sequence. Enter a vector of integers from the table Mapping Amino Acid Integers to Characters below. The array does not have to be of type integer, but it does have to contain only integer numbers.
`CaseValue`	Enter either `'upper'` or `'lower'`. Default is `'upper'`.

Mapping Amino Acid Integers to Characters

Amino Acid		Amino Acid		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		Amino Acid		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

`int2aa` converts a `1`-by-`N` numeric array of integers representing amino acids to a `1`-by-`N` array of type `char` using the table above.

`int2aa(SeqAA)` converts the amino acid sequence from integers into a sequence of characters representing the amino acids.

`int2aa(SeqAA, '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

```
int2nt(SeqNT)

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

Arguments

`SeqNT`	Enter a nucleotide sequence represented as an integer array (vector). See Mapping Nucleotide Characters to Integers on page 5–118.
`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'`.

Mapping Nucleotide Integers to Characters

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` converts a `1`-by-`N` numeric array of integers representing nucleotide bases to a `1`-by-`N` character array.

`int2nt(SeqNT)` maps integers 16 and greater to the unknown character *. Standard nucleotide bases are mapped and ambiguous nucleotide bases are mapped.

`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 nt2int

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)
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)` 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)

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

Description

`maimage(X, FieldName)` 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`

Purpose

Display intensity versus ratio scatter plot for microarray signals

Syntax

```
mairplot(X, Y)

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

Description

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

`mairplot(..., 'FactorLines', N)` 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`

Purpose

Create a loglog plot of microarray data

Syntax

```
maloglog(X, Y)

maloglog(..., 'FactorLines', N)
maloglog(..., 'Title', TitleValue)
maloglog(..., 'Labels', LablesValues)
maloglog(..., HandleGraphics name/value)
H = maloglog(...)
```

Description

`maloglog(X, Y)` 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)

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

Description

`YSmooth = malowess(X, Y)` 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)
[XNorm, MAD] = mamadnorm(X)

mamadnorm(..., 'Global', GlobalValue)
```

Description

`XNorm = mamadnorm(X)` 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)
[XNorm, ColMean] = mameannorm(X)

mameannorm(..., 'Prctile', PrctileValue)
mameannorm(..., 'Global', GlobalValue)
```

Description

`XNorm = mameannorm(X)` 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 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 Standard Genetic Code on page 5–7.

Description `molweight(SeqAA)` calculates the molecular weight for `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 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.
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`

Purpose

Convert a sequence of nucleotides to a sequence of amino acids

Syntax

```
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.
`FrameValue`	Property to select a frame. Enter `1`, `2`, `3`, or `'ALL'`. 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 5–115. 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 `true` or `false`. Default 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

`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 later `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`, `geneticcode`, `revgeneticcode`

Purpose

Convert a nucleotide sequence from a letter to an integer representation

Syntax

```
nt2int(SeqNT, 'PropertyName', PropertyValue)

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

Arguments

`SeqNT`	Nucleotide sequence. Enter a string with the characters `A`, `C`, `T`, `G`, and `U`. If `ACGTOnly` is `false`, you can also include characters for the ambiguous nucleotide bases `N`, `R`, `Y`, `K`, `M`, `S`, `W`, `B`, `D`, `H`, and `V`. See table Mapping Nucleotide Characters to Integers on page 5–118.
`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`.

Mapping Nucleotide Characters to Integers

Base		Base		Base	
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		Base		Base	
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 converts a character string of nucleotide bases to a 1-by-N array of integers.

nt2int(SeqNT, '*PropertyName*', *PropertyValue*) converts a nucleotide sequence of characters into an array of integers. Unknown characters are mapped to 0.

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 numbers.

```
s = nt2int('ACTGCTAGC')

s =
     1     2     4     3     2     4     1     3     2
```

See Also

Bioinformatics Toolbox function int2nt

Purpose

Plot the density of nucleotides along a sequence

Syntax

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

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

Arguments

`SeqNT`	Nucleotide sequence. Enter a string of characters from the table Mapping Nucleotide Characters to Integers on page 5–118.

Description

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

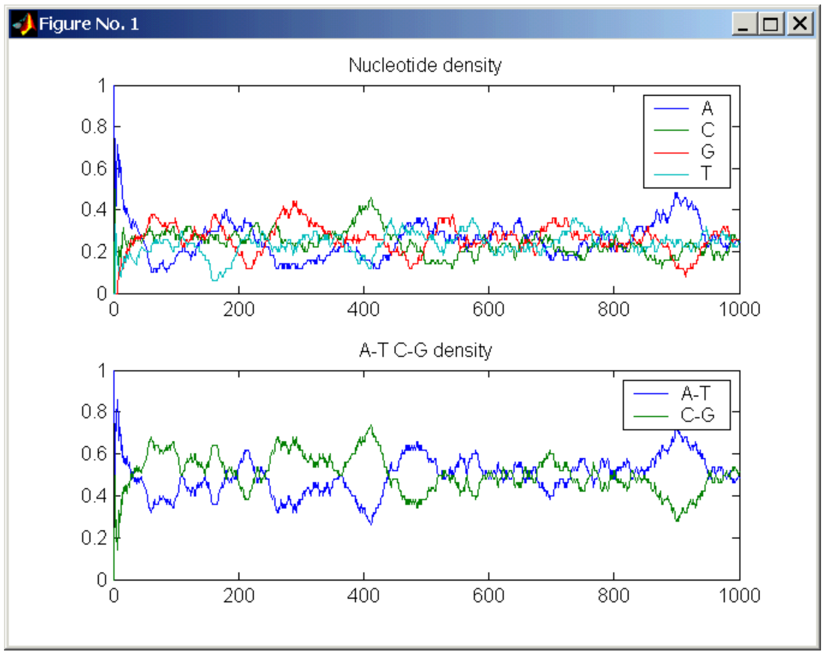
`Denstity = ntdensity(SeqNT, 'PropertyName', PropertyValue)` returns a MATLAB structure with the density of nucleotides `A`, `C`, `G`, and `T`.

`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`

nuc44

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`.

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 sequences.
`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)
[Postion, Length, Pal] = palindromes(SeqNT)

palindromes(..., 'Length', LengthValue)
palindromes(..., 'Complement', ComplementValue)
```

Arguments

`SeqNT`	Nucleotide. Enter a character string.

Description

`[Position, Length] = palindromes(SeqNT)` finds all palindromes in sequence `SeqNT` with a length greater than or equal to `6`, and returns the starting indices, `Position`, and the lengths of the palindromes, `Length`.

`[Position, Length, Pal] = palindromes(SeqNT)` also returns a cell array `Pal` of the palindromes.

`palindromes(..., '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'
```

Purpose

Return a PAM scoring matrix

Syntax

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

ScoringMatrix = pam(..., 'Extended', false)
ScoringMatrix = pam(..., 'Order', 'OrderString')
```

Arguments

`N`	Supported values of `N` are `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 *`.

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')` returns a scoring matrix with the `20` amino acid characters if `Extended` is `false`. It does not include ambiguous and stop characters `B`, `Z`, `X`, and `*`.

`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`

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`, `getpdb`, `getgenpept`

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 `gethmmprof`, `hmmprofalign`, `hmmprofstruct`, `showhmmprof`

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.

Description

`PIRData = pirread('File')` reads data from a Protein Information Resource (PIR-PSD) formatted file *File* and creates a MATLAB structure `PIRData` with the following fields:

```
Entry
EntryType
Title
Organism
Date
Accessions
Reference
Genetics
Classification
Keywords
Feature
Summary
Sequence: [1x105 char]
```

`pirread('String')` attempts to retrieve PIR data from the string `String`.

For more information on the PIR-PSD database, see

```
http://pir.georgetown.edu
```

Examples

Get protein information for cytochrome C from the PIR-PSD database, save the information in the file `cchu.txt`, and then read the information back into MATLAB.

```
getpir('cchu', 'ToFile', 'cchu.txt')
pirdata = pirread('cchu.txt')
```

See Also

Bioinformatics Toolbox functions `genpeptread`, `getpir`, `pdbread`

Purpose Display property values for amino acid sequences

Syntax

```
proteinplot(Seq_AA)
```

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

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`.

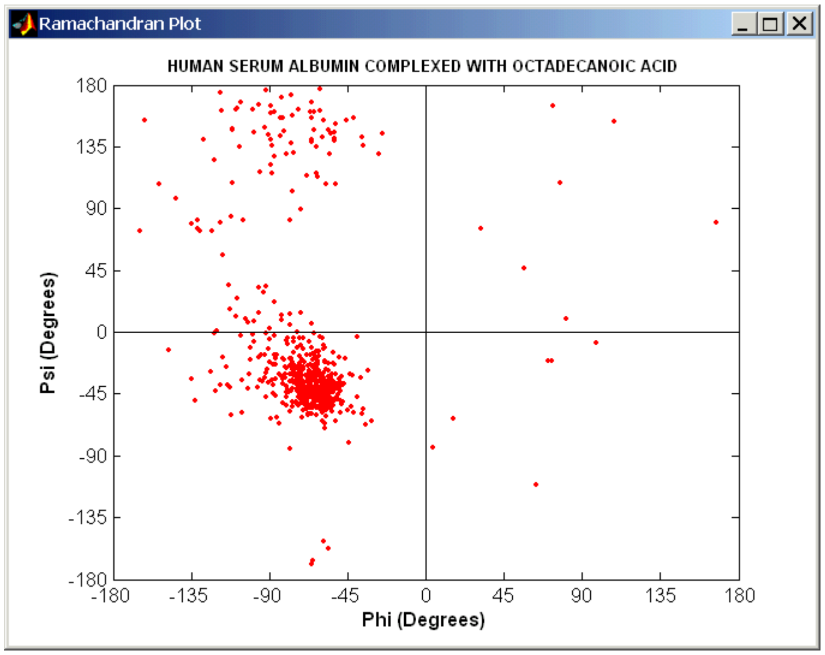
`Angles = ramachandran(...)` returns an array of the torsion angles PHI, PSI, and OMEGA for the residue sequence.

`[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 `i`th letter of the sequence alphabet is selected with weight `W(i)`. The weight vector is usually a probability vector or a frequency count vector. Note that the `i`th element of the nucleotide alphabet is given by `int2nt(i)`, and the `i`th element of the amino acid alphabet is given by `int2aa(i)`.

`randseq(..., 'FromStructure',` *`FromStructureValue`*`)` creates a weighted random sequence with weights given by the output structure from `basecount`, `dimercount`, `codoncount`, or `aacount`.

`randseq(..., 'Case',` *`CaseValue`*`)` specifies the case for a letter sequence.

`randseq(...,'DataType',` *`DataTypeValue`*`)` specifies the data type for the sequence array.

Examples

Generate a random DNA sequence.

```
randseq(20)

ans =
TAGCTGGCCAAGCGAGCTTG
```

Generate a random RNA sequence.

```
randseq(20,'alphabet','rna')

ans =
GCUGCGGCGGUUGUAUCCUG
```

Generate a random protein sequence.

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

ans =
DYKMCLYEFGMFGHFTGHKK
```

See Also 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`

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.
`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`.
`PartialDigest`	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 5–149. 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, geneticcode, nt2aa

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 `fastaread`, `genbankread`, `genpeptread`, `traceplot`

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

MATLAB functions `regexp`, `regexpi`

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.

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

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.
`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 to visualize the match of two sequences.

`seqdotplot(Seq1,Seq2, Window, Number)` plots sequence matches when there are at least *`Number`* matches in a window of size *`Window`*.

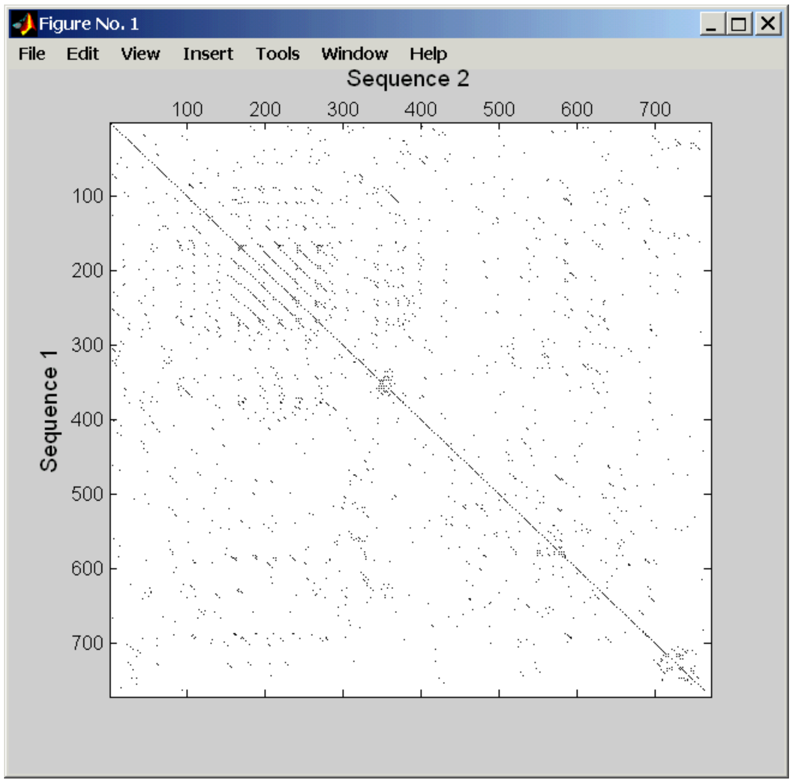
When plotting nucleotide sequences, start with a `Window` of `11` and `Number` of `7`.

`Matches = seqdotplot(...)` returns the number of dots in the dot plot matrix.

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)
```



See Also Bioinformatics Toolbox functions `hmmprofalign`, `nwalign`, `swalign`

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.

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 `seqcomplement`, `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.
`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`

seqshoworfs

Purpose

Graphically display the open reading frames in a sequence

Syntax

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

seqshoworfs(..., 'Frames', FramesValue)
seqshoworfs(..., 'GeneticCode', GeneticCodeValue)
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.
`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`.
`MinimumLength`	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
```

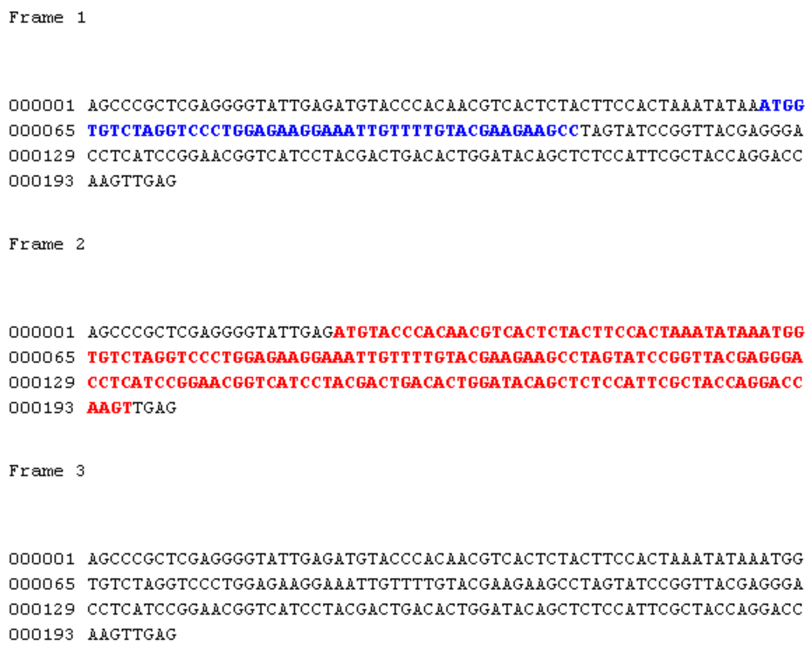
`seqshoworfs(..., 'Color', ColorValue)` selects the color used to highlight the open reading frames in the output display. The default color scheme is blue for the first reading frame, red for the second, and green for the third frame.

`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);
```



```
Frame 1


000001 AGCCCGCTCGAGGGGTATTGAGATGTACCCACAACGTCACTCTACTTCCACTAAATATAAATGG
000065 TGTCTAGGTCCCTGGAGAAGGAAATTGTTTTGTACGAAGAAGCCTAGTATCCGGTTACGAGGGA
000129 CCTCATCCGGAACGGTCATCCTACGACTGACACTGGATACAGCTCTCCATTCGCTACCAGGACC
000193 AAGTTGAG

Frame 2


000001 AGCCCGCTCGAGGGGTATTGAGATGTACCCACAACGTCACTCTACTTCCACTAAATATAAATGG
000065 TGTCTAGGTCCCTGGAGAAGGAAATTGTTTTGTACGAAGAAGCCTAGTATCCGGTTACGAGGGA
000129 CCTCATCCGGAACGGTCATCCTACGACTGACACTGGATACAGCTCTCCATTCGCTACCAGGACC
000193 AAGTTGAG

Frame 3


000001 AGCCCGCTCGAGGGGTATTGAGATGTACCCACAACGTCACTCTACTTCCACTAAATATAAATGG
000065 TGTCTAGGTCCCTGGAGAAGGAAATTGTTTTGTACGAAGAAGCCTAGTATCCGGTTACGAGGGA
000129 CCTCATCCGGAACGGTCATCCTACGACTGACACTGGATACAGCTCTCCATTCGCTACCAGGACC
000193 AAGTTGAG
```

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`, `seqshowwords`, `seqwordcount`

MATLAB function `regexp`

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.
Word	Enter a short character sequence.
ColorValue	Property to select the color for highlighted characters. Enter a 1-by-3 RGB vector specifying the intensity (0 255) of the red, green, and blue components, or enter a character from the following list: 'b'– blue, 'g'– green, 'r'– red, 'c'– cyan, 'm'– magenta, or 'y'– yellow. The default color is red 'r'.
ColumnsValue	Property to specify the number of characters in a line. Default value is 64.

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 `codoncount`, `seqwordcount`, `showhmmprof`

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.
`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`, `seqshowwords`

MATLAB functions `seq2regexp`, `strfind`

Purpose

Display a sequence alignment with color

Syntax

```
showalignment(Alignment, 'PropertyName', PropertyValue)

showalignment(..., 'MatchColor', MatchColorValue)
showalignment(..., 'SimilarColor' SimilarColorValue)
showalignment(..., 'Columns', ColumnsValue)
```

Arguments

`Alignment`	Enter the output from either the function `swalign` or `nwalign`.
`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(..., '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`, `hmmprofstruct`, `pfamhmmread`

Purpose

Read data from a SPOT file

Syntax

```
SPOTData = sptread('File',
                        'PropertyName', PropertyValue)

sptread(..., 'CleanColNames, 'CleanColNamesValues')
```

Arguments

File	SPOT formatted file (ASCII text file). Enter a filename, a path and filename, or URL pointing to a file. `File` can also be a MATLAB character array that contains the text for a SPOT file.
CleanColNamesValue	Property to control using valid MATLAB variable names.

Description

`SPOTData = sptread('File')` reads a SPOT formatted file and creates a MATLAB structure `SPOTData` containing the following fields:

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

`sptread(..., 'CleanColNames, CleanColNamesValue)` The column names in the SPOT file contain periods and some characters that cannot be used in MATLAB variable names. If you plan to use the column names as variable names in a function, use this option with

CleanColNames set to true and the function will return the field ColumnNames with valid variable names.

The Indices field of the structure includes the MATLAB indices that you can use for plotting heat maps of the data.

Examples

```
% Read in a sample SPOT file and plot the median foreground
% intensity for the 635 nm channel.
spotStruct = sptread('spotdata.txt')
maimage(spotStruct,'Rmedian');

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

rmedCol = find(strcmp(spotStruct.ColumnNames,'Rmedian'));
Rmedian = spotStruct.Data(:,rmedCol);
imagesc(Rmedian(spotStruct.Indices));
colormap bone
colorbar
```

See Also

Bioinformatics Toolbox functions 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.
`AlphabetValue`	Property to select an amino acid or nucleotide sequences. Enter either `'AA'` or `'NT'`. The default value is `'AA'`.
`ScaleValue`	Property to specify the scale factor for a scoring matrix.
`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`.
`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`

Index