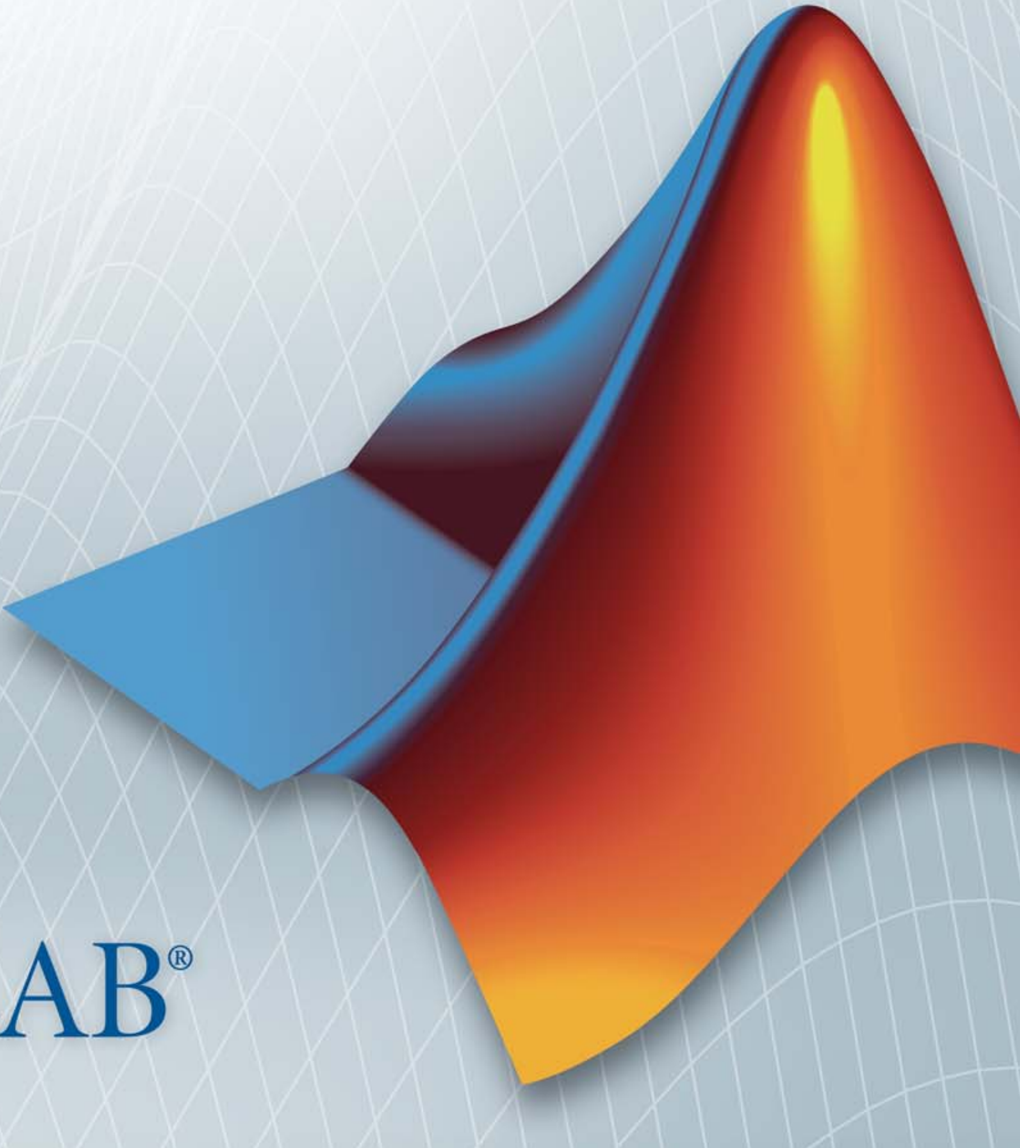
Bioinformatics Toolbox™

User's Guide

R2011b



MATLAB®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Bioinformatics Toolbox™ User's Guide

Trademarks

Patents

Revision History

September 2003	Online only	New for Version 1.0 (Release 13SP1+)
June 2004	Online only	Revised for Version 1.1 (Release 14)
November 2004	Online only	Revised for Version 2.0 (Release 14SP1+)
March 2005	Online only	Revised for Version 2.0.1 (Release 14SP2)
May 2005	Online only	Revised for Version 2.1 (Release 14SP2+)
September 2005	Online only	Revised for Version 2.1.1 (Release 14SP3)
November 2005	Online only	Revised for Version 2.2 (Release 14SP3+)
March 2006	Online only	Revised for Version 2.2.1 (Release 2006a)
May 2006	Online only	Revised for Version 2.3 (Release 2006a+)
September 2006	Online only	Revised for Version 2.4 (Release 2006b)
March 2007	Online only	Revised for Version 2.5 (Release 2007a)
April 2007	Online only	Revised for Version 2.6 (Release 2007a+)
September 2007	Online only	Revised for Version 3.0 (Release 2007b)
March 2008	Online only	Revised for Version 3.1 (Release 2008a)
October 2008	Online only	Revised for Version 3.2 (Release 2008b)
March 2009	Online only	Revised for Version 3.3 (Release 2009a)
September 2009	Online only	Revised for Version 3.4 (Release 2009b)
March 2010	Online only	Revised for Version 3.5 (Release 2010a)
September 2010	Online only	Revised for Version 3.6 (Release 2010b)
April 2011	Online only	Revised for Version 3.7 (Release 2011a)
September 2011	Online only	Revised for Version 4.0 (Release 2011b)

1 Getting Started

2 High-Throughput Sequence Analysis

3 Sequence Analysis

4 Microarray Analysis

5 Phylogenetic Analysis

A Examples

Index

Getting Started

Product Overview

In this section…
"Features" on page 1-2
"Expected Users" on page 1-4

Features

Bioinformatics Toolbox™ provides algorithms and visualization techniques for Next Generation Sequencing (NGS), microarray analysis, mass spectrometry, and gene ontology. Using toolbox functions, you can read genomic and proteomic data from standard file formats such as SAM, FASTA, CEL, and CDF, as well as from online databases such as the NCBI Gene Expression Omnibus and GenBank®. You can explore and visualize this data with sequence browsers, spatial heatmaps, and clustergrams. The toolbox also provides statistical techniques for detecting peaks, imputing values for missing data, and selecting features.

You can combine toolbox functions to support common bioinformatics workflows. You can use ChIP-Seq data to identify transcription factors; analyze RNA-Seq data to identify differentially expressed genes; identify copy number variants and SNPs in microarray data; and classify protein profiles using mass spectrometry data.

The Bioinformatics Toolbox product extends the MATLAB® environment to provide an integrated software environment for genome and proteome analysis. Scientists and engineers can answer questions, solve problems, prototype new algorithms, and build applications for drug discovery and design, genetic engineering, and biological research. An introduction to these features will help you to develop a conceptual model for working with the toolbox and your biological data.

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

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

Toolbox features and functions fall within these categories:

- **Data formats and databases** — Connect to Web-accessible databases containing genomic and proteomic data. Read and convert between multiple data formats.
- **High-throughput sequencing** — Gene expression and transcription factor analysis of next-generation sequencing data, including RNA-Seq and ChIP-Seq.
- **Sequence analysis** — Determine the statistical characteristics of a sequence, align two sequences, and multiply align several sequences. Model patterns in biological sequences using hidden Markov model (HMM) profiles.
- **Phylogenetic analysis** — Create and manipulate phylogenetic tree data.
- **Microarray data analysis** — Read, normalize, and visualize microarray data.
- **Mass spectrometry data analysis** — Analyze and enhance raw mass spectrometry data.
- **Statistical learning** — Classify and identify features in data sets with statistical learning tools.
- **Programming interface** — Use other bioinformatic software (BioPerl and BioJava) within the MATLAB environment.

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

- **Prototype and develop algorithms** — Prototype new ideas in an open and extensible environment. Develop algorithms using efficient string processing and statistical functions, view the source code for existing functions, and use the code as a template for customizing, improving, or creating your own functions. See “Prototyping and Development Environment” on page 1-19.

- **Visualize data** — Visualize sequences and alignments, gene expression data, phylogenetic trees, mass spectrometry data, protein structure, and relationships between data with interconnected graphs. See “Data Visualization” on page 1-19.
- **Share and deploy applications** — Use an interactive GUI builder to develop a custom graphical front end for your data analysis programs. Create standalone applications that run separately from the MATLAB environment. See “Algorithm Sharing and Application Deployment” on page 1-20.

Expected Users

The Bioinformatics Toolbox product is intended for computational biologists and research scientists who need to develop new algorithms or implement published ones, visualize results, and create standalone 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 industries.
- **Education/Professor/Student** — This toolbox is well suited for learning and teaching genome and proteome analysis techniques. Educators and students can concentrate on bioinformatic algorithms instead of programming basic functions such as reading and writing to files.

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

Installation

In this section...

Installing

Install the Bioinformatics Toolbox software from a DVD or Web release using the MathWorks® Installer. For more information, see the installation documentation.

Required Software

The Bioinformatics Toolbox software requires the following MathWorks products to be installed on your computer.

Required Software	Description
MATLAB	Provides a command-line interface and integrated software environment for the Bioinformatics Toolbox software. Version 3.7 of the Bioinformatics Toolbox software requires MATLAB Version 7.12 on the Release 2011a DVD.
Statistics Toolbox™	Provides basic statistics and probability functions used by the functions of the Bioinformatics Toolbox software. Version 3.7 of the Bioinformatics Toolbox software requires Statistics Toolbox Version 7.5 on the Release 2011a DVD.

Optional Software

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

Using the Bioinformatics Toolbox software with other MATLAB toolboxes and products will allow you to do advanced algorithm development and solve multidisciplinary problems.

Optional Software	Description
Parallel Computing Toolbox™	Perform parallel bioinformatic computations on multicore computers and computer clusters. For an example of batch processing through parallel computing, see the Batch Processing of Spectra Using Distributed Computing demo.
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.
SimBiology®	Model, simulate, and analyze biochemical systems.
Optimization Toolbox™	Use nonlinear optimization to predict the secondary structure of proteins and the structure of other biological macromolecules.
Neural Network Toolbox™	Use neural networks to solve problems where algorithms are not available. For example, you can train neural networks for pattern recognition using large sets of sequence data.
Database Toolbox™	Create your own in-house databases for sequence data with custom annotations.

Optional Software	Description
MATLAB® Compiler™	Create standalone applications from MATLAB GUI applications, and create dynamic link libraries from MATLAB functions to use with any programming environment.
MATLAB® Builder™ NE	Create COM objects to use with any COM-based programming environment.
MATLAB Builder JA	Integrate MATLAB applications into your organization's Java™ programs by creating a Java wrapper around the application.
MATLAB Builder EX	Create Microsoft® Excel® add-in functions from MATLAB functions to use with Excel® spreadsheets.
Spreadsheet Link™ EX	Connect Microsoft Excel with the MATLAB Workspace to exchange data and to use MATLAB computational and visualization functions. For more information, see "Importing and Exploring Bioinformatic Data from Microsoft® Excel" on page 1-21.

Features and Functions

In this section...

Data Formats and Databases

The toolbox accesses many of the databases on the Web and other online data sources. It allows you to copy data into the MATLAB Workspace, and read and write to files with standard bioinformatic formats. It also reads many common genome file formats, so that you do not have to write and maintain your own file readers.

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

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

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

Gene Ontology database — Load the database from the Web into a gene ontology object (`geneont.geneont`). Select sections of the ontology with methods for the geneont object (`geneont.getancestors`, `geneont.getdescendants`, `geneont.getmatrix`, `geneont.getrelatives`), and manipulate data with utility functions (`goannotread`, `num2goid`).

Read data from instruments — Read data generated from gene sequencing instruments (`scfread`, `joinseq`, `traceplot`), mass spectrometers (`jcampread`), and Agilent® microarray scanners (`agferead`).

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

- Sequence data: GenBank (`genbankread`), GenPept (`genpeptread`), EMBL (`emblread`), PDB (`pdbread`), and FASTA (`fastaread`)
- Multiply aligned sequences: ClustalW and GCG formats (`multialignread`)
- Gene expression data from microarrays: Gene Expression Omnibus (GEO) data (`geosoftread`), GenePix® data in GPR and GAL files (`gprread`, `galread`), SPOT data (`sptread`), Affymetrix® GeneChip® data (`affyread`), and ImaGene® results files (`imageneread`)
- Hidden Markov model profiles: PFAM-HMM file (`pfamhmmread`)

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

BLAST searches — Request Web-based BLAST searches (`blastncbi`), get the results from a search (`getblast`) and read results from a previously saved BLAST formatted report file (`blastread`).

The MATLAB environment 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 compare nucleotide or amino acid sequences using pairwise or multiple sequence alignment functions.

Pairwise sequence alignment — Efficient 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`). Visualize sequence similarities with `seqdotplot` and sequence alignment results with `showalignment`.

Multiple sequence alignment — Functions for multiple sequence alignment (`multialign`, `profalign`) and functions that support multiple sequences (`multialignread`, `fastaread`, `showalignment`). There is also a graphical interface (`multialignviewer`) for viewing the results of a multiple sequence alignment and manually making adjustment.

Multiple sequence profiles — Implementations for multiple alignment and profile hidden Markov model algorithms (`gethmmprof`, `gethmmalignment`, `gethmmtree`, `pfamhmmread`, `hmmprofalign`, `hmmprofestimate`, `hmmprofgenerate`, `hmmprofmerge`, `hmmprofstruct`, `showhmmprof`).

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

Sequence Utilities and Statistics

You can manipulate and analyze your sequences to gain a deeper understanding of the physical, chemical, and biological characteristics of your data. Use a graphical user interface (GUI) with many of the sequence functions in the toolbox (`seqtool`).

Sequence conversion and 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 and protein sequences (`aa2int`, `aa2nt`, `dna2rna`, `rna2dna`, `int2aa`, `int2nt`, `nt2aa`, `nt2int`, `seqcomplement`, `seqrcomplement`, `seqreverse`).

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

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

Sequence utilities — Determine a consensus sequence from a set of multiply aligned amino acid, nucleotide sequences (`seqconsensus`, or a sequence profile (`seqprofile`). Format a sequence for display (`seqdisp`) or graphically show a sequence alignment with frequency data (`seqlogo`).

Additional MATLAB functions efficiently handle string operations with regular expressions (`regexp`, `seq2regexp`) to look for specific patterns in a sequence and search through a library for string matches (`seqmatch`).

Look for possible cleavage sites in a DNA/RNA sequence by searching for palindromes (`palindromes`).

Protein Property Analysis

You can use a collection of protein analysis methods to extract information from your data. You can determine protein characteristics and simulate enzyme cleavage reactions. The toolbox provides functions to calculate various properties of a protein sequence, such as the atomic composition (`atomiccomp`), molecular weight (`molweight`), and isoelectric point (`isoelectric`). You can cleave a protein with an enzyme (`cleave`, `rebasecuts`) and create distance and Ramachandran plots for PDB data (`pdbdistplot`, `ramachandran`). The toolbox contains a graphical user interface for protein analysis (`proteinplot`) and plotting 3-D protein and other molecular structures with information from molecule model files, such as PDB files (`molviewer`).

Amino acid sequence utilities — Calculate amino acid statistics for a sequence (`aacount`) and get information about character codes (`aminolookup`).

Phylogenetic Analysis

You can use functions for phylogenetic tree building and analysis. There is also a GUI to draw phylograms (trees).

Phylogenetic tree data — Read and write Newick-formatted tree files (`phytreeread`, `phytreewrite`) into the MATLAB Workspace as phylogenetic tree objects (`phytree`).

Create a phylogenetic tree — Calculate the pairwise distance between biological sequences (`seqpdist`), estimate the substitution rates (`dnds`, `dndsml`), build a phylogenetic tree from pairwise distances (`seqlinkage`, `seqneighjoin`, `reroot`), and view the tree in an interactive GUI that allows you to view, edit, and explore the data (`phytreetool` or `view`). This GUI also allows you to prune branches, reorder, rename, and explore distances.

Phylogenetic tree object methods — You can access the functionality of the `phytreetool` GUI using methods for a phylogenetic tree object (`phytree`). Get property values (`get`) and node names (`getbyname`). Calculate the patristic distances between pairs of leaf nodes (`pdist`, `weights`) and draw a phylogenetic tree object in a MATLAB Figure window as a phylogram, cladogram, or radial treeplot (`plot`). Manipulate tree data by selecting branches and leaves using a specified criterion (`select`, `subtree`) and removing nodes (`prune`). Compare trees (`getcanonical`) and use Newick-formatted strings (`getnewickstr`).

Microarray Data Analysis

The MATLAB environment is widely used for microarray data analysis, including reading, filtering, normalizing, and visualizing microarray data. However, the standard normalization and visualization tools that scientists use can be difficult to implement. The toolbox includes these standard functions:

Microarray data — Read Affymetrix GeneChip files (`affyread`) and plot data (`probesetplot`), ImaGene results files (`imageneread`), SPOT files (`sptread`) and Agilent microarray scanner files (`agferead`). Read GenePix

GPR files (gprread) and GAL files (galread). Get Gene Expression Omnibus (GEO) data from the Web (getgeodata) and read GEO data from files (geosoftread).

A utility function (magetfield) extracts data from one of the microarray reader functions (gprread, agferead, sptread, imageneread).

Microarray normalization and filtering — The toolbox provides a number of methods for normalizing microarray data, such as lowess normalization (malowess) and mean normalization (manorm), or across multiple arrays (quantilenorm). 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 spatial plots of microarray data (maimage, redgreencmap), box plots (maboxplot), loglog plots (maloglog), and intensity-ratio plots (mairplot). You can also view clustered expression profiles (clustergram, redgreencmap). You can create 2-D scatter plots of principal components from the microarray data (mapcaplot).

Microarray utility functions — Use the following functions to work with Affymetrix GeneChip data sets. Get library information for a probe (probelibraryinfo), gene information from a probe set (probesetlookup), and probe set values from CEL and CDF information (probesetvalues). Show probe set information from NetAffx™ Analysis Center (probesetlink) and plot probe set values (probesetplot).

The toolbox accesses statistical routines to perform cluster analysis and to visualize the results, and you can view your data through statistical visualizations such as dendrograms, classification, and regression trees.

Microarray Data Storage

The toolbox includes functions, objects, and methods for creating, storing, and accessing microarray data.

The object constructor function, DataMatrix, lets you create a DataMatrix object to encapsulate data and metadata from a microarray experiment. A

DataMatrix object stores experimental data in a matrix, with rows typically corresponding to gene names or probe identifiers, and columns typically corresponding to sample identifiers. A DataMatrix object also stores metadata, including the gene names or probe identifiers (as the row names) and sample identifiers (as the column names).

You can reference microarray expression values in a DataMatrix object the same way you reference data in a MATLAB array, that is, by using linear or logical indexing. Alternately, you can reference this experimental data by gene (probe) identifiers and sample identifiers. Indexing by these identifiers lets you quickly and conveniently access subsets of the data without having to maintain additional index arrays.

Many MATLAB operators and arithmetic functions are available to DataMatrix objects by means of methods. These methods let you modify, combine, compare, analyze, plot, and access information from DataMatrix objects. Additionally, you can easily extend the functionality by using general element-wise functions, `dmarrayfun` and `dmbsxfun`, and by manually accessing the properties of a DataMatrix object.

Note For more information on creating and using DataMatrix objects, see "Representing Expression Data Values in DataMatrix Objects" on page 4-5.

Mass Spectrometry Data Analysis

The mass spectrometry functions preprocess and classify raw data from SELDI-TOF and MALDI-TOF spectrometers and use statistical learning functions to identify patterns.

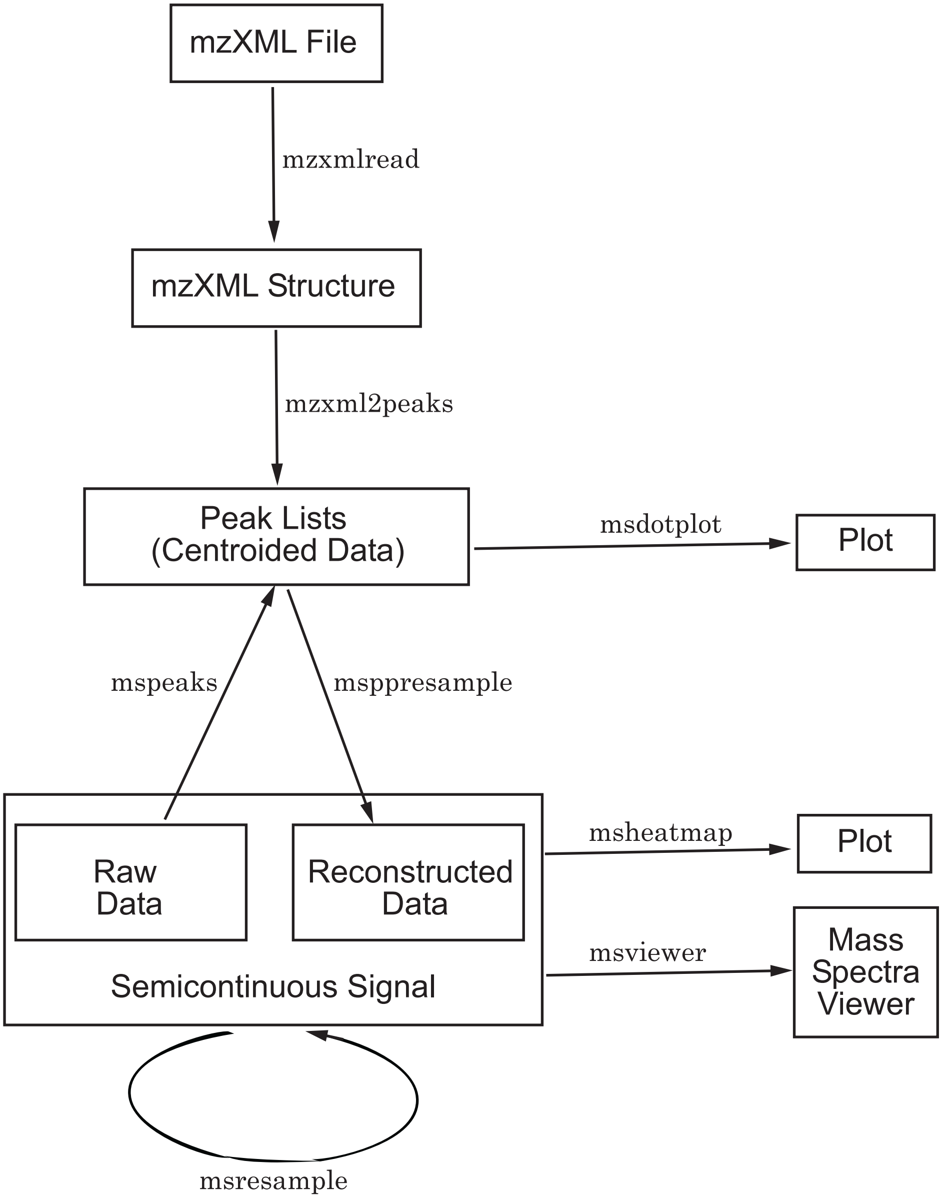
Reading raw data — Load raw mass/charge and ion intensity data from comma-separated-value (CSV) files, or read a JCAMP-DX-formatted file with mass spectrometry data (`jcampread`) into the MATLAB environment.

You can also have data in TXT files and use the `importdata` function.

Preprocessing raw data — Resample high-resolution data to a lower resolution (`msresample`) where the extra data points are not needed. Correct the baseline (`msbackadj`). Align a spectrum to a set of reference masses (msalign) and visually verify the alignment (msheatmap). Normalize the area between spectra for comparing (msnorm), and filter out noise (mslowess and mssgolay).

Spectrum analysis — Load spectra into a GUI (msviewer) for selecting mass peaks and further analysis.

The following graphic illustrates the roles of the various mass spectrometry functions in the toolbox.


mzXML File
mzxmlread
mzXML Structure
mzxml2peaks
Peak Lists
(Centroided Data)
msdotplot
Plot
mspeaks
msppresample
Raw
Data
Reconstructed
Data
Semicontinuous Signal
msheatmap
Plot
msviewer
Mass
Spectra
Viewer
msresample

Graph Theory Functions

Graph theory functions in the toolbox apply basic graph theory algorithms to sparse matrices. A sparse matrix represents a graph, any nonzero entries in the matrix represent the edges of the graph, and the values of these entries represent the associated weight (cost, distance, length, or capacity) of the edge. Graph algorithms that use the weight information will cancel the edge if a `NaN` or an `Inf` is found. Graph algorithms that do not use the weight information will consider the edge if a `NaN` or an `Inf` is found, because these algorithms look only at the connectivity described by the sparse matrix and not at the values stored in the sparse matrix.

Sparse matrices can represent four types of graphs:

- **Directed Graph** — Sparse matrix, either double real or logical. Row (column) index indicates the source (target) of the edge. Self-loops (values in the diagonal) are allowed, although most of the algorithms ignore these values.
- **Undirected Graph** — Lower triangle of a sparse matrix, either double real or logical. An algorithm expecting an undirected graph ignores values stored in the upper triangle of the sparse matrix and values in the diagonal.
- **Direct Acyclic Graph (DAG)** — Sparse matrix, double real or logical, with zero values in the diagonal. While a zero-valued diagonal is a requirement of a DAG, it does not guarantee a DAG. An algorithm expecting a DAG will *not* test for cycles because this will add unwanted complexity.
- **Spanning Tree** — Undirected graph with no cycles and with one connected component.

There are no attributes attached to the graphs; sparse matrices representing all four types of graphs can be passed to any graph algorithm. All functions will return an error on nonsquare sparse matrices.

Graph algorithms do not pretest for graph properties because such tests can introduce a time penalty. For example, there is an efficient shortest path algorithm for DAG, however testing if a graph is acyclic is expensive compared to the algorithm. Therefore, it is important to select a graph theory function and properties appropriate for the type of the graph represented by your input matrix. If the algorithm receives a graph type that differs from what it expects, it will either:

- Return an error when it reaches an inconsistency. For example, if you pass a cyclic graph to the `graphshortestpath` function and specify `Acyclic` as the `method` property.
- Produce an invalid result. For example, if you pass a directed graph to a function with an algorithm that expects an undirected graph, it will ignore values in the upper triangle of the sparse matrix.

The graph theory functions include `graphallshortestpaths`, `graphconncomp`, `graphisdag`, `graphisomorphism`, `graphisspantree`, `graphmaxflow`, `graphminspantree`, `graphpred2path`, `graphshortestpath`, `graphtopoorder`, and `graphtraverse`.

Graph Visualization

The toolbox includes functions, objects, and methods for creating, viewing, and manipulating graphs such as interactive maps, hierarchy plots, and pathways. This allows you to view relationships between data.

The object constructor function (`biograph`) lets you create a biograph object to hold graph data. Methods of the biograph object let you calculate the position of nodes (`dolayout`), draw the graph (`view`), get handles to the nodes and edges (`getnodesbyid` and `getedgesbynodeid`) to further query information, and find relations between the nodes (`getancestors`, `getdescendants`, and `getrelatives`). There are also methods that apply basic graph theory algorithms to the biograph object.

Various properties of a biograph object let you programmatically change the properties of the rendered graph. You can customize the node representation, for example, drawing pie charts inside every node (`CustomNodeDrawFcn`). Or you can associate your own callback functions to nodes and edges of the graph, for example, opening a Web page with more information about the nodes (`NodeCallback` and `EdgeCallback`).

Statistical Learning and Visualization

You can classify and identify features in data sets, set up cross-validation experiments, and compare different classification methods.

The toolbox provides functions that build on the classification and statistical learning tools in the Statistics Toolbox software (`classify`, `kmeans`, and `treefit`).

These functions include imputation tools (`knnimpute`), and K-nearest neighbor classifiers (`knnclassify`).

Other functions include set up of cross-validation experiments (`crossvalind`) and comparison of the performance of different classification methods (`classperf`). In addition, there are tools for selecting diversity and discriminating features (`rankfeatures`, `randfeatures`).

Prototyping and Development Environment

The MATLAB environment lets you prototype and develop algorithms and easily compare alternatives.

- **Integrated environment** — Explore biological data in an environment that integrates programming and visualization. Create reports and plots with the built-in functions for mathematics, graphics, and statistics.
- **Open environment** — Access the source code for the 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 matrices for sequences or groups of sequences allows you to work efficiently and not worry about writing loops or other programming controls.
- **Programming tools** — Use a visual debugger for algorithm development and refinement and an algorithm performance profiler to accelerate development.

Data Visualization

You can visually compare pairwise sequence alignments, multiply aligned sequences, gene expression data from microarrays, and plot nucleic acid and protein characteristics. The 2-D and volume visualization features let you create custom graphical representations of multidimensional data sets. You can also create montages and overlays, and export finished graphics to an Adobe® PostScript® image file or copy directly into Microsoft® PowerPoint®.

Algorithm Sharing and Application Deployment

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

- **Share algorithms with other users** — You can share data analysis algorithms created in the MATLAB language across all supported platforms by giving files to other users. You can also create GUIs within the MATLAB environment using the Graphical User Interface Development Environment (GUIDE).
- **Deploy MATLAB GUIs** — Create a GUI within the MATLAB environment using GUIDE, and then use MATLAB Compiler software to create a standalone GUI application that runs separately from the MATLAB environment.
- **Create dynamic link libraries (DLLs)** — Use MATLAB Compiler software to create DLLs for your functions, and then link these libraries to other programming environments such as C and C++.
- **Create COM objects** — Use MATLAB Builder NE software to create COM objects, and then use a COM-compatible programming environment (Visual Basic®) to create a standalone application.
- **Create Excel add-ins** — Use MATLAB Builder EX software to create Excel add-in functions, and then use these functions with Excel spreadsheets.
- **Create Java classes** — Use MATLAB Builder JA software to automatically generate Java classes from algorithms written in the MATLAB programming language. You can run these classes outside the MATLAB environment.

Importing and Exploring Bioinformatic Data from Microsoft Excel

In this section…

Using Excel and MATLAB Together

If you have bioinformatic data in an Excel (2007 or 2010) spreadsheet, use Spreadsheet Link EX to:

- Connect Excel with the MATLAB Workspace to exchange data
- Use MATLAB and Bioinformatics Toolbox computational and visualization functions

About the Example

Note The following example assumes you have Spreadsheet Link EX software installed on your system.

The Excel file used in the following example contains data from DeRisi, J.L., Iyer, V.R., and Brown, P.O. (Oct. 24, 1997). Exploring the metabolic and genetic control of gene expression on a genomic scale. Science *278(5338)*, 680–686. PMID: 9381177. The data was filtered using the steps described in the Gene Expression Profile Analysis demo.

Before Running the Example

1. If not already done, modify your system path to include the MATLAB root folder as described in “Modifying Your System Path” in the Spreadsheet Link EX documentation.

2. If not already done, enable the Spreadsheet Link EX Add-In as described in “Configuring the Spreadsheet Link EX Software” in the Spreadsheet Link EX documentation.

3. Close MATLAB and Excel if they are open.

4. Start Excel 2007 or 2010 software. MATLAB and Spreadsheet Link EX software automatically start.

5. From Excel, open the following file provided with the Bioinformatics Toolbox software:

   ```
   matlabroot\toolbox\bioinfo\biodemos\Filtered_Yeastdata.xlsm
   ```

   Note *`matlabroot`* is the MATLAB root folder, which is where MATLAB software is installed on your system.

6. In the Excel software, enable macros. Click the **Developer** tab, and then select **Macro Security** from the Code group. (If the **Developer** tab is not displayed on the Excel ribbon, consult Excel Help to display it.)

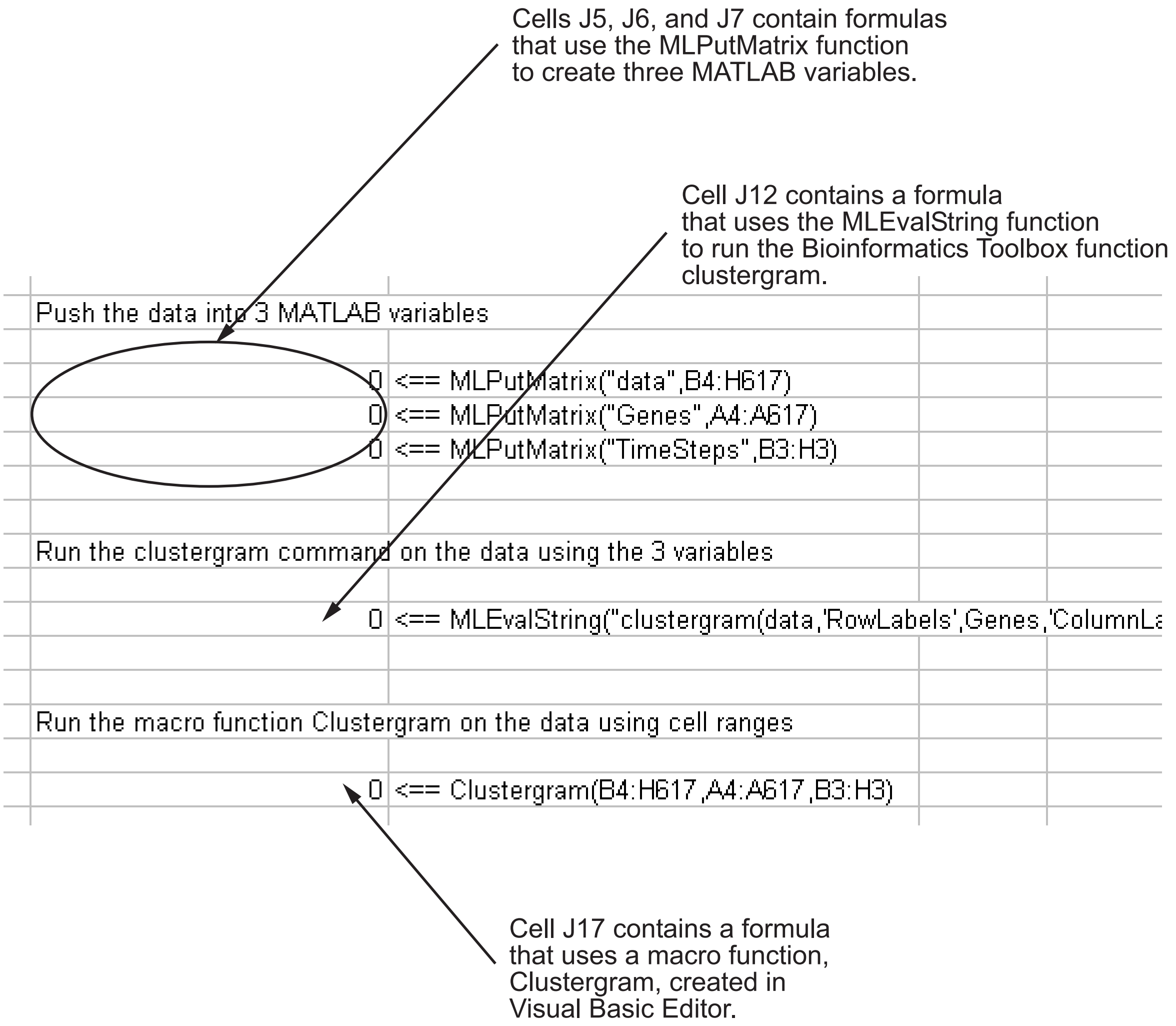
Running the Example for the Entire Data Set

1. In the provided Excel file, note that columns A through H contain data from DeRisi et al. Also note that cells J5, J6, J7, and J12 contain formulas using Spreadsheet Link EX functions `MLPutMatrix` and `MLEvalString`.

   Tip To view a cell’s formula, select the cell, and then view the formula in the formula bar *fx* at the top of the Excel window.

2 Execute the formulas in cells J5, J6, J7, and J12, by selecting the cell, pressing **F2**, and then pressing **Enter**.

Each of the first three cells contains a formula using the Spreadsheet Link EX function `MLPutMatrix`, which creates a MATLAB variable from the data in the spreadsheet. Cell J12 contains a formula using the Spreadsheet Link EX function `MLEvalString`, which runs the Bioinformatics Toolbox `clustergram` function using the three variables as input. For more information on adding formulas using Spreadsheet Link EX functions, see "Entering Functions into Worksheet Cells" in the Spreadsheet Link EX documentation.



Cells J5, J6, and J7 contain formulas that use the MLPutMatrix function to create three MATLAB variables.

Cell J12 contains a formula that uses the MLEvalString function to run the Bioinformatics Toolbox function clustergram.

```
Push the data into 3 MATLAB variables

                0 <== MLPutMatrix("data",B4:H617)
                0 <== MLPutMatrix("Genes",A4:A617)
                0 <== MLPutMatrix("TimeSteps",B3:H3)


Run the clustergram command on the data using the 3 variables

                0 <== MLEvalString("clustergram(data,'RowLabels',Genes,'ColumnLa


Run the macro function Clustergram on the data using cell ranges

                0 <== Clustergram(B4:H617,A4:A617,B3:H3)
```

Cell J17 contains a formula that uses a macro function, Clustergram, created in Visual Basic Editor.

3 Note that cell J17 contains a formula using a macro function `Clustergram`, which was created in the Visual Basic Editor. Running this macro does the same as the formulas in cells J5, J6, J7, and J12. Optionally, view the `Clustergram` macro function by clicking the **Developer** tab, and then clicking the Visual Basic button. (If the **Developer** tab is not on the Excel ribbon, consult Excel Help to display it.)

For more information on creating macros using Visual Basic Editor, see "Examples: Using Spreadsheet Link EX Functions in Macros" in the Spreadsheet Link EX documentation.

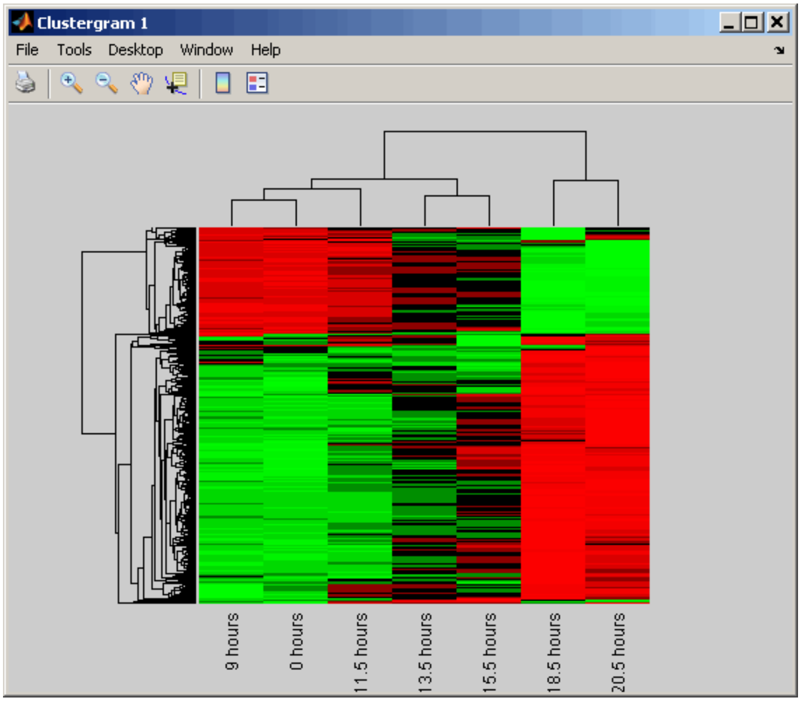
4 Execute the formula in cell J17 to analyze and visualize the data:

a Select cell **J17**.

b Press **F2**.

c Press **Enter**.

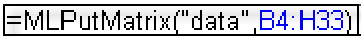
The macro function `Clustergram` runs creating three MATLAB variables (`data`, `Genes`, and `TimeSteps`) and displaying a Clustergram window containing dendrograms and a heat map of the data.



Editing Formulas to Run the Example on a Subset of the Data

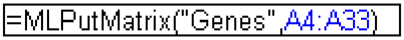
1 Edit the formulas in cells J5 and J6 to analyze a subset of the data. Do this by editing the formulas' cell ranges to include data for only the first 30 genes:

   a Select cell **J5**, and then press **F2** to display the formula for editing. Change **H617** to **H33**, and then press **Enter**.

   ```
   =MLPutMatrix("data",B4:H33)
   ```

   b Select cell **J6**, then press **F2** to display the formula for editing. Change **A617** to **A33**, and then press **Enter**.

   ```
   =MLPutMatrix("Genes",A4:A33)
   ```

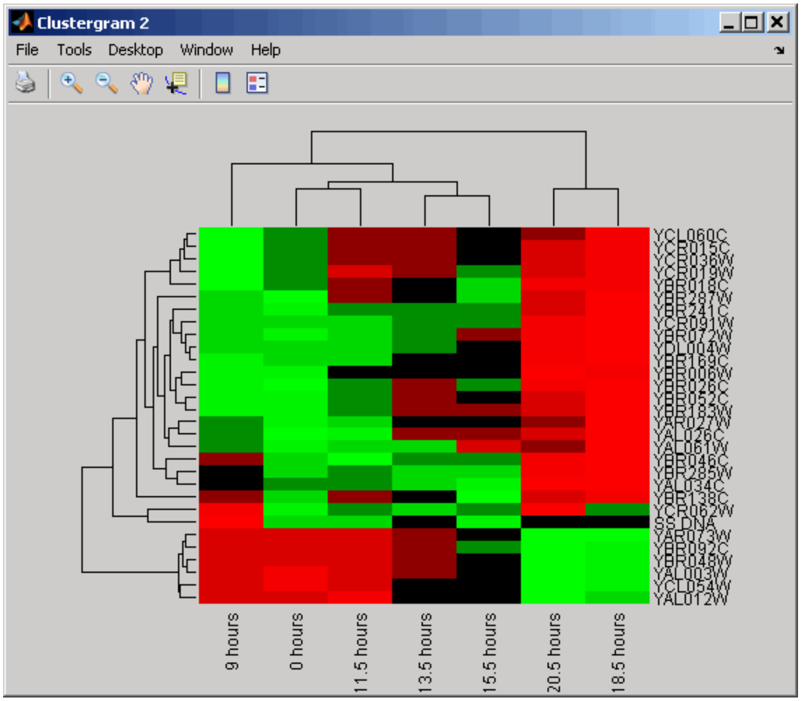
2 Run the formulas in cells J5, J6, J7, and J12 to analyze and visualize a subset of the data:

   a Select cell **J5**, press **F2**, and then press **Enter**.

   b Select cell **J6**, press **F2**, and then press **Enter**.

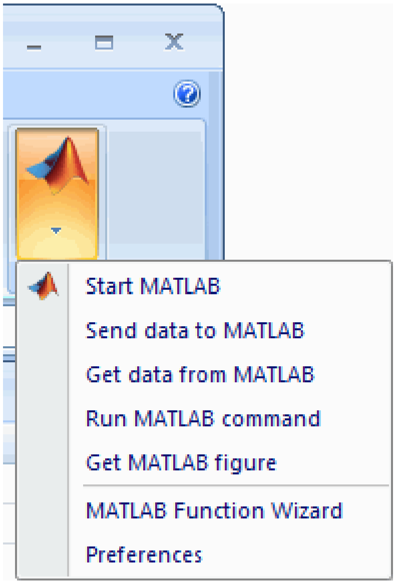
   c Select cell **J7**, press **F2**, and then press **Enter**.

   d Select cell **J12**, press **F2**, and then press **Enter**.



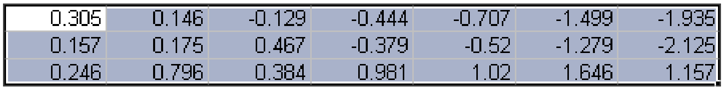
Using the Spreadsheet Link EX Interface to Interact With the Data in MATLAB

Use the MATLAB group on the right side of the **Home** tab to interact with the data:



For example, create a variable in MATLAB containing a 3-by-7 matrix of the data, plot the data in a Figure window, and then add the plot to your spreadsheet:

1 Click-drag to select cells **B5** through **H7**.



0.305	0.146	-0.129	-0.444	-0.707	-1.499	-1.935
0.157	0.175	0.467	-0.379	-0.52	-1.279	-2.125
0.246	0.796	0.384	0.981	1.02	1.646	1.157

2 From the MATLAB group, select **Send data to MATLAB**.

3 Type **YAGenes** for the variable name, and then click **OK**.

The variable **YAGenes** is added to the MATLAB Workspace as a 3-by-7 matrix.

4 From the MATLAB group, select **Run MATLAB command**.

5 Type **plot(YAGenes')** for the command, and then click **OK**.

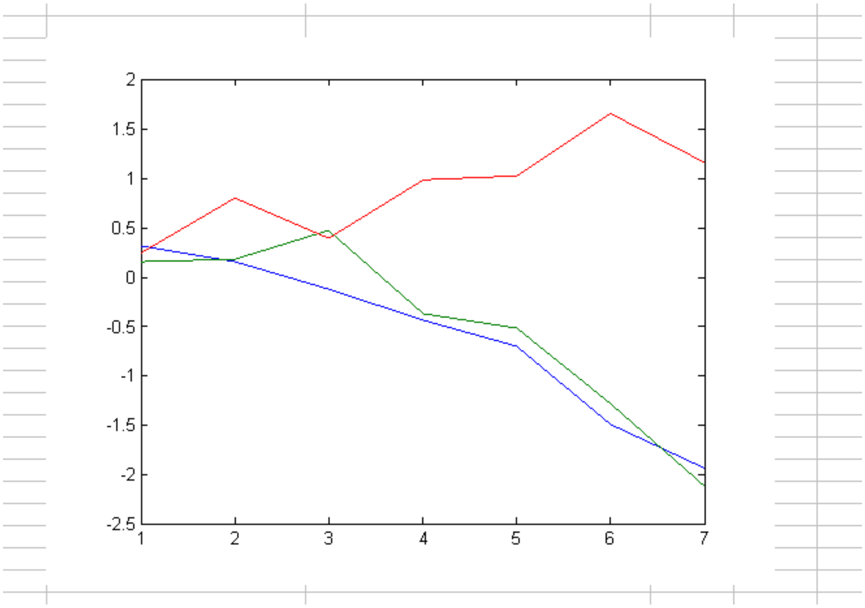
A Figure window displays a plot of the data.

Note Make sure you use the `'` (transpose) symbol when plotting the data in this step. You need to transpose the data in `YAGenes` so that it plots as three genes over seven time intervals.

---

6 Select cell **J20**, and then click from the MATLAB group, select **Get MATLAB figure**.

The figure is added to the spreadsheet.



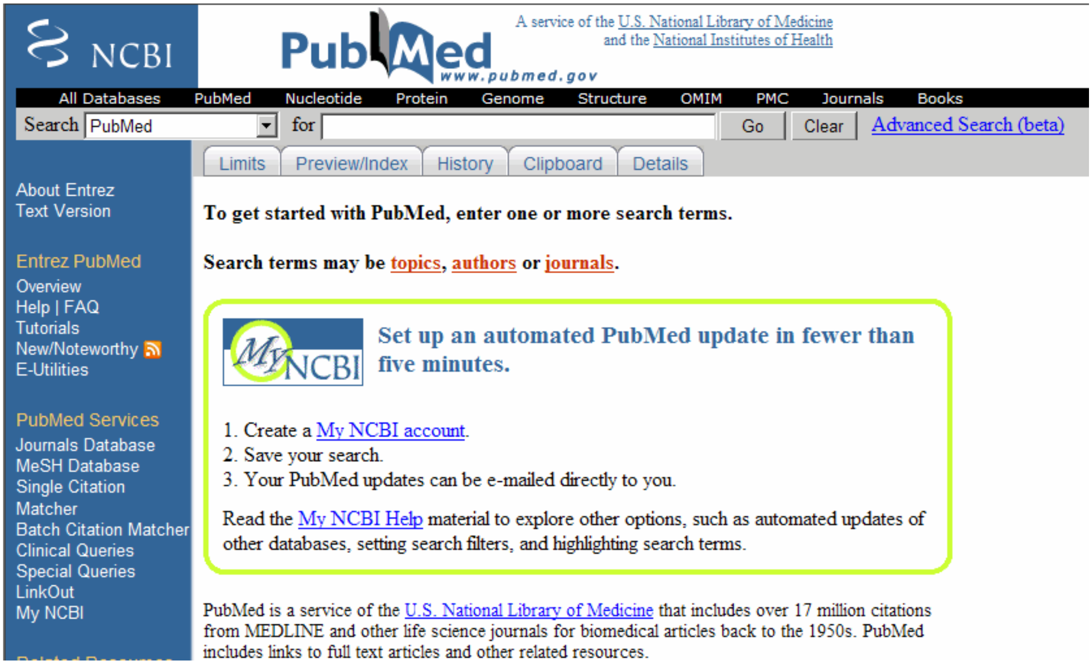
Creating get Functions

In this section...

What Are get Functions?

Bioinformatics Toolbox includes several get functions that retrieve information from various Web databases. Additionally, with some basic MATLAB programming skills, you can create your own get function to retrieve information from a specific Web database.

The following procedure illustrates how to create a function to retrieve information from the NCBI PubMed database and read the information into a MATLAB structure. The NCBI PubMed database contains biomedical literature citations and abstracts.



Search PubMed for

To get started with PubMed, enter one or more search terms.

Search terms may be topics, authors or journals.

Set up an automated PubMed update in fewer than five minutes.

1. Create a My NCBI account.
2. Save your search.
3. Your PubMed updates can be e-mailed directly to you.

Read the My NCBI Help material to explore other options, such as automated updates of other databases, setting search filters, and highlighting search terms.

PubMed is a service of the U.S. National Library of Medicine that includes over 17 million citations from MEDLINE and other life science journals for biomedical articles back to the 1950s. PubMed includes links to full text articles and other related resources.

Creating the getpubmed Function

The following procedure shows you how to create a function named getpubmed using the MATLAB Editor. This function will retrieve citation and abstract information from PubMed literature searches and write the data to a MATLAB structure.

Specifically, this function will take one or more search terms, submit them to the PubMed database for a search, then return a MATLAB structure or structure array, with each structure containing information for an article found by the search. The returned information will include a PubMed identifier, publication date, title, abstract, authors, and citation.

The function will also include property name/property value pairs that let the user of the function limit the search by publication date and limit the number of records returned.

1 From MATLAB, open the MATLAB Editor by selecting **File > New > Function**.

2 Define the getpubmed function, its input arguments, and return values by typing:

```
function pmstruct = getpubmed(searchterm,varargin)
% GETPUBMED Search PubMed database & write results to MATLAB structure
```

3 Add code to do some basic error checking for the required input SEARCHTERM.

```
% Error checking for required input SEARCHTERM
if(nargin<1)
    error('GETPUBMED:NotEnoughInputArguments',...
          'SEARCHTERM is missing.');
end
```

4 Create variables for the two property name/property value pairs, and set their default values.

```
% Set default settings for property name/value pairs,
% 'NUMBEROFRECORDS' and 'DATEOFPUBLICATION'
maxnum = 50; % NUMBEROFRECORDS default is 50
pubdate = ''; % DATEOFPUBLICATION default is an empty string
```

5 Add code to parse the two property name/property value pairs if provided as input.

```
% Parsing the property name/value pairs
num_argin = numel(varargin);
for n = 1:2:num_argin
    arg = varargin{n};
    switch lower(arg)

        % If NUMBEROFRECORDS is passed, set MAXNUM
        case 'numberofrecords'
            maxnum = varargin{n+1};

        % If DATEOFPUBLICATION is passed, set PUBDATE
        case 'dateofpublication'
            pubdate = varargin{n+1};

    end
end
```

6 You access the PubMed database through a search URL, which submits a search term and options, and then returns the search results in a specified format. This search URL is comprised of a base URL and defined parameters. Create a variable containing the base URL of the PubMed database on the NCBI Web site.

```
% Create base URL for PubMed db site
baseSearchURL = 'http://www.ncbi.nlm.nih.gov/sites/entrez?cmd=search';
```

7 Create variables to contain five defined parameters that the `getpubmed` function will use, namely, db (database), term (search term), report (report type, such as MEDLINE®), format (format type, such as text), and dispmax (maximum number of records to display).

```
% Set db parameter to pubmed
dbOpt = '&db=pubmed';

% Set term parameter to SEARCHTERM and PUBDATE
% (Default PUBDATE is '')
termOpt = ['&term=',searchterm,'+AND+',pubdate];
```

```
% Set report parameter to medline
reportOpt = '&report=medline';

% Set format parameter to text
formatOpt = '&format=text';

% Set dispmax to MAXNUM
% (Default MAXNUM is 50)
maxOpt = ['&dispmax=',num2str(maxnum)];
```

8 Create a variable containing the search URL from the variables created in the previous steps.

```
% Create search URL
searchURL = [baseSearchURL,dbOpt,termOpt,reportOpt,formatOpt,maxOpt];
```

9 Use the `urlread` function to submit the search URL, retrieve the search results, and return the results (as text in the MEDLINE report type) in `medlineText`, a character array.

```
medlineText = urlread(searchURL);
```

10 Use the MATLAB `regexp` function and regular expressions to parse and extract the information in `medlineText` into `hits`, a cell array, where each cell contains the MEDLINE-formatted text for one article. The first input is the character array to search, the second input is a search expression, which tells the `regexp` function to find all records that start with `PMID-`, while the third input, `'match'`, tells the `regexp` function to return the actual records, rather than the positions of the records.

```
hits = regexp(medlineText,'PMID-.*?(?=PMID|</pre>$)','match');
```

11 Instantiate the `pmstruct` structure returned by `getpubmed` to contain six fields.

```
pmstruct = struct('PubMedID','','PublicationDate','','Title','',...
             'Abstract','','Authors','','Citation','');
```

12 Use the MATLAB `regexp` function and regular expressions to loop through each article in `hits` and extract the PubMed ID, publication date, title, abstract, authors, and citation. Place this information in the `pmstruct` structure array.

```
for n = 1:numel(hits)
    pmstruct(n).PubMedID = regexp(hits{n},'(?<=PMID- ).*?(?=\n)','match', 'once');
    pmstruct(n).PublicationDate = regexp(hits{n},'(?<=DP  - ).*?(?=\n)','match', 'once');
    pmstruct(n).Title = regexp(hits{n},'(?<=TI  - ).*?(?=PG  -|AB  -)','match', 'once');
    pmstruct(n).Abstract = regexp(hits{n},'(?<=AB  - ).*?(?=AD  -)','match', 'once');
    pmstruct(n).Authors = regexp(hits{n},'(?<=AU  - ).*?(?=\n)','match');
    pmstruct(n).Citation = regexp(hits{n},'(?<=SO  - ).*?(?=\n)','match', 'once');
end
```

13 Select **File > Save As**.

When you are done, your file should look similar to the `getpubmed.m` file included with the Bioinformatics Toolbox software. The sample `getpubmed.m` file, including help, is located at:

```
matlabroot\toolbox\bioinfo\biodemos\getpubmed.m
```

Note The notation *`matlabroot`* is the MATLAB root directory, which is the directory where the MATLAB software is installed on your system.

Support Vector Machines (SVM)

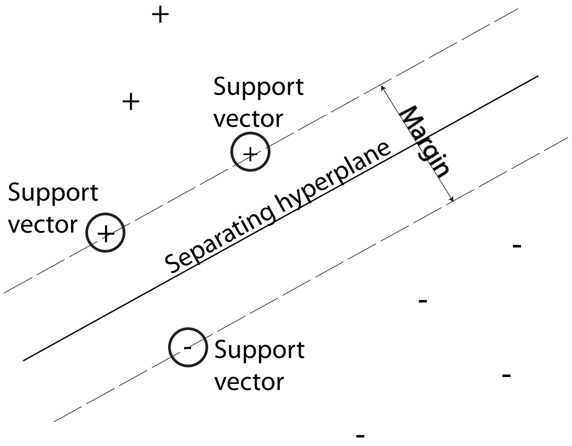
In this section...

Understanding Support Vector Machines

Separable Data

You can use a support vector machine (SVM) when your data has exactly two classes. An SVM classifies data by finding the best hyperplane that separates all data points of one class from those of the other class. The *best* hyperplane for an SVM means the one with the largest *margin* between the two classes. Margin means the maximal width of the slab parallel to the hyperplane that has no interior data points.

The *support vectors* are the data points that are closest to the separating hyperplane; these points are on the boundary of the slab. The following figure illustrates these definitions, with + indicating data points of type 1, and – indicating data points of type –1.



Mathematical Formulation: Primal. This discussion follows Hastie, Tibshirani, and Friedman [3] and Christianini and Shawe-Taylor [2].

The data for training is a set of points (vectors) x_i along with their categories y_i. For some dimension d, the $x_i \epsilon R^d$, and the $y_i = \pm 1$. The equation of a hyperplane is

$$<w,x> + b = 0,$$

where $w \epsilon R^d$, $<w,x>$ is the inner (dot) product of w and x, and b is real.

The following problem defines the *best* separating hyperplane. Find w and b that minimize $||w||$ such that for all data points (x_i,y_i),

$$y_i(<w,x_i> + b) \geq 1.$$

The support vectors are the x_i on the boundary, those for which $y_i(<w,x_i> + b) = 1$.

For mathematical convenience, the problem is usually given as the equivalent problem of minimizing $<w,w>/2$. This is a quadratic programming problem. The optimal solution w, b enables classification of a vector z as follows:

$$\text{class}(z) = \text{sign}(<w,z> + b).$$

Mathematical Formulation: Dual. It is computationally simpler to solve the dual quadratic programming problem. To obtain the dual, take positive Lagrange multipliers α_i multiplied by each constraint, and subtract from the objective function:

$$L_P = \frac{1}{2}\langle w,w\rangle - \sum_i \alpha_i \left(y_i \left(\langle w,x_i \rangle + b \right) - 1 \right),$$

where you look for a stationary point of L_P over w and b. Setting the gradient of L_P to 0, you get

$$\begin{aligned} w &= \sum_i \alpha_i y_i x_i \\ 0 &= \sum_i \alpha_i y_i . \end{aligned} \qquad \textbf{(1-1)}$$

Substituting into L_P, you get the dual L_D:

$$L_D = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \left\langle x_i, x_j \right\rangle,$$

which you maximize over $\alpha_i \geq 0$. In general, many α_i are 0 at the maximum. The nonzero α_i in the solution to the dual problem define the hyperplane, as seen in Equation 1-1, which gives w as the sum of $\alpha_i y_i x_i$. The data points x_i corresponding to nonzero α_i are the *support vectors*.

The derivative of L_D with respect to a nonzero α_i is 0 at an optimum. This gives

$$y_i(<w,x_i> + b) - 1 = 0.$$

In particular, this gives the value of b at the solution, by taking any i with nonzero α_i.

The dual is a standard quadratic programming problem. For example, the Optimization Toolbox `quadprog` solver solves this type of problem.

Nonseparable Data

Your data might not allow for a separating hyperplane. In that case, SVM can use a *soft margin*, meaning a hyperplane that separates many, but not all data points.

There are two standard formulations of soft margins. Both involve adding slack variables s_i and a penalty parameter C.

- The L^1-norm problem is:

  $$\min_{w,b,s}\left(\frac{1}{2}\langle w,w\rangle + C\sum_i s_i\right)$$

  such that

  $$\begin{aligned} y_i\left(\langle w,x_i\rangle + b\right) &\geq 1 - s_i \\ s_i &\geq 0. \end{aligned}$$

  The L^1-norm refers to using s_i as slack variables instead of their squares. The `SMO` `svmtrain` method minimizes the L^1-norm problem.

- The L^2-norm problem is:

  $$\min_{w,b,s}\left(\frac{1}{2}\langle w,w\rangle + C\sum_i s_i^2\right)$$

  subject to the same constraints. The `QP` `svmtrain` method minimizes the L^2-norm problem.

In these formulations, you can see that increasing C places more weight on the slack variables s_i, meaning the optimization attempts to make a stricter separation between classes. Equivalently, reducing C towards 0 makes misclassification less important.

Mathematical Formulation: Dual. For easier calculations, consider the L^1 dual problem to this soft-margin formulation. Using Lagrange multipliers μ_i, the function to minimize for the L^1-norm problem is:

$$L_P = \frac{1}{2}\langle w,w\rangle + C\sum_i s_i - \sum_i \alpha_i \left(y_i \left(\langle w,x_i \rangle + b \right) - (1 - s_i) \right) - \sum_i \mu_i s_i,$$

where you look for a stationary point of L_P over w, b, and positive s_i. Setting the gradient of L_P to 0, you get

$$\begin{aligned} b &= \sum_i \alpha_i y_i x_i \\ \sum_i \alpha_i y_i &= 0 \\ \alpha_i &= C - \mu_i \\ \alpha_i, \mu_i, s_i &\geq 0. \end{aligned}$$

These equations lead directly to the dual formulation:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \left\langle x_i, x_j \right\rangle$$

subject to the constraints

$$\begin{aligned} \sum_i y_i \alpha_i &= 0 \\ 0 \leq \alpha_i &\leq C. \end{aligned}$$

The final set of inequalities, $0 \leq \alpha_i \leq C$, shows why C is sometimes called a *box constraint*. C keeps the allowable values of the Lagrange multipliers α_i in a “box”, a bounded region.

The gradient equation for b gives the solution b in terms of the set of nonzero α_i, which correspond to the support vectors.

You can write and solve the dual of the L^2-norm problem in an analogous manner. For details, see Christianini and Shawe-Taylor [2], Chapter 6.

svmtrain Implementation. Both dual soft-margin problems are quadratic programming problems. Internally, `svmtrain` has several different algorithms for solving the problems. The default Sequential Minimal Optimization (SMO) algorithm minimizes the one-norm problem. SMO is a relatively fast algorithm. If you have an Optimization Toolbox license, you can choose to use `quadprog` as the algorithm. `quadprog` minimizes the L^2-norm problem. `quadprog` uses a good deal of memory, but solves quadratic programs to a high degree of precision (see Bottou and Lin [1]). For details, see the `svmtrain` function reference page.

Nonlinear Transformation with Kernels

Some binary classification problems do not have a simple hyperplane as a useful separating criterion. For those problems, there is a variant of the mathematical approach that retains nearly all the simplicity of an SVM separating hyperplane.

This approach uses these results from the theory of reproducing kernels:

- There is a class of functions $K(x,y)$ with the following property. There is a linear space S and a function φ mapping x to S such that

  $$K(x,y) = <\varphi(x),\varphi(y)>.$$

  The dot product takes place in the space S.
- This class of functions includes:
  - Polynomials: For some positive integer d,

    $$K(x,y) = (1 + <x,y>)^d.$$

  - Radial basis function: For some positive number σ,

    $$K(x,y) = \exp(-<(x-y),(x - y)>/(2\sigma^2)).$$

  - Multilayer perceptron (neural network): For a positive number p_1 and a negative number p_2,

    $$K(x,y) = \tanh(p_1<x,y> + p_2).$$

Note Not every set of p_1 and p_2 gives a valid reproducing kernel.

The mathematical approach using kernels relies on the computational method of hyperplanes. All the calculations for hyperplane classification use nothing more than dot products. Therefore, nonlinear kernels can use identical calculations and solution algorithms, and obtain classifiers that are nonlinear. The resulting classifiers are hypersurfaces in some space S, but the space S does not have to be identified or examined.

Using Support Vector Machines

As with any supervised learning model, you first train a support vector machine, then use the trained machine to classify (predict) new data. In addition, to obtain satisfactory predictive accuracy, you can use various SVM kernel functions, and you must tune the parameters of the kernel functions.

- "Training an SVM Classifier" on page 1-41
- "Classifying New Data with an SVM Classifier" on page 1-42
- "Tuning an SVM Classifier" on page 1-42

Training an SVM Classifier

Train an SVM classifier with the `svmtrain` function. The most common syntax is:

```
SVMstruct = svmtrain(data,groups,'Kernel_Function','rbf');
```

The inputs are:

- `data` — Matrix of data points, where each row is one observation, and each column is one feature.
- `groups` — Column vector with each row corresponding to the value of the corresponding row in `data`. `groups` should have only two types of entries. So `groups` can have logical entries, or can be a double vector or cell array with two values.

- Kernel_Function — The default value of 'linear' separates the data by a hyperplane. The value 'rbf' uses a Gaussian radial basis function. Hsu, Chang, and Lin [4] suggest using 'rbf' as your first try.

The resulting structure, SVMstruct, contains the optimized parameters from the SVM algorithm, enabling you to classify new data.

For more name-value pairs you can use to control the training, see the svmtrain reference page.

Classifying New Data with an SVM Classifier

Classify new data with the svmclassify function. The syntax for classifying new data with a SVMstruct structure is:

```
newClasses = svmclassify(SVMstruct,newData)
```

The resulting vector, newClasses, represents the classification of each row in newData.

Tuning an SVM Classifier

Hsu, Chang, and Lin [4] recommend tuning parameters of your classifier according to this scheme:

- Start with Kernel_Function set to 'rbf' and default parameters.
- Try different parameters for training, and check via cross validation to obtain the best parameters.

The most important parameters to try changing are:

- boxconstraint — One strategy is to try a geometric sequence of the box constraint parameter. For example, take 11 values, from 1e-5 to 1e5 by a factor of 10.
- rbf_sigma — One strategy is to try a geometric sequence of the RBF sigma parameter. For example, take 11 values, from 1e-5 to 1e5 by a factor of 10.

For the various parameter settings, try cross validating the resulting classifier. Use crossval with 5-way or the default 10-way cross validation.

After obtaining a reasonable initial parameter, you might want to refine your parameters to obtain better accuracy. Start with your initial parameters and perform another cross validation step, this time using a factor of 1.2. Alternatively, optimize your parameters with `fminsearch`, as shown in "Example: SVM Classification with Cross Validation" on page 1-47.

Example: Nonlinear Classifier with Gaussian Kernel

This example generates one class of points inside the unit disk in two dimensions, and another class of points in the annulus from radius 1 to radius 2. It then generates a classifier based on the data with the Gaussian radial basis function kernel. The default linear classifier is obviously unsuitable for this problem, since the model is circularly symmetric. Set the box constraint parameter to `Inf` to make a strict classification, meaning no misclassified training points.

Note Other kernel functions might not work with this strict box constraint, since they might be unable to provide a strict classification. Even though the `rbf` classifier can separate the classes, the result can be overtrained.

1 Generate 100 points uniformly distributed in the unit disk. To do so, generate a radius r as the square root of a uniform random variable, generate an angle t uniformly in $(0,2\pi)$, and put the point at $(r\cos(t),r\sin(t)$.

```
r = sqrt(rand(100,1)); % radius
t = 2*pi*rand(100,1); % angle
data1 = [r.*cos(t), r.*sin(t)]; % points
```

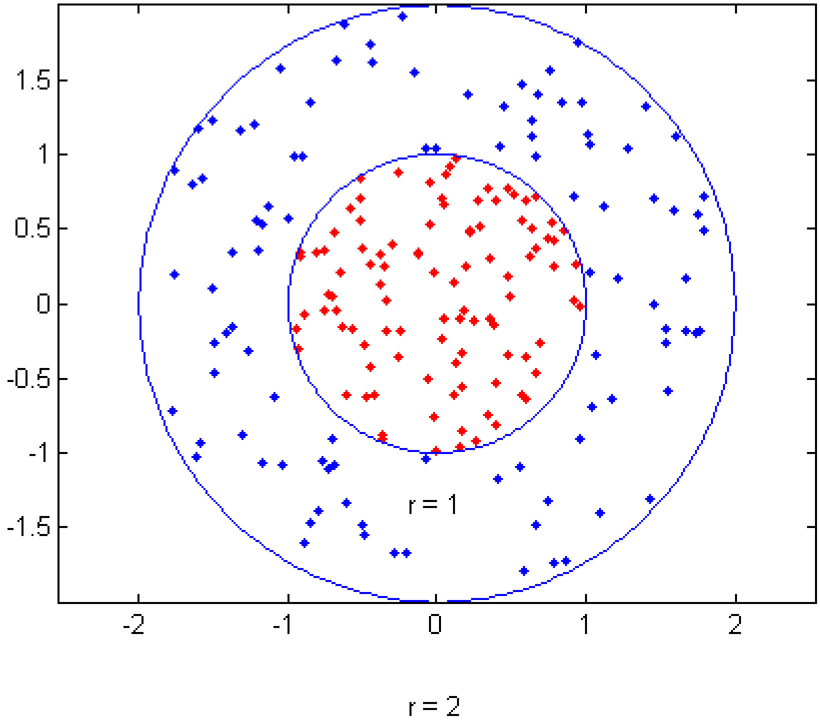
2 Generate 100 points uniformly distributed in the annulus. The radius is again proportional to a square root, this time a square root of the uniform distribution from 1 through 4.

```
r2 = sqrt(3*rand(100,1)+1); % radius
t2 = 2*pi*rand(100,1); % angle
data2 = [r2.*cos(t2), r2.*sin(t2)]; % points
```

3 Plot the points, and plot circles of radii 1 and 2 for comparison:

```
plot(data1(:,1),data1(:,2),'r.')
```

```
hold on
plot(data2(:,1),data2(:,2),'b.')
ezpolar(@(x)1);ezpolar(@(x)2);
axis equal
hold off
```



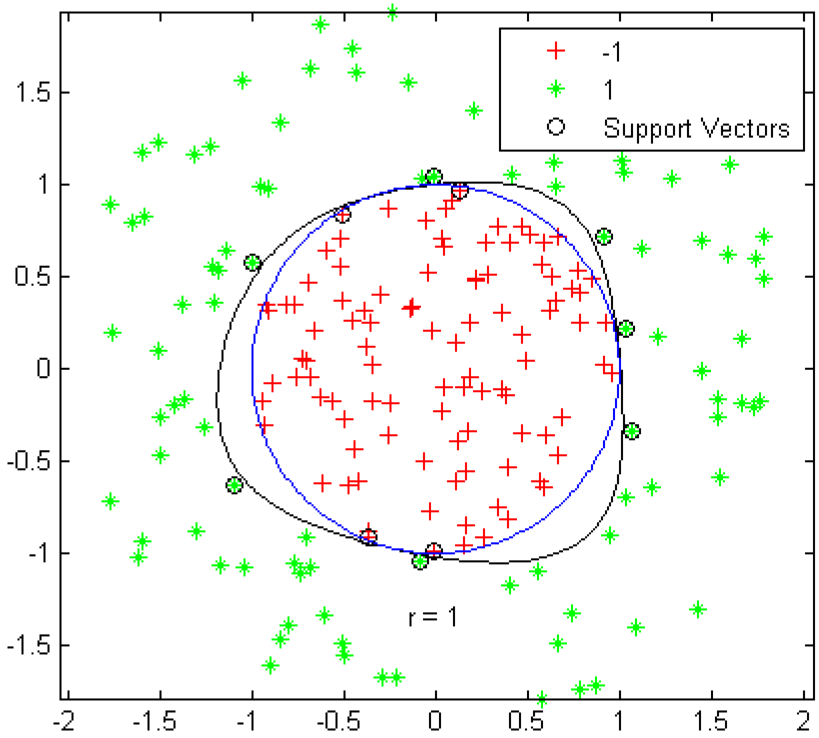
4 Put the data in one matrix, and make a vector of classifications:

```
data3 = [data1;data2];
theclass = ones(200,1);
theclass(1:100) = -1;
```

5 Train an SVM classifier with:

- `Kernel_Function` set to `'rbf'`
- `boxconstraint` set to `Inf`

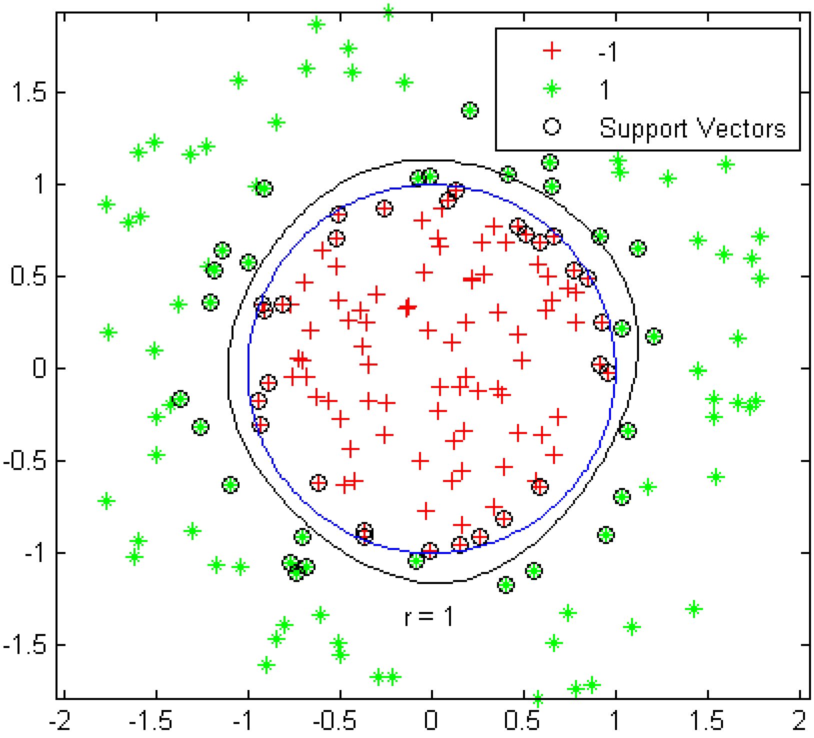
```
cl = svmtrain(data3,theclass,'Kernel_Function','rbf',...
    'boxconstraint',Inf,'showplot',true);
hold on
axis equal
ezpolar(@(x)1)
hold off
```



`svmtrain` generates a classifier that is close to a circle of radius 1. The difference is due to the random training data.

6 Training with the default parameters makes a more nearly circular classification boundary, but one that misclassifies some training data.

```
cl = svmtrain(data3,theclass,'Kernel_Function','rbf',...
    'showplot',true);
hold on
axis equal
ezpolar(@(x)1)
hold off
```



Example: SVM Classification with Cross Validation

This example classifies points from a Gaussian mixture model. The model is described in Hastie, Tibshirani, and Friedman [3], page 17. It begins with generating 10 base points for a “green” class, distributed as 2-D independent normals with mean (1,0) and unit variance. It also generates 10 base points for a “red” class, distributed as 2-D independent normals with mean (0,1) and unit variance. For each class (green and red), generate 100 random points as follows:

1. Choose a base point *m* of the appropriate color uniformly at random.
2. Generate an independent random point with 2-D normal distribution with mean *m* and variance I/5, where I is the 2-by-2 identity matrix.

After generating 100 green and 100 red points, classify them using `svmtrain`, and tune the classification using cross validation.

To generate the points and classifier:

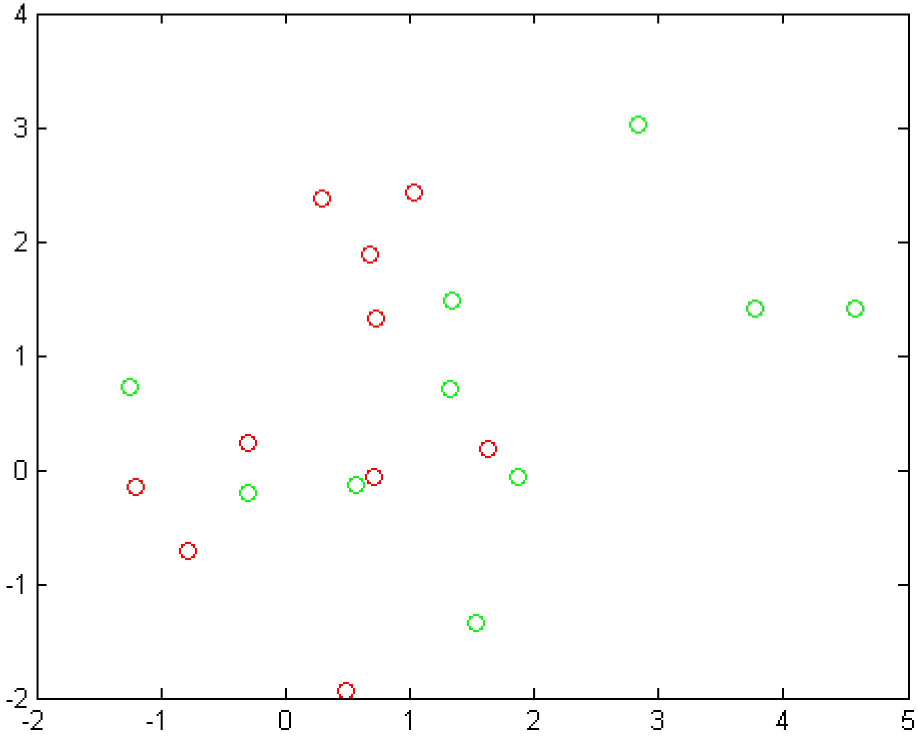
1. Generate the 10 base points for each class:

   ```
   grnpop = mvnrnd([1,0],eye(2),10);
   redpop = mvnrnd([0,1],eye(2),10);
   ```

2. View the base points:

   ```
   plot(grnpop(:,1),grnpop(:,2),'go')
   hold on
   plot(redpop(:,1),redpop(:,2),'ro')
   hold off
   ```

   Since many red base points are close to green base points, it is difficult to classify the data points.



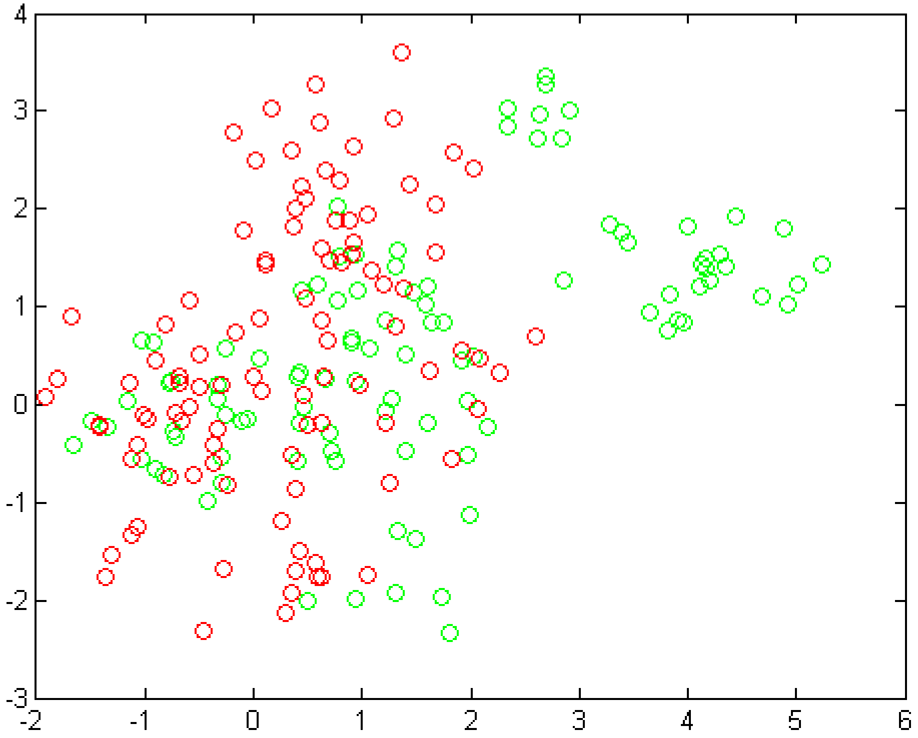
3 Generate the 100 data points of each class:

```
redpts = zeros(100,2);grnpts = redpts;
for i = 1:100
    grnpts(i,:) = mvnrnd(grnpop(randi(10),:),eye(2)*0.2);
    redpts(i,:) = mvnrnd(redpop(randi(10),:),eye(2)*0.2);
end
```

4 View the data points:

```
figure
plot(grnpts(:,1),grnpts(:,2),'go')
hold on
```

```
plot(redpts(:,1),redpts(:,2),'ro')
hold off
```

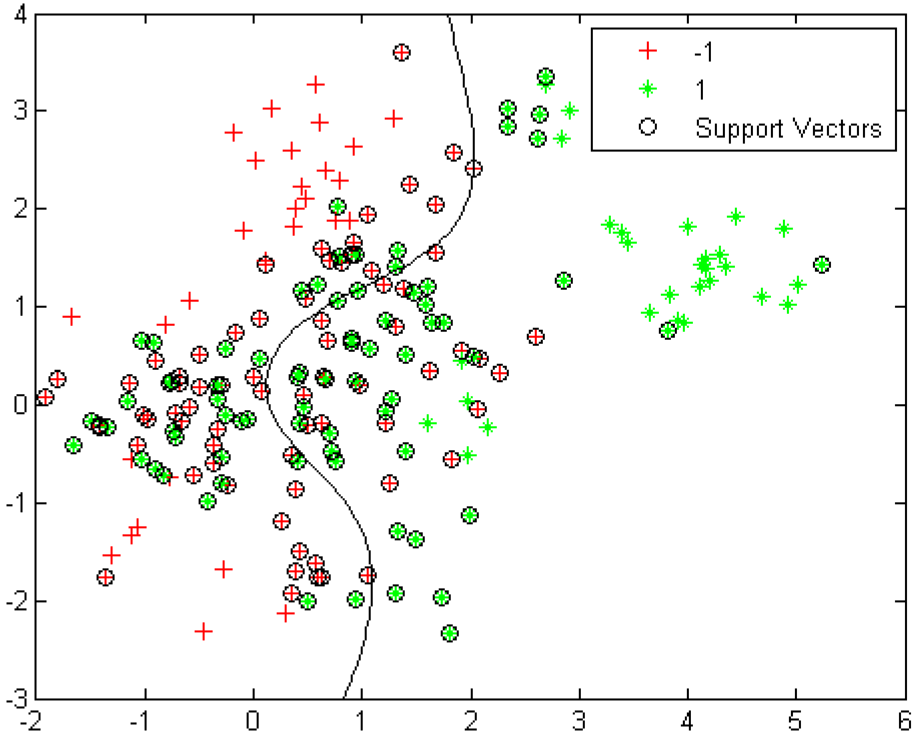


5 Put the data into one matrix, and make a vector grp that labels the class of each point:

```
cdata = [grnpts;redpts];
grp = ones(200,1);
% green label 1, red label -1
grp(101:200) = -1;
```

6 Check the basic classification of all the data using the default parameters:

```
svmStruct = svmtrain(cdata,grp,'Kernel_Function','rbf',...
    'showplot',true);
```



7 Write a function called `crossfun` to calculate the predicted classification `yfit` from a test vector `xtest`, when the SVM is trained on a sample `xtrain` that has classification `ytrain`. Since you want to find the best parameters `rbf_sigma` and `boxconstraint`, include those in the function.

```
function yfit = ...
    crossfun(xtrain,ytrain,xtest,rbf_sigma,boxconstraint)

% Train the model on xtrain, ytrain,
% and get predictions of class of xtest
svmStruct = svmtrain(xtrain,ytrain,'Kernel_Function','rbf',...
    'rbf_sigma',rbf_sigma,'boxconstraint',boxconstraint);
yfit = svmclassify(svmStruct,xtest);
```

8 Set up a partition for cross validation. This step causes the cross validation to be fixed. Without this step, the cross validation is random, so a minimization procedure can find a spurious local minimum.

```
c = cvpartition(200,'kfold',10);
```

9 Set up a function that takes an input `z=[rbf_sigma,boxconstraint]`, and returns the cross-validation value of `exp(z)`. The reason to take `exp(z)` is twofold:

- `rbf_sigma` and `boxconstraint` must be positive.
- You should look at points spaced approximately exponentially apart.

This function handle computes the cross validation at parameters `exp([rbf_sigma,boxconstraint])`:

```
minfn = @(z)crossval('mcr',cdata,grp,'Predfun', ...
    @(xtrain,ytrain,xtest)crossfun(xtrain,ytrain,...
    xtest,exp(z(1)),exp(z(2))),'partition',c);
```

10 Search for the best parameters `[rbf_sigma,boxconstraint]` with `fminsearch`, setting looser tolerances than the defaults.

Tip If you have a Global Optimization Toolbox license, use `patternsearch` for faster, more reliable minimization. Give bounds on the components of `z` to keep the optimization in a sensible region, such as [–5,5], and give a relatively loose `TolMesh` tolerance.

```
opts = optimset('TolX',5e-4,'TolFun',5e-4);
[searchmin fval] = fminsearch(minfn,randn(2,1),opts)

searchmin =
    0.9758
   -0.1569
```

```
fval =
    0.3350
```

The best parameters [rbf_sigma;boxconstraint] in this run are:

```
z = exp(searchmin)
z =
    2.6534
    0.8548
```

11 Since the result of fminsearch can be a local minimum, not a global minimum, try again with a different starting point to check that your result is meaningful:

```
[searchmin fval] = fminsearch(minfn,randn(2,1),opts)

searchmin =
    0.2778
    0.6395

fval =
    0.3100
```

The best parameters [rbf_sigma;boxconstraint] in this run are:

```
z = exp(searchmin)
z =
    1.3202
    1.8956
```

12 Try another search:

```
[searchmin fval] = fminsearch(minfn,randn(2,1),opts)

searchmin =
   -0.0749
    0.6085

fval =
    0.2850
```

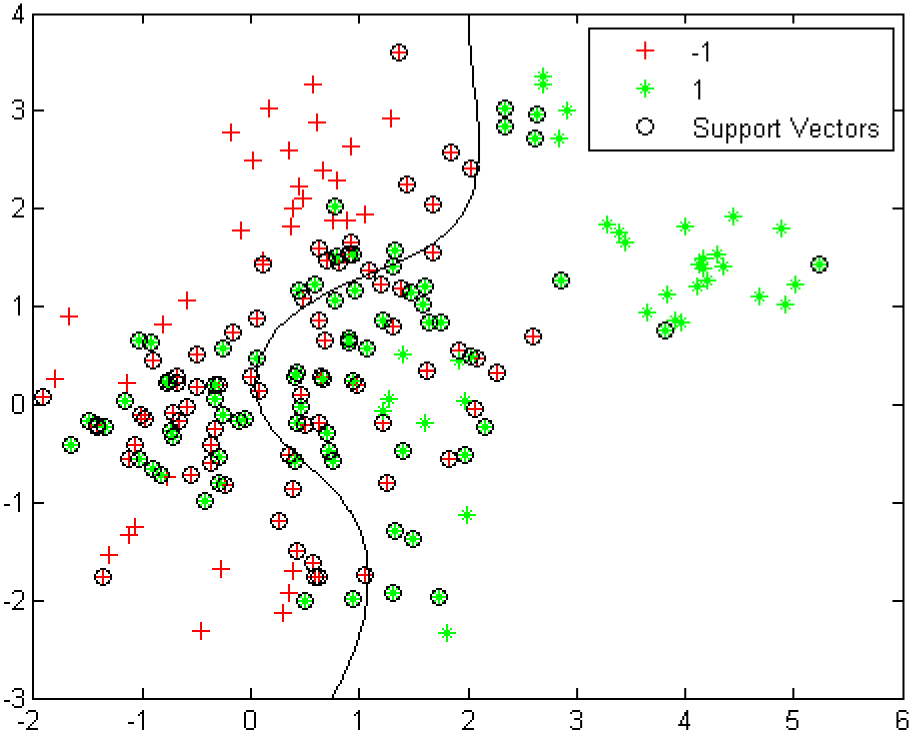
The third search obtains the lowest function value. The final parameters are:

```
z = exp(searchmin)
z =
    0.9278
    1.8376
```

The default parameters [1,1] are close to optimal for this data and partition.

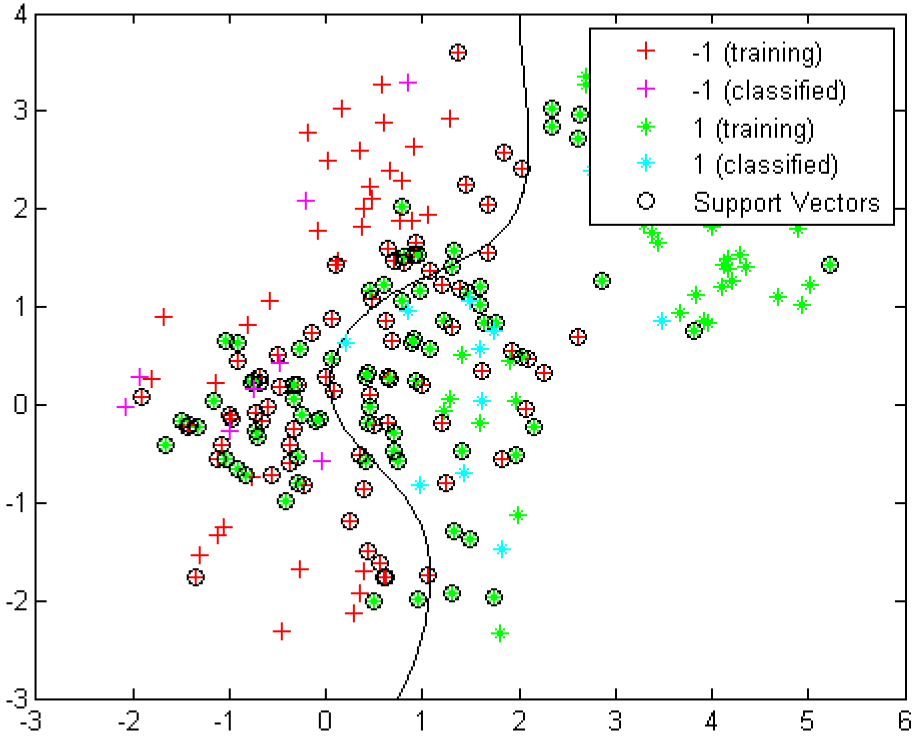
13 Use the z parameters to train a new SVM classifier:

```
svmStruct = svmtrain(cdata,grp,'Kernel_Function','rbf',...
'rbf_sigma',z(1),'boxconstraint',z(2),'showplot',true);
```



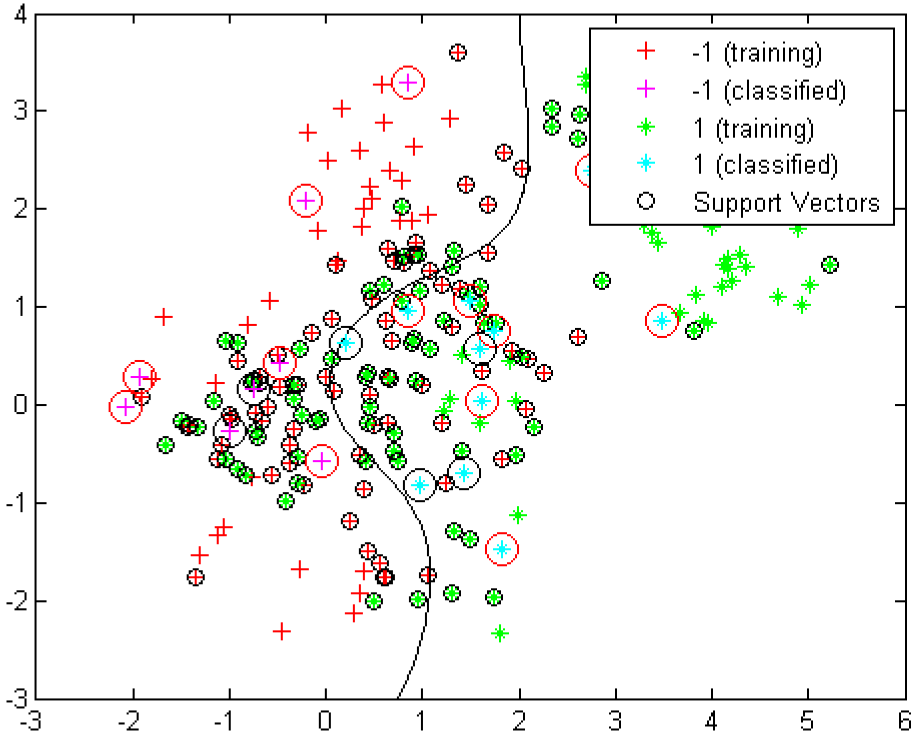
14 Generate and classify some new data points:

```
grnobj = gmdistribution(grnpop,.2*eye(2));
redobj = gmdistribution(redpop,.2*eye(2));

newData = random(grnobj,10);
newData = [newData;random(redobj,10)];
grpData = ones(20,1);
grpData(11:20) = -1; % red = -1

v = svmclassify(svmStruct,newData,'showplot',true);
```



15 See which new data points are correctly classified. Circle the correctly classified points in red, and the incorrectly classified points in black.

```
mydiff = (v == grpData); % classified correctly
hold on
for ii = mydiff % plot red circles around correct pts
    plot(newData(ii,1),newData(ii,2),'ro','MarkerSize',12)
end

for ii = not(mydiff) % plot black circles around incorrect pts
    plot(newData(ii,1),newData(ii,2),'ko','MarkerSize',12)
end
hold off
```



References

[1] Bottou, L., and Chih-Jen Lin. *Support Vector Machine Solvers*. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.4209&rep=rep1&type=pdf`.

[2] Christianini, N., and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, Cambridge, UK, 2000.

[3] Hastie, T., R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, second edition. Springer, New York, 2008.

[4] Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin. *A Practical Guide to Support Vector Classification*. Available at `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

High-Throughput Sequence Analysis

Extracting Information from Large Multi-Entry Text Files

In this section...

Overview

Many biological experiments produce huge data files that are difficult to access due to their size, which can cause memory issues when reading the file into the MATLAB Workspace. You can construct a `BioIndexedFile` object to access the contents of a large text file (up to 4 GB) containing nonuniform size entries, such as sequences, annotations, and cross-references to data sets. The `BioIndexedFile` object lets you quickly and efficiently access this data without loading the source file into memory.

You can use the `BioIndexedFile` object to access individual entries or a subset of entries when the source file is too big to fit into memory. You can access entries using indices or keys. You can read and parse one or more entries using provided interpreters or a custom interpreter function.

Use the `BioIndexedFile` object in conjunction with your large source file to:

- Access a subset of the entries for validation or further analysis.
- Parse entries using a custom interpreter function.

What Files Can You Access?

You can use the `BioIndexedFile` object to access large text files up to 4 GB in size.

Your source file can have these application-specific formats:

- FASTA
- FASTQ
- SAM

Your source file can also have these general formats:

- **Table** — Tab-delimited table with multiple columns. Keys can be in any column. Rows with the same key are considered separate entries.
- **Multi-row Table** — Tab-delimited table with multiple columns. Keys can be in any column. Contiguous rows with the same key are considered a single entry. Noncontiguous rows with the same key are considered separate entries.
- **Flat** — Flat file with concatenated entries separated by a character string, typically `//`. Within an entry, the key is separated from the rest of the entry by a white space.

Before You Begin

Before constructing a `BioIndexedFile` object, locate your source file on your hard drive or a local network.

When you construct a `BioIndexedFile` object from your source file for the first time, you also create an auxiliary index file, which by default is saved to the same location as your source file. However, if your source file is in a read-only location, you can specify a different location to save the index file.

Tip If you construct a `BioIndexedFile` object from your source file on subsequent occasions, it takes advantage of the existing index file, which saves time. However, the index file must be in the same location or a location specified by the subsequent construction syntax.

Tip If insufficient memory is not an issue when accessing your source file, you may want to try an appropriate read function, such as genbankread, for importing data from GenBank files. For a complete list of read functions in the Bioinformatics Toolbox, see “Data Formats and Databases” in the Bioinformatics Toolbox Reference.

Additionally, several read functions such as fastaread, fastqread, samread, and sffread include a Blockread property, which lets you read a subset of entries from a file, thus saving memory.

Creating a BioIndexedFile Object to Access Your Source File

To construct a BioIndexedFile object from a multi-row table file:

1 Create a variable containing the full absolute path of your source file. For your source file, use the yeastgenes.sgd file, which is included with the Bioinformatics Toolbox software.

```
sourcefile = which('yeastgenes.sgd');
```

2 Use the BioIndexedFile constructor function to construct a BioIndexedFile object from the yeastgenes.sgd source file, which is a multi-row table file. Save the index file in the Current Folder. Indicate that the source file keys are in column 3. Also, indicate that the header lines in the source file are prefaced with !, so the constructor ignores them.

```
gene2goObj = BioIndexedFile('mrtab', sourcefile, '.', ...
                            'KeyColumn', 3, 'HeaderPrefix','!')
```

The BioIndexedFile constructor function constructs gene2goObj, a BioIndexedFile object, and also creates an index file with the same name as the source file, but with an IDX extension. It stores this index file in the Current Folder because we specified this location. However, the default location for the index file is the same location as the source file.

Caution Do not modify the index file. If you modify it, you can get invalid results. Also, the constructor function cannot use a modified index file to construct future objects from the associated source file.

Determining the Number of Entries Indexed By a BioIndexedFile Object

To determine the number of entries indexed by a `BioIndexedFile` object, use the `NumEntries` property of the `BioIndexedFile` object. For example, for the `gene2goObj` object:

```
gene2goObj.NumEntries

ans =

        6476
```

Note For a list and description of all properties of a `BioIndexedFile` object, see `BioIndexedFile class`.

Retrieving Entries from Your Source File

Retrieve entries from your source file using either:

- The index of the entry
- The entry key

Retrieving Entries Using Indices

Use the `getEntryByIndex` method to retrieve a subset of entries from your source file that correspond to specified indices. For example, retrieve the first 12 entries from the `yeastgenes.sgd` source file:

```
subset_entries = getEntryByIndex(gene2goObj, [1:12]);
```

Retrieving Entries Using Keys

Use the getEntryByKey method to retrieve a subset of entries from your source file that are associated with specified keys. For example, retrieve all entries with keys of AAC1 and AAD10 from the yeastgenes.sgd source file:

```
subset_entries = getEntryByKey(gene2goObj, {'AAC1' 'AAD10'});
```

The output subset_entries is a single string of concatenated entries. Because the keys in the yeastgenes.sgd source file are not unique, this method returns all entries that have a key of AAC1 or AAD10.

Reading Entries from Your Source File

The BioIndexedFile object includes a read method, which you can use to read and parse a subset of entries from your source file. The read method parses the entries using an interpreter function specified by the Interpreter property of the BioIndexedFile object.

Setting the Interpreter Property

Before using the read method, make sure the Interpreter property of the BioIndexedFile object is set appropriately.

If you constructed a BioIndexedFile object from ...	The Interpreter property ...
A source file with an application-specific format (FASTA, FASTQ, or SAM)	By default is a handle to a function appropriate for that file type and typically does not require you to change it.
A source file with a table, multi-row table, or flat format	By default is [], which means the interpreter is an anonymous function in which the output is equivalent to the input. You can change this to a handle to a function that accepts a single string of one or more concatenated entries and returns a structure or an array of structures containing the interpreted data.

There are two ways to set the `Interpreter` property of the `BioIndexedFile` object:

- When constructing the `BioIndexedFile` object, use the `Interpreter` property name/property value pair
- After constructing the `BioIndexedFile` object, set the `Interpreter` property

Note For more information on setting the `Interpreter` property of a `BioIndexedFile` object, see `BioIndexedFile class`.

Reading a Subset of Entries

The `read` method reads and parses a subset of entries that you specify using either entry indices or keys.

Example

To quickly find all the gene ontology (GO) terms associated with a particular gene because the entry keys are gene names:

1. Set the `Interpreter` property of the `gene2goObj` `BioIndexedFile` object to a handle to a function that reads entries and returns only the column containing the GO term. In this case the interpreter is a handle to an anonymous function that accepts strings and extracts strings that start with the characters `GO.`

   ```
   gene2goObj.Interpreter = @(x) regexp(x,'GO:\d+','match')
   ```

2. Read only the entries that have a key of YAT2, and return their GO terms.

   ```
   GO_YAT2_entries = read(gene2goObj, 'YAT2')

   GO_YAT2_entries =

   'GO:0004092' 'GO:0005737' 'GO:0006066' 'GO:0006066' 'GO:0009437'
   ```

Storing and Managing Short-Read Sequence Data in Objects

In this section...

Overview

High-throughput sequencing instruments produce large amounts of short-read sequence data that can be challenging to store and manage. Using objects to contain this data lets you easily access, manipulate, and filter the data.

Bioinformatics Toolbox includes two objects for working with short-read sequence data.

Object	Contains This Information	Construct from One of These
`BioRead`	• Sequence headers • Read sequences • Sequence qualities (base calling)	• FASTQ file • SAM file • FASTQ structure (created using the `fastqread` function) • SAM structure (created using the `samread` function)

Object	Contains This Information	Construct from One of These
		• Cell arrays containing header, sequence, and quality information (created using the `fastqread` function)
`BioMap`	• Sequence headers • Read sequences • Sequence qualities (base calling) • Sequence alignment and mapping information (relative to a single reference sequence), including mapping quality	• SAM file • BAM file • SAM structure (created using the `samread` function) • BAM structure (created using the `bamread` function) • Cell arrays containing header, sequence, quality, and mapping/alignment information (created using the `samread` or `bamread` function)

Representing Sequence and Quality Data in a BioRead Object

Prerequisites

A `BioRead` object represents a collection of short-read sequences. Each element in the object is associated with a sequence, sequence header, and sequence quality information.

To construct a `BioRead` object from a FASTQ or SAM file, the file must be less than 4 GB.

Construct a `BioRead` object in one of two ways:

- **Indexed** — The data remains in the source file. Constructing the object and accessing its contents is memory efficient. However, you cannot modify object properties, other than the Name property. This is the default method if you construct a BioRead object from a FASTQ- or SAM-formatted file.
- **In Memory** — The data is read into memory. Constructing the object and accessing its contents is limited by the amount of available memory. However, you can modify object properties. When you construct a BioRead object from a FASTQ structure or cell arrays, the data is read into memory. When you construct a BioRead object from a FASTQ- or SAM-formatted file, use the InMemory name-value pair argument to read the data into memory.

Constructing a BioRead Object from a FASTQ- or SAM-Formatted File

Note This example constructs a BioRead object from a FASTQ-formatted file. Use similar steps to construct a BioRead object from a SAM-formatted file.

Use the BioRead constructor function to construct a BioRead object from a FASTQ-formatted file and set the Name property:

```
BRObj1 = BioRead('SRR005164_1_50.fastq', 'Name', 'MyObject')

BRObj1 =

  BioRead

    Properties:
       Quality: {50x1 cell}
      Sequence: {50x1 cell}
        Header: {50x1 cell}
         NSeqs: 50
          Name: 'MyObject'

  Methods, Superclasses
```

The constructor function construct a BioRead object and, if an index file does not already exist, it also creates an index file with the same file name, but with an .IDX extension. This index file, by default, is stored in the same location as the source file.

Caution Your source file and index file must always be in sync.

- After constructing a BioRead object, do not modify the index file, or you can get invalid results when using the existing object or constructing new objects.
- If you modify the source file, delete the index file, so the object constructor creates a new index file when constructing new objects.

Note Because you constructed this BioRead object from a source file, you cannot modify the properties (except for Name) of the BioRead object.

Representing Sequence, Quality, and Alignment/Mapping Data in a BioMap Object

Prerequisites

A BioMap object represents a collection of short-read sequences that map against a single reference sequence. Each element in the object is associated with a read sequence, sequence header, sequence quality information, and alignment/mapping information.

To construct a BioMap object from a SAM file, the file must be less than 4 GB.

When constructing a BioMap object from a BAM file, the maximum size of the file is limited by your operating system and available memory.

Construct a BioMap object in one of two ways:

- **Indexed** — The data remains in the source file. Constructing the object and accessing its contents is memory efficient. However, you cannot modify object properties, other than the Name property. This is the default method if you construct a BioMap object from a SAM- or BAM-formatted file.

- **In Memory** — The data is read into memory. Constructing the object and accessing its contents is limited by the amount of available memory. However, you can modify object properties. When you construct a BioMap object from a structure, the data is read into memory. When you construct a BioMap object from a SAM- or BAM-formatted file, use the InMemory name-value pair argument to read the data into memory.

Constructing a BioMap Object from a SAM- or BAM-Formatted File

Note This example constructs a BioMap object from a SAM-formatted file. Use similar steps to construct a BioMap object from a BAM-formatted file.

1 To construct a valid BioMap object from a SAM- or BAM-formatted file, the file must contain only one reference sequence. Determine the number and names of the reference sequences in your source file using the saminfo or baminfo function and the ScanDictionary name-value pair argument.

```
samstruct = saminfo('ex2.sam', 'ScanDictionary', true);
samstruct.ScannedDictionary

ans =

    'seq1'
    'seq2'
```

Tip The previous syntax scans the entire SAM file, which is time consuming. If you are confident that the Header information of the SAM file is correct, omit the ScanDictionary name-value pair argument, and inspect the SequenceDictionary field instead.

2 Use the BioMap constructor function to construct a BioMap object from the SAM file and set the Name property. Because the SAM-formatted file

in this example, ex2.sam, contains multiple reference sequences, use the SelectRef name-value pair argument to specify one reference sequence, seq1:

```
BMObj2 = BioMap('ex2.sam', 'SelectRef', 'seq1', 'Name', 'MyObject')

BMObj2 =

  BioMap

Properties:
         Reference: 'seq1'
         Signature: {1501x1 cell}
             Start: [1501x1 uint32]
    MappingQuality: [1501x1 uint8]
              Flag: [1501x1 uint16]
      MatePosition: [1501x1 uint32]
           Quality: {1501x1 cell}
          Sequence: {1501x1 cell}
            Header: {1501x1 cell}
             NSeqs: 1501
              Name: 'MyObject'

  Methods, Superclasses
```

The constructor function constructs a BioMap object and, if an index file does not already exist, it also creates an index file with the same file name, but with an .IDX or .BAI extension. This index file, by default, is stored in the same location as the source file.

Caution Your source file and index file must always be in sync.

- After constructing a BioMap object, do not modify the index file, or you can get invalid results when using the existing object or constructing new objects.
- If you modify the source file, delete the index file, so the object constructor creates a new index file when constructing new objects.

Note Because you constructed this BioMap object from a source file, you cannot modify the properties (except for Name and Reference) of the BioMap object.

Constructing a BioMap Object from a SAM or BAM Structure

Note This example constructs a BioMap object from a SAM structure using samread. Use similar steps to construct a BioMap object from a BAM structure using bamread.

1 Use the samread function to create a SAM structure from a SAM-formatted file:

```
SAMStruct = samread('ex2.sam');
```

2 To construct a valid BioMap object from a SAM-formatted file, the file must contain only one reference sequence. Determine the number and names of the reference sequences in your SAM-formatted file using the unique function to find unique names in the ReferenceName field of the structure:

```
unique({SAMStruct.ReferenceName})

ans =

    'seq1'    'seq2'
```

3 Use the BioMap constructor function to construct a BioMap object from a SAM structure. Because the SAM structure contains multiple reference sequences, use the SelectRef name-value pair argument to specify one reference sequence, seq1:

```
BMObj1 = BioMap(SAMStruct, 'SelectRef', 'seq1')

BMObj1 =

  BioMap
```

```
Properties:
        Reference: 'seq1'
        Signature: {1501x1 cell}
            Start: [1501x1 uint32]
   MappingQuality: [1501x1 uint8]
             Flag: [1501x1 uint16]
     MatePosition: [1501x1 uint32]
          Quality: {1501x1 cell}
         Sequence: {1501x1 cell}
           Header: {1501x1 cell}
            NSeqs: 1501
             Name: ''

 Methods, Superclasses
```

Retrieving Information from a BioRead or BioMap Object

You can retrieve all or a subset of information from a `BioRead` or `BioMap` object.

Retrieving All Values of a Property from a BioRead or BioMap Object

Use the `get` method to retrieve a specific property from all elements in a `BioRead` or `BioMap` object. For example, to retrieve all headers from a `BioRead` object, use the `get` method with the `Header` property:

```
allHeaders = get(BRObj1, 'Header');
```

The previous syntax returns a cell array containing the headers for all elements in the `BioRead` object. For example, to retrieve all start positions of aligned read sequences from a `BioMap` object, use the `get` method with the `Start` property:

```
allStarts = get(BMObj1, 'Start');
```

The previous syntax returns a vector containing the start positions of aligned read sequences with respect to the position numbers in the reference sequence in a `BioMap` object.

Note Property names are case sensitive.

For a list and description of all properties of a BioRead object, see BioRead class. For a list and description of all properties of a BioMap object, see BioMap class.

Retrieving a Subset of Information from a BioRead or BioMap Object

Use specialized get methods with a numeric vector, logical vector, or cell array of headers to retrieve a subset of information from an object. For example, to retrieve the first 10 elements from a BioRead object, use the getSubset method:

```
newBRObj = getSubset(BRObj1, [1:10]);
```

The previous syntax returns a new BioRead object containing the first 10 elements in the original BioRead object.

For example, to retrieve the first 12 positions of sequences with headers SRR005164.1, SRR005164.7, and SRR005164.16, use the getSubsequence method:

```
subSeqs = getSubsequence(BRObj1, ...
         {'SRR005164.1', 'SRR005164.7', 'SRR005164.16'}, [1:12]')

subSeqs =

    'TGGCTTTAAAGC'
    'CCCGAAAGCTAG'
    'AATTTTGCGGCT'
```

For example, to retrieve information about the third element in a BioMap object, use the getInfo method:

```
Info_3 = getInfo(BMObj1, 3);
```

The previous syntax returns a tab-delimited string containing this information for the third element:

- Sequence header
- SAM flags for the sequence
- Start position of the aligned read sequence with respect to the reference sequence
- Mapping quality score for the sequence
- Signature (CIGAR-formatted string) for the sequence
- Sequence
- Quality scores for sequence positions

Note Method names are case sensitive.

For a complete list and description of methods of a `BioRead` object, see `BioRead` class. For a complete list and description of methods of a `BioMap` object, see `BioMap` class.

Setting Information in a BioRead or BioMap Object

Prerequisites

Several specialized `set` methods let you set the properties of a subset of elements in a `BioRead` or `BioMap` object.

To modify properties (other than `Name` and `Reference`) of a `BioRead` or `BioMap` object, the data must be in memory, and not indexed. To ensure the data is in memory, do one of the following:

- Construct the object from a structure as described in “Constructing a BioMap Object from a SAM or BAM Structure” on page 2-14.
- Construct the object from a source file using the `InMemory` name-value pair argument.

Providing Custom Headers for Sequences

To provide custom headers for sequences of interest (in this case sequences 2, 4, and 6), use the `setHeader` method:

```
newBRObj = setHeader(BRObj1, {'H2', 'H4', 'H6'}, [2 4 6]);
```

The previous syntax returns a new object containing the new headers.

Renaming the Reference Sequence

To rename the reference sequence in a `BioMap` object, use the `setReference` method:

```
BMObj1 = setReference(BMObj1, 'Chromosome7');
```

The previous syntax updates the name of the reference sequence from `seq1` to `Chromosome7` in the `BioMap` object.

Note Method names are case sensitive.

For a complete list and description of methods of a `BioRead` object, see `BioRead` class. For a complete list and description of methods of a `BioMap` object, see `BioMap` class.

Determining Coverage of a Reference Sequence

When working with a `BioMap` object, you can determine the number of read sequences that:

- Align within a specific region of the reference sequence
- Align to each position within a specific region of the reference sequence

For example, you can compute the number, indices, and start positions of the read sequences that align within the first 25 positions of the reference sequence. To do so, use the `getCounts`, `getIndex`, and `getStart` methods:

```
Cov = getCounts(BMObj1, 1, 25)

Cov =

    12

Indices = getIndex(BMObj1, 1, 25)
Indices =

     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
    11
    12

startPos = getStart(BMObj1, Indices)

startPos =

           1
           3
           5
           6
           9
          13
          13
          15
          18
          22
          22
          24
```

The first two syntaxes return the number and indices of the read sequences that align within the specified region of the reference sequence. The last syntax returns a vector containing the start position of each aligned read sequence, corresponding to the position numbers of the reference sequence.

For example, you can also compute the number of the read sequences that align to *each* of the first 10 positions of the reference sequence. For this computation, use the `getBaseCoverage` method:

```
Cov = getBaseCoverage(BMObj1, 1, 10)

Cov =

     1     1     2     2     3     4     4     4     5     5
```

Constructing Sequence Alignments to a Reference Sequence

It is useful to construct and view the alignment of the read sequences that align to a specific region of the reference sequence. It is also helpful to know which read sequences align to this region in a BioMap object.

For example, to retrieve the alignment of read sequences to the first 12 positions of the reference sequence in a BioMap object, use the getAlignment method:

```
[Alignment_1_12, Indices] = getAlignment(BMObj2, 1, 12)

Alignment_1_12 =

CACTAGTGGCTC
  CTAGTGGCTC
    AGTGGCTC
     GTGGCTC
        GCTC


Indices =

     1
     2
     3
     4
     5
```

Return the headers of the read sequences that align to a specific region of the reference sequence:

```
alignedHeaders = getHeader(BMObj2, Indices)
alignedHeaders =

    'B7_591:4:96:693:509'
    'EAS54_65:7:152:368:113'
    'EAS51_64:8:5:734:57'
    'B7_591:1:289:587:906'
    'EAS56_59:8:38:671:758'
```

Filtering Read Sequences Using SAM Flags

SAM- and BAM-formatted files include the status of 11 binary flags for each read sequence. These flags describe different sequencing and alignment aspects of a read sequence. For more information on the flags, see the The SAM Format Specification. The `filterByFlag` method lets you filter the read sequences in a `BioMap` object by using these flags.

Filtering Unmapped Read Sequences

1 Construct a `BioMap` object from a SAM-formatted file.

```
BMObj2 = BioMap('ex1.sam');
```

2 Use the `filterByFlag` method to create a logical vector indicating the read sequences in a `BioMap` object that are mapped.

```
LogicalVec_mapped = filterByFlag(BMObj2, 'unmappedQuery', false);
```

3 Use this logical vector and the `getSubset` method to create a new `BioMap` object containing only the mapped read sequences.

```
filteredBMObj_1 = getSubset(BMObj2, LogicalVec_mapped);
```

Filtering Read Sequences That Are Not Mapped in a Pair

1 Construct a `BioMap` object from a SAM-formatted file.

```
BMObj2 = BioMap('ex1.sam');
```

2 Use the `filterByFlag` method to create a logical vector indicating the read sequences in a `BioMap` object that are mapped in a proper pair, that is, both the read sequence and its mate are mapped to the reference sequence.

```
LogicalVec_paired = filterByFlag(BMObj2, 'pairedInMap', true);
```

3 Use this logical vector and the getSubset method to create a new BioMap object containing only the read sequences that are mapped in a proper pair.

```
filteredBMObj_2 = getSubset(BMObj2, LogicalVec_paired);
```

Storing and Managing Feature Annotations in Objects

In this section...

Representing Feature Annotations in a GFFAnnotation or GTFAnnotation Object

The `GFFAnnotation` and `GTFAnnotation` objects represent a collection of feature annotations for one or more reference sequences. You construct these objects from GFF (General Feature Format) and GTF (Gene Transfer Format) files. Each element in the object represents a single annotation. The properties and methods associated with the objects let you investigate and filter the data based on reference sequence, a feature (such as CDS or exon), or a specific gene or transcript.

Constructing an Annotation Object

Use the `GFFAnnotation` constructor function to construct a `GFFAnnotation` object from either a GFF- or GTF-formatted file:

```
GFFAnnotObj = GFFAnnotation('tair8_1.gff')

GFFAnnotation

  Properties:
    FieldNames: {1x9 cell}
    NumEntries: 3331

  Methods, Superclasses
```

Use the GTFAnnotation constructor function to construct a GTFAnnotation object from a GTF-formatted file:

```
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf')

GTFAnnotObj =

  GTFAnnotation

  Properties:
    FieldNames: {1x11 cell}
    NumEntries: 308

  Methods, Superclasses
```

Retrieving General Information from an Annotation Object

Determine the field names and the number of entries in an annotation object by accessing the FieldNames and NumEntries properties. For example, to see the field names for each annotation object constructed in the previous section, query the FieldNames property:

```
GFFAnnotObj.FieldNames

ans =

  Columns 1 through 6

   'Reference'   'Start'   'Stop'   'Feature'   'Source'   'Score'

  Columns 7 through 9

   'Strand'   'Frame'   'Attributes'

GTFAnnotObj.FieldNames

ans =

  Columns 1 through 6
```

```
    'Reference'    'Start'    'Stop'    'Feature'    'Gene'    'Transcript'

   Columns 7 through 11

    'Source'    'Score'    'Strand'    'Frame'    'Attributes'
```

Determine the range of the reference sequences that are covered by feature annotations by using the `getRange` method with the annotation object constructed in the previous section:

```
range = getRange(GFFAnnotObj)

range =

        3631      498516
```

Accessing Data in an Annotation Object

Creating a Structure of the Annotation Data

Creating a structure of the annotation data lets you access the field values. Use the `getData` method to create a structure containing a subset of the data in a `GFFAnnotation` object constructed in the previous section.

```
% Extract annotations for positions 1 through 10000 of the
% reference sequence
AnnotStruct = getData(GFFAnnotObj,1,10000)

AnnotStruct =

60x1 struct array with fields:
    Reference
    Start
    Stop
    Feature
    Source
    Score
    Strand
    Frame
    Attributes
```

Accessing Field Values in the Structure

Use dot indexing to access all or specific field values in a structure.

For example, extract the start positions for all annotations:

```
Starts = AnnotStruct.Start;
```

Extract the start positions for annotations 12 through 17. Notice that you must use square brackets when indexing a range of positions:

```
Starts_12_17 = [AnnotStruct(12:17).Start]

Starts_12_17 =

   4706        5174        5174        5439        5439        5631
```

Extract the start position and the feature for the 12th annotation:

```
Start_12 = AnnotStruct(12).Start

Start_12 =

        4706

Feature_12 = AnnotStruct(12).Feature

Feature_12 =

CDS
```

Using Feature Annotations with Short-Read Sequence Data

Investigate the results of short-read sequence experiments by using `GFFAnnotation` and `GTFAnnotation` objects with `BioMap` objects. For example, you can:

- Determine counts of short-read sequences aligned to regions of a reference sequence associated with specific annotations, such as in RNA-Seq workflows.

- Find annotations within a specific range of a peak of interest in a reference sequence, such as in ChIP-Seq workflows.

Determining Annotations of Interest

1. Construct a GTFAnnotation object from a GTF- formatted file:

```
GTFAnnotObj = GTFAnnotation('hum37_2_1M.gtf');
```

2. Use the getReferenceNames method to return the names for the reference sequences for the annotation object:

```
refNames = getReferenceNames(GTFAnnotObj)

refNames =

    'chr2'
```

3. Use the getFeatureNames method to retrieve the feature names from the annotation object:

```
featureNames = getFeatureNames(GTFAnnotObj)

featureNames =

    'CDS'
    'exon'
    'start_codon'
    'stop_codon'
```

4. Use the getGeneNames method to retrieve a list of the unique gene names from the annotation object:

```
geneNames = getGeneNames(GTFAnnotObj)

geneNames =

    'uc002qvu.2'
    'uc002qvv.2'
    'uc002qvw.2'
    'uc002qvx.2'
```

```
           'uc002qvy.2'
           'uc002qvz.2'
           'uc002qwa.2'
           'uc002qwb.2'
           'uc002qwc.1'
           'uc002qwd.2'
           'uc002qwe.3'
           'uc002qwf.2'
           'uc002qwg.2'
           'uc002qwh.2'
           'uc002qwi.3'
           'uc002qwk.2'
           'uc002qwl.2'
           'uc002qwm.1'
           'uc002qwn.1'
           'uc002qwo.1'
           'uc002qwp.2'
           'uc002qwq.2'
           'uc010ewe.2'
           'uc010ewf.1'
           'uc010ewg.2'
           'uc010ewh.1'
           'uc010ewi.2'
           'uc010yim.1'
```

The previous steps gave us a list of available reference sequences, features, and genes associated with the available annotations. Use this information to determine annotations of interest. For instance, you might be interested only in annotations that are exons associated with the uc002qvv.2 gene on chromosome 2.

Filtering Annotations

Use the `getData` method to filter the annotations and create a structure containing only the annotations of interest, which are annotations that are exons associated with the uc002qvv.2 gene on chromosome 2.

```
AnnotStruct = getData(GTFAnnotObj,'Reference','chr2',...
                      'Feature','exon','Gene','uc002qvv.2')
```

```
AnnotStruct =

12x1 struct array with fields:
    Reference
    Start
    Stop
    Feature
    Gene
    Transcript
    Source
    Score
    Strand
    Frame
    Attributes
```

The return structure contains 12 elements, indicating there are 12 annotations that meet your filter criteria.

Extracting Position Ranges for Annotations of Interest

After filtering the data to include only annotations that are exons associated with the uc002qvv.2 gene on chromosome 2, use the Start and Stop fields to create vectors of the start and end positions for the ranges associated with the 12 annotations.

```
StartPos = [AnnotStruct.Start];
EndPos = [AnnotStruct.Stop];
```

Determining Counts of Short-Read Sequences Aligned to Annotations

Construct a `BioMap` object from a BAM-formatted file containing short-read sequence data aligned to chromosome 2.

```
BMObj3 = BioMap('ex3.bam');
```

Then use the range for the annotations of interest as input to the `getCounts` method of a `BioMap` object. This returns the counts of short reads aligned to the annotations of interest.

```
counts = getCounts(BMObj3,StartPos,EndPos,'independent', true)
counts =

        1399
           1
          54
         221
          97
         125
           0
           1
           0
          65
           9
          12
```

Visualizing and Investigating Short-Read Alignments

In this section...

When to Use the NGS Browser to Visualize and Investigate Data

The NGS Browser lets you visually verify and investigate the alignment of short-read sequences to a reference sequence, in support of analyses that measure genetic variations and gene expression. The NGS Browser lets you:

- Visualize short-read data aligned to a nucleotide reference sequence.
- Compare multiple data sets aligned against a common reference sequence.
- View coverage of different bases and regions of the reference sequence.
- Investigate quality and other details of aligned reads.
- Identify mismatches due to base calling errors or polymorphisms.
- Visualize insertions and deletions.

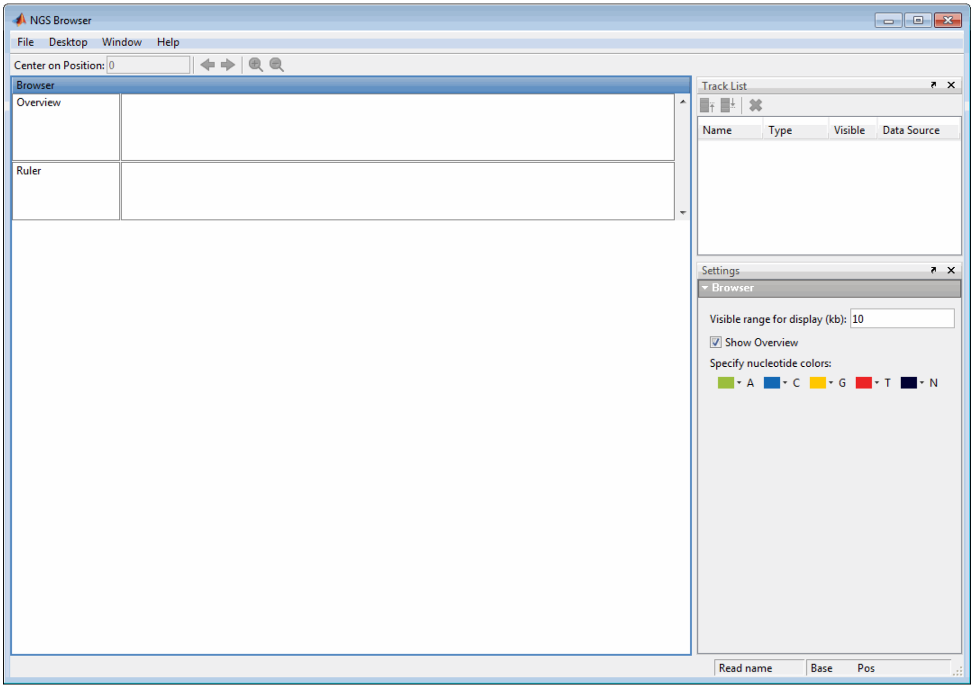
- Retrieve feature annotations relative to a specific region of the reference sequence.
- Investigate regions of interest in the alignment, determined by various analyses.

You can visualize and investigate the aligned data before, during, or after any preprocessing (filtering, quality recalibration) or analysis steps you perform on the aligned data.

Opening the NGS Browser

To open the NGS Browser, type the following in the MATLAB Command Window:

```
ngsbrowser
```

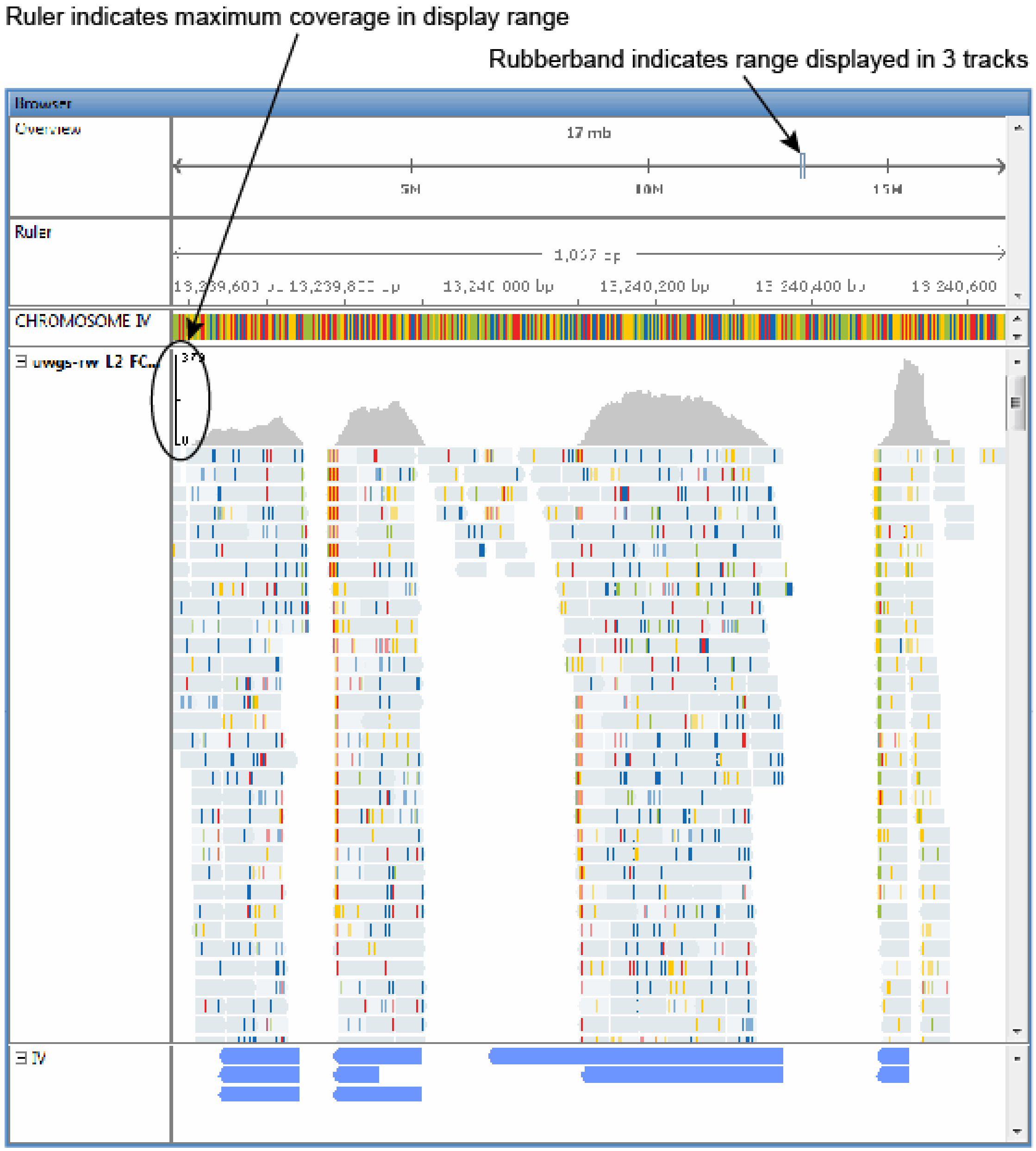


Visible range for display (kb): 10

☑ Show Overview

Specify nucleotide colors: 🟩 A 🟦 C 🟨 G 🟥 T ⬛ N

Importing Data into the NGS Browser



Ruler indicates maximum coverage in display range

Rubberband indicates range displayed in 3 tracks

Browser Displaying Reference Track, One Alignment Track, and One Annotation Track

Importing a Reference Sequence

You can import a single reference sequence into the NGS Browser. The reference sequence must be in a FASTA file.

1 Select **File > Add Data from File**.

2 In the Open dialog box, select a FASTA file, and then click **Open**.

Tip You can use the `getgenbank` function with the `ToFile` and `SequenceOnly` name-value pair arguments to retrieve a reference sequence from the GenBank database and save it to a FASTA-formatted file.

Importing Short-Read Alignment Data

You can import multiple data sets of short-read alignment data. The alignment data must be in either of the following:

- `BioMap` object

  Tip Construct a BioMap object from a SAM- or BAM-formatted file to investigate, subset, and filter the data before importing it into the NGS Browser.

- SAM- or BAM-formatted file

Note Your SAM- or BAM-formatted file must:

- Be less than 4 GB (SAM file)
- Be a size that your operating system and available memory can support (BAM file)
- Contain alignment information for a single reference sequence
- Have reads ordered by start position in the reference sequence
- Have an index file (IDX or BAI) stored in the same location as your source file, or stored in a location to which you have write access, because the browser needs to create and store an index file in this location

Tip To check if your SAM- or BAM-formatted file has alignment information for multiple reference sequences, use the `saminfo` or `baminfo` function and the `ScanDictionary` name-value pair argument. If your source file contains alignment information for multiple reference sequences, construct a BioMap object from the file, specifying one reference sequence, as described in “Constructing a BioMap Object from a SAM- or BAM-Formatted File” on page 2-12, and then import the BioMap object into the browser.

Tip MathWorks recommends using SAMtools to check if the reads in your SAM- or BAM-formatted file are ordered by position in the reference sequence, and also to reorder them, if needed.

Tip If you do not have an index file (IDX or BAI) stored in the same location as your source file, and your source file is stored in a location to which you do not have write access, construct a `BioMap` object from the source file using the `IndexDir` name-value pair argument, and then import the BioMap object into the browser.

To import short-read alignment data:

1 Select **File > Add Data from File** or **File > Import Alignment Data from MATLAB Workspace**.

2 Select a SAM-formatted file, BAM-formatted file, or BioMap object.

3 Repeat the previous steps to import additional data sets.

Importing Feature Annotations

You can import multiple sets of feature annotations from GFF- or GTF-formatted files that contain data for a single reference sequence.

1 Select **File > Add Data from File**

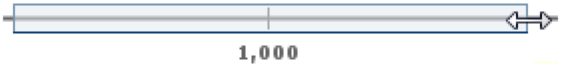
2 In the Open dialog box, select a GFF- or GTF-formatted file, and then click **Open**.

3 Repeat the previous steps to import additional annotations.

Zooming and Panning to a Specific Region of the Alignment

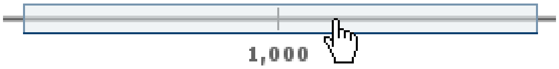
To zoom in and out:

Use the 🔍+ 🔍− toolbar buttons,
or click-drag an edge of the rubberband in the Overview area.



To pan across the alignment:

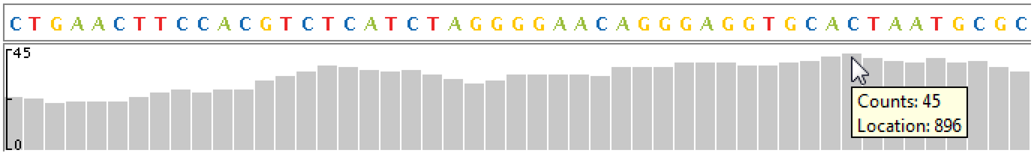
Use the ⇦ ⇨ toolbar buttons,
or click-drag the rubberband in the Overview area.



Tip Use the left and right arrow keys to pan in one base pair (bp) increments.

Viewing Coverage of the Reference Sequence

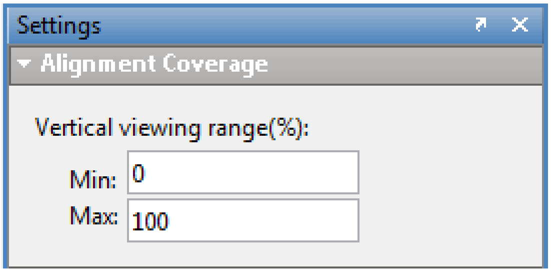
At the top of each alignment track, the *coverage view* displays the coverage of each base in the reference sequence. The vertical ruler on the left edge of the coverage view indicates the maximum coverage in the display range. Hover the mouse pointer over a position in the coverage view to display the location and counts.


C T G A A C T T C C A C G T C T C A T C T A G G G G A A C A G G G A G G T G C A C T A A T G C G C
45
0

Counts: 45
Location: 896

Note The browser computes coverage at the base pair resolution, instead of binning, even when zoomed out.

To change the percent coverage displayed, click anywhere in the alignment track, and then edit the Alignment Coverage settings.



Vertical viewing range(%):

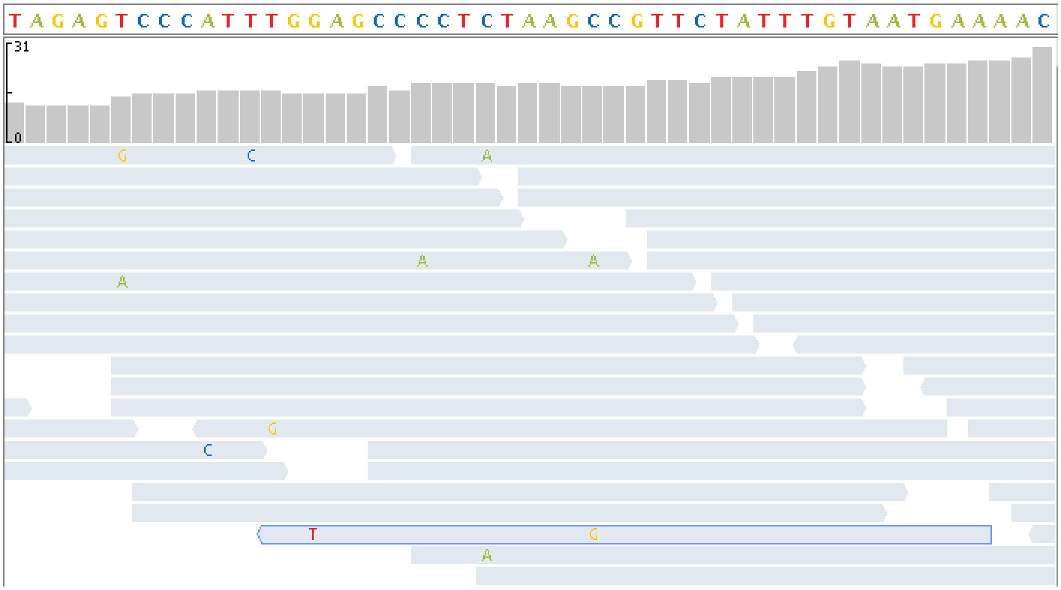
Min: 0

Max: 100

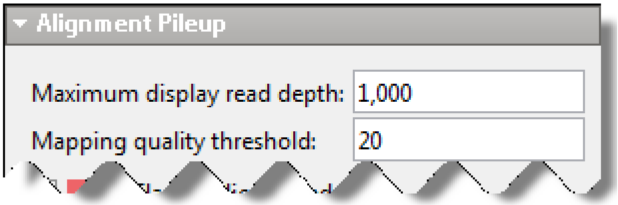
Tip Set **Max** to a value greater than 100, if needed, when comparing the coverage of multiple tracks of reads.

Viewing the Pileup View of Short Reads

Each alignment track includes a *pileup view* of the short reads aligned to the reference sequence.


T A G A G T C C C A T T T G G A G C C C C T C T A A G C C G T T C T A T T T G T A A T G A A A A C
31
0
G
C
A
A
A
A
G
C
T
G
A

Limit the depth of the reads displayed in the pileup view by setting the **Maximum display read depth** in the Alignment Pileup settings.



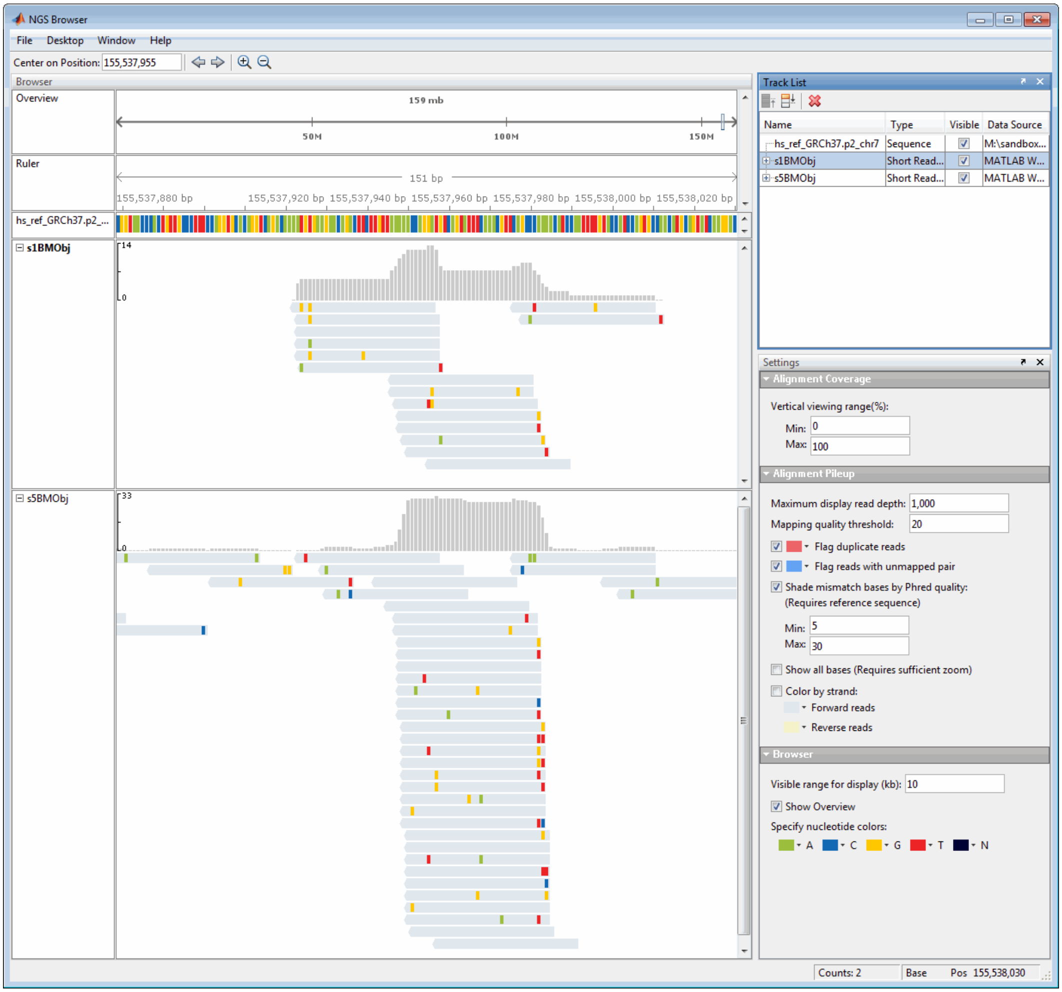
Maximum display read depth: 1,000

Mapping quality threshold: 20

Tip Limiting the depth of short reads in the pileup view does not change the counts displayed in the coverage view.

Comparing Alignments of Multiple Data Sets

Compare multiple data sets, with each data set in its own track, against a common reference sequence. Use the **Track List** to show/hide, order, and delete tracks of data.



Center on Position: 155,537,955

Name	Type	Visible	Data Source
hs_ref_GRCh37.p2_chr7	Sequence	☑	M:\sandbox...
s1BMObj	Short Read...	☑	MATLAB W...
s5BMObj	Short Read...	☑	MATLAB W...

Alignment Coverage

Vertical viewing range(%):

Min: 0
Max: 100

Alignment Pileup

Maximum display read depth: 1,000

Mapping quality threshold: 20

☑ 🟥 Flag duplicate reads

☑ 🟦 Flag reads with unmapped pair

☑ Shade mismatch bases by Phred quality:
(Requires reference sequence)

Min: 5
Max: 30

□ Show all bases (Requires sufficient zoom)

□ Color by strand:
Forward reads
Reverse reads

Browser

Visible range for display (kb): 10

☑ Show Overview

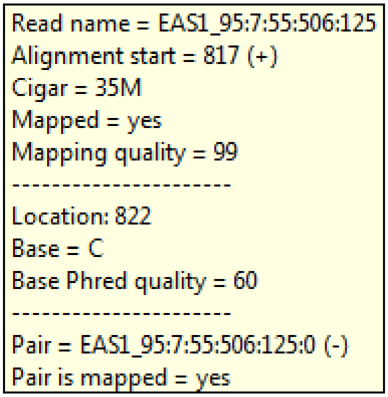
Specify nucleotide colors:

🟩 A 🟦 C 🟨 G 🟥 T ■ N

Counts: 2 | Base | Pos 155,538,030

Viewing Location, Quality Scores, and Mapping Information

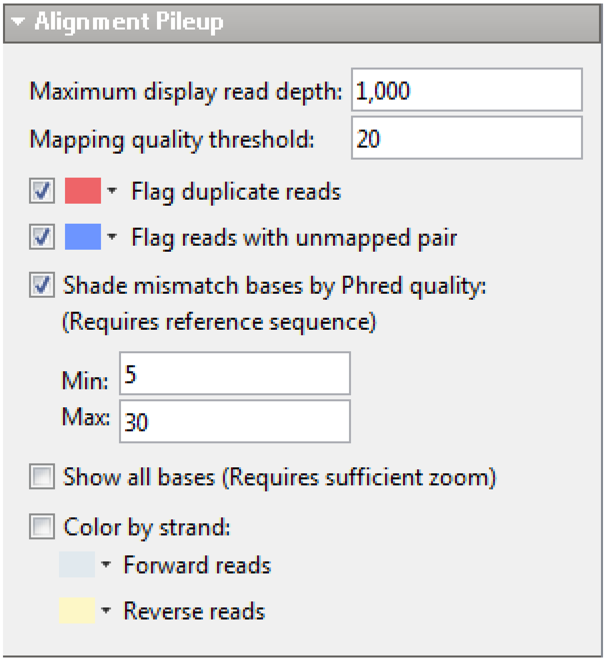
Hover the mouse pointer over a position in a read to display strand direction, location, quality, and mapping information for the base, the read, and its paired mate.



```
Read name = EAS1_95:7:55:506:125
Alignment start = 817 (+)
Cigar = 35M
Mapped = yes
Mapping quality = 99
----------------------
Location: 822
Base = C
Base Phred quality = 60
----------------------
Pair = EAS1_95:7:55:506:125:0 (-)
Pair is mapped = yes
```

Flagging Reads

Click anywhere in an alignment track to display the Alignment Pileup settings.



Alignment Pileup

Maximum display read depth: 1,000

Mapping quality threshold: 20

☑ 🟥 Flag duplicate reads

☑ 🟦 Flag reads with unmapped pair

☑ Shade mismatch bases by Phred quality:
(Requires reference sequence)

Min: 5
Max: 30

□ Show all bases (Requires sufficient zoom)

□ Color by strand:

⬜ Forward reads

🟨 Reverse reads

Flagging Reads with Low Mapping Quality

Set the **Mapping quality threshold** in the Alignment Pileup section to flag low-quality reads. Reads with a mapping quality below this level appear in a lighter shade of gray.

Flagging Duplicate Reads

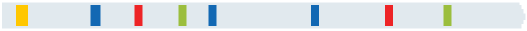
Select **Flag duplicate reads** and select an outline color.

Flagging Reads with Unmapped Pairs

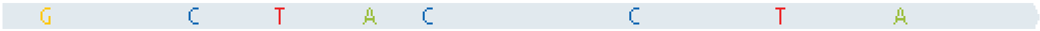
Select **Flag reads with unmapped pair** and select an outline color.

Evaluating and Flagging Mismatches

Mismatches display as colored blocks or letters, depending on the zoom level.

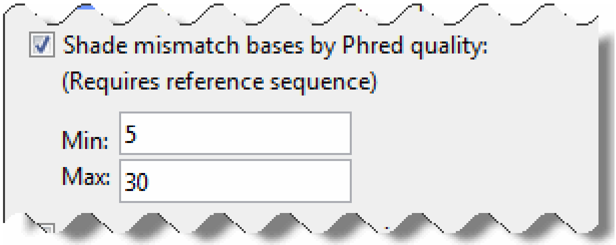


Zoomed out view of read — Mismatches display as bars



Zoomed in view of read — Mismatches display as letters

In addition to the base Phred quality information that displays in the tooltip, you can visualize quality differences by using the **Shade mismatch bases by Phred quality** settings.



☑ Shade mismatch bases by Phred quality:
(Requires reference sequence)

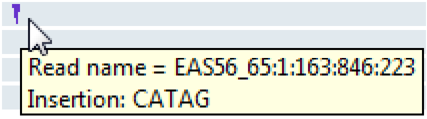
Min: 5
Max: 30

The mismatch blocks or letters display in:

- Light shade — Mismatch bases with Phred scores below the minimum
- Graduation of medium shades — Mismatch bases with Phred scores within the minimum to maximum range
- Dark shade — Mismatch bases with Phred scores above the maximum

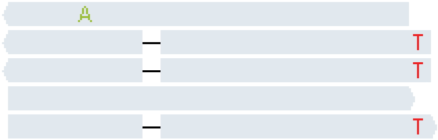
Viewing Insertions and Deletions

The NGS Browser designates insertions with a symbol. Hover the mouse pointer over the insertion symbol to display information about it.



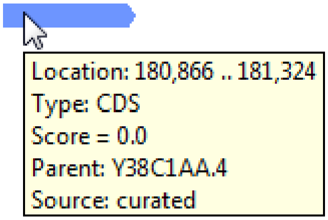
Read name = EAS56_65:1:163:846:223
Insertion: CATAG

The NGS Browser designates deletions with dashes.



Viewing Feature Annotations

After importing a feature annotation file, you can zoom and pan to view feature annotations associated with a region of interest in the alignment. Hover the mouse pointer over the feature annotation.



Location: 180,866 .. 181,324
Type: CDS
Score = 0.0
Parent: Y38C1AA.4
Source: curated

Printing and Exporting the Browser Image

Print or export the browser image by selecting **File > Print Image** or **File > Export Image**.

Identifying Differentially Expressed Genes from RNA-Seq Data

This example loads RNA-seq data and tests for differential expression using a statistical model. It uses MATLAB, Bioinformatics Toolbox™, and Statistics Toolbox™ functions.

Introduction

RNA-seq is an emerging technology for surveying gene expression and transcriptome content by directly sequencing the mRNA molecules in a sample. RNA-seq can provide gene expression measurements and is regarded as an attractive approach to analyze a transcriptome in an unbiased and comprehensive manner.

In this example, you will used Bioinformatics Toolbox and Statistics Toolbox functions to load publicly available transcriptional profiling sequencing data into MATLAB, compute the digital gene expression, and then identify differentially expressed genes in RNA-seq data from hormone treated prostate cancer cell line samples [1].

The Prostate Cancer Data Set

In the prostate cancer study, the prostate cancer cell line LNCap was treated with androgen/DHT. Mock-treated and androgen-stimulated LNCap cells were sequenced using the Illumina 1G Genome Analyzer [1]. For the mock-treated cells, there were four lanes totaling ~10 million reads. For the DHT-treated cells, there were three lanes totaling ~7 million reads. All replicates were technical replicates. Samples labeled s1 through s4 are from mock-treated cells. Samples labeled s5, s6, and s8 are from DHT-treated cells. The read sequences are stored in FASTA files. The sequence IDs break down as follows: seq_(unique sequence id)_(number of times this sequence was seen in this lane).

This example assumes that you:

(1) Downloaded and uncompressed the seven FASTA files (`s1.fa`, `s2.fa`, `s3.fa`, `s4.fa`, `s5.fa`, `s6.fa` and `s8.fa`) containing the raw, 35bp, unmapped short reads from the author’s Web Site.

(2) Produced a SAM-formatted file for each of the seven FASTA files by mapping the short reads to the NCBI version 37 of the human genome using a mapper such as Bowtie [2],

(3) Ordered the SAM-formatted files by reference name first, then by genomic position.

For the published version of this example, 4,388,997 short reads were mapped using the Bowtie aligner [2]. The aligner was instructed to report one best valid alignment. No more than two mismatches were allowed for alignment. Reads with more than one reportable alignment were suppressed, i.e. any read that mapped to multiple locations was discarded. The alignment was output to seven SAM files (s1.sam, s2.sam, s3.sam, s4.sam, s5.sam, s6.sam and s8.sam). Because the input files were FASTA files, all quality values were assumed to be 40 on the Phred quality scale [2]. We then used SAMtools [3] to sort the mapped reads in the seven SAM files, one for each replicate.

Creating a Gene Annotation File of Target Genes

Using the Homo sapiens genes data set from Ensembl Genes 59 annotation database from BioMart, we downloaded the annotations for the protein coding genes to a text file, rnaseqdedemo_ensembl_genes.txt. The annotation file contains Ensembl IDs for 20,024 target genes, their chromosome numbers, start and end positions on the chromosomes, and strand direction.

Load the annotation file into a DataMatrix and display the first 10 entries.

```
ensemble_genes = bioma.data.DataMatrix('file', 'rnaseqdedemo_ensembl_gene

ensemble_genes(1:10, :)


ans = 

                       Chromosome Name    Gene Start (bp)    Gene End (bp
    ENSG00000237613    1                            34554              36081
    ENSG00000177693    1                            69055              70108
    ENSG00000235249    1                      3.6764e+005        3.6863e+005
    ENSG00000185097    1                       6.211e+005        6.2203e+005
    ENSG00000197049    1                      7.2132e+005        7.2251e+005
    ENSG00000187634    1                   8.6026e+005         8.7996e+005
    ENSG00000188976    1                   8.7958e+005         8.9469e+005
    ENSG00000187961    1                   8.9597e+005          9.011e+005
    ENSG00000187583    1                   9.0188e+005         9.1125e+005
    ENSG00000187642    1                   9.1058e+005          9.175e+005
```

Importing Mapped Short Read Alignment Data

The size of the sorted SAM files in this data set are in the order of 250-360MB. Use the function `saminfo` to get information about a SAM file, including a list of all the reference names present in the file. For example, to get information about the `s1.sam` file.

```
s1_sam_info = saminfo('s1.sam', 'ScanDictionary', true)


s1_sam_info =

                   Filename: 's1.sam'
                   FilePath: 'C:\RNA-Seq\examples'
                   FileSize: 272376614
                FileModDate: '19-Oct-2010 22:44:22'
                   NumReads: []
          ScannedDictionary: {26x1 cell}
     ScannedDictionaryCount: [26x1 uint64]
```

Display a list of all the reference names present in `s1.sam`.

```
s1_sam_info.ScannedDictionary


ans =

    'gi|224589800|ref|NC_000001.10|'
    'gi|224589811|ref|NC_000002.11|'
    'gi|224589815|ref|NC_000003.11|'
    'gi|224589816|ref|NC_000004.11|'
    'gi|224589817|ref|NC_000005.9|'
    'gi|224589818|ref|NC_000006.11|'
    'gi|224589819|ref|NC_000007.13|'
    'gi|224589820|ref|NC_000008.10|'
    'gi|224589821|ref|NC_000009.11|'
    'gi|224589801|ref|NC_000010.10|'
    'gi|224589802|ref|NC_000011.9|'
    'gi|224589803|ref|NC_000012.11|'
    'gi|224589804|ref|NC_000013.10|'
    'gi|224589805|ref|NC_000014.8|'
    'gi|224589806|ref|NC_000015.9|'
    'gi|224589807|ref|NC_000016.9|'
    'gi|224589808|ref|NC_000017.10|'
    'gi|224589809|ref|NC_000018.9|'
    'gi|224589810|ref|NC_000019.9|'
    'gi|224589812|ref|NC_000020.10|'
    'gi|224589813|ref|NC_000021.8|'
    'gi|224589814|ref|NC_000022.10|'
    'gi|224589822|ref|NC_000023.10|'
    'gi|224589823|ref|NC_000024.9|'
    'gi|17981852|ref|NC_001807.4|'
    'Unmapped'
```

The SAM file s1.sam contains 25 reference names. This is because s1.sam contains reads mapped to all 24 chromosomes and one mitochodrial chromosome in the human genome.

The BioMap class in Bioinformatics Toolbox represents short read alignment information. You can access the mapped reads in s1.sam by creating a BioMap using the BioIndexedFile s1_BioIndexedFile as the interface to the actual data. For example, create a BioMap to access the short reads mapped to chromosome 1 of sample s1.

```
s1_chr1_BioMap = BioMap('s1.sam',...
                        'SelectReference', 'gi|224589800|ref|NC_000001.1


s1_chr1_BioMap =

  BioMap
```

```
Properties:
       Reference: 'gi|224589800|ref|NC_000001.10|'
       Signature: [39037x1 File indexed property]
           Start: [39037x1 File indexed property]
  MappingQuality: [39037x1 File indexed property]
            Flag: [39037x1 File indexed property]
    MatePosition: [39037x1 File indexed property]
         Quality: [39037x1 File indexed property]
        Sequence: [39037x1 File indexed property]
          Header: [39037x1 File indexed property]
           NSeqs: 39037
            Name: ''
```

You can access the alignments, and perform operations like getting counts and coverage from s1_chr1_BioMap. For more examples of getting read coverage at the chromosome level, see Exploring Protein-DNA Binding Sites from Paired-End ChIP-Seq Data.

Determining Digital Gene Expression

Next, you will determine the mapped reads associated with each Ensembl gene in chromosome 1. For each gene, count the mapped reads that overlap any part of the gene. The read counts for each gene are the digital gene expression of that gene. Use the getCounts method of a BioMap to compute the read count within a specified range.

Get the Ensembl genes in chromosome 1:

```
chr1_genes = ensemble_genes(ensemble_genes(:, 1) == 1, :);
length(chr1_genes)


ans =

        2016
```

There are 2,016 genes in chromosome 1. Get the number of reads mapped to the genes annotated on chromosome 1.

```
s1_chr1_counts = getCounts(s1_chr1_BioMap, chr1_genes.(':')(2),...
                                            chr1_genes.(':')(3),...
                                            'Independent', true);
```

Determine the number of genes that have counts greater than or equal to 50.

```
sum(s1_chr1_counts >= 50)


ans =

   184
```

The helper function `getGeneCounts` captures the previous steps and returns a vector of gene counts for the 24 chromosomes in the SAM file `s1.sam`.

```
s1_gene_counts = getGeneCounts('s1.sam', ensemble_genes, 1:24);
```

Repeat this step for the other six samples (SAM files) in the data set to get their gene counts.

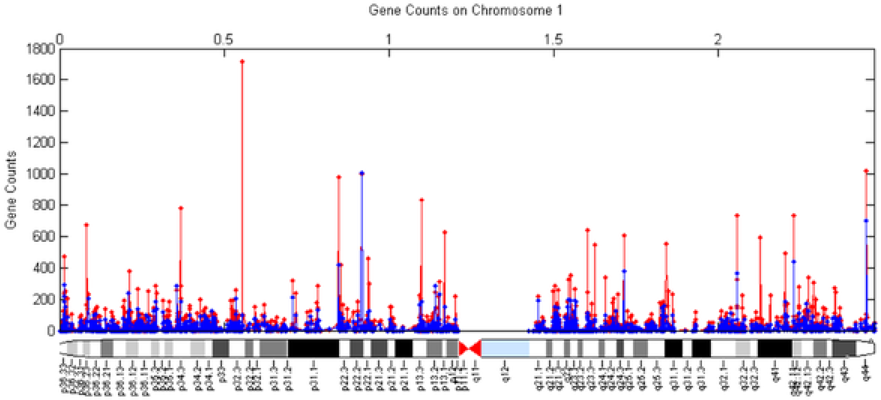
```
s2_gene_counts = getGeneCounts('s2.sam', ensemble_genes, 1:24);
s3_gene_counts = getGeneCounts('s3.sam', ensemble_genes, 1:24);
s4_gene_counts = getGeneCounts('s4.sam', ensemble_genes, 1:24);

s5_gene_counts = getGeneCounts('s5.sam', ensemble_genes, 1:24);
s6_gene_counts = getGeneCounts('s6.sam', ensemble_genes, 1:24);
s8_gene_counts = getGeneCounts('s8.sam', ensemble_genes, 1:24);
```

You can plot the counts for a chromosome along the chromosome genome coordinate. For example, plot the counts for chromosome 1 for mock-treated sample `s1` and DHT-treated sample `s5`. First, get the start positions of the genes on chromosome 1 and their counts.

```
x = ensemble_genes.(ensemble_genes(:, 1) == 1)(2);
chr1_counts_mock = s1_gene_counts(ensemble_genes(:, 1) == 1);
chr1_counts_dht  = s5_gene_counts(ensemble_genes(:, 1) == 1);
```

Plot the counts on chromosome 1. Add the ideogram for chromosome 1 to the plot using the `chromosomeplot` function of Bioinformatics Toolbox.

```
[x, sidx]= sort(x);
figure
plot(x, chr1_counts_dht(sidx),  '.-r',...
     x, chr1_counts_mock(sidx), '.-b')
ylabel('Gene Counts')
title('Gene Counts on Chromosome 1')
chromosomeplot('hs_cytoBand.txt', 1, 'AddToPlot', gca)
set(gcf,'Position',max(get(gcf,'Position'),[0 0 1000 0]))
```



Store the gene counts of all the samples as a `DataMatrix` with Ensembl gene IDs as the row names.

```
lncap_counts = bioma.data.DataMatrix([s1_gene_counts,...
                                      s2_gene_counts ,...
                                      s3_gene_counts ,...
                                      s4_gene_counts ,...
                                      s5_gene_counts,...
                                      s6_gene_counts,...
                                      s8_gene_counts],...
                                      rownames(ensemble_genes),...
                                      {'Mock_1', 'Mock_2', 'Mock_3', 'Moc
```

```
                                        'DHT_1', 'DHT_2', 'DHT_3'});
```

Inspect the first 10 rows in the count table.

```
lncap_counts(1:10, :)


ans = 

                          Mock_1    Mock_2    Mock_3    Mock_4    DHT_1    D
    ENSG00000237613        0         0         0         0         0
    ENSG00000177693        0         0         0         0         0
    ENSG00000235249        0         0         0         0         0
    ENSG00000185097        0         0         0         0         0
    ENSG00000197049        1         0         1         1         0
    ENSG00000187634        2         5        11         7         5       1
    ENSG00000188976       33        49        70        75        58       5
    ENSG00000187961        0         0         0         0         0
    ENSG00000187583       21        16        31        25        18       2
    ENSG00000187642        5         3         7         5         4
```

The DataMatrix `lncap_counts` contains counts for samples from two biological conditions: mock-treated and DHT-treated.

```
cond_Mock = logical([1 1 1 1 0 0 0]);
cond_DHT  = logical([0 0 0 0 1 1 1]);
```

Inference of Differential Signal in RNA Expression

For RNA-seq experiments, the read counts have been found to be linearly related to the abundance of the target transcripts [4]. The interest lies in comparing the read counts between different biological conditions. Current observations suggest that typical RNA-seq experiments have low background noise, and the gene counts are discrete and could follow the Poisson distribution. While it has been noted that the assumption of the Poisson distribution often predicts smaller variation in count data by ignoring the extra variation due to the actual differences between replicate samples [5]. Anders *et.al.*,(2010) proposed an error model for statistical inference of differential signal in RNA-seq expression data that could address the overdispersion problem [6]. Their approach uses the negative binomial distribution to model the null distribution of the read counts. The mean and variance of the negative binomial distribution are linked by local regression, and these two parameters can be reliably estimated even when the number of replicates is small [6].

In this example, you will apply this statistical model to process the count data and test for differential expression. The details of the algorithm can be found in reference [6]. The model of Anders *et.al.*, (2010) has three sets of parameters that need to be estimated from the data:

1. Library size parameters;

2. Gene abundance parameters under each experimental condition;

3. The smooth functions that model the dependence of the raw variance on the expected mean.

Estimating Library Size Factor

The expectation values of all gene counts from a sample are proportional to the sample's library size. The effective library size can be estimated from the count data.

Compute the geometric mean of the gene counts (rows in `lncap_counts`) across all samples in the experiment as a pseudo-reference sample.

```
geoMeans = exp(mean(log(lncap_counts), 2));
```

Each library size parameter is computed as the median of the ratio of the sample's counts to those of the pseudo-reference sample.

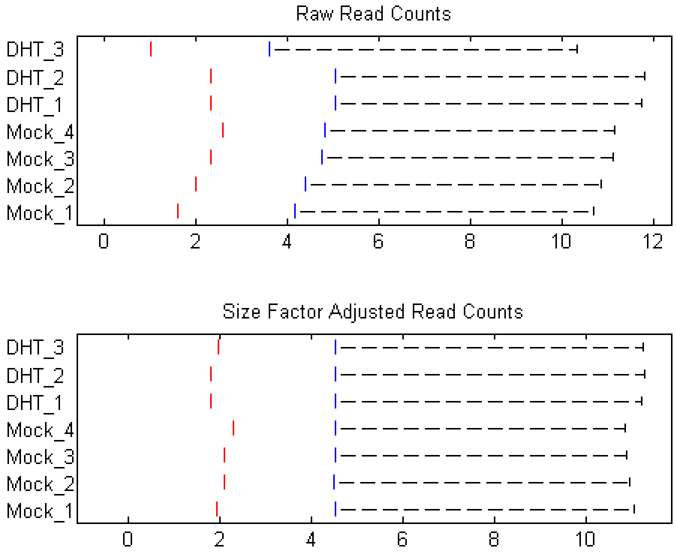
```
ratios = dmbsxfun(@rdivide, lncap_counts(geoMeans >0, :), geoMeans(geoMea
sizeFactors = median(ratios, 1);
```

The counts can be transformed to a common scale using size factor adjustment.

```
base_counts = dmbsxfun(@rdivide, lncap_counts, sizeFactors);
```

Use the `boxplot` function to inspect the count distribution of the mock-treated and DHT-treated samples and the size factor adjustment.

```
figure
subplot(2,1,1)
maboxplot(log2(lncap_counts), 'title','Raw Read Counts',...
                              'orientation', 'horizontal')
subplot(2,1,2)
maboxplot(log2(base_counts), 'title','Size Factor Adjusted Read Counts',
                             'orientation', 'horizontal')
```



Estimating Negative Binomial Distribution Parameters

The expectation value of counts for a gene is also proportional to the gene abundance parameter. You can estimate the gene abundance parameter from the average of counts from samples corresponding to an experimental condition. For example, compute the mean counts and sample variances from mock-treated samples.

```
base_mean_mock = mean(base_counts(:, cond_Mock), 2);
base_var_mock = var(base_counts(:, cond_Mock), 0, 2);
```

To avoid code duplication in the example for computing parameters for samples of different conditions, we provide a helper function, `estimateBaseParams`, to compute the mean, the variance, the smooth function fit data for raw variance estimation, and the diagnostic variance residual distribution from replicates under the same condition. For example, compute the base means and variances for DHT-treated samples.

```
[base_mean_dht, base_var_dht] = estimateBaseParams(lncap_counts(:, cond_D
                                                   sizeFactors(cond_DHT),
                                                   'MeanAndVar');
```

In the model, the full variances of the negative binomial distribution of the counts of a gene are considered as the sum of a shot noise term and a raw variance term. The shot noise term is the read counts of the gene, while the raw variance can be predicted from the mean, i.e., genes with a similar expression level have similar variance across the replicates (samples of the same biological condition). A smooth function that models the dependence of the raw variance on the mean is obtained by fitting the sample mean and variance within replicates for each gene using the local regression function `malowess`. For example, get the smooth fit data from the sample mean and variance of the mock-treated samples.

```
[rawVarSmooth_X_mock, rawVarSmooth_Y_mock] = ...
                              estimateBaseParams(lncap_counts(:, cond_M
                                                 sizeFactors(cond_Mock)
                                                 'SmoothFunc');
```

Find the raw variances for each gene from its base mean value by interpolation.

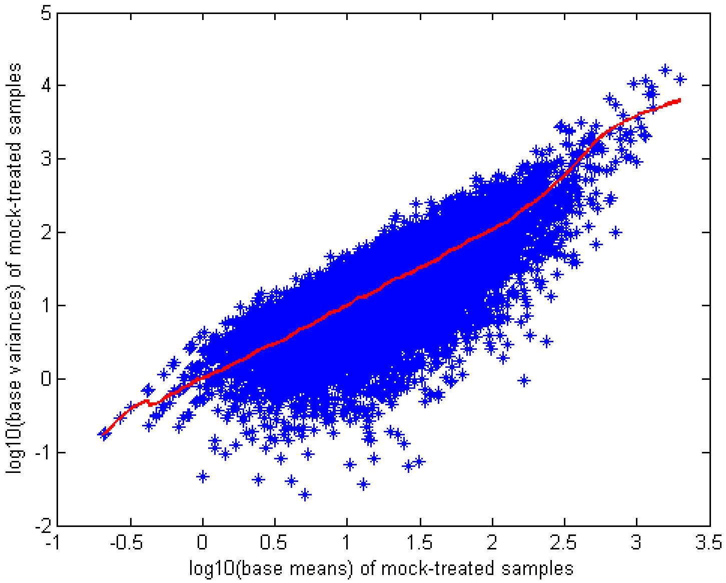
```
raw_var_mock_fit = interp1(rawVarSmooth_X_mock, rawVarSmooth_Y_mock,...
                           log(base_mean_mock), 'linear', 'extrap');
```

Add the bias correction term [6] to get the raw variances.

```
zConst = sum(1 ./sizeFactors(cond_Mock), 2) / length(sizeFactors(cond_Moc
raw_var_mock = raw_var_mock_fit - base_mean_mock * zConst;
```

Plot the sample variance and the raw variance data to check the fit of the variance function.

```
[base_mean_mock_sort, sidx] = sort(log10(base_mean_mock));
raw_var_mock_sort = log10(raw_var_mock_fit(sidx));

figure
plot(log10(base_mean_mock), log10(base_var_mock), '*')
hold on
line(base_mean_mock_sort, real(raw_var_mock_sort), 'Color', 'r', 'LineWid
ylabel('log10(base variances) of mock-treated samples')
xlabel('log10(base means) of mock-treated samples')
```



The fit (red line) follows the single-gene estimates well, even though the spread of the latter is considerable, as one would expect, given that each raw variance value is estimated from only four values (four mock-treaded replicates).

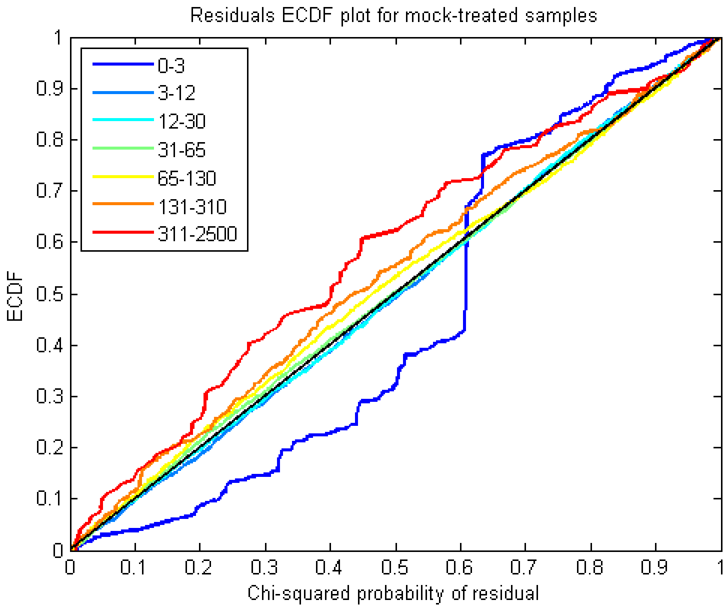
As RNA-seq experiments typically have few replicates, the single-gene estimate of the base variance can deviate wildly from the fitted value. To see whether this might be too wild, the cumulative probability for the ratio of single-gene estimate of the base variance to the fitted value is calculated from the chi-square distribution, as explained in reference [6].

Compute the cumulative probabilities of the variance ratios of mock-treated samples.

```
df_mock = sum(cond_Mock) - 1;
varRatio_mock = base_var_mock ./ raw_var_mock_fit;
pchisq_mock = chi2cdf(df_mock * varRatio_mock, df_mock);
```

Compute the empirical cumulative density functions (ECDF) stratified by base count levels, and show the ECDFs curves. Group the counts into seven levels.

```
count_levels = [0 3; 3.1 12; 12.1 30; 30.1 65; 65.1 130; 130.1 310; 310.

figure;
hold on
cm = jet(7);
for i = 1:7
   [Y1,X1] = ecdf(pchisq_mock(base_mean_mock>count_levels(i, 1) &...
                               base_mean_mock<count_levels(i,2)));
   plot(X1,Y1,'LineWidth',2,'color',cm(i,:))
end
plot([0,1],[0,1] ,'k', 'linewidth', 2)
set(gca, 'Box', 'on')
legend('0-3', '3-12', '12-30', '31-65', '65-130', '131-310', '311-2500',
       'Location','NorthWest')
xlabel('Chi-squared probability of residual')
ylabel('ECDF')
title('Residuals ECDF plot for mock-treated samples')
```



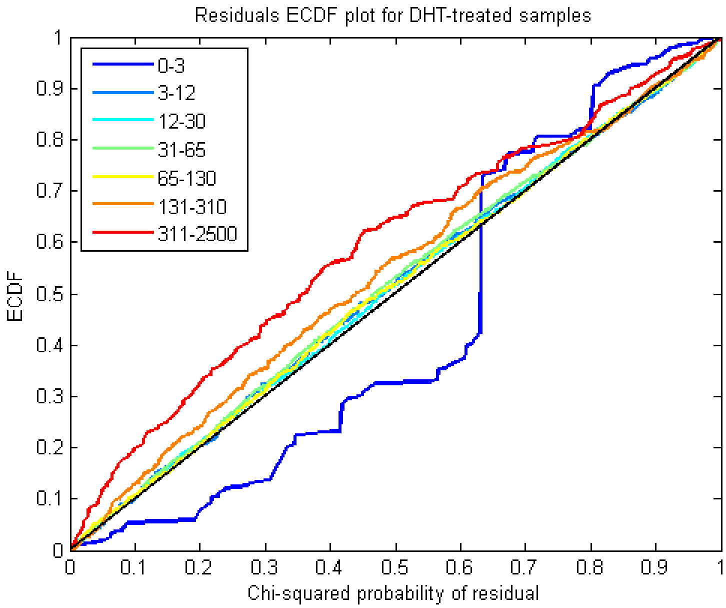
The ECDF curves of count levels greater than 3 and below 130 follows the diagonal well (black line). If the ECDF curves are below the black line, variance is underestimated. If the ECDF curves are above the black line, variance is overestimated [6]. For very low counts (below 3), the deviations become stronger, but at these levels, shot noise dominates. For the high count cases, the variance is overestimated. The reason might be there aren't enough genes with high counts. Get the number of genes in each of the count levels.

```
num_in_count_levels = zeros(1, 7);
for i = 1:7
    num_in_count_levels(i) = sum(base_mean_mock>count_levels(i, 1) & ...
                                 base_mean_mock<count_levels(i,2));
end

num_in_count_levels


num_in_count_levels =
        4048         3355         3405         2354         1132         408
```

Increasing the sequence depth, which in turn increases the number of genes with higher counts, improves the variance estimation.

You can produce the same ECDF plot for the DHT-treated samples by following the same steps.

```
pchisq_dht = estimateBaseParams(lncap_counts(:, cond_DHT),...
                                                sizeFactors(1, cond_DHT)
                                                'Diagnostic');

figure;
hold on
for i = 1:7
   [Y1,X1] = ecdf(pchisq_dht(base_mean_dht>count_levels(i, 1) & ...
                             base_mean_dht<count_levels(i,2)));
   plot(X1,Y1,'LineWidth',2,'color',cm(i,:))
end
plot([0,1],[0,1] ,'k', 'linewidth', 2)
set(gca, 'Box', 'on')
legend('0-3', '3-12', '12-30', '31-65', '65-130', '131-310', '311-2500',
       'Location','NorthWest')
xlabel('Chi-squared probability of residual')
ylabel('ECDF')
title('Residuals ECDF plot for DHT-treated samples')
```



In both cases, most of the ECDF curves follow the diagonal well. The fits are reasonably good.

Testing for Differential Expression

Having estimated and verified the mean-variance dependence, you can test for differentially expressed genes between the samples from the mock- and DHT- treated conditions. For your convenience, we provide the helper function `estimateNBParams` to estimate the mean and full variance of the two-parametric negative binomial distribution for each gene from the three sets of parameters discussed above.

```
[mu_mock, full_var_mock, mu_dht, full_var_dht] =...
        estimateNBParams(lncap_counts, sizeFactors, cond_DHT, cond_Mock
```

Compute the p-values for the statistical significance of the change from DHT-treated condition to mock-treated condition. The helper function `computePVal` implements the numerical computation of the p-values presented in the reference [6]. We use the `nbinpdf` function to compute the negative binomial probability density. Note: The computation was not optimized, and it will take several minutes to run.

Get the gene counts for each condition:

```
k_mock = sum(lncap_counts(:, cond_Mock), 2);
k_dht = sum(lncap_counts(:, cond_DHT), 2);

pvals =  computePVal(k_dht, mu_dht, full_var_dht,...
                          k_mock, mu_mock, full_var_mock);
```

You can empirically adjust the p-values from the multiple tests for false discovery rate (FDR) with the Benjamini-Hochberg procedure [7] using the `mafdr` function in Bioinformatics Toolbox.

```
p_fdr = mafdr(pvals, 'BHFDR', true);
```

Determine the fold change estimated from the DHT-treated to the mock-treated condition.

```
foldChange = base_mean_dht ./ base_mean_mock;
```

Determine the base 2 logarithm of the fold change.

```
log2FoldChange = log2(foldChange);
```

Determine the mean expression level estimated from both conditions.

```
base_mean_com = estimateBaseParams(lncap_counts, sizeFactors, 'MeanAndVar
```

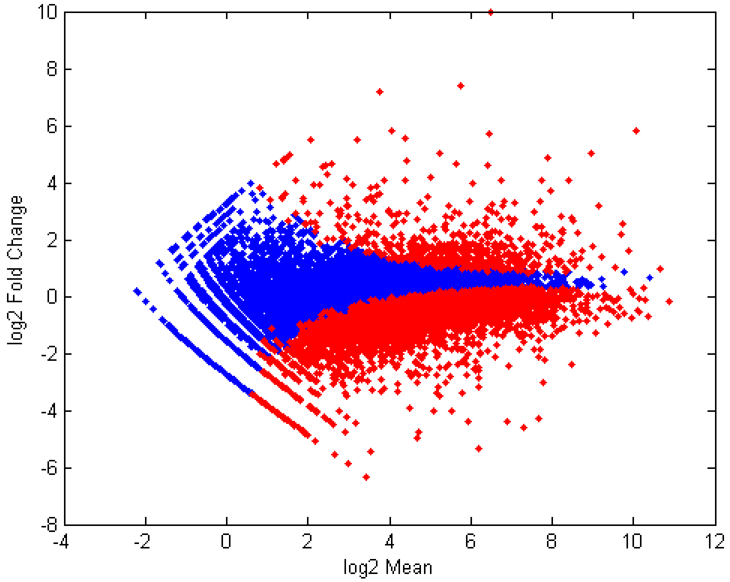
Assume a p-value cutoff of 0.01.

```
de_idx = p_fdr < 0.01;
```

Plot the log2 fold changes against the base means, and color those genes with p-values less than the cutoff value red.

```
figure;
plot(log2(base_mean_com(~de_idx, :)), log2FoldChange(~de_idx,:), 'b.')
hold on
plot(log2(base_mean_com(de_idx, :)), log2FoldChange(de_idx, :), 'r.')
```

```
xlabel('log2 Mean')
ylabel('log2 Fold Change')
```



You can identify up- or down- regulated genes for mean base count levels over 3.

```
up_idx = find(p_fdr < 0.01 & log2FoldChange >= 2 & base_mean_com > 3 );
numel(up_idx)


ans =

   179


down_idx = find(p_fdr < 0.01 & log2FoldChange <= -2 & base_mean_com > 3 )
numel(down_idx)
```

```
ans =

   284
```

This analysis identified 461 genes (out of 20,024 genes) that were differentially up- or down- regulated by hormone treatment.

References

[1] Li, H., Lovci, M.T., Kwon, Y-S., Rosenfeld, M.G., Fu, X-D., and Yeo, G.W. (2008). Determination of Tag Density Required for Digital Transcriptome Analysis: Application to an Androgen-Sensitive Prostate Cancer Model. PNAS, 105(51), 20179-20184.

[2] Langmead, B., Trapnell, C., Pop, M., and Salzberg, S.L. (2009). Ultrafast and Memory-efficient Alignment of Short DNA Sequences to the Human Genome. Genome Biol, 10:R25, 1-10.

[3] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R. and 1000 Genome Project Data Processing Subgroup (2009). The Sequence Alignment/map (SAM) Format and SAMtools. Bioinformatics, 25, 2078-2079.

[4] Mortazavi, A., Williams, B.A., McCue, K., Schaeffer, L., and Wold, B. (2008). Mapping and quantifying mammalian transcriptomes by RNA-Seq. Nature Methods, 5, 621-628.

[5] Robinson, M.D., and Oshlack, A. (2010). A Scaling Normalization method for differential Expression Analysis of RNA-seq Data. Genome Biol, 11:R25, 1-9.

[6] Anders, S. and Huber W. (2010). Differential Expression Analysis for Sequence Count Data. Genome Biology, 11:R106.

[7] Benjamini, Y., and Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. J. Royal Stat. Soc., B 57, 289-300.

Suggest an enhancement for this example.

Exploring Protein-DNA Binding Sites from Paired-End ChIP-Seq Data

This example performs a genome-wide analysis of a transcription factor in the *Arabidopsis Thaliana* (Thale Cress) model organism.

For enhanced performance, it is recommended that you run this example on a 64-bit platform, because the memory footprint is close to 2 Gb. On a 32-bit platform, if you receive `"Out of memory"` errors when running this example, try increasing the virtual memory (or swap space) of your operating system or try setting the `3GB` switch (32-bit Windows XP only). These techniques are described in this document.

Introduction

ChIP-Seq is a technology that is used to identify transcription factors that interact with specific DNA sites. First chromatin immunoprecipitation enriches DNA-protein complexes using an antibody that binds to a particular protein of interest. Then, all the resulting fragments are processed using high-throughput sequencing. Sequencing fragments are mapped back to the reference genome. By inspecting over-represented regions it is possible to mark the genomic location of DNA-protein interactions.

In this example, short reads are produced by the paired-end Illumina platform. Each fragment is reconstructed from two short reads successfully mapped, with this the exact length of the fragment can be computed. Using paired-end information from sequence reads maximizes the accuracy of predicting DNA-protein binding sites.

Data Set

This example explores the paired-end ChIP-Seq data generated by Wang *et.al.* [1] using the Illumina platform. The data set has been courteously submitted to the Gene Expression Omnibus repository with accession number GSM424618. The unmapped paired-end reads can be obtained from the NCBI FTP site.

This example assumes that you:

(1) downloaded the file SRR054715.sra containing the unmapped short read and converted it to FASTQ formatted files using the NCBI SRA Toolkit.

(2) produced a SAM formatted file by mapping the short reads to the Thale Cress reference genome, using a mapper such as BWA [2], Bowtie, or SSAHA2 (which is the mapper used by authors of [1]), and,

(3) ordered the SAM formatted file by reference name first, then by genomic position.

For the published version of this example, 8,655,859 paired-end short reads are mapped using the BWA mapper [2]. BWA produced a SAM formatted file (aratha.sam) with 17,311,718 records (8,655,859 x 2). Repetitive hits were randomly chosen, and only one hit is reported, but with lower mapping quality. The SAM file was ordered and converted to a BAM formatted file using SAMtools [3] before being loaded into MATLAB.

The last part of the example also assumes that you downloaded the reference genome for the Thale Cress model organism (which includes five chromosomes). Uncomment the following lines of code to download the reference from the NCBI repository:

```
% getgenbank('NC_003070','FileFormat','fasta','tofile','ach1.fasta');
% getgenbank('NC_003071','FileFormat','fasta','tofile','ach2.fasta');
% getgenbank('NC_003074','FileFormat','fasta','tofile','ach3.fasta');
% getgenbank('NC_003075','FileFormat','fasta','tofile','ach4.fasta');
% getgenbank('NC_003076','FileFormat','fasta','tofile','ach5.fasta');
```

Creating a MATLAB Interface to a BAM Formatted File

To create local alignments and look at the coverage we need to construct a BioMap. BioMap has an interface that provides direct access to the mapped short reads stored in the BAM formatted file, thus minimizing the amount of data that is actually loaded to the workspace. Use the function baminfo to obtain a list of the existing references and the actual number of short read mapped to each one.

```
info = baminfo('aratha.bam','ScanDictionary',true);

infoStr = sprintf('Reference    Number of Reads\n');
```

```
for i = 1:numel(info.ScannedDictionary)
    infoStr = sprintf('%s%8s\t\t%d \n',infoStr,...
        info.ScannedDictionary{i},info.ScannedDictionaryCount(i));
end
disp(infoStr)

Reference    Number of Reads
    Chr1  3151847
    Chr2  3080417
    Chr3  3062917
    Chr4  2218868
    Chr5  3123275
Unmapped  2674394
```

The remainder of this example focuses on the analysis of one of the five chromosomes, Chr1. Create a BioMap to access the short reads mapped to the first chromosome.

```
bm1 = BioMap('aratha.bam','SelectReference','Chr1');
```

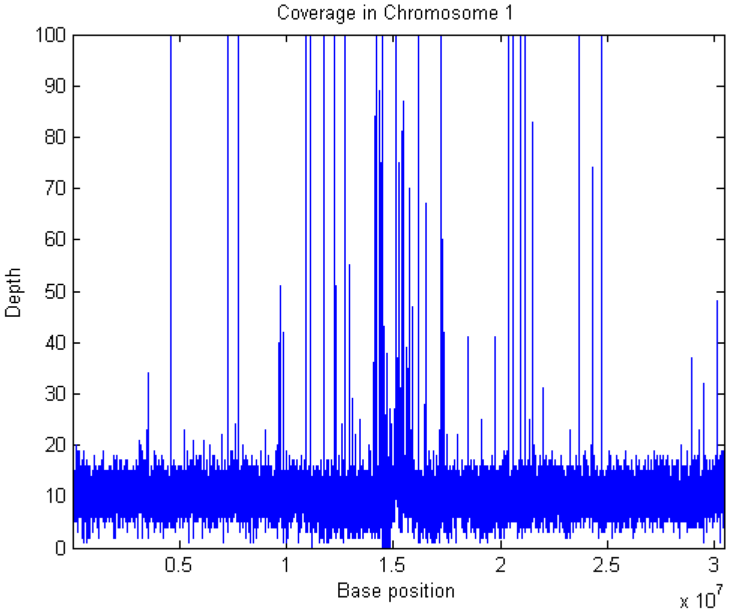
Further inspection of BioMap can indicate the range of the mapped short reads and how many short reads are mapped to the first chromosome.

```
x1 = min(getStart(bm1));
x2 = max(getStop(bm1));
fprintf('Chromosome 1 Range: %d-%d\n',x1,x2)
fprintf('Chromosome 1 Number of Reads: %d\n',bm1.NSeqs)

Chromosome 1 Range: 1-30427671
Chromosome 1 Number of Reads: 3151847
```

Exploring the Coverage at Different Resolutions

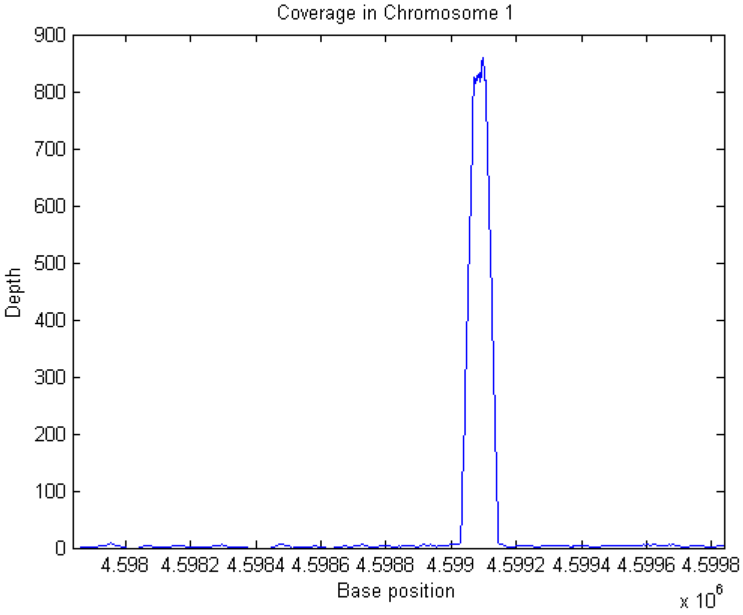
To explore the coverage for the whole range of the chromosome, a binning algorithm is required. The getBaseCoverage method produces a coverage signal based on effective alignments. It also allows you to specify a bin width to control the size (or resolution) of the output signal. However internal computations are still performed at the base pair (bp) resolution. This means that despite setting a large bin size, narrow peaks in the coverage signal can still be observed. Once the coverage signal is plotted you can program the figure's data cursor to display the genomic position when using the tooltip. You can zoom and pan the figure to determine the position and height of the ChIP-Seq peaks.

```
[cov,bin] = getBaseCoverage(bm1,x1,x2,'binWidth',1000,'binType','max');
figure
plot(bin,cov)
xlim([x1,x2])
ylim([0 100])
xlabel('Base position')
ylabel('Depth')
title('Coverage in Chromosome 1')

mdc = @(h,e)  {['Position: ',num2str(get(e,'Position')*[1;0])], ['Y: ',nu
set(datacursormode(gcf),'UpdateFcn',mdc)
datacursormode on


mdc =

    @(h,e){['Position: ',num2str(get(e,'Position')*[1;0])],['Y: ',num2str
```



It is also possible to explore the coverage signal at the bp resolution (also referred to as the *pile-up* profile). Explore one of the large peaks observed in the data at position 4598837.

```
p1 = 4598837-1000;
p2 = 4598837+1000;

figure
plot(p1:p2,getBaseCoverage(bm1,p1,p2))
xlim([p1,p2])
xlabel('Base position')
ylabel('Depth')
title('Coverage in Chromosome 1')
set(datacursormode(gcf),'UpdateFcn',mdc)
datacursormode on
```



Identifying and Filtering Regions with Artifacts

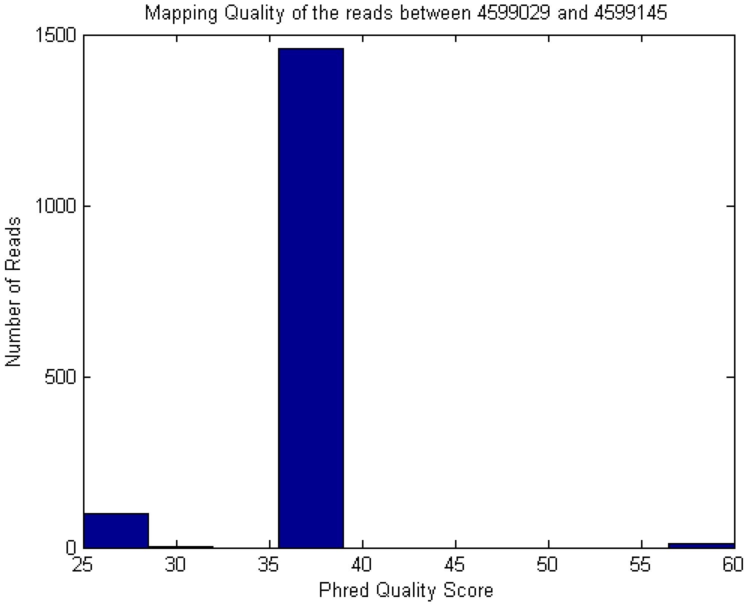
Observe the large peak with coverage depth of 800+ between positions 4599029 and 4599145. Investigate how these reads are aligning to the reference chromosome. You can retrieve a subset of these reads enough to satisfy a coverage depth of 25, since this is sufficient to understand what is happening in this region. Use `getIndex` to obtain indices to this subset. Then use `getCompactAlignment` to display the corresponding multiple alignment of the short-reads.

```
i = getIndex(bm1,4599029,4599145,'depth',25);
bmx = getSubset(bm1,i,'inmemory',false)
getCompactAlignment(bmx,4599029,4599145)


bmx =

  BioMap
  Properties:
         Reference: 'Chr1'
         Signature: [62x1 File indexed property]
             Start: [62x1 File indexed property]
    MappingQuality: [62x1 File indexed property]
              Flag: [62x1 File indexed property]
      MatePosition: [62x1 File indexed property]
           Quality: [62x1 File indexed property]
          Sequence: [62x1 File indexed property]
            Header: [62x1 File indexed property]
             NSeqs: 62
              Name: ''



ans =

AGTT AATCAAATAGAAAGCCCCGAGGGCGCCATATCCTAGGCGC  AAACTATGTGATTGAATAAATCCTCC
AGTGC  TCAAATAGAAAGCCCCGAGGGCGCCATATTCTAGGAGCCC             GAATAAATCCTCC
AGTTCAA             CCCGAGGGCGCCATATTCTAGGAGCCCAAACTATGTGATT
AGTTCAATCAAATAGAAAGC               TTCTAGGAGCCCAAACTATGTGATTGAATAAATCCTCC
AGTT                                  AAGGAGCCCAAAATATGTGATTGAATAAATCCACC
AGTACAATCAAATAGAAAGCCCCGAGGGCGCCATA   TAGGAGCCCAAACTATGTGATTGAATAAATCCTCC
CGTACAATCAAATAGAAAGCCCCGAGGGCGCCATATTC  GGAGCCCAAACTATGTGATTGAATAAATCCTCC
CGTACAATCAAATAGAAAGCCCCGAGGGCGCCATATTC  GGAGCCCAAACTATGTGATTGAATAAATCCTCC
CGTACAATCAAATAGAAAGCCCCGAGGGCGCCATATTC  GGAGCCCAAGCTATGTGATTGAATAAATCCTCC
CGTACAATCAAATAGAAAGCCCCGAGGGCGCCATATTC  GGAGCCCAAACTATGTGATTGAATAAATCCTCC
AGTTCAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTA GAGCCCAAACTATGTGATTGAATAAATCCTCC
GATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTA GAGCCCAAACTATGTGATTGAATAAATCTTCC
GATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTA GAGCCCAAACTATGTGATTGAATAAATCCTCC
GATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTA GAGCCCAAACTATGTGATTGAATAAATCCTCC
GATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTA GAGCCCAAATTATGTGATTGAATAAATCCTCC
 ATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTAG   CCCAAACTATGTGATTGAATAAATCCTCC
 ATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTAG   CACAAACTATGTGATTGAATAAATCCTCC
 ATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTAG    CCAAACTATGTGATTGAATAAATCCTCC
 ATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTAG
 ATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTCG
 ATACAATCAAATAGAAAGCCCCGGGGGCGCCATATTCTAG
 ATTGAGTCAAATAGAAAGCCCCGAGGGCGCCATATTCTAG
 ATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTAG
```

```
ATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTAG
ATACAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTAG
   CAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTAGGAG
   CAATCAAATAGAAAGCCCCGAGGGCGCCATATTCTAGGAG
                                     TAGGAGCCCAAACTATGTGATTGAATAAATCCTCC
                                     TAGGAGCCCAAACTATGCCATTGAATAAATCCTCC
                                       GGAGCCCAAGCTATGTGATTGAATAAATCCTCC
                                        GAGCCCAAACTATGTGATTGAATAAATCCTCC
                                        GAGCCCAAACTATGTGATTGAATAAATCCTCC
                                        GAGCCCAAACTATGTGATTGAATAAATCCTCC
                                        GAGCCCAAACTATGTGATTGAATAAATCCTCC
                                        GAGCCCAAACTATGTGATTGAATAAATCCTCC
```

In addition to visually confirming the alignment, you can also explore the mapping quality for all the short reads in this region, as this may hint to a potential problem. In this case, less than one percent of the short reads have a Phred quality of 60, indicating that the mapper most likely found multiple hits within the reference genome, hence assigning a lower mapping quality.

```
figure
i = getIndex(bm1,4599029,4599145);
hist(double(getMappingQuality(bm1,i)))
title('Mapping Quality of the reads between 4599029 and 4599145')
xlabel('Phred Quality Score')
ylabel('Number of Reads')
```



Most of the large peaks in this data set occur due to satellite repeat regions or due to its closeness to the centromere [4], and show characteristics similar to the example just explored. You may explore other regions with large peaks using the same procedure.

To prevent these problematic regions, two techniques are used. First, given that the provided data set uses paired-end sequencing, by removing the reads that are not aligned in a proper pair reduces the number of potential aligner errors or ambiguities. You can achieve this by exploring the `flag` field of the SAM formatted file, in which the second less significant bit is used to indicate if the short read is mapped in a proper pair.

```
i = find(bitget(getFlag(bm1),2));
bm1_filtered = getSubset(bm1,i)


bm1_filtered = 

  BioMap
  Properties:
         Reference: 'Chr1'
         Signature: [3040724x1 File indexed property]
             Start: [3040724x1 File indexed property]
    MappingQuality: [3040724x1 File indexed property]
              Flag: [3040724x1 File indexed property]
      MatePosition: [3040724x1 File indexed property]
           Quality: [3040724x1 File indexed property]
          Sequence: [3040724x1 File indexed property]
            Header: [3040724x1 File indexed property]
             NSeqs: 3040724
              Name: ''
```

Second, consider only uniquely mapped reads. You can detect reads that are equally mapped to different regions of the reference sequence by looking at the mapping quality, because BWA assigns a lower mapping quality (less than 60) to this type of short read.

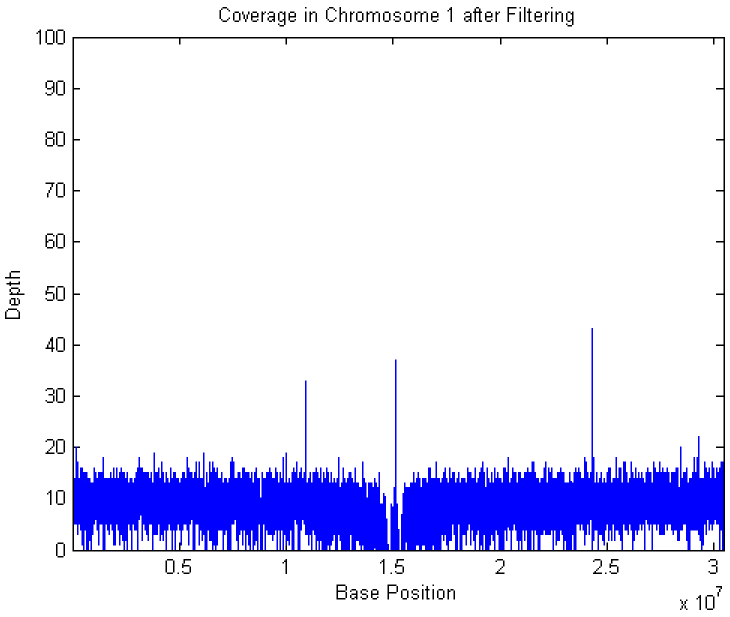
```
i = find(getMappingQuality(bm1_filtered)==60);
bm1_filtered = getSubset(bm1_filtered,i)
```

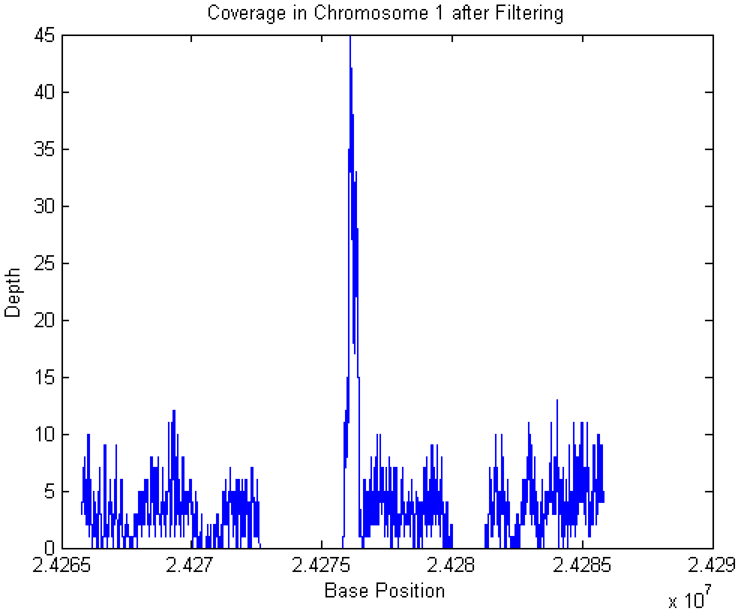
```
bm1_filtered =

  BioMap

  Properties:
         Reference: 'Chr1'
         Signature: [2313252x1 File indexed property]
             Start: [2313252x1 File indexed property]
    MappingQuality: [2313252x1 File indexed property]
              Flag: [2313252x1 File indexed property]
      MatePosition: [2313252x1 File indexed property]
           Quality: [2313252x1 File indexed property]
          Sequence: [2313252x1 File indexed property]
            Header: [2313252x1 File indexed property]
             NSeqs: 2313252
```

```
               Name: ''
```

Visualize again the filtered data set using both, a coarse resolution with 1000 bp bins for the whole chromosome, and a fine resolution for a small region of 20,000 bp. Most of the large peaks due to artifacts have been removed.

```
[cov,bin] = getBaseCoverage(bm1_filtered,x1,x2,'binWidth',1000,'binType',
figure
plot(bin,cov)
xlim([x1,x2])
ylim([0 100])
xlabel('Base Position')
ylabel('Depth')
title('Coverage in Chromosome 1 after Filtering')
set(datacursormode(gcf),'UpdateFcn',mdc)
datacursormode on

p1 = 24275801-10000;
p2 = 24275801+10000;

figure
plot(p1:p2,getBaseCoverage(bm1_filtered,p1,p2))
xlabel('Base Position')
ylabel('Depth')
title('Coverage in Chromosome 1 after Filtering')
set(datacursormode(gcf),'UpdateFcn',mdc)
datacursormode on
```


Coverage in Chromosome 1 after Filtering
Depth
100
90
80
70
60
50
40
30
20
10
0
0.5
1
1.5
2
2.5
3
Base Position
x 10^7



Recovering Sequencing Fragments from the Paired-End Reads

In Wang's paper [1] it is hypothesized that paired-end sequencing data has the potential to increase the accuracy of the identification of chromosome binding sites of DNA associated proteins because the fragment length can be derived accurately, while when using single-end sequencing it is necessary to resort to a statistical approximation of the fragment length, and use it indistinctly for all putative binding sites.

Use the paired-end reads to reconstruct the sequencing fragments. First, get the indices for the forward and the reverse reads in each pair. This information is captured in the fifth bit of the `flag` field, according to the SAM file format.

```
fow_idx = find(~bitget(getFlag(bm1_filtered),5));
rev_idx = find(bitget(getFlag(bm1_filtered),5));
```

SAM-formatted files use the same header strings to identify pair mates. By pairing the header strings you can determine how the short reads in `BioMap` are paired. To pair the header strings, simply order them in ascending order and use the sorting indices (`hf` and `hr`) to link the unsorted header strings.

```
[~,hf] = sort(getHeader(bm1_filtered,fow_idx));
[~,hr] = sort(getHeader(bm1_filtered,rev_idx));
mate_idx = zeros(numel(fow_idx),1);
mate_idx(hf) = rev_idx(hr);
```

Use the resulting `fow_idx` and `mate_idx` variables to retrieve pair mates. For example, retrieve the paired-end reads for the first 10 fragments.

```
for j = 1:10
  disp(getInfo(bm1_filtered, fow_idx(j)))
  disp(getInfo(bm1_filtered, mate_idx(j)))
end
```

```
SRR054715.sra.6849385 163 20 60 40M AACCCTAAACCTCTGAATCCTTAATCCCTAAATCCCT
SRR054715.sra.6849385 83 229 60 40M CCTATTTCTTGTGGTTTTCTTTCCTTCACTTAGCTAT
SRR054715.sra.6992346 99 20 60 40M AACCCTAAACCTCTGAATCCTTAATCCCTAAATCCCTA
SRR054715.sra.6992346 147 239 60 40M GTGGTTTTCTTTCCTTCACTTAGCTATGGATGGTTT
SRR054715.sra.8438570 163 47 60 40M CTAAATCCCTAAATCTTTAAATCCTACATCCATGAAT
SRR054715.sra.8438570 83 274 60 40M TATCTTCATTTGTTATATTGGATACAAGCTTTGCTAC
SRR054715.sra.1676744 163 67 60 40M ATCCTACATCCATGAATCCCTAAATACCTAATCCCCT
SRR054715.sra.1676744 83 283 60 40M TTGTTATATTGGATACAAGCTTTGCTACGATCTACAT
SRR054715.sra.6820328 163 73 60 40M CATCCATGAATCCCTAAATACCTAATTCCCTAAACCC
SRR054715.sra.6820328 83 267 60 40M GTTGGTGTATCTTCATTTGTTATATTGGATACGAGCT
SRR054715.sra.1559757 163 103 60 40M TAAACCCGAAACCGGTTTCTCTGGTTGAAACTCATT
SRR054715.sra.1559757 83 311 60 40M GATCTACATTTGGGAATGTGAGTCTCTTATTGTAACC
SRR054715.sra.5658991 163 103 60 40M CAAACCCGAAACCGGTTTCTCTGGTTGAAACTCATT
SRR054715.sra.5658991 83 311 60 40M GATCTACATTTGGGAATGTGAGTCTCTTATTGTAACC
SRR054715.sra.4625439 163 143 60 40M ATATAATGATAATTTTAGCGTTTTTATGCAATTGCT
SRR054715.sra.4625439 83 347 60 40M CTTAGTGTTGGTTTATCTCAAGAATCTTATTAATTGT
SRR054715.sra.1007474 163 210 60 40M ATTTGAGGTCAATACAAATCCTATTTCTTGTGGTTT
SRR054715.sra.1007474 83 408 60 40M TATTGTCATTCTTACTCCTTTGTGGAAATGTTTGTTC
SRR054715.sra.7345693 99 213 60 40M TGAGGTCAATACAAATCCTATTTCTTGTGGTTTTCTT
SRR054715.sra.7345693 147 393 60 40M TTATTTTTGGACATTTATTGTCATTCTTACTCCTTT
```

Use the paired-end indices to construct a new `BioMap` with the minimal information needed to represent the sequencing fragments. First, calculate the insert sizes.

```
J = getStop(bm1_filtered, fow_idx);
K = getStart(bm1_filtered, mate_idx);
L = K - J - 1;
```

Obtain the new signature (or CIGAR string) for each fragment by using the short read original signatures separated by the appropriate number of skip CIGAR symbols (N).

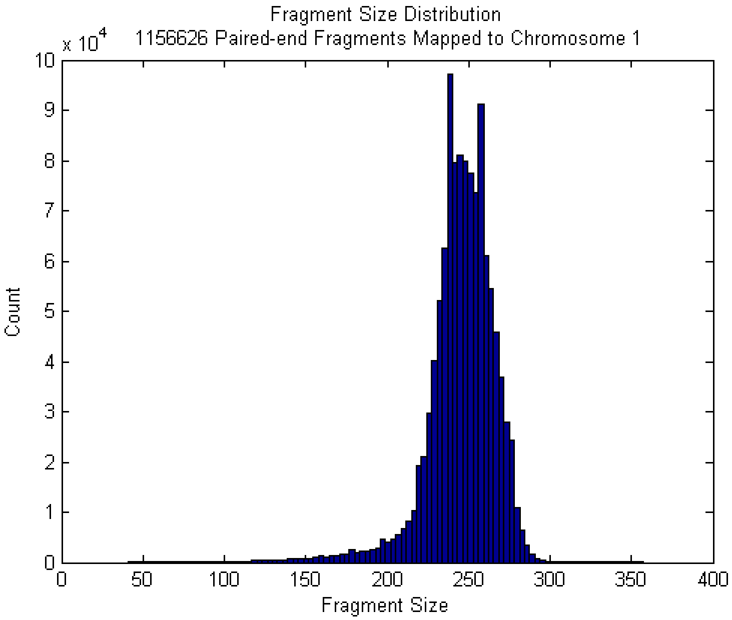
```
n = numel(L);
cigars = cell(n,1);
for i = 1:n
   cigars{i} = sprintf('%dN' ,L(i));
end
cigars = strcat( getSignature(bm1_filtered, fow_idx),...
                 cigars,...
                 getSignature(bm1_filtered, mate_idx));
```

Reconstruct the sequences for the fragments by concatenating the respective sequences of the paired-end short reads.

```
seqs = strcat( getSequence(bm1_filtered, fow_idx),...
               getSequence(bm1_filtered, mate_idx));
```

Calculate and plot the fragment size distribution.

```
J = getStart(bm1_filtered,fow_idx);
K = getStop(bm1_filtered,mate_idx);
L = K - J + 1;
figure
hist(double(L),100)
title(sprintf('Fragment Size Distribution\n %d Paired-end Fragments Mappe
xlabel('Fragment Size')
ylabel('Count')
```



Construct a new `BioMap` to represent the sequencing fragments. With this, you will be able explore the coverage signals as well as local alignments of the fragments.

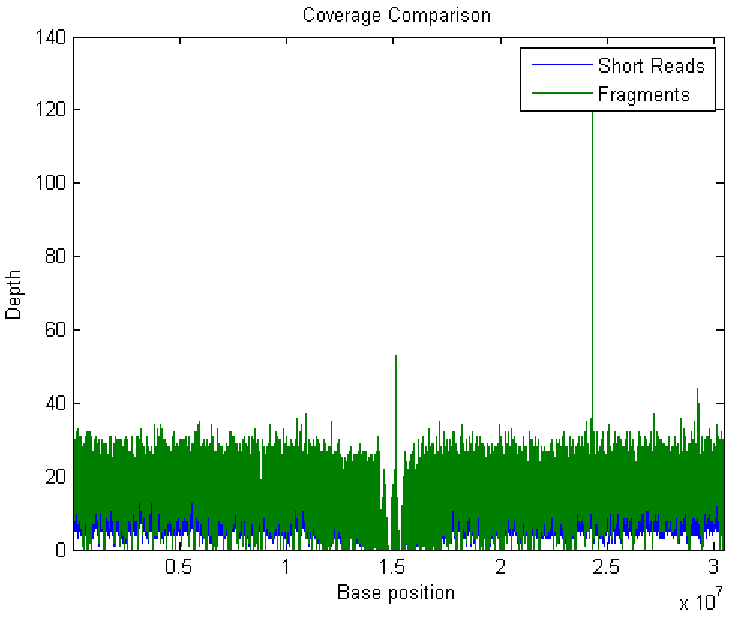
```
bm1_fragments = BioMap('Sequence',seqs,'Signature',cigars,'Start',J)


bm1_fragments =

  BioMap

  Properties:
         Reference: ''
         Signature: {1156626x1 cell}
             Start: [1156626x1 uint32]
    MappingQuality: [0x1 uint8]
              Flag: [0x1 uint16]
      MatePosition: [0x1 uint32]
           Quality: {0x1 cell}
```

```
            Sequence: {1156626x1 cell}
              Header: {0x1 cell}
               NSeqs: 1156626
                Name: ''
```

Exploring the Coverage Using Fragment Alignments

Compare the coverage signal obtained by using the reconstructed fragments with the coverage signal obtained by using individual paired-end reads. Notice that enriched binding sites, represented by peaks, can be better discriminated from the background signal.

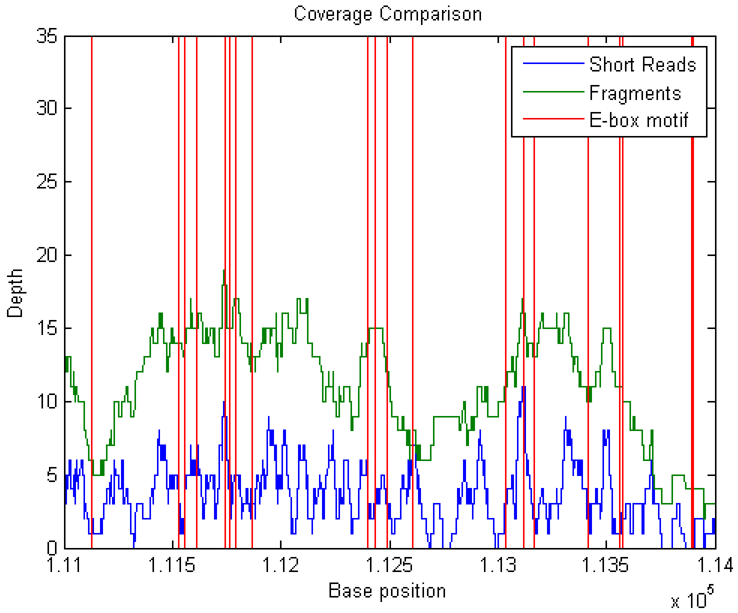
```
cov_reads = getBaseCoverage(bm1_filtered,x1,x2,'binWidth',1000,'binType',
[cov_fragments,bin] = getBaseCoverage(bm1_fragments,x1,x2,'binWidth',100

figure
plot(bin,cov_reads,bin,cov_fragments)
xlim([x1,x2])
xlabel('Base position')
ylabel('Depth')
title('Coverage Comparison')
legend('Short Reads','Fragments')
set(datacursormode(gcf),'UpdateFcn',mdc)
datacursormode on
```



Perform the same comparison at the bp resolution. In this dataset, Wang et.al. [1] investigated a basic helix-loop-helix (*bHLH*) transcription factor. *bHLH* proteins typically bind to a consensus sequence called an *E-box* (with a CANNTG motif). Use fastaread to load the reference chromosome, search for the *E-box* motif in the 3' and 5' directions, and then overlay the motif positions on the coverage signals. This example works over the region 1-200,000, however the figure limits are narrowed to a 3000 bp region in order to better depict the details.

```
p1 = 1;
p2 = 200000;

cov_reads = getBaseCoverage(bm1_filtered,p1,p2);
[cov_fragments,bin] = getBaseCoverage(bm1_fragments,p1,p2);

chr1 = fastaread('ach1.fasta');
mp1 = regexp(chr1.Sequence(p1:p2),'CA..TG')+3+p1;
mp2 = regexp(chr1.Sequence(p1:p2),'GT..AC')+3+p1;
motifs = [mp1 mp2];
```

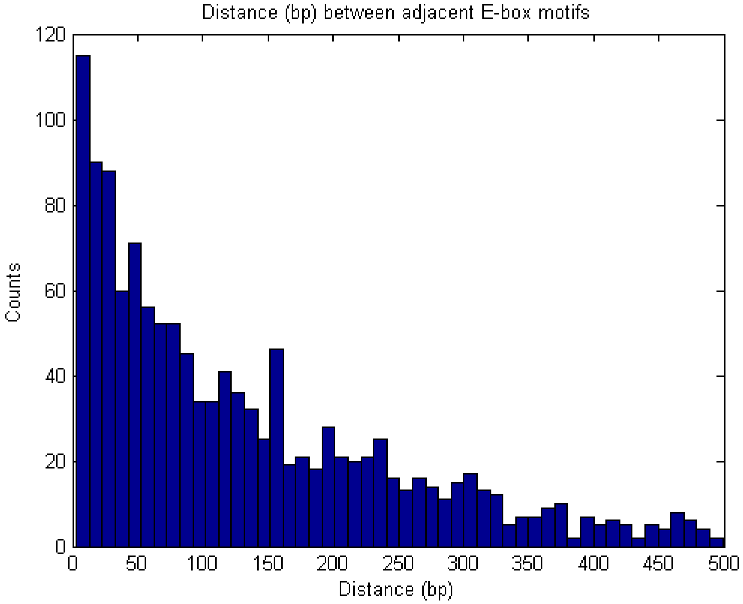
```
figure
plot(bin,cov_reads,bin,cov_fragments)
hold on
plot([1;1;1]*motifs,[0;max(ylim);NaN],'r')
xlabel('Base position')
ylabel('Depth')
title('Coverage Comparison')
legend('Short Reads','Fragments','E-box motif')
set(datacursormode(gcf),'UpdateFcn',mdc)
datacursormode on
xlim([111000 114000])
```



Observe that it is not possible to associate each peak in the coverage signals with an *E-box* motif. This is because the length of the sequencing fragments is comparable to the average motif distance, blurring peaks that are close. Plot the distribution of the distances between the *E-box* motif sites.

```
motif_sep = diff(sort(motifs));
```

```
figure
hist(motif_sep(motif_sep<500),50)
title('Distance (bp) between adjacent E-box motifs')
xlabel('Distance (bp)')
ylabel('Counts')
```

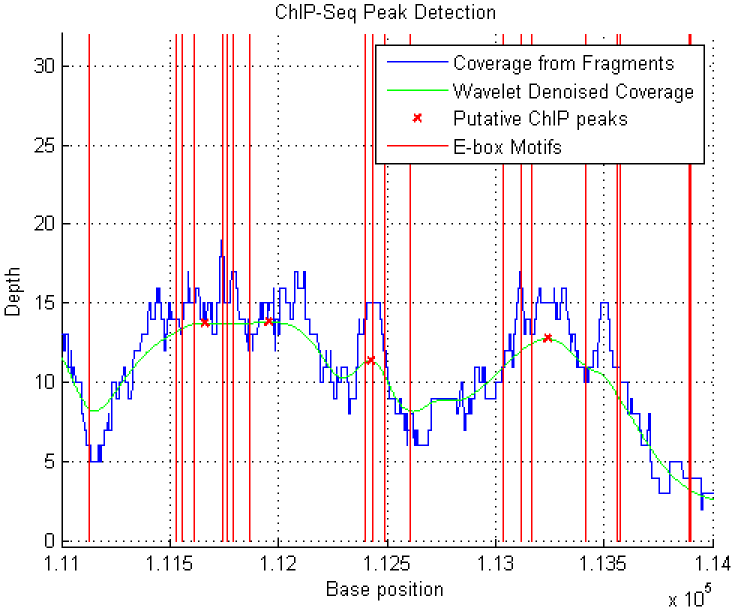


Finding Significant Peaks in the Coverage Signal

Use the function mspeaks to perform peak detection with Wavelets denoising on the coverage signal of the fragment alignments. Filter putative ChIP peaks using a height filter to remove peaks that are not enriched by the binding process under consideration.

```
putative_peaks = mspeaks(bin,cov_fragments,'noiseestimator',20,...
                         'heightfilter',10,'showplot',true);
hold on
plot([1;1;1]*motifs(motifs>p1 & motifs<p2),[0;max(ylim);NaN],'r')
legend('Coverage from Fragments','Wavelet Denoised Coverage','Putative Ch
xlabel('Base position')
```

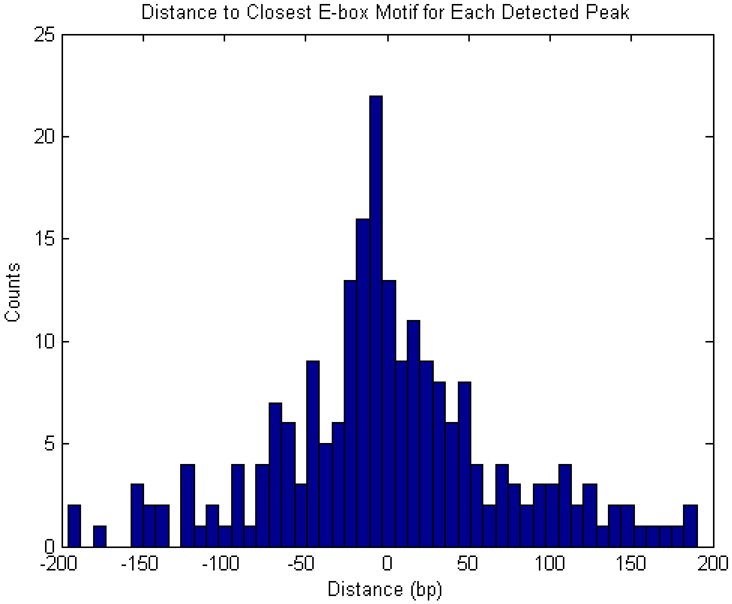
```
ylabel('Depth')
title('ChIP-Seq Peak Detection')
set(datacursormode(gcf),'UpdateFcn',mdc)
datacursormode on
xlim([111000 114000])
```



Use the knnsearch function to find the closest motif to each one of the putative peaks. As expected, most of the enriched ChIP peaks are close to an *E-box* motif [1]. This reinforces the importance of performing peak detection at the finest resolution possible (bp resolution) when the expected density of binding sites is high, as it is in the case of the *E-box* motif. It also demonstrates that for this type of analysis, paired-end sequencing should be considered over single-end sequencing [1].

```
h = knnsearch(motifs',putative_peaks(:,1));
distance = putative_peaks(:,1)-motifs(h(:))';
figure
hist(distance(abs(distance)<200),50)
title('Distance to Closest E-box Motif for Each Detected Peak')
```

```
xlabel('Distance (bp)')
ylabel('Counts')
```



References

[1] Wang C., Xu J., Zhang D., Wilson Z.A., and Zhang D. "An effective approach for identification of in vivo protein-DNA binding sites from paired-end ChIP-Seq data", BMC Bioinformatics, 11:81, Feb 9, 2010.

[2] Li H. and Durbin R. "Fast and accurate short read alignment with Burrows-Wheeler transform", Bioinformatics, 25, pp 1754-60, 2009.

[3] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R. and 1000 Genome Project Data Processing Subgroup "The Sequence Alignment/map (SAM) Format and SAMtools", Bioinformatics, 25, pp 2078-2079, 2009.

[4] Jothi R, Cuddapah S, Barski A, Cui K, Zhao K. "Genome-wide identification of in vivo protein-DNA binding sites from ChIP-Seq data", Nucleic Acids Research, 36(16), pp 5221-31, Sep 2008.

[5] Hoofman B.G., and Jones S.J.M. "Genome-wide identification of DNA–protein interactions using chromatin immunoprecipitation coupled with flow cell sequencing", Journal of Endocrinology 201, pp 1-13, 2009.

[6] Ramsey SA, Knijnenburg TA, Kennedy KA, Zak DE, Gilchrist M, Gold ES, Johnson CD, Lampano AE, Litvak V, Navarro G, Stolyar T, Aderem A, Shmulevich I. "Genome-wide histone acetylation data improve prediction of mammalian transcription factor binding sites", Bioinformatics, 26(17), pp 2071-5, Sep 1, 2010.

Provide feedback for this demo.

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 known function.

Reading, Analyzing, and Converting a Nucleotide Sequence Using the Command Line

In this section...

Overview of Example

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

Searching the Web for Sequence Information

The following procedure illustrates how to use the MATLAB Help browser to search the Web for information. In this example you are interested in studying the human mitochondrial genome. While many genes that code for mitochondrial proteins are found in the cell nucleus, the mitochondrial has genes that code for proteins used to produce energy.

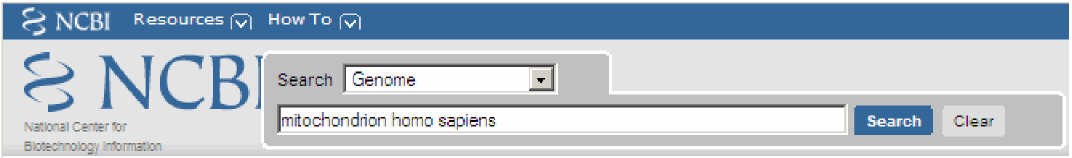
First research information about the human mitochondria and find the nucleotide sequence for the genome. Next, look at the nucleotide content for the entire sequence. And finally, determine open reading frames and extract specific gene sequences.

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

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

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

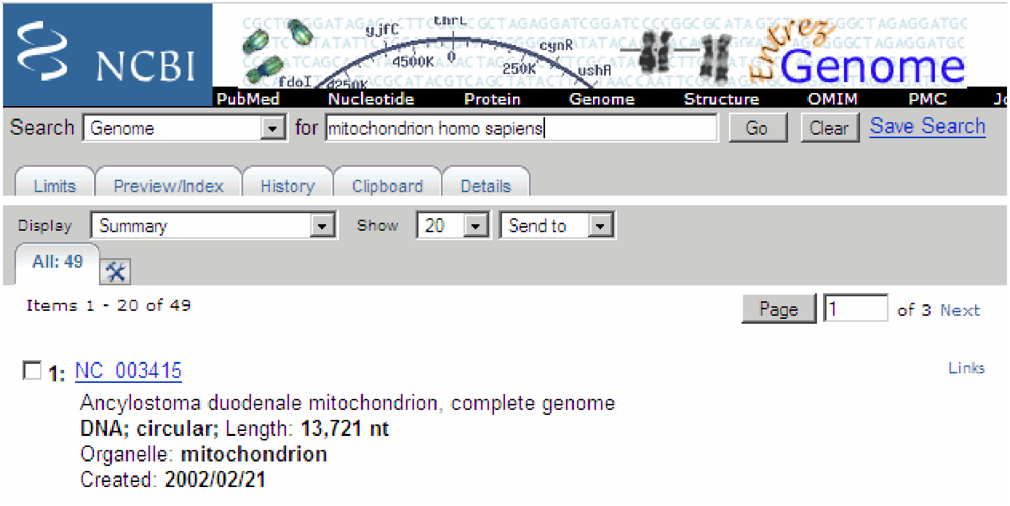
2 Search the NCBI Web site for information. For example, to search for the human mitochondrion genome, from the **Search** list, select `Genome` , and in the **Search** list, enter `mitochondrion homo sapiens`.



Search Genome

mitochondrion homo sapiens

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



Search Genome for mitochondrion homo sapiens

Display Summary Show 20 Send to

Items 1 - 20 of 49

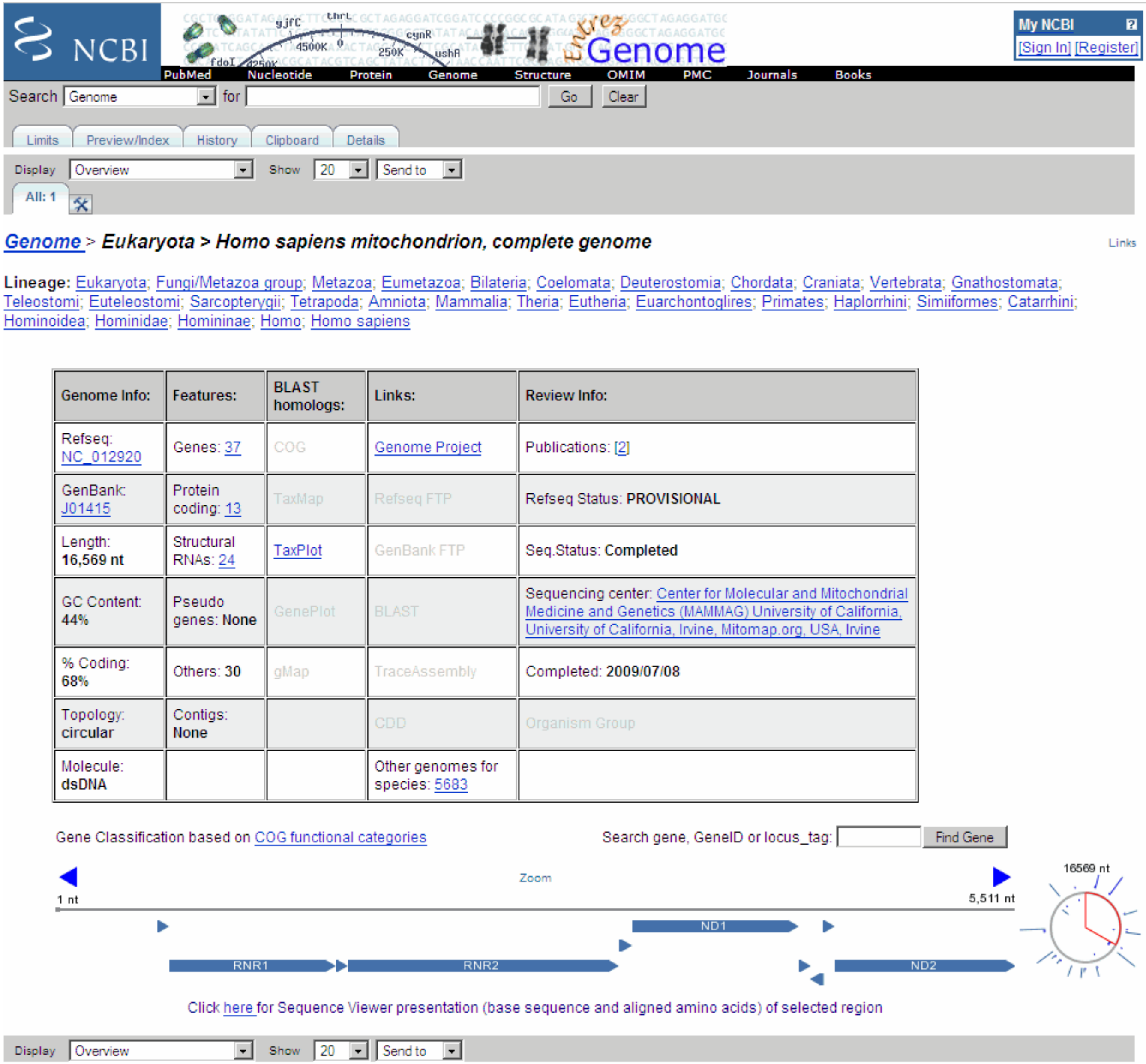
Page 1 of 3

□ 1: NC_003415

Ancylostoma duodenale mitochondrion, complete genome
DNA; circular; Length: **13,721 nt**
Organelle: **mitochondrion**
Created: **2002/02/21**

3 Select a result page. For example, click the link labeled **NC_012920**.

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



Genome > Eukaryota > Homo sapiens mitochondrion, complete genome

Lineage: Eukaryota; Fungi/Metazoa group; Metazoa; Eumetazoa; Bilateria; Coelomata; Deuterostomia; Chordata; Craniata; Vertebrata; Gnathostomata; Teleostomi; Euteleostomi; Sarcopterygii; Tetrapoda; Amniota; Mammalia; Theria; Eutheria; Euarchontoglires; Primates; Haplorrhini; Simiiformes; Catarrhini; Hominoidea; Hominidae; Homininae; Homo; Homo sapiens

Genome Info:	Features:	BLAST homologs:	Links:	Review Info:
Refseq: NC_012920	Genes: 37	COG	Genome Project	Publications: [2]
GenBank: J01415	Protein coding: 13	TaxMap	Refseq FTP	Refseq Status: **PROVISIONAL**
Length: **16,569 nt**	Structural RNAs: 24	TaxPlot	GenBank FTP	Seq.Status: **Completed**
GC Content: **44%**	Pseudo genes: **None**	GenePlot	BLAST	Sequencing center: Center for Molecular and Mitochondrial Medicine and Genetics (MAMMAG) University of California, University of California, Irvine, Mitomap.org, USA, Irvine
% Coding: **68%**	Others: **30**	gMap	TraceAssembly	Completed: **2009/07/08**
Topology: **circular**	Contigs: **None**		CDD	Organism Group
Molecule: **dsDNA**			Other genomes for species: 5683	

Gene Classification based on COG functional categories

Search gene, GeneID or locus_tag: Find Gene

Click here for Sequence Viewer presentation (base sequence and aligned amino acids) of selected region

Reading Sequence Information from the Web

The following procedure illustrates how to find a nucleotide sequence in a public database and read the sequence information into the MATLAB environment. Many public databases for nucleotide sequences are accessible from the Web. The MATLAB Command Window provides an integrated environment for bringing sequence information into the MATLAB environment.

The consensus sequence for the human mitochondrial genome has the GenBank accession number NC_012920. 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 retrieve sequence information for the human mitochondrial genome, in the MATLAB Command Window, type

```
mitochondria = getgenbank('NC_012920','SequenceOnly',true)
```

The getgenbank function retrieves the nucleotide sequence from the GenBank database and creates a character array.

```
mitochondria =
GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCATGCAT
TTGGTATTTTCGTCTGGGGGGTGTGCACGCGATAGCATTGCGAGACGCTG
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 software, using the command

```
load mitochondria
```

The load function loads the sequence mitochondria into the MATLAB Workspace.

3 Get information about the sequence. Type

```
whos mitochondria
```

Information about the size of the sequence displays in the MATLAB Command Window.

```
Name               Size                 Bytes  Class    Attributes

mitochondria       1x16569              33138  char
```

Determining Nucleotide Composition

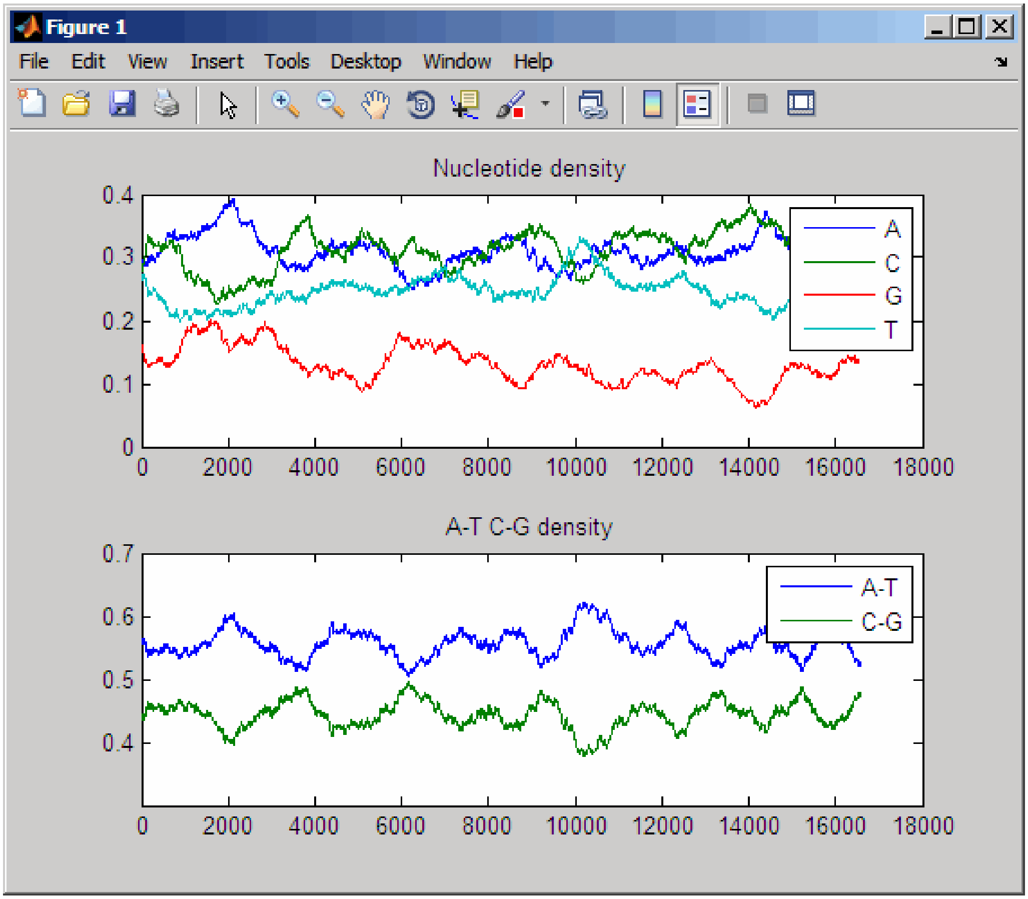
The following procedure illustrates how to determine the monomers and dimers, and then visualize data in graphs and bar plots. Sections of a DNA sequence with a high percent of A+T nucleotides usually indicate intergenic parts of the sequence, while low A+T and higher G+C nucleotide percentages indicate possible genes. Many times high CG dinucleotide content is located before a gene.

After you read a sequence into the MATLAB environment, 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 "Reading Sequence Information from the Web" on page 3-5.

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 `basecount` function.

```
basecount(mitochondria)
```

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

```
ans =
    A: 5124
    C: 5181
    G: 2169
    T: 4094
```

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

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

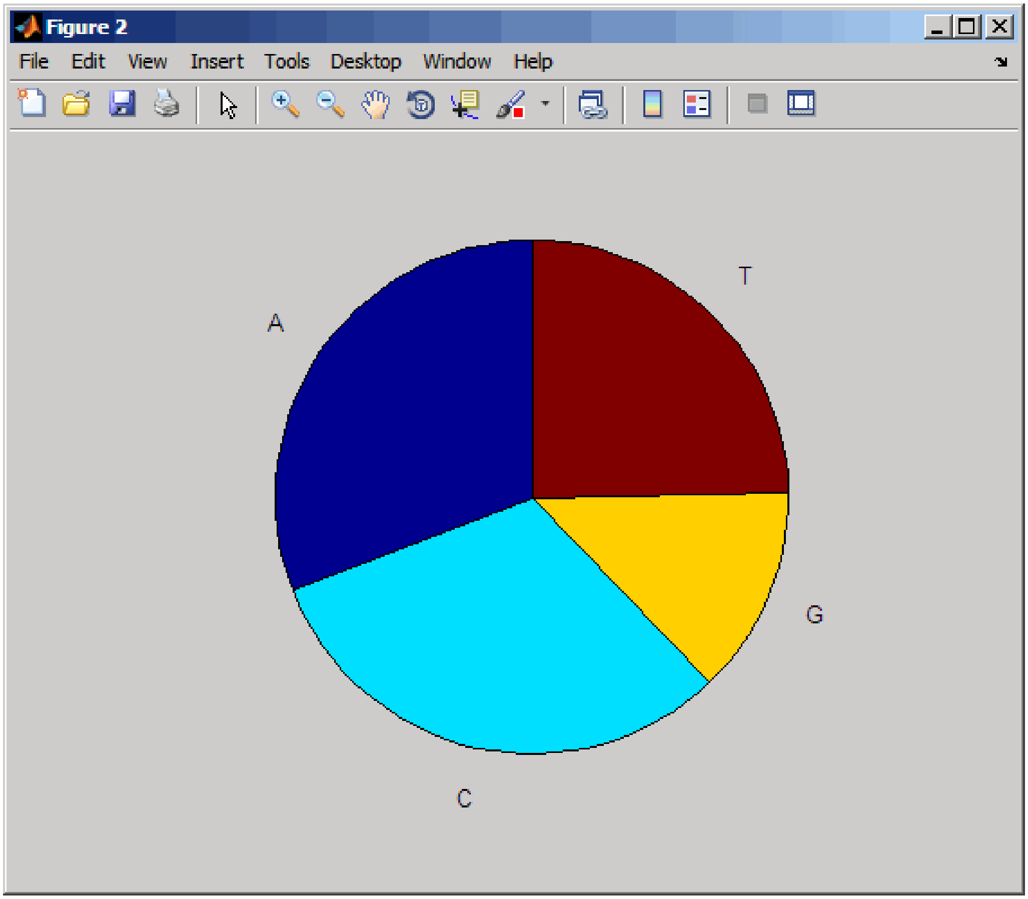
As expected, the nucleotide counts on the reverse complement strand are complementary to the 5'-3' strand.

```
ans =
    A: 4094
    C: 2169
    G: 5181
    T: 5124
```

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

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

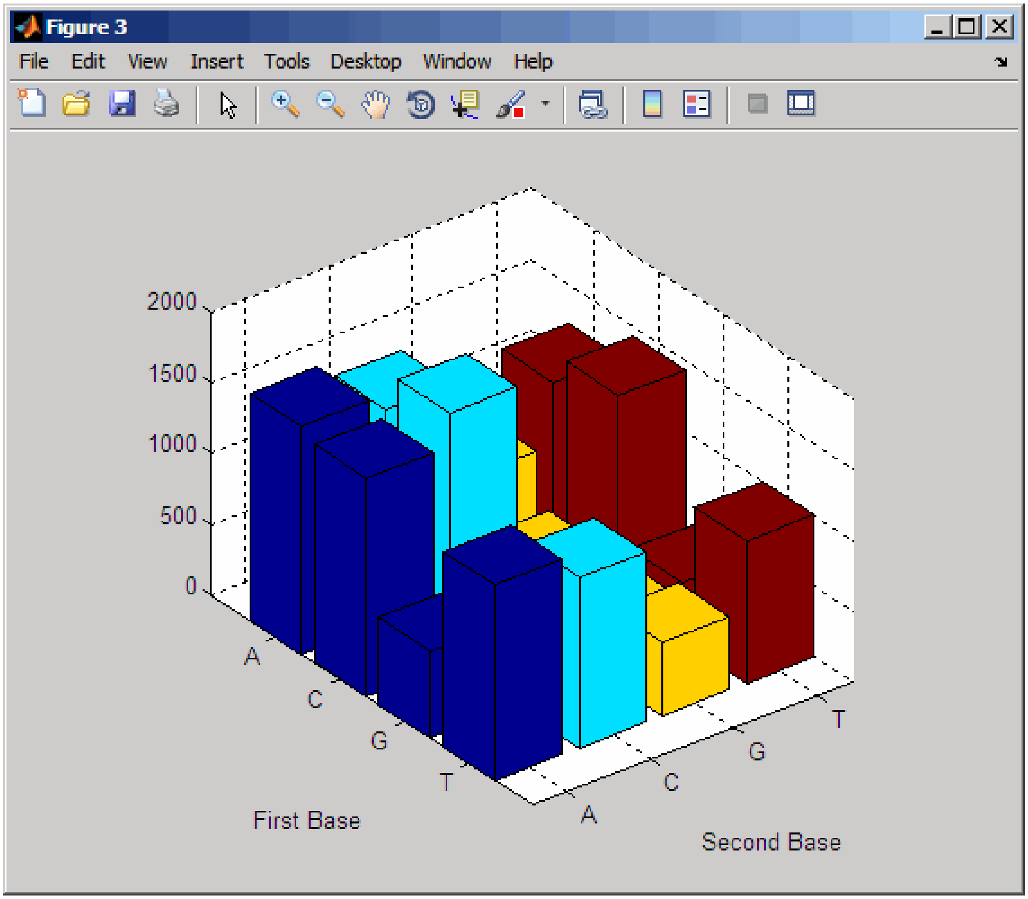
A pie chart displays in the MATLAB Figure window.



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

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

ans =

    AA: 1604
    AC: 1495
    AG: 795
    AT: 1230
```

```
CA: 1534
CC: 1771
CG: 435
CT: 1440
GA: 613
GC: 711
GG: 425
GT: 419
TA: 1373
TC: 1204
TG: 513
TT: 1004
```



Determining Codon Composition

The following procedure illustrates how to look at codons for the six reading frames. 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 the MATLAB environment, you can analyze the sequence for codon composition. This procedure uses the human mitochondria genome as an example. See "Reading Sequence Information from the Web" on page 3-5.

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

```
codoncount(mitochondria)
```

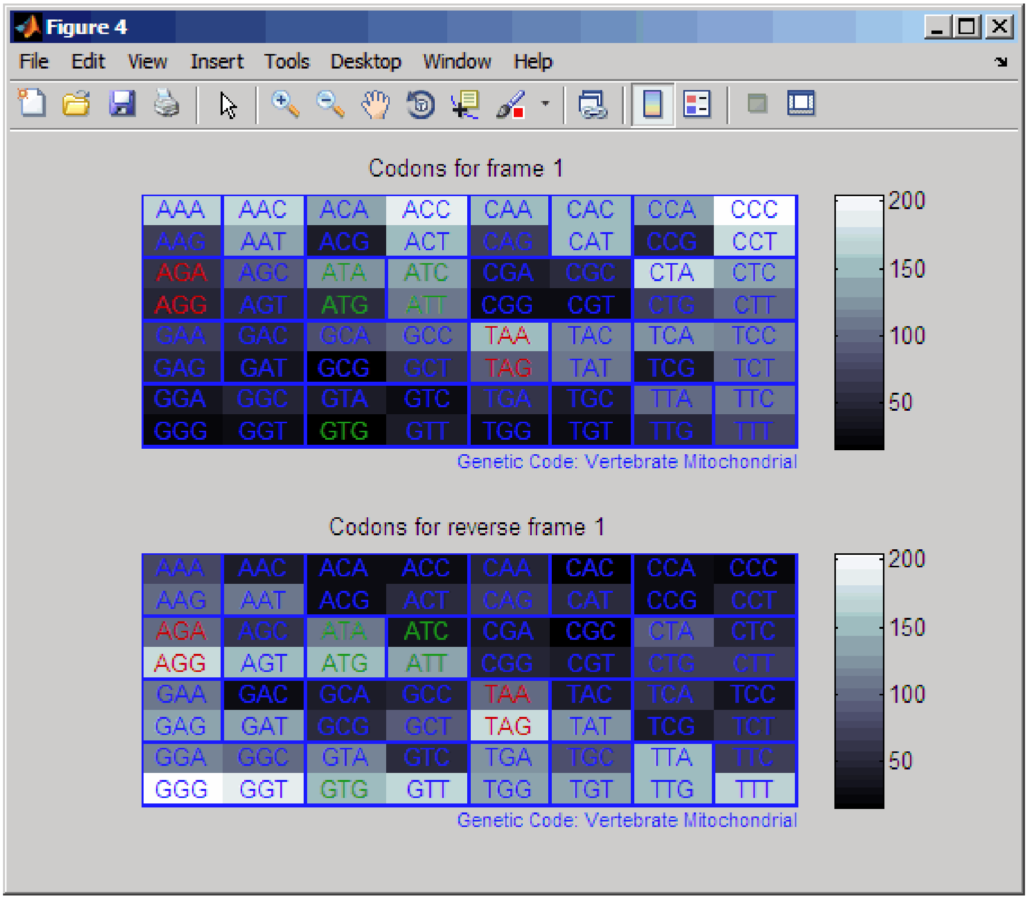
The codon counts for the first reading frame displays.

```
AAA - 167     AAC - 171     AAG -  71     AAT - 130
ACA - 137     ACC - 191     ACG -  42     ACT - 153
AGA -  59     AGC -  87     AGG -  51     AGT -  54
ATA - 126     ATC - 131     ATG -  55     ATT - 113
CAA - 146     CAC - 145     CAG -  68     CAT - 148
CCA - 141     CCC - 205     CCG -  49     CCT - 173
CGA -  40     CGC -  54     CGG -  29     CGT -  27
CTA - 175     CTC - 142     CTG -  74     CTT - 101
GAA -  67     GAC -  53     GAG -  49     GAT -  35
GCA -  81     GCC - 101     GCG -  16     GCT -  59
GGA -  36     GGC -  47     GGG -  23     GGT -  28
GTA -  43     GTC -  26     GTG -  18     GTT -  41
TAA - 157     TAC - 118     TAG -  94     TAT - 107
TCA - 125     TCC - 116     TCG -  37     TCT - 103
TGA -  64     TGC -  40     TGG -  29     TGT -  26
TTA -  96     TTC - 107     TTG -  47     TTT -  78
```

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

```
for frame = 1:3
    figure
    subplot(2,1,1);
    codoncount(mitochondria,'frame',frame,'figure',true,...
               'geneticcode','Vertebrate Mitochondrial');
    title(sprintf('Codons for frame %d',frame));
    subplot(2,1,2);
    codoncount(mitochondria,'reverse',true,'frame',frame,...
               'figure',true,'geneticcode','Vertebrate Mitochondrial');
    title(sprintf('Codons for reverse frame %d',frame));
end
```

Heat maps display all 64 codons in the 6 reading frames.



Codons for frame 1

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

Genetic Code: Vertebrate Mitochondrial

Codons for reverse frame 1

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

Genetic Code: Vertebrate Mitochondrial

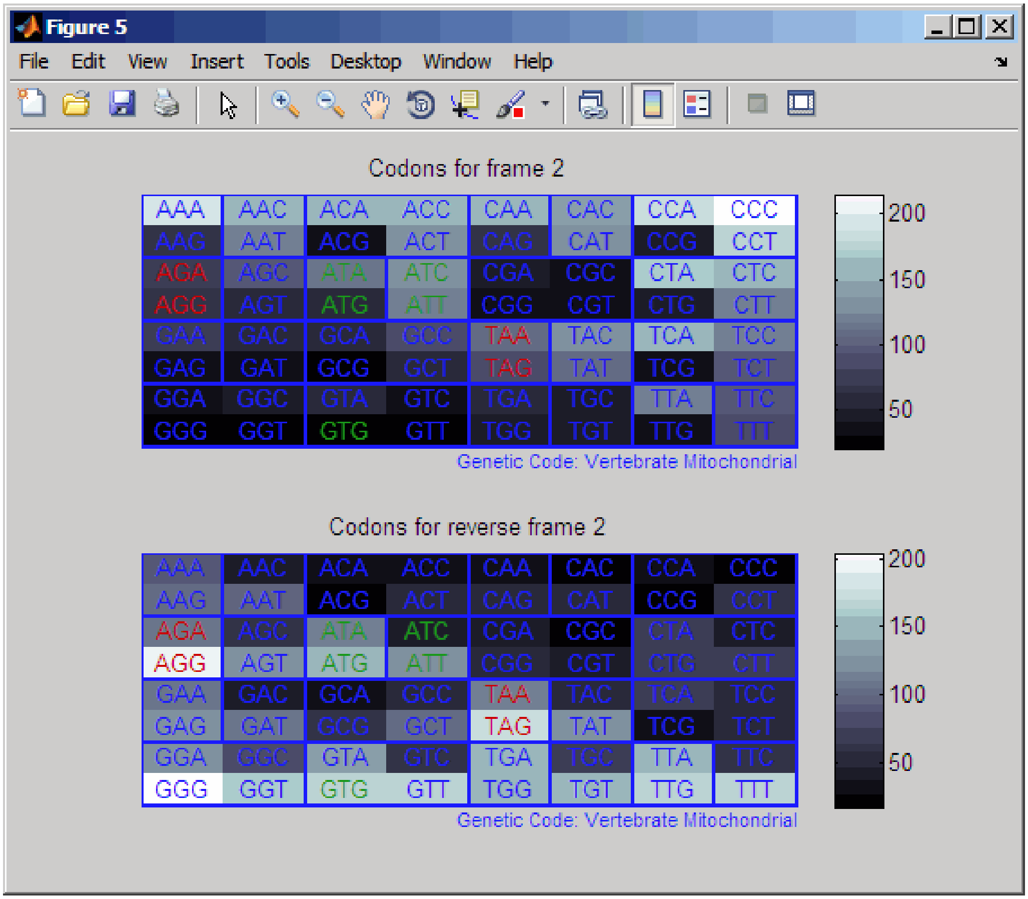
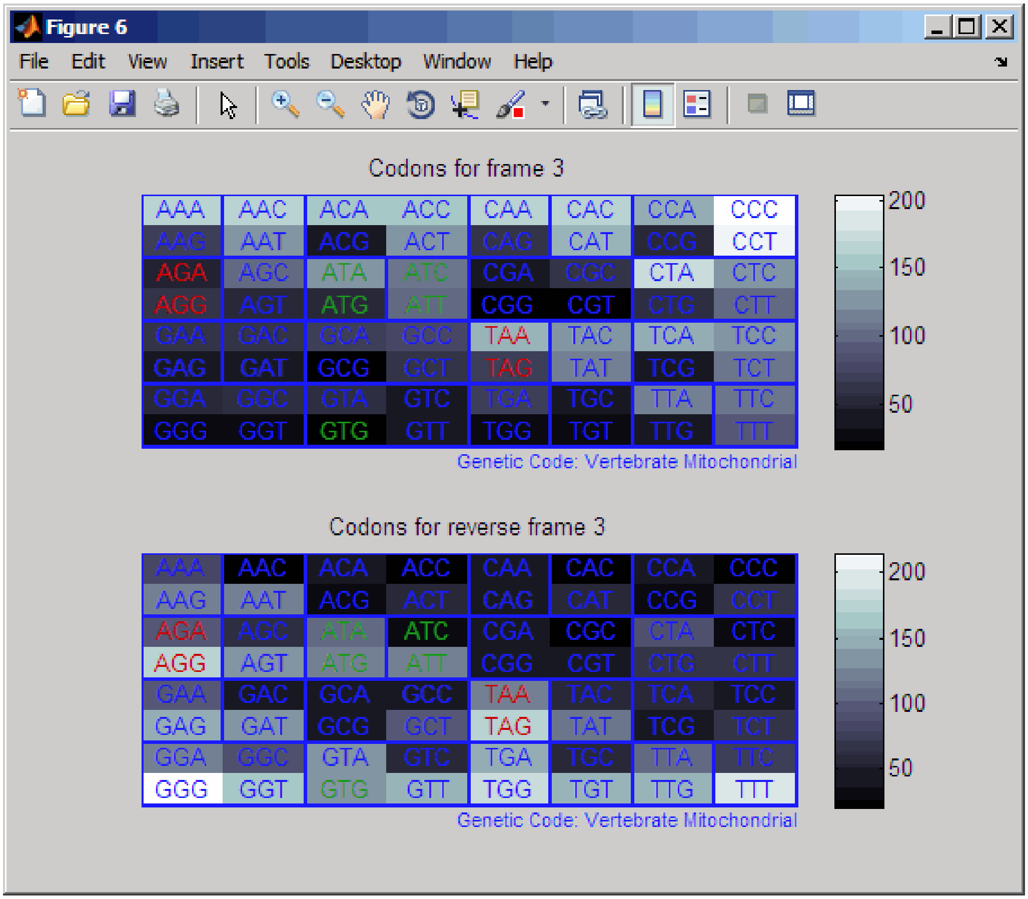


Figure 5
File Edit View Insert Tools Desktop Window Help
Codons for frame 2
AAA AAC ACA ACC CAA CAC CCA CCC
AAG AAT ACG ACT CAG CAT CCG CCT
AGA AGC ATA ATC CGA CGC CTA CTC
AGG AGT ATG ATT CGG CGT CTG CTT
GAA GAC GCA GCC TAA TAC TCA TCC
GAG GAT GCG GCT TAG TAT TCG TCT
GGA GGC GTA GTC TGA TGC TTA TTC
GGG GGT GTG GTT TGG TGT TTG TTT
200
150
100
50
Genetic Code: Vertebrate Mitochondrial
Codons for reverse frame 2
AAA AAC ACA ACC CAA CAC CCA CCC
AAG AAT ACG ACT CAG CAT CCG CCT
AGA AGC ATA ATC CGA CGC CTA CTC
AGG AGT ATG ATT CGG CGT CTG CTT
GAA GAC GCA GCC TAA TAC TCA TCC
GAG GAT GCG GCT TAG TAT TCG TCT
GGA GGC GTA GTC TGA TGC TTA TTC
GGG GGT GTG GTT TGG TGT TTG TTT
200
150
100
50
Genetic Code: Vertebrate Mitochondrial



Codons for frame 3

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

Genetic Code: Vertebrate Mitochondrial

Codons for reverse frame 3

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

Genetic Code: Vertebrate Mitochondrial

Open Reading Frames

The following procedure illustrates how to locate the open reading frames using a specific genetic code. 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 the MATLAB environment, you can analyze the sequence for open reading frames. This procedure uses the human mitochondria genome as an example. See "Reading Sequence Information from the Web" on page 3-5.

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

   ```
   seqshoworfs(mitochondria);
   ```

   If you compare this output to the genes shown on the NCBI page for `NC_012920`, there are fewer genes than expected. This is because vertebrate mitochondria use a genetic code slightly different from the standard genetic code. For a list of genetic codes, see the "Genetic Code" table in the `aa2nt` reference page in the *Bioinformatics Toolbox Reference*.

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 third reading frame. One starts at position 4470 and the other starts at 5904. 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 = 4470;
   StartIndex = find(orfs(3).Start == ND2Start)
   ND2Stop = orfs(3).Stop(StartIndex)
   ```

   The stop position displays.

   ```
   ND2Stop =

          5511
   ```

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

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

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 - 23     ATC - 24     ATG -  1     ATT -  8
CAA -  8     CAC -  3     CAG -  2     CAT -  1
CCA -  4     CCC - 12     CCG -  2     CCT -  5
CGA -  0     CGC -  3     CGG -  0     CGT -  1
CTA - 26     CTC - 18     CTG -  4     CTT -  7
GAA -  5     GAC -  0     GAG -  1     GAT -  0
GCA -  8     GCC -  7     GCG -  1     GCT -  4
GGA -  5     GGC -  7     GGG -  0     GGT -  1
GTA -  3     GTC -  2     GTG -  0     GTT -  3
TAA -  0     TAC -  8     TAG -  0     TAT -  2
TCA -  7     TCC - 11     TCG -  1     TCT -  4
TGA - 10     TGC -  0     TGG -  1     TGT -  0
TTA -  8     TTC -  7     TTG -  1     TTT -  8
```

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

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

The following displays:

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

Amino Acid Conversion and Composition

The following procedure illustrates how to extract the protein-coding sequence from a gene sequence and convert it to the amino acid sequence for the protein. 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 3-15.

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 the GenPept database.

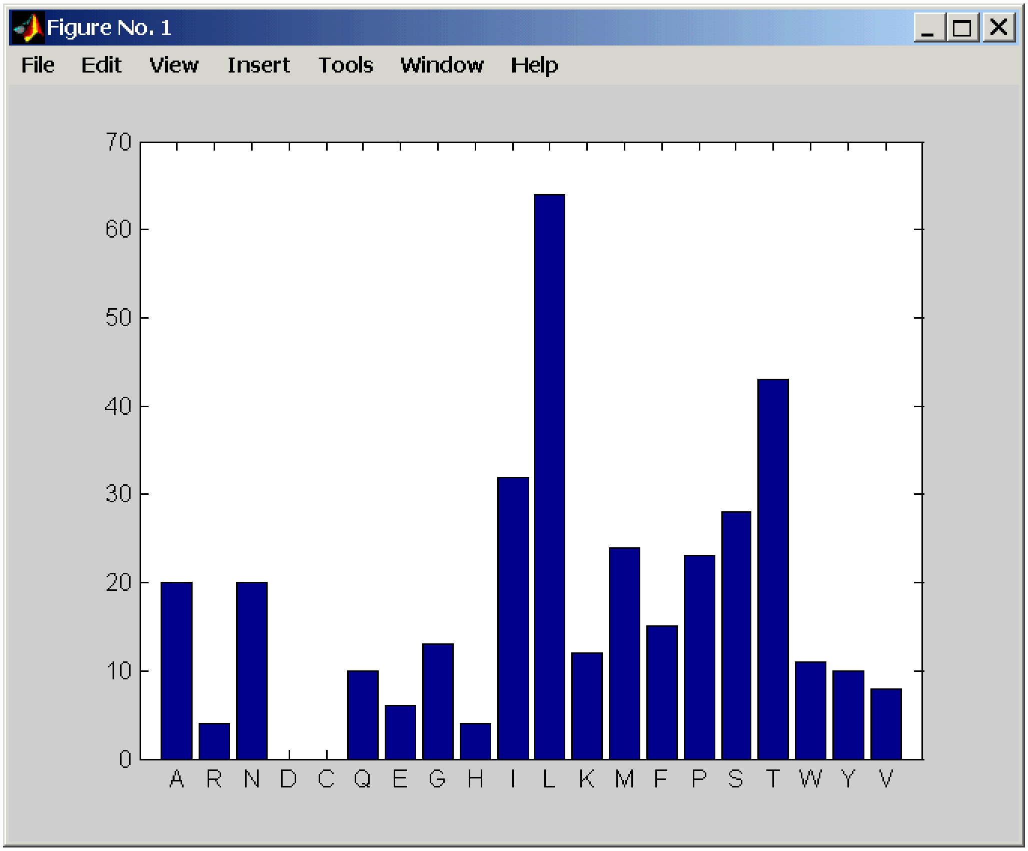
```
ND2protein = getgenpept('YP_003024027','sequenceonly',true)
```

The `getgenpept` function retrieves 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')
```

A bar graph displays. 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)
```

The following displays in the MATLAB Workspace:

```
ans =

      C: 1818
      H: 2882
      N: 420
      O: 471
```

```
        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.

Importing, Viewing, and Exploring a Nucleotide Sequence Using a Graphical Interface

In this section...

Overview of the Sequence Tool

The Sequence Tool window integrates many of the sequence functions in the toolbox. Instead of entering commands in the MATLAB Command Window, you can select and enter options.

Importing a Sequence into the Sequence Tool

The first step when analyzing a nucleotide or amino acid sequence is to import sequence information into the MATLAB environment. The Sequence Tool can connect to Web databases such as NCBI and EMBL and read information into the MATLAB environment.

The following procedure illustrates how to retrieve sequence information from the NCBI database on the Web. This example uses the GenBank accession number NM_000520, which is the human gene HEXA that is associated with Tay-Sachs disease.

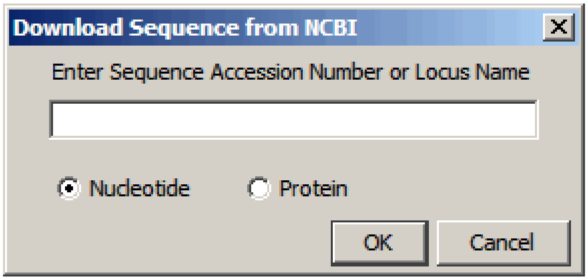
1 In the MATLAB Command Window, type

```
seqtool
```

   The Sequence Tool window opens without a sequence loaded. Notice that the panes to the right and bottom are blank.

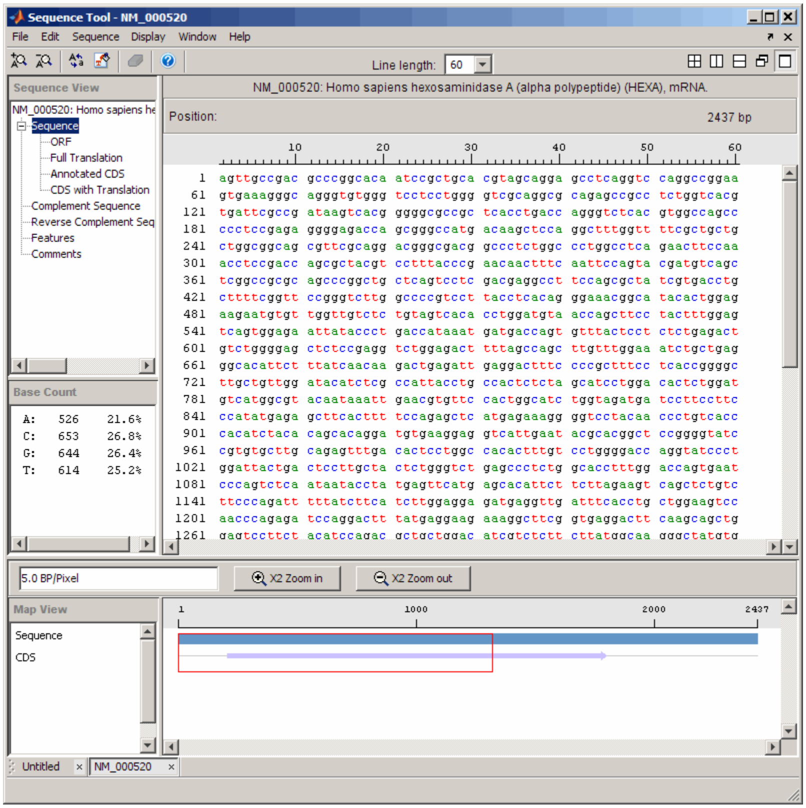
2 To retrieve a sequence from the NCBI database, select **File > Download Sequence from > NCBI**.

The Download Sequence from NCBI dialog box opens.


Download Sequence from NCBI
Enter Sequence Accession Number or Locus Name
● Nucleotide
○ Protein
OK
Cancel

3 In the **Enter Sequence** box, type an accession number for an NCBI database entry, for example, **NM_000520**. Click the **Nucleotide** option button, and then click **OK**.

The MATLAB software accesses the NCBI database on the Web, loads nucleotide sequence information for the accession number you entered, and calculates some basic statistics.



NM_000520: Homo sapiens hexosaminidase A (alpha polypeptide) (HEXA), mRNA.

Position: 2437 bp

```
A:   526    21.6%
C:   653    26.8%
G:   644    26.4%
T:   614    25.2%
```

```
          10         20         30         40         50         60
   1 agttgccgac gcccggcaca atccgctgca cgtagcagga gcctcaggtc caggccggaa
  61 gtgaaagggc agggtgtggg tcctcctggg gtcgcaggcg cagagccgcc tctggtcacg
 121 tgattcgccg ataagtcacg ggggcgccgc tcacctgacc agggtctcac gtggccagcc
 181 ccctccgaga ggggagacca gcgggccatg acaagctcca ggctttggtt ttcgctgctg
 241 ctggcggcag cgttcgcagg acgggcgacg gccctctggc cctggcctca gaacttccaa
 301 acctccgacc agcgctacgt cctttacccg aacaactttc aattccagta cgatgtcagc
 361 tcggccgcgc agcccggctg ctcagtcctc gacgaggcct tccagcgcta tcgtgacctg
 421 cttttcggtt ccgggtcttg gccccgtcct tacctcacag ggaaacggca tacactggag
 481 aagaatgtgt tggttgtctc tgtagtcaca cctggatgta accagcttcc tactttggag
 541 tcagtggaga attataccct gaccataaat gatgaccagt gtttactcct ctctgagact
 601 gtctggggag ctctccgagg tctggagact tttagccagc ttgtttggaa atctgctgag
 661 ggcacattct ttatcaacaa gactgagatt gaggactttc cccgctttcc tcaccggggc
 721 ttgctgttgg atacatctcg ccattacctg ccactctcta gcatcctgga cactctggat
 781 gtcatggcgt acaataaatt gaacgtgttc cactggcatc tggtagatga tccttccttc
 841 ccatatgaga gcttcacttt tccagagctc atgagaaagg ggtcctacaa ccctgtcacc
 901 cacatctaca cagcacagga tgtgaaggag gtcattgaat acgcacggct ccggggtatc
 961 cgtgtgcttg cagagtttga cactcctggc cacactttgt cctggggacc aggtatccct
1021 ggattactga ctccttgcta ctctgggtct gagccctctg gcacctttgg accagtgaat
1081 cccagtctca ataataccta tgagttcatg agcacattct tcttagaagt cagctctgtc
1141 ttcccagatt tttatcttca tcttggagga gatgaggttg atttcacctg ctggaagtcc
1201 aacccagaga tccaggactt tatgaggaag aaaggcttcg gtgaggactt caagcagctg
1261 gagtccttct acatccagac gctgctggac atcgtctctt cttatggcaa gggctatgtg
```

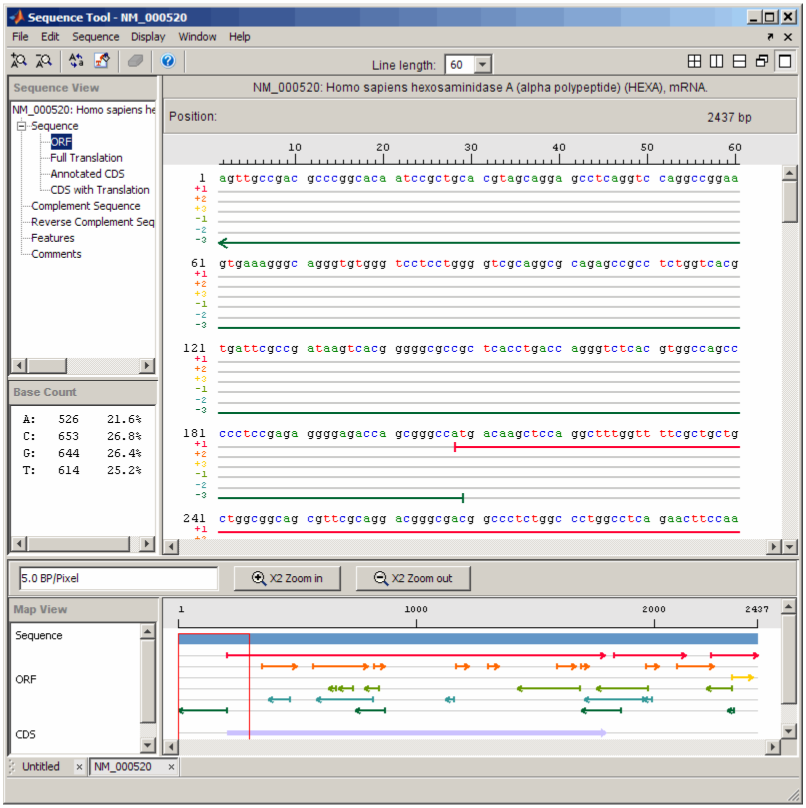
Viewing Nucleotide Sequence Information

After you import a sequence into the Sequence Tool window, you can read information stored with the sequence, or you can view graphic representations for ORFs and CDSs.

1 In the left pane tree, click **Comments**. The right pane displays general information about the sequence.

2 Now click **Features**. The right pane displays NCBI feature information, including index numbers for a gene and any CDS sequences.

3 Click **ORF** to show the search results for ORFs in the six reading frames.



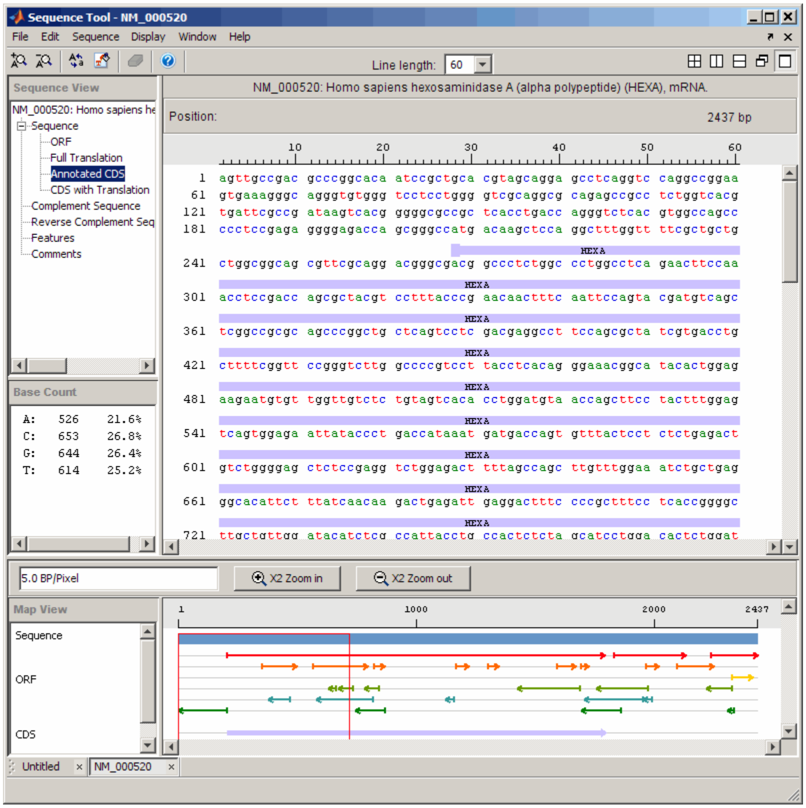
NM_000520: Homo sapiens hexosaminidase A (alpha polypeptide) (HEXA), mRNA.

Position: 2437 bp

```
  1 agttgccgac gcccggcaca atccgctgca cgtagcagga gcctcaggtc caggccggaa
 61 gtgaaagggc agggtgtggg tcctcctggg gtcgcaggcg cagagccgcc tctggtcacg
121 tgattcgccg ataagtcacg ggggcgccgc tcacctgacc agggtctcac gtggccagcc
181 ccctccgaga ggggagacca gcgggccatg acaagctcca ggctttggtt ttcgctgctg
241 ctggcggcag cgttcgcagg acgggcgacg gccctctggc cctggcctca gaacttccaa
```

```
A:   526    21.6%
C:   653    26.8%
G:   644    26.4%
T:   614    25.2%
```

4 Click **Annotated CDS** to show the protein coding part of a nucleotide sequence.



NM_000520: Homo sapiens hexosaminidase A (alpha polypeptide) (HEXA), mRNA.

Position: 2437 bp

```
A:   526    21.6%
C:   653    26.8%
G:   644    26.4%
T:   614    25.2%
```

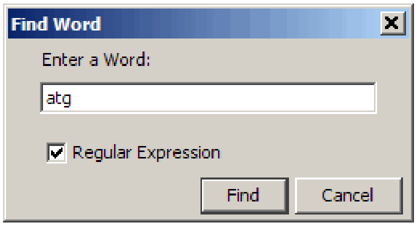
```
              10         20         30         40         50         60
  1 agttgccgac gcccggcaca atccgctgca cgtagcagga gcctcaggtc caggccggaa
 61 gtgaaagggc agggtgtggg tcctcctggg gtcgcaggcg cagagccgcc tctggtcacg
121 tgattcgccg ataagtcacg gggcgccgc tcacctgacc agggtctcac gtggccagcc
181 ccctccgaga ggggagacca gcgggccatg acaagctcca ggctttggtt ttcgctgctg
                                                HEXA
241 ctggcggcag cgttcgcagg acgggcgacg gccctctggc cctggcctca gaacttccaa
                                HEXA
301 acctccgacc agcgctacgt cctttacccg aacaactttc aattccagta cgatgtcagc
                                HEXA
361 tcggccgcgc agcccggctg ctcagtcctc gacgaggcct tccagcgcta tcgtgacctg
                                HEXA
421 cttttcggtt ccgggtcttg gccccgtcct tacctcacag ggaaacggca tacactggag
                                HEXA
481 aagaatgtgt tggttgtctc tgtagtcaca cctggatgta accagcttcc tactttggag
                                HEXA
541 tcagtggaga attataccct gaccataaat gatgaccagt gtttactcct ctctgagact
                                HEXA
601 gtctggggag ctctccgagg tctggagact tttagccagc ttgtttggaa atctgctgag
                                HEXA
661 ggcacattct ttatcaacaa gactgagatt gaggactttc cccgctttcc tcaccggggc
                                HEXA
721 ttgctgttgg atacatctcg ccattacctg ccactctcta gcatcctgga cactctggat
```

Searching for Words

The following procedure illustrates how to search for characteristic words and sequence patterns. You will search for sequence patterns like the TATAA box and patterns for specific restriction enzymes.

1 Select **Sequence > Find Word**.

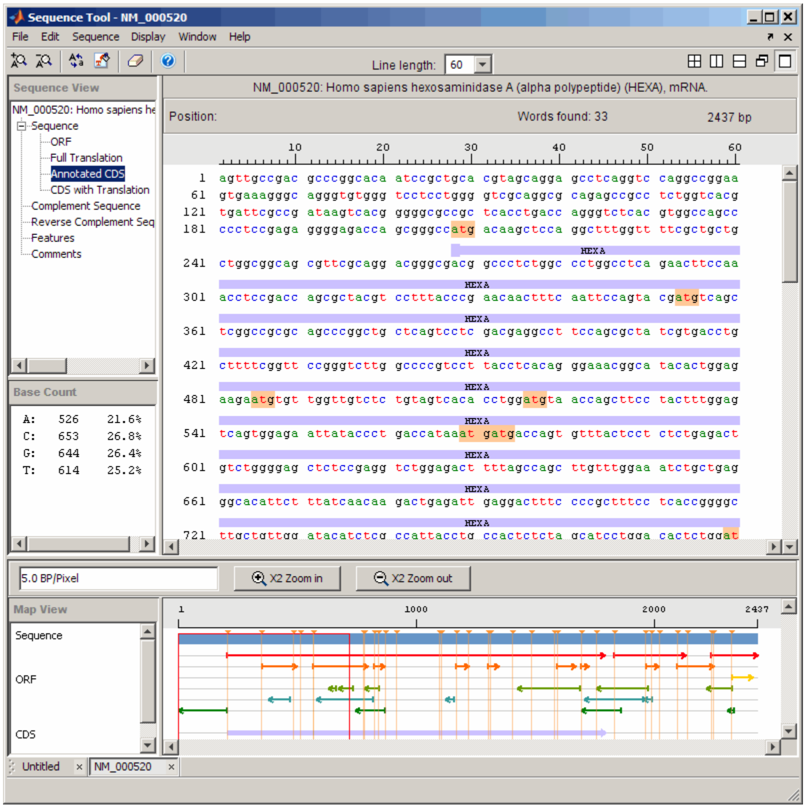
2 In the Find Word dialog box, type a sequence word or pattern, for example, **atg**, and then click **Find**.



Enter a Word: atg

☑ Regular Expression

The Sequence Tool window searches and displays the location of the selected word.



Line length: 60

NM_000520: Homo sapiens hexosaminidase A (alpha polypeptide) (HEXA), mRNA.

Position: Words found: 33 2437 bp

```
            10         20         30         40         50         60
  1 agttgccgac gcccggcaca atccgctgca cgtagcagga gcctcaggtc caggccggaa
 61 gtgaaagggc agggtgtggg tcctcctggg gtcgcaggcg cagagccgcc tctggtcacg
121 tgattcgccg ataagtcacg gggcgccgc tcacctgacc agggtctcac gtggccagcc
181 ccctccgaga ggggagacca gcgggccatg acaagctcca ggctttggtt ttcgctgctg
                                                    HEXA
241 ctggcggcag cgttcgcagg acgggcgacg gccctctggc cctggcctca gaacttccaa
                                  HEXA
301 acctccgacc agcgctacgt cctttacccg aacaactttc aattccagta cgatgtcagc
                                  HEXA
361 tcggccgcgc agcccggctg ctcagtcctc gacgaggcct tccagcgcta tcgtgacctg
                                  HEXA
421 cttttcggtt ccgggtcttg gccccgtcct tacctcacag ggaaacggca tacactggag
                                  HEXA
481 aagaatgtgt tggttgtctc tgtagtcaca cctggatgta accagcttcc tactttggag
                                  HEXA
541 tcagtggaga attataccct gaccataaat gatgaccagt gtttactcct ctctgagact
                                  HEXA
601 gtctggggag ctctccgagg tctggagact tttagccagc ttgtttggaa atctgctgag
                                  HEXA
661 ggcacattct ttatcaacaa gactgagatt gaggactttc cccgctttcc tcaccggggc
                                  HEXA
721 ttgctgttgg atacatctcg ccattacctg ccactctcta gcatcctgga cactctggat
```

```
A:   526   21.6%
C:   653   26.8%
G:   644   26.4%
T:   614   25.2%
```

5.0 BP/Pixel

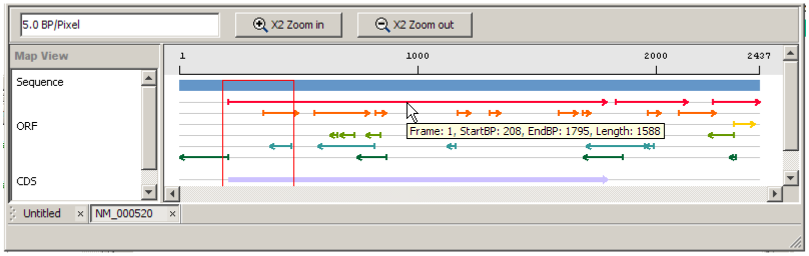
3 Clear the display by clicking the Clear Word Selection button on the toolbar.

Exploring Open Reading Frames

The following procedure illustrates how to identify the protein coding part of a nucleotide sequence and copy it into a new view. Identifying coding sections of a nucleotide sequence is a common bioinformatics task. After locating the coding part of a sequence, you can copy it to a new view, translate it to an amino acid sequence, and continue with your analysis.

1 In the left pane, click **ORF**.

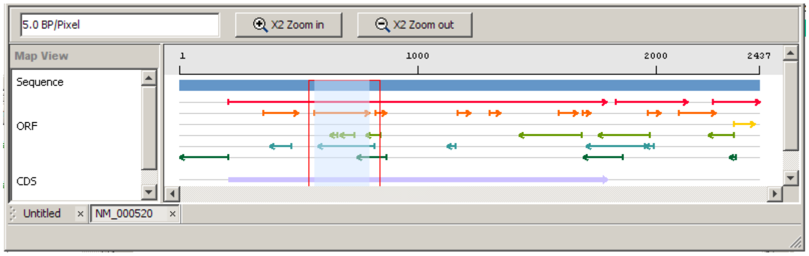
The Sequence Tool window displays the ORFs for the six reading frames in the lower-right pane. Hover the cursor over a frame to display information about it.



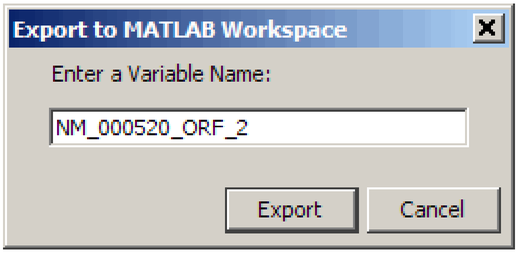
Frame: 1, StartBP: 208, EndBP: 1795, Length: 1588

2 Click the longest ORF on reading frame 2.

The ORF is highlighted to indicate the part of the sequence that is selected.



3 Right-click the selected ORF and then select **Export to Workspace**. In the Export to MATLAB Workspace dialog box, type a variable name, for example, **NM_000520_ORF_2**, then click **Export**.

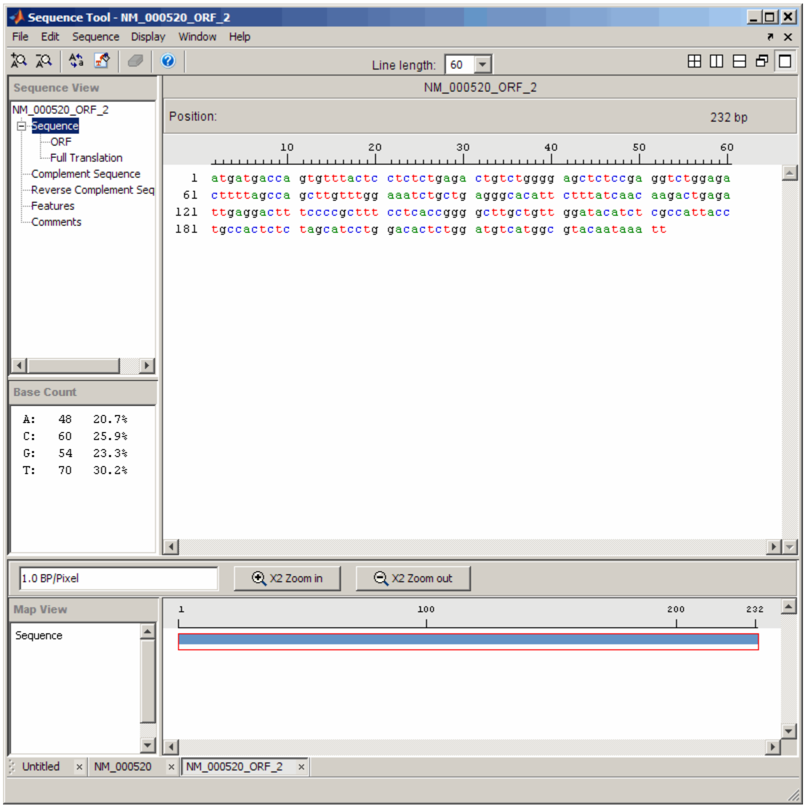


Enter a Variable Name: NM_000520_ORF_2

The **NM_000520_ORF_2** variable is added to the MATLAB Workspace.

4 Select **File > Import from Workspace**. Type the name of a variable with an exported ORF, for example, **NM_000520_ORF_2**, and then click **Import**.

The Sequence Tool window adds a tab at the bottom for the new sequence while leaving the original sequence open.



NM_000520_ORF_2

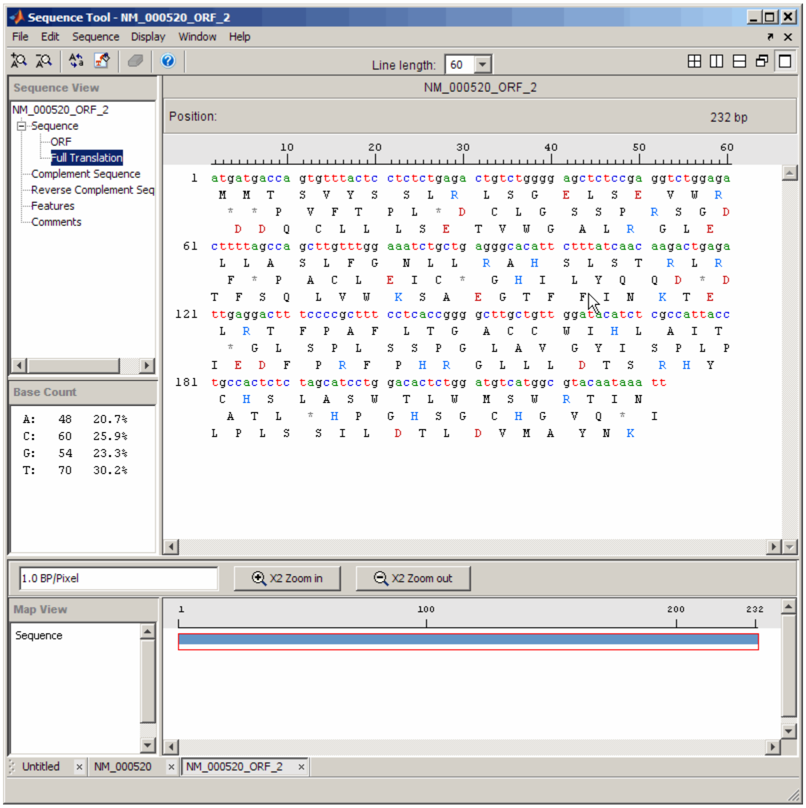
Position: 232 bp

```
          10         20         30         40         50         60
  1 atgatgacca gtgtttactc ctctctgaga ctgtctgggg agctctccga ggtctggaga
 61 cttttagcca gcttgtttgg aaatctgctg agggcacatt ctttatcaac aagactgaga
121 ttgaggactt tccccgcttt cctcaccggg gcttgctgtt ggatacatct cgccattacc
181 tgccactctc tagcatcctg gacactctgg atgtcatggc gtacaataaa tt
```

```
A:   48   20.7%
C:   60   25.9%
G:   54   23.3%
T:   70   30.2%
```

5 In the left pane, click **Full Translation**. Select **Display > Amino Acid Residue Display > One Letter Code**.

The Sequence Tool window displays the amino acid sequence below the nucleotide sequence.



Line length: 60

NM_000520_ORF_2

Position: 232 bp

```
          10         20         30         40         50         60
  1 atgatgacca gtgtttactc ctctctgaga ctgtctgggg agctctccga ggtctggaga
     M  M  T   S  V  Y  S   S  L  R   L  S  G   E  L  S  E   V  W  R
      *  *  P   V  F  T   P  L  *  D   C  L  G   S  S  P   R  S  G  D
       D  D  Q   C  L  L   L  S  E   T  V  W  G   A  L  R   G  L  E
 61 cttttagcca gcttgtttgg aaatctgctg agggcacatt ctttatcaac aagactgaga
     L  L  A   S  L  F  G   N  L  L   R  A  H   S  L  S  T   R  L  R
      F  *  P   A  C  L   E  I  C  *   G  H  I   L  Y  Q   Q  D  *  D
    T  F  S  Q   L  V  W   K  S  A   E  G  T  F   F  I  N   K  T  E
121 ttgaggactt tccccgcttt cctcaccggg gcttgctgtt ggatacatct cgccattacc
     L  R  T   F  P  A  F   L  T  G   A  C  C   W  I  H  L   A  I  T
      *  G  L   S  P  L   S  S  P  G   L  A  V   G  Y  I   S  P  L  P
    I  E  D  F   P  R  F   P  H  R   G  L  L  L   D  T  S   R  H  Y
181 tgccactctc tagcatcctg gacactctgg atgtcatggc gtacaataaa tt
     C  H  S   L  A  S  W   T  L  W   M  S  W   R  T  I  N
      A  T  L   *  H  P   G  H  S  G   C  H  G   V  Q  *   I
    L  P  L  S   S  I  L   D  T  L   D  V  M  A   Y  N  K
```

```
A:   48   20.7%
C:   60   25.9%
G:   54   23.3%
T:   70   30.2%
```

1.0 BP/Pixel

Closing the Sequence Tool

Close the Sequence Tool window from the MATLAB command line using the following syntax:

```
seqtool('close')
```

Importing, Viewing, and Exploring a Protein Sequence Using a Graphical Interface

In this section...

Overview of the Sequence Tool

The Sequence Tool window integrates many of the sequence functions in the toolbox. Instead of entering commands in the MATLAB Command Window, you can select and enter options.

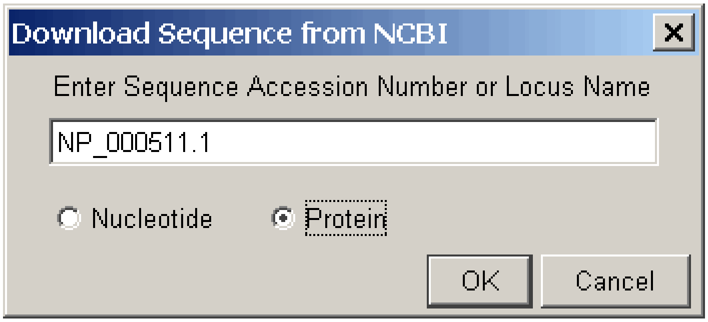
Viewing Amino Acid Sequence Statistics

The following procedure illustrates how to view an amino acid sequence for an ORF located in a nucleotide sequence. You can import your own amino acid sequence, or you can get a protein sequence from the GenBank database. This example uses the GenBank accession number NP_000511.1, which is the alpha subunit for a human enzyme associated with Tay-Sachs disease.

1 Select **File > Download Sequence from > NCBI**.

   The Download Sequence from NCBI dialog box opens.

2 In the **Enter Sequence** box, type an accession number for an NCBI database entry, for example, **NP_000511.1**. Click the **Protein** option button, and then click **OK**.



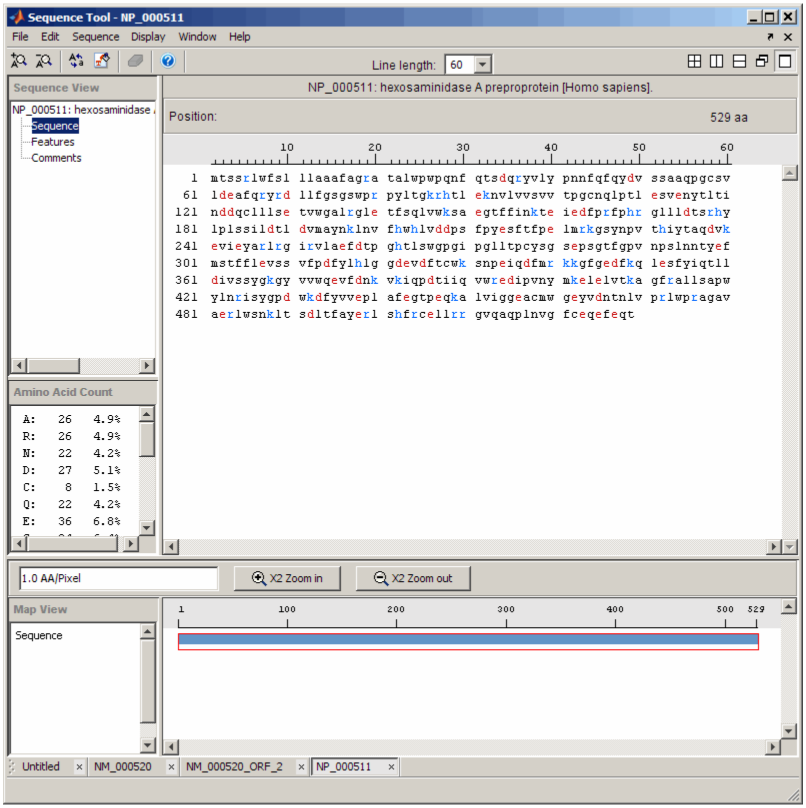
Download Sequence from NCBI

Enter Sequence Accession Number or Locus Name

NP_000511.1

○ Nucleotide ● Protein

The MATLAB software accesses the NCBI database on the Web and loads amino acid sequence information for the accession number you entered.



Line length: 60

NP_000511: hexosaminidase A preproprotein [Homo sapiens].

Position: 529 aa

```
             10         20         30         40         50         60
  1 mtssrlwfsl llaaafagra talwpwpqnf qtsdqryvly pnnfqfqydv ssaaqpgcsv
 61 ldeafqryrd llfgsgswpr pyltgkrhtl eknvlvvsvv tpgcnqlptl esvenytlti
121 nddqclllse tvwgalrgle tfsqlvwksa egtffinkte iedfprfphr gllldtsrhy
181 lplssildtl dvmaynklnv fhwhlvddps fpyesftfpe lmrkgsynpv thiytaqdvk
241 evieyarlrg irvlaefdtp ghtlswgpgi pglltpcysg sepsgtfgpv npslnntyef
301 mstfflevss vfpdfylhlg gdevdftcwk snpeiqdfmr kkgfgedfkq lesfyiqtll
361 divssygkgy vvwqevfdnk vkiqpdtiiq vwredipvny mkelelvtka gfrallsapw
421 ylnrisygpd wkdfyvvepl afegtpeqka lviggeacmw geyvdntnlv prlwpragav
481 aerlwsnklt sdltfayerl shfrcellrr gvqaqplnvg fceqefeqt
```

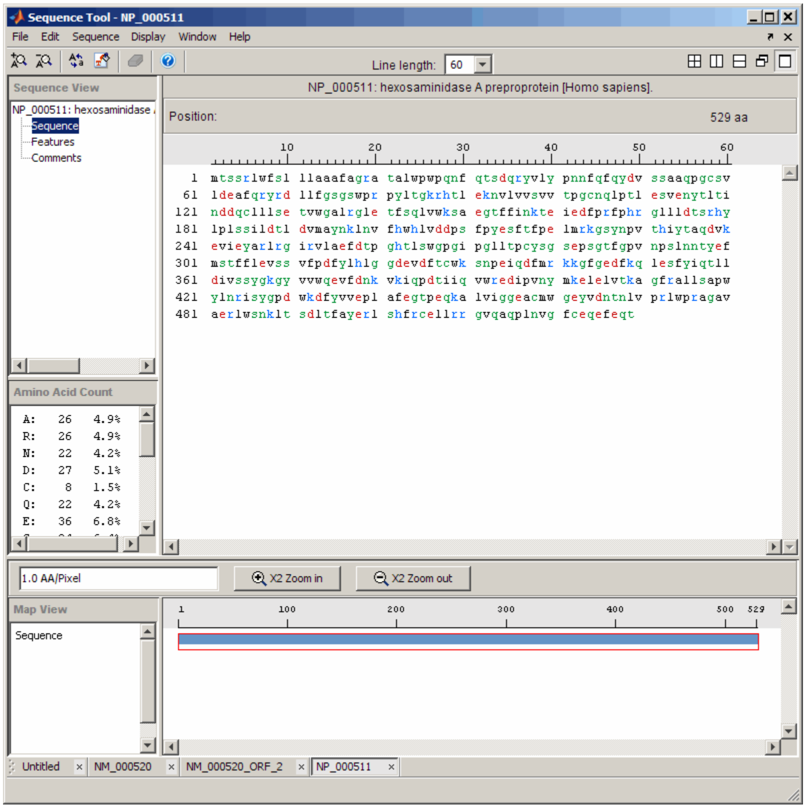
Amino Acid Count

```
A:   26   4.9%
R:   26   4.9%
N:   22   4.2%
D:   27   5.1%
C:    8   1.5%
Q:   22   4.2%
E:   36   6.8%
```

1.0 AA/Pixel

3 Select **Display > Amino Acid Color Scheme**, and then select **Charge**, **Function**, **Hydrophobicity**, **Structure**, or **Taylor**. For example, select **Function**.

The display colors change to highlight charge information about the amino acid residues. The following table shows color legends for the amino acid color schemes.



NP_000511: hexosaminidase A preproprotein [Homo sapiens].

Position: 529 aa

```
                10         20         30         40         50         60
  1 mtssrlwfsl llaaafagra talwpwpqnf qtsdqryvly pnnfqfqydv ssaaqpgcsv
 61 ldeafqryrd llfgsgswpr pyltgkrhtl eknvlvvsvv tpgcnqlptl esvenytlti
121 nddqcllllse tvwgalrgle tfsqlvwksa egtffinkte iedfprfphr gllldtsrhy
181 lplssildtl dvmaynklnv fhwhlvddps fpyesftfpe lmrkgsynpv thiytaqdvk
241 evieyarlrg irvlaefdtp ghtlswgpgi pglltpcysg sepsgtfgpv npslnntyef
301 mstfflevss vfpdfylhlg gdevdftcwk snpeiqdfmr kkgfgedfkq lesfyiqtll
361 divssygkgy vvwqevfdnk vkiqpdtiiq vwredipvny mkelelvtka gfrallsapw
421 ylnrisygpd wkdfyvvepl afegtpeqka lviggeacmw geyvdntnlv prlwpragav
481 aerlwsnklt sdltfayerl shfrcellrr gvqaqplnvg fceqefeqt
```

```
A:   26   4.9%
R:   26   4.9%
N:   22   4.2%
D:   27   5.1%
C:    8   1.5%
Q:   22   4.2%
E:   36   6.8%
```

1.0 AA/Pixel

Amino Acid Color Scheme	Color Legend
Charge	• Acidic — Red • Basic — Light Blue • Neutral — Black
Function	• Acidic — Red • Basic — Light Blue • Hydropobic, nonpolar — Black • Polar, uncharged — Green
Hydrophobicity	• Hydrophilic — Light Blue • Hydrophobic — Black
Structure	• Ambivalent — Dark Green • External — Light Blue • Internal — Orange
Taylor	Each amino acid is assigned its own color, based on the colors proposed by W.R. Taylor.

Closing the Sequence Tool

Close the Sequence Tool window from the MATLAB command line using the following syntax:

```
seqtool('close')
```

References

[1] Taylor, W.R. (1997). Residual colours: a proposal for aminochromography. Protein Engineering *10, 7*, 743–746.

Example: Sequence Alignment

In this section...

Overview of 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 and verify a corresponding gene in a model organism.

Finding a Model Organism to Study

The following procedure illustrates how to use the MATLAB Help browser to search the Web for information. 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.

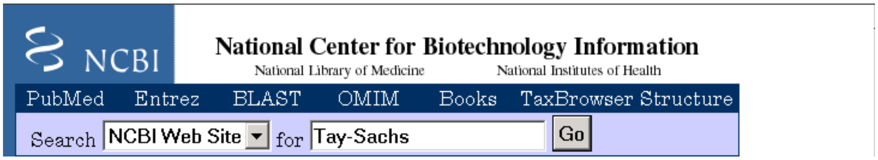
First, research information about Tay-Sachs and the enzyme that is associated with this disease, then find the nucleotide sequence for the human gene that codes for the enzyme, and finally find a corresponding gene in another organism to use as a model for study.

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

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

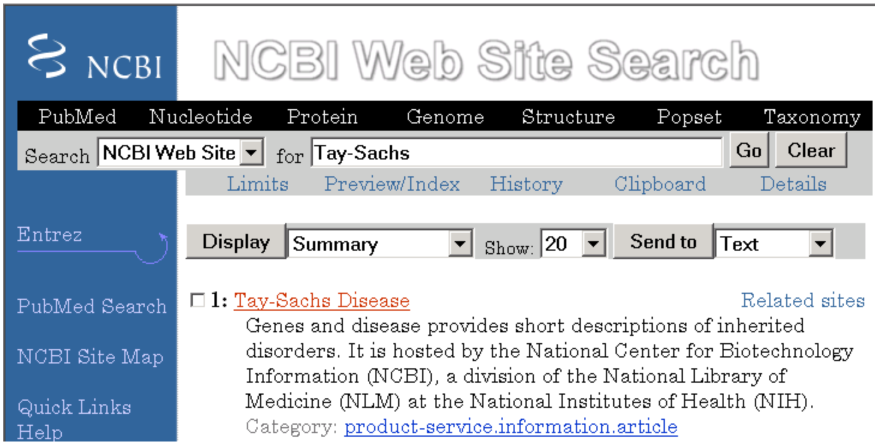
The MATLAB Help browser opens with the home page for the NCBI Web site.

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
Go

Search NCBI Web Site for Tay-Sachs

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



Search NCBI Web Site for Tay-Sachs

Summary Show: 20 Text

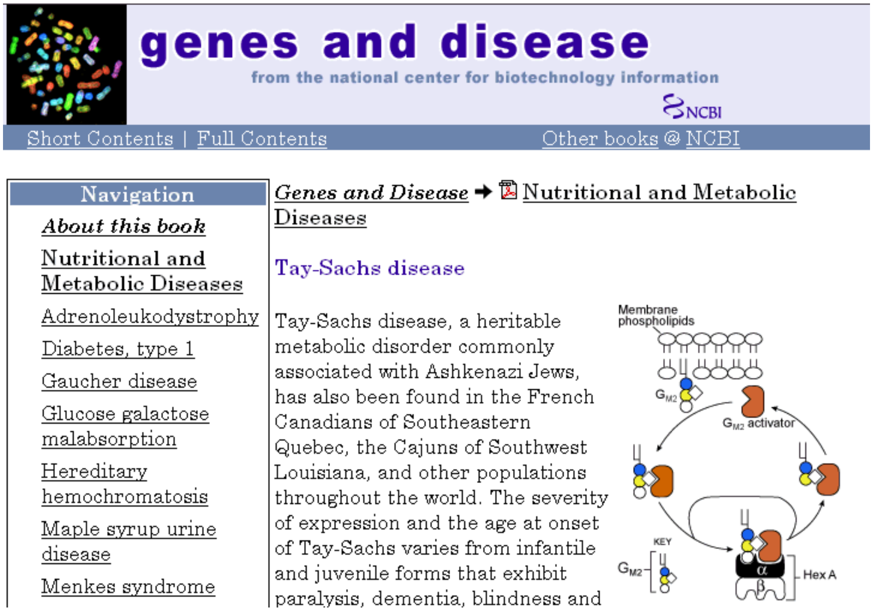
□ 1: Tay-Sachs Disease

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.

Retrieving Sequence Information from a Public Database

The following procedure illustrates how to find the nucleotide sequence for a human gene in a public database and read the sequence information into the MATLAB environment. 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 the MATLAB environment.

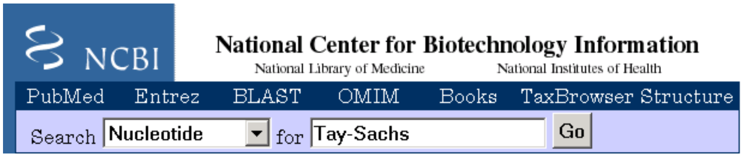
After you locate a sequence, you need to move the sequence data into the MATLAB Workspace.

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

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

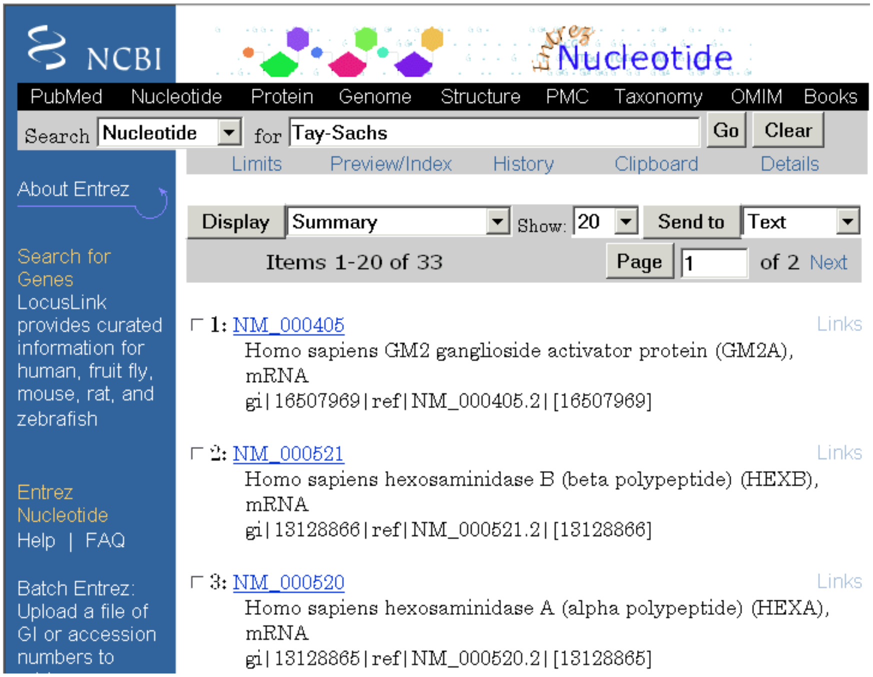
The MATLAB Help browser window opens with the NCBI home page.

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
Go

Search Nucleotide for Tay-Sachs

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 Nucleotide for Tay-Sachs

Display Summary Show: 20 Send to Text

Items 1-20 of 33 Page 1 of 2 Next

About Entrez

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

Entrez Nucleotide
Help | FAQ

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

□ 1: NM_000405
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 the MATLAB environment. For example, to get sequence information for the human gene HEXA, type

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

Note 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: 'NM_000520'
      LocusSequenceLength: '2255'
     LocusNumberofStrands: ''
            LocusTopology: 'linear'
```

```
   LocusMoleculeType: 'mRNA'
  LocusGenBankDivision: 'PRI'
 LocusModificationDate: '13-AUG-2006'
            Definition: 'Homo sapiens hexosaminidase A (alpha polypeptide) (HEXA), mRNA.'
             Accession: 'NM_000520'
               Version: 'NM_000520.2'
                    GI: '13128865'
               Project: []
              Keywords: []
               Segment: []
                Source: 'Homo sapiens (human)'
        SourceOrganism: [4x65 char]
             Reference: {1x58 cell}
               Comment: [15x67 char]
              Features: [74x74 char]
                   CDS: [1x1 struct]
              Sequence: [1x2255 char]
             SearchURL: [1x108 char]
           RetrieveURL: [1x97 char]
```

Searching a Public Database for Related Genes

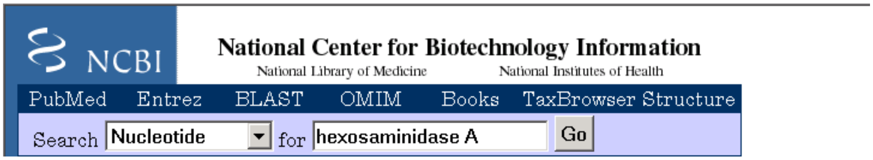
The following procedure illustrates how to find the nucleotide sequence for a mouse gene related to a human gene, and read the sequence information into the MATLAB environment. The sequence and function of many genes is conserved during the evolution of species through homologous genes. Homologous genes 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

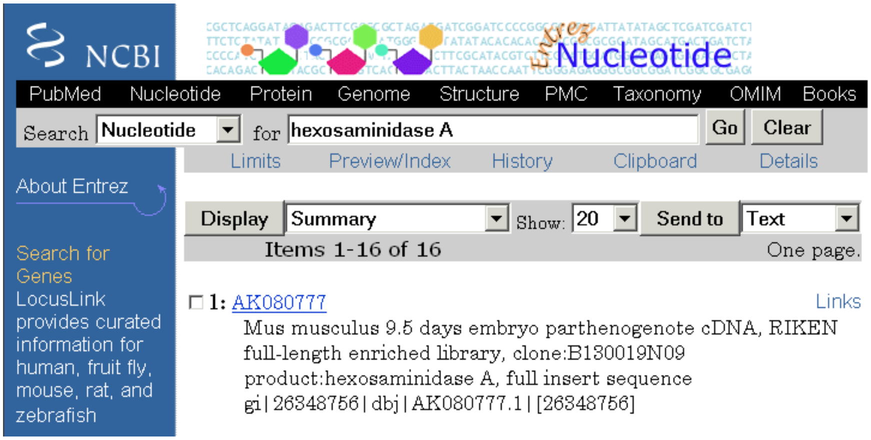
```
web('http://www.ncbi.nlm.nih.gov')
```

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
Go

Search Nucleotide for hexosaminidase A

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



Search Nucleotide for hexosaminidase A

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

Display Summary Show: 20 Send to Text

Items 1-16 of 16 One page.

□ 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 the MATLAB environment. Type

```
mouseHEXA = getgenbank('AK080777')
```

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: '02-SEP-2005'
               Definition: [1x150 char]
                Accession: 'AK080777'
                  Version: 'AK080777.1'
                       GI: '26348756'
                  Project: []
                 Keywords: 'HTC; CAP trapper.'
                  Segment: []
                   Source: 'Mus musculus (house mouse)'
           SourceOrganism: [4x65 char]
                Reference: {1x8 cell}
                  Comment: [8x66 char]
                 Features: [33x74 char]
                      CDS: [1x1 struct]
                 Sequence: [1x1839 char]
                SearchURL: [1x107 char]
              RetrieveURL: [1x97 char]
```

Locating Protein Coding Sequences

The following procedure illustrates how to convert a sequence from nucleotides to amino acids and identify the open reading frames. 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 software. In the MATLAB Command window, type

```
load hexosaminidase
```

The structures humanHEXA and mouseHEXA load into the MATLAB Workspace.

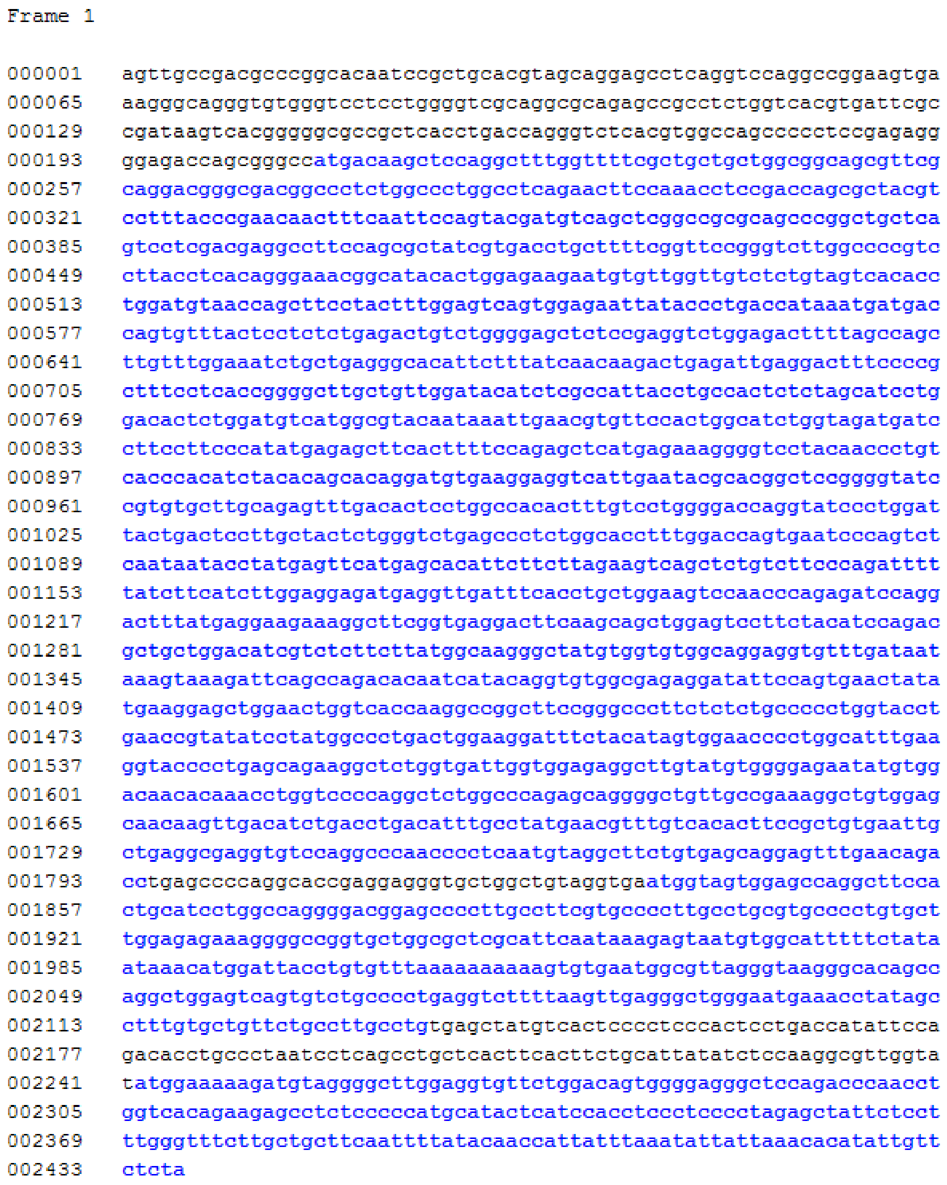
2 Locate open reading frames (ORFs) in the human gene. For example, for the human gene HEXA, type

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

seqshoworfs creates the output structure humanORFs. This structure contains 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 displaying the three reading frames with the ORFs colored blue, red, and green. Notice that the longest ORF is in the first reading frame.



```
Frame 1

000001   agttgccgacgcccggcacaatccgctgcacgtagcaggagcctcaggtccaggccggaagtga
000065   aagggcagggtgtgggtcctcctggggtcgcaggcgcagagccgcctctggtcacgtgattcgc
000129   cgataagtcacgggggcgccgctcacctgaccagggtctcacgtggccagccccctccgagagg
000193   ggagaccagcgggccatgacaagctccaggctttggttttcgctgctgctggcggcagcgttcg
000257   caggacgggcgacggccctctggccctggcctcagaacttccaaacctccgaccagcgctacgt
000321   cctttacccgaacaactttcaattccagtacgatgtcagctcggccgcgcagcccggctgctca
000385   gtcctcgacgaggccttccagcgctatcgtgacctgcttttcggttccgggtcttggccccgtc
000449   cttacctcacagggaaacggcatacactggagaagaatgtgttggttgtctctgtagtcacacc
000513   tggatgtaaccagcttcctactttggagtcagtggagaattataccctgaccataaatgatgac
000577   cagtgtttactcctctctgagactgtctggggagctctccgaggtctggagacttttagccagc
000641   ttgtttggaaatctgctgagggcacattctttatcaacaagactgagattgaggactttccccg
000705   ctttcctcaccggggcttgctgttggatacatctcgccattacctgccactctctagcatcctg
000769   gacactctggatgtcatggcgtacaataaattgaacgtgttccactggcatctggtagatgatc
000833   cttccttcccatatgagagcttcacttttccagagctcatgagaaaggggtcctacaaccctgt
000897   cacccacatctacacagcacaggatgtgaaggaggtcattgaatacgcacggctccggggtatc
000961   cgtgtgcttgcagagtttgacactcctggccacactttgtcctggggaccaggtatccctggat
001025   tactgactccttgctactctgggtctgagccctctggcacctttggaccagtgaatcccagtct
001089   caataatacctatgagttcatgagcacattcttcttagaagtcagctctgtcttcccagatttt
001153   tatcttcatcttggaggagatgaggttgatttcacctgctggaagtccaacccagagatccagg
001217   actttatgaggaagaaaggcttcggtgaggacttcaagcagctggagtccttctacatccagac
001281   gctgctggacatcgtctcttcttatggcaagggctatgtggtgtggcaggaggtgtttgataat
001345   aaagtaaagattcagccagacacaatcatacaggtgtggcgagaggatattccagtgaactata
001409   tgaaggagctggaactggtcaccaaggccggcttccgggcccttctctctgccccctggtacct
001473   gaaccgtatatcctatggccctgactggaaggatttctacatagtggaacccctggcatttgaa
001537   ggtacccctgagcagaaggctctggtgattggtggagaggcttgtatgtggggagaatatgtgg
001601   acaacacaaacctggtccccaggctctggcccagagcaggggctgttgccgaaaggctgtggag
001665   caacaagttgacatctgacctgacatttgcctatgaacgtttgtcacacttccgctgtgaattg
001729   ctgaggcgaggtgtccaggcccaacccctcaatgtaggcttctgtgagcaggagtttgaacaga
001793   cctgagccccaggcaccgaggagggtgctggctgtaggtgaatggtagtggagccaggcttcca
001857   ctgcatcctggccaggggacggagccccttgccttcgtgccccttgcctgcgtgcccctgtgct
001921   tggagagaaaggggccggtgctggcgctcgcattcaataaagagtaatgtggcatttttctata
001985   ataaacatggattacctgtgtttaaaaaaaaaagtgtgaatggcgttagggtaagggcacagcc
002049   aggctggagtcagtgtctgcccctgaggtcttttaagttgagggctgggaatgaaacctatagc
002113   ctttgtgctgttctgccttgcctgtgagctatgtcactcccctcccactcctgaccatattcca
002177   gacacctgccctaatcctcagcctgctcacttcacttctgcattatatctccaaggcgttggta
002241   tatggaaaaagatgtaggggcttggaggtgttctggacagtggggagggctccagacccaacct
002305   ggtcacagaagagcctctcccccatgcatactcatccacctccctcccctagagctattctcct
002369   ttgggtttcttgctgcttcaattttatacaaccattatttaaatattattaaacacatattgtt
002433   ctcta
```

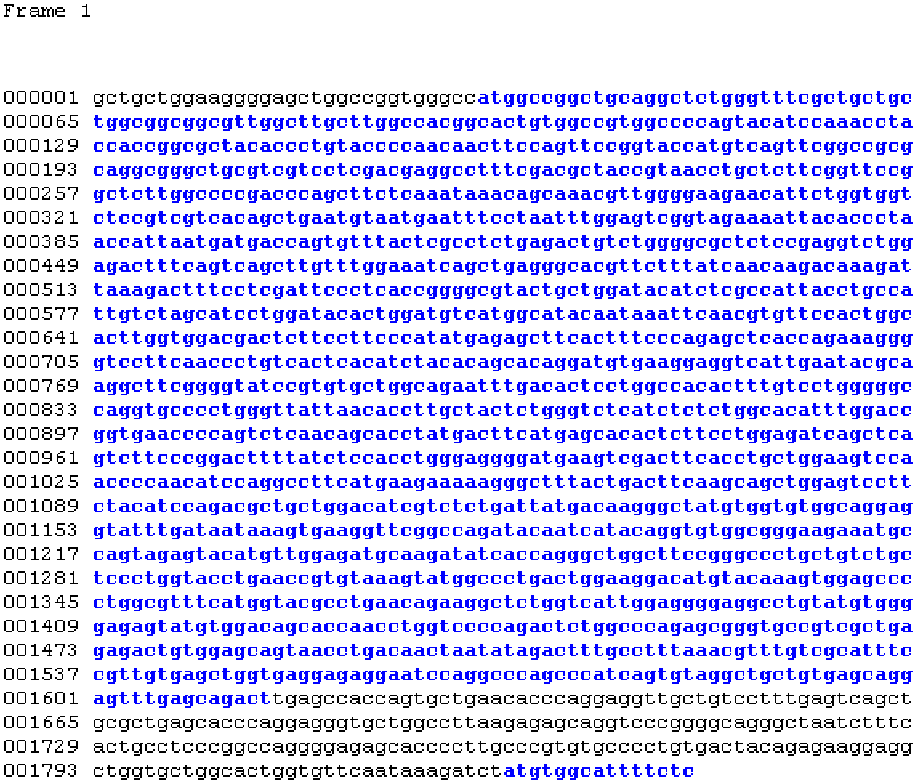
3 Locate open reading frames (ORFs) in the mouse gene. Type:

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

`seqshoworfs` creates the structure `mouseORFS`.

```
mouseORFs =

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

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



```
Frame 1


000001 gctgctggaaggggagctggccggtgggccatggccggctgcaggctctgggtttcgctgctgc
000065 tggcggcggcgttggcttgcttggccacggcactgtggccgtggccccagtacatccaaaccta
000129 ccaccggcgctacaccctgtaccccaacaacttccagttccggtaccatgtcagttcggccgcg
000193 caggcgggctgcgtcgtcctcgacgaggcctttcgacgctaccgtaacctgctcttcggttccg
000257 gctcttggccccgacccagcttctcaaataaacagcaaacgttggggaagaacattctggtggt
000321 ctccgtcgtcacagctgaatgtaatgaatttcctaatttggagtcggtagaaaattacaccctа
000385 accattaatgatgaccagtgtttactcgcctctgagactgtctggggcgctctccgaggtctgg
000449 agactttcagtcagcttgtttggaaatcagctgagggcacgttctttatcaacaagacaaagat
000513 taaagactttcctcgattccctcaccggggcgtactgctggatacatctcgccattacctgcca
000577 ttgtctagcatcctggatacactggatgtcatggcatacaataaattcaacgtgttccactggc
000641 acttggtggacgactcttccttcccatatgagagcttcactttcccagagctcaccagaaaggg
000705 gtccttcaaccctgtcactcacatctacacagcacaggatgtgaaggaggtcattgaatacgca
000769 aggcttcggggtatccgtgtgctggcagaatttgacactcctggccacactttgtcctgggggc
000833 caggtgccccctgggttattaacaccttgctactctgggtctcatctctctggcacatttggacc
000897 ggtgaaccccagtctcaacagcacctatgacttcatgagcacactcttcctggagatcagctca
000961 gtcttcccggacttttatctccacctgggaggggatgaagtcgacttcacctgctggaagtcca
001025 accccaacatccaggccttcatgaagaaaagggctttactgacttcaagcagctggagtcctt
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

The following procedure illustrates how to use global and local alignment functions to compare two 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 open reading frames identified previously, convert the human and mouse DNA sequences to the amino acid sequences. Because both the human and mouse HEXA genes were in the first reading frames (default), you do not need to indicate which frame. Type

   ```
   humanProtein = nt2aa(humanHEXA.Sequence);
   mouseProtein = nt2aa(mouseHEXA.Sequence);
   ```

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

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

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

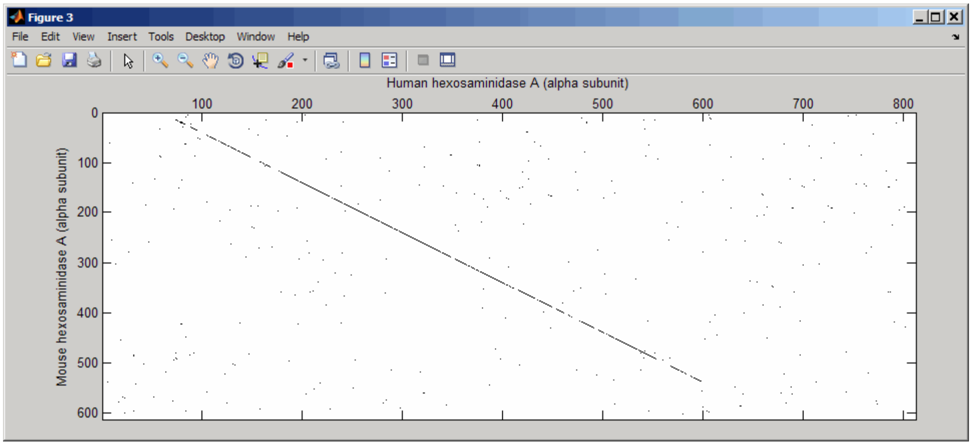
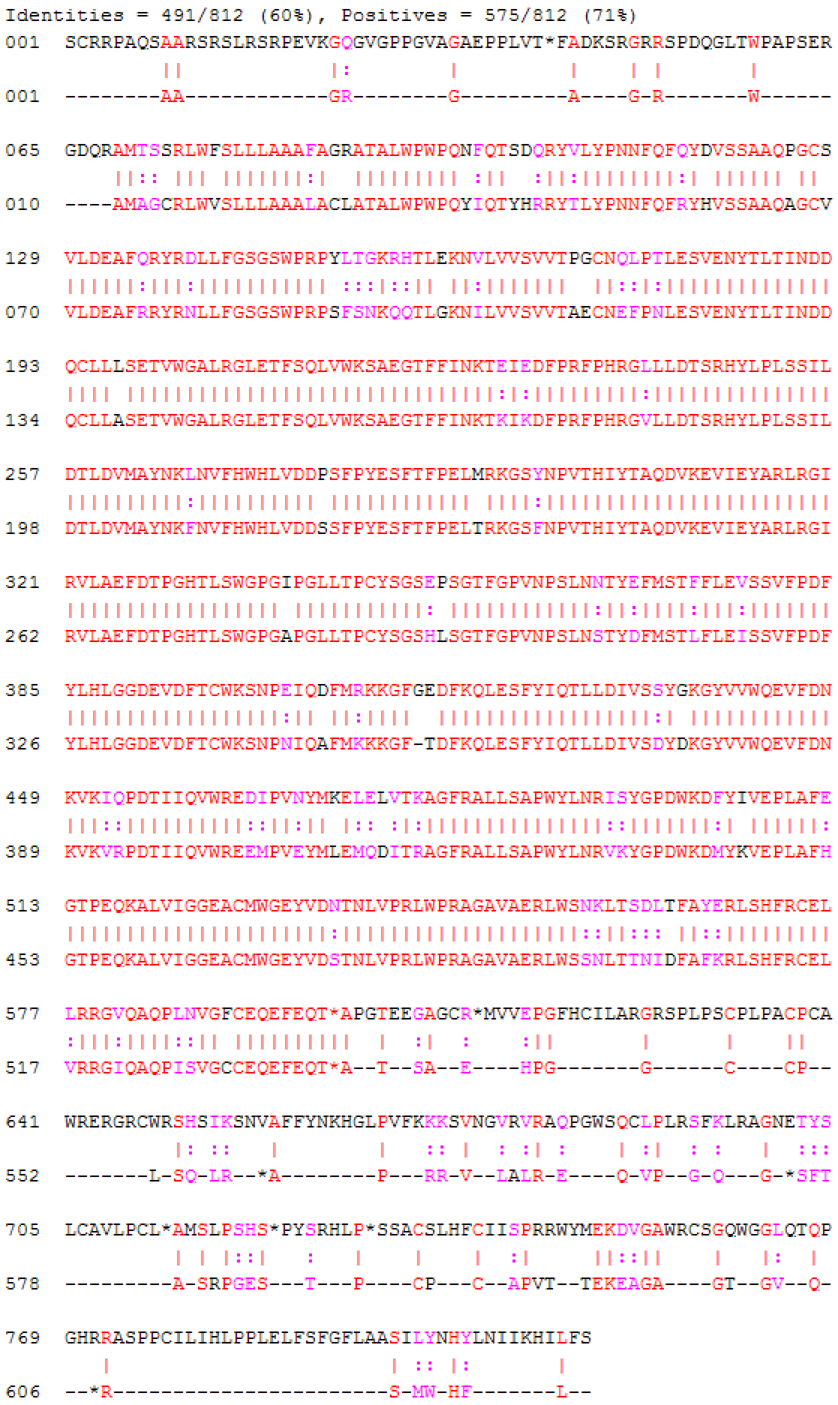


Figure 3
File Edit View Insert Tools Desktop Window Help
Human hexosaminidase A (alpha subunit)
100 200 300 400 500 600 700 800
Mouse hexosaminidase A (alpha subunit)
0 100 200 300 400 500 600

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

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

`showalignment` displays the global alignment of the two sequences in the Help browser. Notice that the calculated identity between the two sequences is 60%.



```
Identities = 491/812 (60%), Positives = 575/812 (71%)
001  SCRRPAQSAARSRSLRSRPEVKGQGVGPPGVAGAEPPLVT*FADKSRGRRSPDQGLTWPAPSER
             ||            |:        |         |    | |       |
001  --------AA------------GR--------G---------A----G-R-------W------

065  GDQRAMTSSRLWFSLLLAAAFAGRATALWPWPQNFQTSDQRYVLYPNNFQFQYDVSSAAQPGCS
         ||:: ||| |||||||:|  ||||||||| :||  :||:||||||||:| |||||| ||
010  ----AMAGCRLWVSLLLAAALACLATALWPWPQYIQTYHRRYTLYPNNFQFRYHVSSAAQAGCV

129  VLDEAFQRYRDLLFGSGSWPRPYLTGKRHTLEKNVLVVSVVTPGCNQLPTLESVENYTLTINDD
     ||||||:|||:||||||||||| :::|::|| ||:|||||||  ||::|:||||||||||||||
070  VLDEAFRRYRNLLFGSGSWPRPSFSNKQQTLGKNILVVSVVTAECNEFPNLESVENYTLTINDD

193  QCLLLSETVWGALRGLETFSQLVWKSAEGTFFINKTEIEDFPRFPHRGLLLDTSRHYLPLSSIL
     |||| |||||||||||||||||||||||||||||||:|:|||||||||:|||||||||||||||
134  QCLLASETVWGALRGLETFSQLVWKSAEGTFFINKTKIKDFPRFPHRGVLLDTSRHYLPLSSIL

257  DTLDVMAYNKLNVFHWHLVDDPSFPYESFTFPELMRKGSYNPVTHIYTAQDVKEVIEYARLRGI
     ||||||||||:|||||||||| |||||||||||| ||||:||||||||||||||||||||||||
198  DTLDVMAYNKFNVFHWHLVDDSSFPYESFTFPELTRKGSFNPVTHIYTAQDVKEVIEYARLRGI

321  RVLAEFDTPGHTLSWGPGIPGLLTPCYSGSEPSGTFGPVNPSLNNTYEFMSTFFLEVSSVFPDF
     |||||||||||||||||| |||||||||||: ||||||||||||:||:||||:|||:|||||||
262  RVLAEFDTPGHTLSWGPGAPGLLTPCYSGSHLSGTFGPVNPSLNSTYDFMSTLFLEISSVFPDF

385  YLHLGGDEVDFTCWKSNPEIQDFMRKKGFGEDFKQLESFYIQTLLDIVSSYGKGYVVWQEVFDN
     ||||||||||||||||||:|| ||:||||  ||||||||||||||||||:| ||||||||||||
326  YLHLGGDEVDFTCWKSNPNIQAFMKKKGF-TDFKQLESFYIQTLLDIVSDYDKGYVVWQEVFDN

449  KVKIQPDTIIQVWREDIPVNYMKELELVTKAGFRALLSAPWYLNRISYGPDWKDFYIVEPLAFE
     |||::||||||||||::||:|| |:: :|:|||||||||||||||::|||||||:| ||||||:
389  KVKVRPDTIIQVWREEMPVEYMLEMQDITRAGFRALLSAPWYLNRVKYGPDWKDMYKVEPLAFH

513  GTPEQKALVIGGEACMWGEYVDNTNLVPRLWPRAGAVAERLWSNKLTSDLTFAYERLSHFRCEL
     ||||||||||||||||||||||:||||||||||||||||||||::||::: ||::|||||||||
453  GTPEQKALVIGGEACMWGEYVDSTNLVPRLWPRAGAVAERLWSSNLTTNIDFAFKRLSHFRCEL

577  LRRGVQAQPLNVGFCEQEFEQT*APGTEEGAGCR*MVVEPGFHCILARGRSPLPSCPLPACPCA
     :|||:||||::|| ||||||||||  |  :|  :    :||       |      |    ||
517  VRRGIQAQPISVGCCEQEFEQT*A--T--SA--E----HPG-------G------C----CP--

641  WRERGRCWRSHSIKSNVAFFYNKHGLPVFKKKSVNGVRVRAQPGWSQCLPLRSFKLRAGNETYS
              |: ::   |        |   :: |  : :| :    | :|  : :   |  :::
552  -------L-SQ-LR--*A--------P---RR-V--LALR-E----Q-VP--G-Q---G-*SFT

705  LCAVLPCL*AMSLPSHS*PYSRHLP*SSACSLHFCIISPRRWYMEKDVGAWRCSGQWGGLQTQP
              | | |::|   :   |    |    |  :|     ||::||    |   |:  |
578  ---------A-SRPGES---T---P----CP---C--APVT--TEKEAGA----GT--GV--Q-

769  GHRRASPPCILIHLPPLELFSFGFLAASILYNHYLNIIKHILFS
        |                       |  :: |:       |
606  --*R-----------------------S-MW-HF-------L--
```

The alignment is very good between amino acid position 69 and 599, 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 sequences to include only the amino acids that are in the protein you might get a better alignment. Include the amino acid positions from the first methionine (M) to the first stop (*) that occurs after the first methionine.

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

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

humanStops =

    41   599   611   713   722   730


mouseStops = find(mouseProtein == '*')

mouseStops =

   539   557   574   606
```

Looking at the amino acid sequence for humanProtein, the first M is at position 70, and the first stop after that position is actually the second stop in the sequence (position 599). Looking at the amino acid sequence for mouseProtein, the first M is at position 11, and the first stop after that position is the first stop in the sequence (position 557).

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

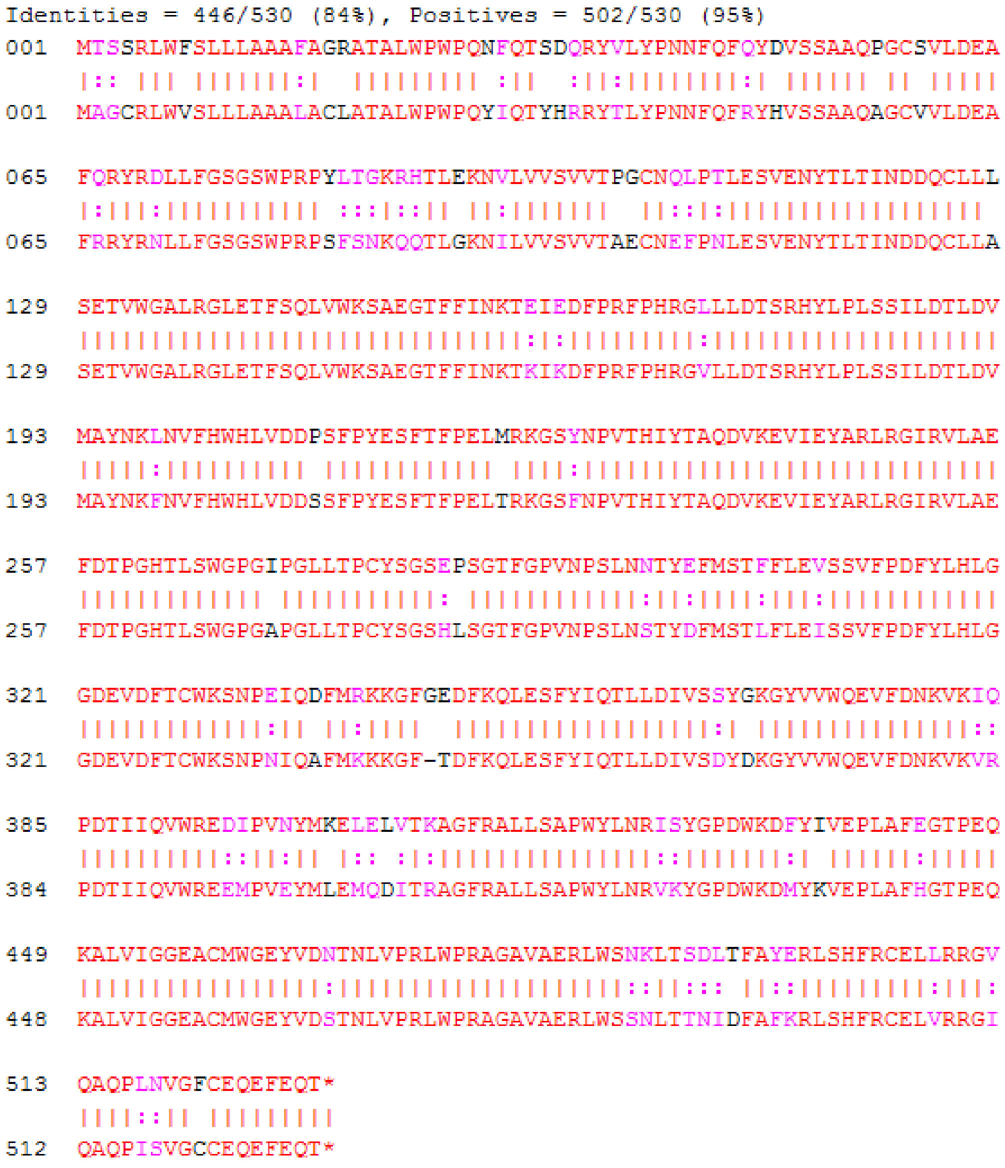
```
humanProteinORF = humanProtein(70:humanStops(2))

humanProteinORF =

MTSSRLWFSLLLAAAFAGRATALWPWPQNFQTSDQRYVLYPNNFQFQYDV
SSAAQPGCSVLDEAFQRYRDLLFGSGSWPRPYLTGKRHTLEKNVLVVSVV
TPGCNQLPTLESVENYTLTINDDQCLLLSETVWGALRGLETFSQLVWKSA
EGTFFINKTEIEDFPRFPHRGLLLDTSRHYLPLSSILDTLDVMAYNKLNV
```

```
FHWHLVDDPSFPYESFTFPELMRKGSYNPVTHIYTAQDVKEVIEYARLRG
IRVLAEFDTPGHTLSWGPGIPGLLTPCYSGSEPSGTFGPVNPSLNNTYEF
MSTFFLEVSSVFPDFYLHLGGDEVDFTCWKSNPEIQDFMRKKGFGEDFKQ
LESFYIQTLLDIVSSYGKGYVVWQEVFDNKVKIQPDTIIQVWREDIPVNY
MKELELVTKAGFRALLSAPWYLNRISYGPDWKDFYIVEPLAFEGTPEQKA
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

```
[GlobalScore_trim, GlobalAlignment_trim] = nwalign(humanProteinORF,...
                               mouseProteinORF);
showalignment(GlobalAlignment_trim)
```

`showalignment` displays the results for the second global alignment. Notice that the percent identity for the untrimmed sequences is `60%` and `84%` for trimmed sequences.



```
Identities = 446/530 (84%), Positives = 502/530 (95%)
001   MTSSRLWFSLLLAAAFAGRATALWPWPQNFQTSDQRYVLYPNNFQFQYDVSSAAQPGCSVLDEA
      |:: ||| |||||||:|  ||||||||| :||  :||:||||||||:| |||||| || |||||
001   MAGCRLWVSLLLAAALACLATALWPWPQYIQTYHRRYTLYPNNFQFRYHVSSAAQAGCVVLDEA

065   FQRYRDLLFGSGSWPRPYLTGKRHTLEKNVLVVSVVTPGCNQLPTLESVENYTLTINDDQCLLL
      |:|||:||||||||||| :::|::|| ||:|||||||  ||::|:||||||||||||||||||
065   FRRYRNLLFGSGSWPRPSFSNKQQTLGKNILVVSVVTAECNEFPNLESVENYTLTINDDQCLLA

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

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

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

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

385   PDTIIQVWREDIPVNYMKELELVTKAGFRALLSAPWYLNRISYGPDWKDFYIVEPLAFEGTPEQ
      ||||||||||::||:|| |:: :|:|||||||||||||||::|||||||:| ||||||:|||||
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 seqshoworfs function. Remember that the ORF for the human HEXA gene and the ORF for the mouse HEXA were both on the first reading frame.

```
humanORFs = seqshoworfs(humanHEXA.Sequence)

humanORFs =

1x3 struct array with fields:
    Start
    Stop


mouseORFs = seqshoworfs(mouseHEXA.Sequence)

mouseORFs =

1x3 struct array with fields:
    Start
    Stop


humanPORF = nt2aa(humanHEXA.Sequence(humanORFs(1).Start(1):...
                                     humanORFs(1).Stop(1)));

mousePORF = nt2aa(mouseHEXA.Sequence(mouseORFs(1).Start(1):...
                                     mouseORFs(1).Stop(1)));

[GlobalScore2, GlobalAlignment2] = nwalign(humanPORF, mousePORF);
```

Show the alignment in the Help browser.

```
showalignment(GlobalAlignment2)
```

The result from first truncating a nucleotide sequence before converting it 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.

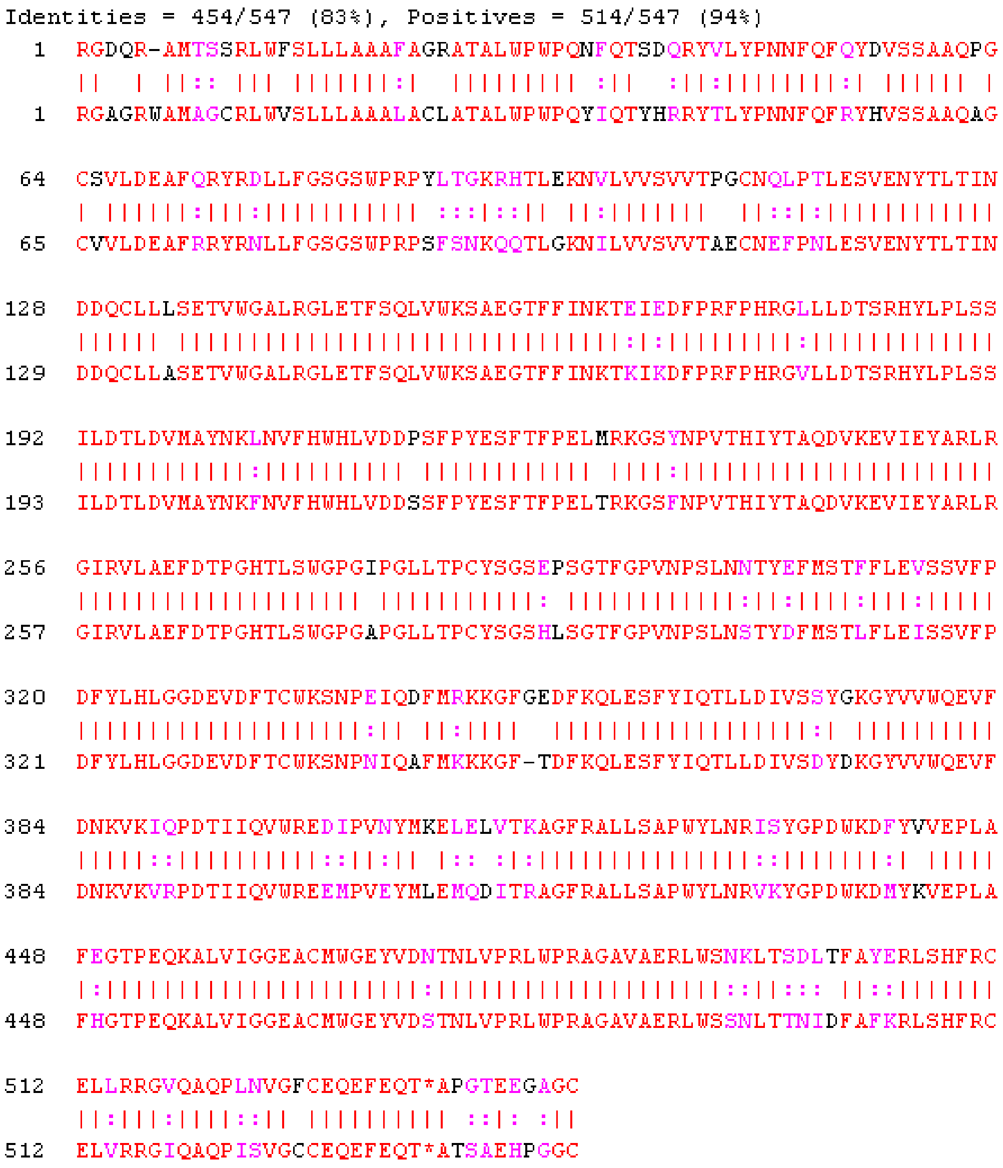
An alternative method to working with subsequences is to use a local alignment function with the nontruncated sequences.

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 . . .
```

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
```

Viewing and Aligning Multiple Sequences

In this section...

Overview of the Multiple Sequence Alignment Viewer

The Multiple Sequence Alignment Viewer is a graphical user interface (GUI) that integrates many sequence and multiple alignment functions in the toolbox. Instead of entering commands in the MATLAB Command Window, you can use this interface to visually inspect a multiple alignment and make manual adjustments.

Loading Sequence Data and Viewing the Phylogenetic Tree

Load unaligned sequence data into the MATLAB environment and view it in a phylogenetic tree.

1 Load sequence data.

```
load primatesdemodata
```

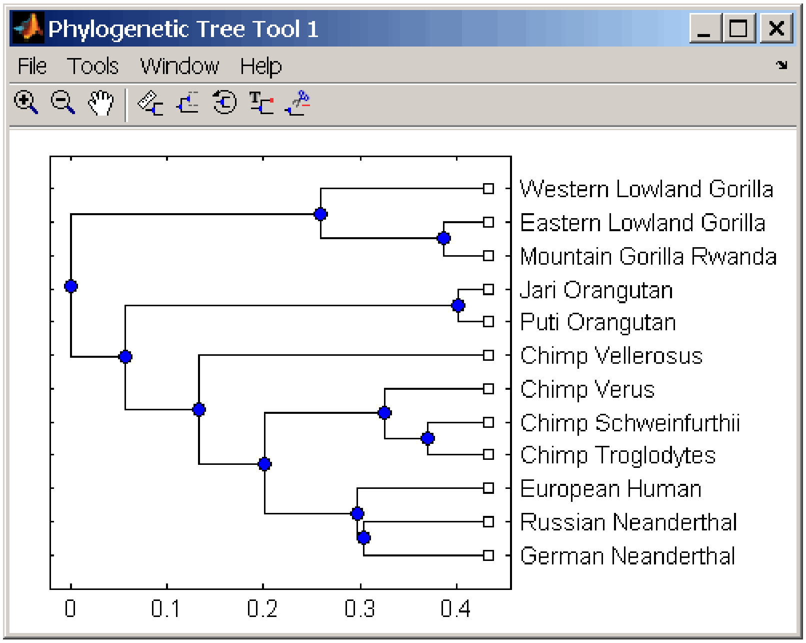
2 Create a phylogenetic tree.

```
tree = seqlinkage(seqpdist(primates),'single', primates);
```

3 View the phylogenetic tree.

```
view(tree)
```

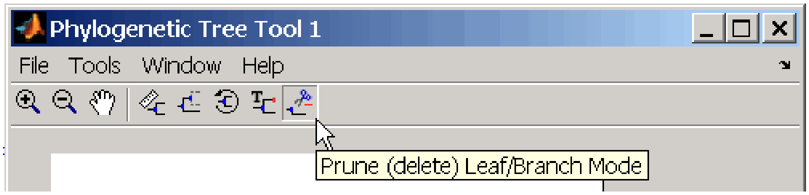
The MATLAB software creates a phytree object in the workspace and loads the sequence data into the Phylogenetic Tree Tool.



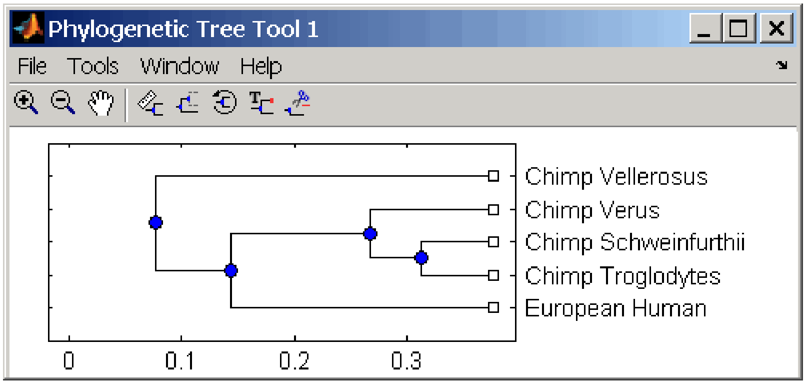
Selecting a Subset of Data from the Phylogenetic Tree

Select the human and chimp branches.

1 From the toolbar, click the **Prune** icon.

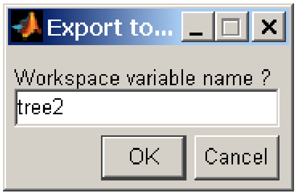


2 Click the branches to prune (remove) from the tree. For this example, click the branch nodes for gorillas, orangutans, and Neanderthals.



3 Export the selected branches to a second tree. Select **File** > **Export to Workspace**, and then select **Only Displayed**.

4 In the Export to dialog box, enter the name of a variable. For example, enter `tree2`, and then click **OK**.



Workspace variable name ? tree2

5 Extract sequences from the tree object.

```
primates2 = primates(seqmatch(get(tree2, 'Leafnames'),{primates.Header}));
```

Aligning Multiple Sequences

After selecting a set of related sequences, you can multiply align them and view the results.

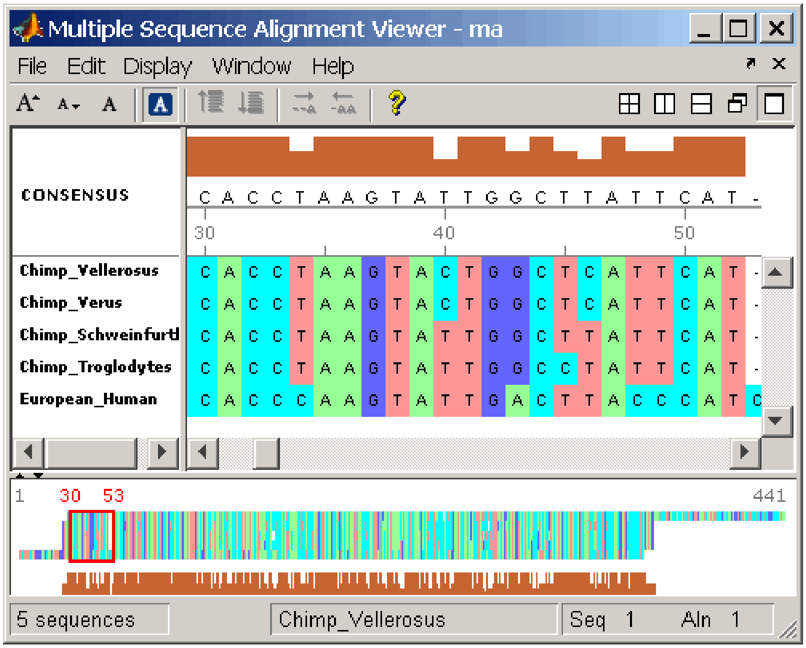
1 Align multiple sequences.

```
ma = multialign(primates2);
```

2 Load aligned sequences in the Multiple Sequence Alignment Viewer.

```
multialignviewer(ma);
```

The aligned sequences appear in the Multiple Sequence Alignment Viewer.



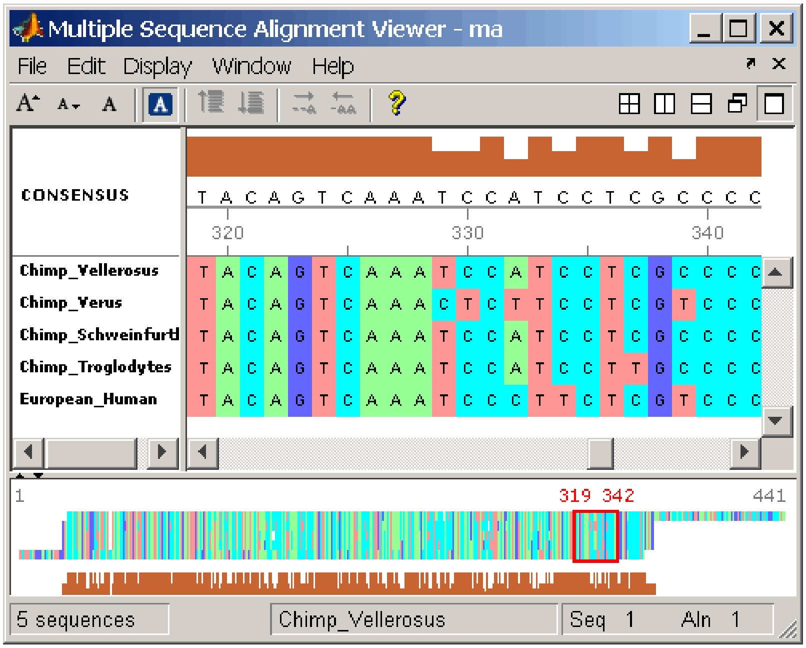
```
CONSENSUS            C A C C T A A G T A T T G G C T T A T T C A T -
Chimp_Vellerosus     C A C C T A A G T A C T G G C T C A T T C A T -
Chimp_Verus          C A C C T A A G T A C T G G C T C A T T C A T -
Chimp_Schweinfurtl   C A C C T A A G T A T T G G C T T A T T C A T -
Chimp_Troglodytes    C A C C T A A G T A T T G G C C T A T T C A T -
European_Human       C A C C C A A G T A T T G A C T T A C C C A T C
```

5 sequences | Chimp_Vellerosus | Seq 1 Aln 1

Adjusting Multiple Sequence Alignments Manually

Algorithms for aligning multiple sequences do not always produce an optimal result. By visually inspecting the alignment, you can identify areas that could use a manual adjustment to improve the alignment.

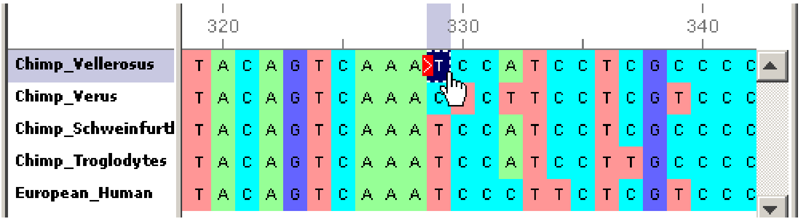
1 Identify an area where you could improve the alignment.



```
CONSENSUS           T A C A G T C A A A T C C A T C C T C G C C C C
Chimp_Vellerosus    T A C A G T C A A A T C C A T C C T C G C C C C
Chimp_Verus         T A C A G T C A A A C T C T T C C T C G T C C C
Chimp_Schweinfurtl  T A C A G T C A A A T C C A T C C T C G C C C C
Chimp_Troglodytes   T A C A G T C A A A T C C A T C C T T G C C C C
European_Human      T A C A G T C A A A T C C C T T C T C G T C C C
```

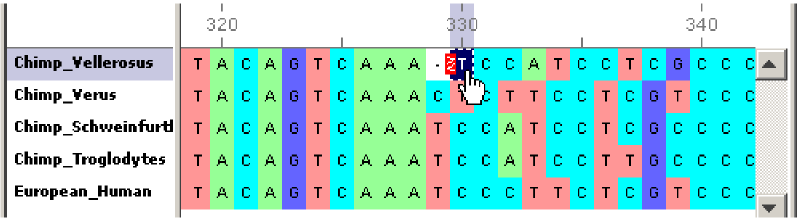
5 sequences | Chimp_Vellerosus | Seq 1 Aln 1

2 Click a letter to select it, and then move the cursor over the red direction bar. The cursor changes to a hand.



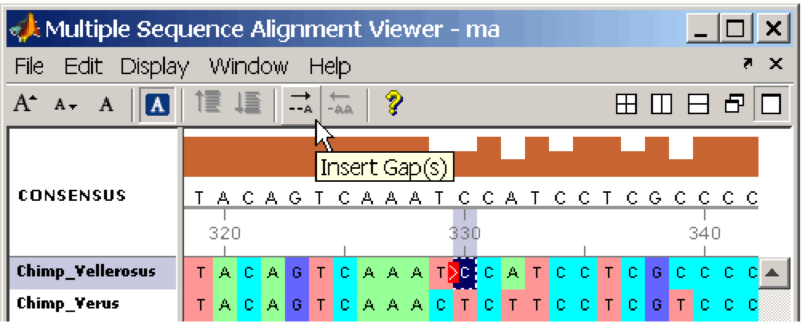
```
Chimp_Vellerosus    T A C A G T C A A A T C C A T C C T C G C C C C
Chimp_Verus         T A C A G T C A A A C   C T T C C T C G T C C C
Chimp_Schweinfurtl  T A C A G T C A A A T C C A T C C T C G C C C C
Chimp_Troglodytes   T A C A G T C A A A T C C A T C C T T G C C C C
European_Human      T A C A G T C A A A T C C C T T C T C G T C C C
```

3 Click and drag the sequence to the right to insert a gap. If there is a gap to the left, you can also move the sequence to the left and eliminate the gap.



```
Chimp_Vellerosus     T A C A G T C A A A - T C C A T C C T C G C C C
Chimp_Verus          T A C A G T C A A A C T C T T C C T C G T C C C
Chimp_Schweinfurtl   T A C A G T C A A A T C C A T C C T C G C C C C
Chimp_Troglodytes    T A C A G T C A A A T C C A T C C T T G C C C C
European_Human       T A C A G T C A A A T C C C T T C T C G T C C C
```

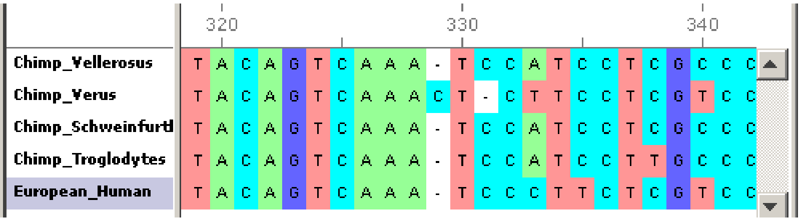
Alternately, to insert a gap, select a character, and then click the **Insert Gap** icon on the toolbar or press the spacebar.



```
CONSENSUS            T A C A G T C A A A T C C A T C C T C G C C C C
Chimp_Vellerosus     T A C A G T C A A A T C C A T C C T C G C C C C
Chimp_Verus          T A C A G T C A A A C T C T T C C T C G T C C C
```

Note You cannot delete or add letters to a sequence, but you can add or delete gaps. If all of the sequences at one alignment position have gaps, you can delete that column of gaps.

4 Continue adding gaps and moving sequences to improve the alignment.



```
Chimp_Vellerosus     T A C A G T C A A A - T C C A T C C T C G C C C
Chimp_Verus          T A C A G T C A A A C T - C T T C C T C G T C C
Chimp_Schweinfurtl   T A C A G T C A A A - T C C A T C C T C G C C C
Chimp_Troglodytes    T A C A G T C A A A - T C C A T C C T T G C C C
European_Human       T A C A G T C A A A - T C C C T T C T C G T C C
```

Closing the Multiple Sequence Alignment Viewer

Close the Multiple Sequence Alignment Viewer window from the MATLAB command line using the following syntax:

```
multialignviewer('close')
```

Microarray Analysis

Storing and Managing Gene Expression Data in Objects

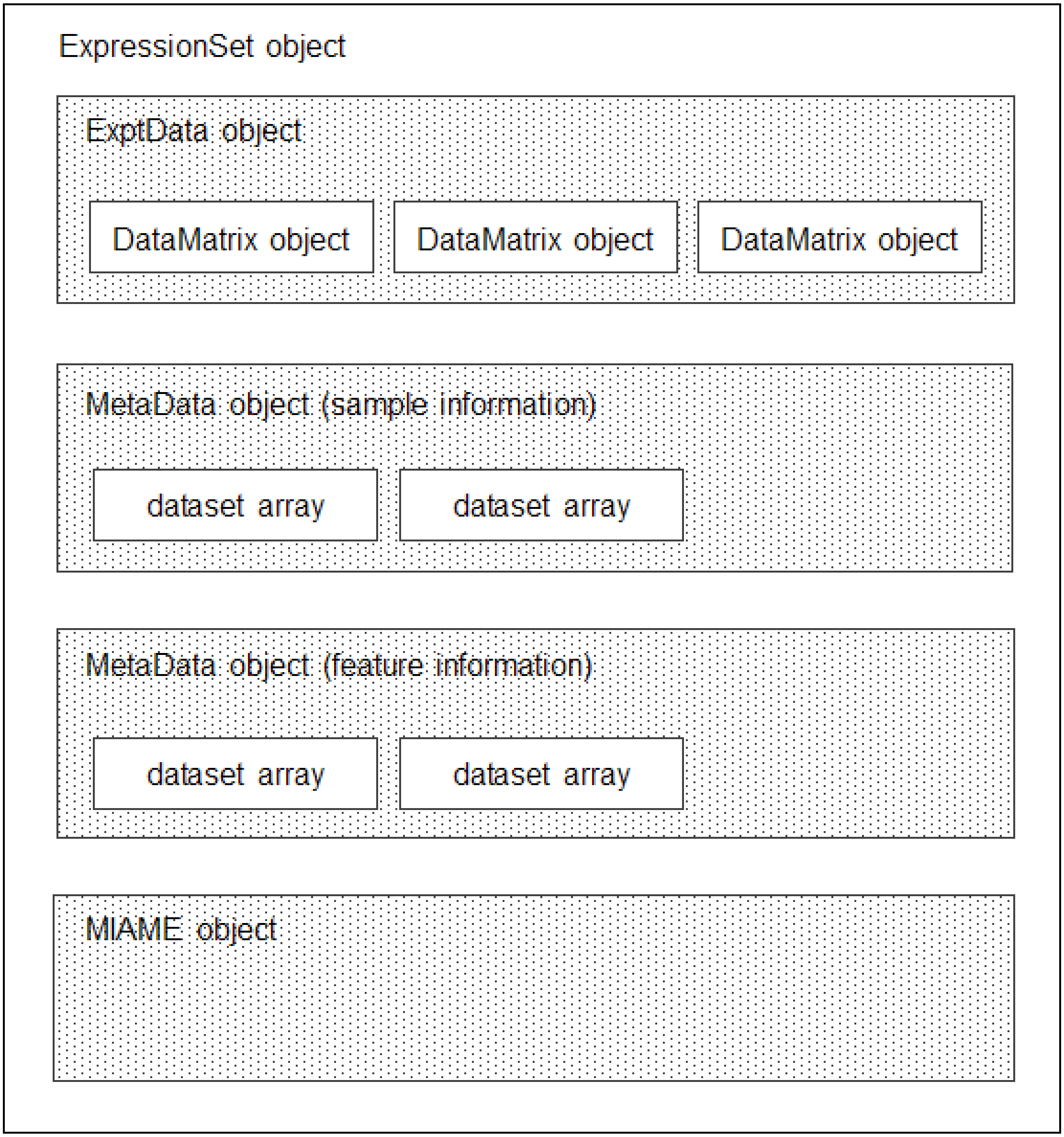
Microarray gene expression experiments are complex, containing data and information from various sources. The data and information from such an experiment is typically subdivided into four categories:

- Measured expression data values
- Sample metadata
- Microarray feature metadata
- Descriptions of experiment methods and conditions

In MATLAB, you can represent all the previous data and information in an ExpressionSet object, which typically contains the following objects:

- One ExptData object containing expression values from a microarray experiment in one or more DataMatrix objects
- One MetaData object containing *sample* metadata in two dataset arrays
- One MetaData object containing *feature* metadata in two dataset arrays
- One MIAME object containing experiment descriptions

The following graphic illustrates a typical ExpressionSet object and its component objects.


ExpressionSet object
ExptData object
DataMatrix object
DataMatrix object
DataMatrix object
MetaData object (sample information)
dataset array
dataset array
MetaData object (feature information)
dataset array
dataset array
MIAME object

Each element (DataMatrix object) in the ExpressionSet object has an element name. Also, there is always one DataMatrix object whose element name is `Expressions`.

An ExpressionSet object lets you store, manage, and subset the data from a microarray gene expression experiment. An ExpressionSet object includes properties and methods that let you access, retrieve, and change data, metadata, and other information about the microarray experiment. These properties and methods are useful to view and analyze the data. For a list of the properties and methods, see ExpressionSet class.

To learn more about constructing and using objects for microarray gene expression data and information, see:

- "Representing Expression Data Values in DataMatrix Objects" on page 4-5
- "Representing Expression Data Values in ExptData Objects" on page 4-11
- "Representing Sample and Feature Metadata in MetaData Objects" on page 4-15
- "Representing Experiment Information in a MIAME Object" on page 4-22
- "Representing All Data and Information in an ExpressionSet Object" on page 4-27

Representing Expression Data Values in DataMatrix Objects

In this section...

Overview of DataMatrix Objects

The toolbox includes functions, objects, and methods for creating, storing, and accessing microarray data.

The object constructor function, `DataMatrix`, lets you create a DataMatrix object to encapsulate data and metadata (row and column names) from a microarray experiment. A DataMatrix object stores experimental data in a matrix, with rows typically corresponding to gene names or probe identifiers, and columns typically corresponding to sample identifiers. A DataMatrix object also stores metadata, including the gene names or probe identifiers (as the row names) and sample identifiers (as the column names).

You can reference microarray expression values in a DataMatrix object the same way you reference data in a MATLAB array, that is, by using linear or logical indexing. Alternately, you can reference this experimental data by gene (probe) identifiers and sample identifiers. Indexing by these identifiers lets you quickly and conveniently access subsets of the data without having to maintain additional index arrays.

Many MATLAB operators and arithmetic functions are available to DataMatrix objects by means of methods. These methods let you modify, combine, compare, analyze, plot, and access information from DataMatrix objects. Additionally, you can easily extend the functionality by using general element-wise functions, `dmarrayfun` and `dmbsxfun`, and by manually accessing the properties of a DataMatrix object.

Note For tables describing the properties and methods of a DataMatrix object, see the DataMatrix object reference page.

Constructing DataMatrix Objects

1 Load the MAT-file, provided with the Bioinformatics Toolbox software, that contains yeast data. This MAT-file includes three variables: `yeastvalues`, a 614-by-7 matrix of gene expression data, `genes`, a cell array of 614 GenBank accession numbers for labeling the rows in `yeastvalues`, and `times`, a 1-by-7 vector of time values for labeling the columns in `yeastvalues`.

```
load filteredyeastdata
```

2 Create variables to contain a subset of the data, specifically the first five rows and first four columns of the `yeastvalues` matrix, the `genes` cell array, and the `times` vector.

```
yeastvalues = yeastvalues(1:5,1:4);
genes = genes(1:5,:);
times = times(1:4);
```

3 Import the microarray object package so that the `DataMatrix` constructor function will be available.

```
import bioma.data.*
```

4 Use the `DataMatrix` constructor function to create a small DataMatrix object from the gene expression data in the variables you created in step 2.

```
dmo = DataMatrix(yeastvalues,genes,times)

dmo =

                   0       9.5     11.5       13.5
    SS DNA     -0.131     1.699    -0.026      0.365
    YAL003W     0.305     0.146    -0.129     -0.444
    YAL012W     0.157     0.175     0.467     -0.379
    YAL026C     0.246     0.796     0.384      0.981
    YAL034C    -0.235    0.487    -0.184    -0.669
```

Getting and Setting Properties of a DataMatrix Object

You use the `get` and `set` methods to retrieve and set properties of a DataMatrix object.

1 Use the `get` method to display the properties of the DataMatrix object, `dmo`.

```
get(dmo)
            Name: ''
        RowNames: {5x1 cell}
        ColNames: {'   0'  ' 9.5'  '11.5'  '13.5'}
           NRows: 5
           NCols: 4
           NDims: 2
    ElementClass: 'double'
```

2 Use the `set` method to specify a name for the DataMatrix object, `dmo`.

```
dmo = set(dmo,'Name','MyDMObject');
```

3 Use the `get` method again to display the properties of the DataMatrix object, `dmo`.

```
get(dmo)
            Name: 'MyDMObject'
        RowNames: {5x1 cell}
        ColNames: {'   0'  ' 9.5'  '11.5'  '13.5'}
           NRows: 5
           NCols: 4
           NDims: 2
    ElementClass: 'double'
```

Note For a description of all properties of a DataMatrix object, see the DataMatrix object reference page.

Accessing Data in DataMatrix Objects

DataMatrix objects support the following types of indexing to extract, assign, and delete data:

- Parenthesis ( ) indexing
- Dot . indexing

Parentheses () Indexing

Use parenthesis indexing to extract a subset of the data in `dmo` and assign it to a new DataMatrix object `dmo2`:

```
dmo2 = dmo(1:5,2:3)
dmo2 =
                9.5     11.5
    SS DNA     1.699    -0.026
    YAL003W    0.146    -0.129
    YAL012W    0.175     0.467
    YAL026C    0.796     0.384
    YAL034C    0.487    -0.184
```

Use parenthesis indexing to extract a subset of the data using row names and column names, and assign it to a new DataMatrix object `dmo3`:

```
dmo3 = dmo({'SS DNA','YAL012W','YAL034C'},'11.5')

dmo3 =

               11.5
    SS DNA     -0.026
    YAL012W     0.467
    YAL034C    -0.184
```

Note If you use a cell array of row names or column names to index into a DataMatrix object, the names must be unique, even though the row names or column names within the DataMatrix object are not unique.

Use parenthesis indexing to assign new data to a subset of the elements in dmo2:

```
dmo2({'SS DNA', 'YAL003W'}, 1:2) = [1.700 -0.030; 0.150 -0.130]
dmo2 =

                 9.5      11.5
    SS DNA        1.7      -0.03
    YAL003W      0.15      -0.13
    YAL012W     0.175      0.467
    YAL026C     0.796      0.384
    YAL034C     0.487     -0.184
```

Use parenthesis indexing to delete a subset of the data in dmo2:

```
dmo2({'SS DNA', 'YAL003W'}, :) = []
dmo2 =

                 9.5      11.5
    YAL012W     0.175      0.467
    YAL026C     0.796      0.384
    YAL034C     0.487     -0.184
```

Dot . Indexing

Note In the following examples, notice that when using dot indexing with DataMatrix objects, you specify all rows or all columns using a colon within single quotation marks, (':').

Use dot indexing to extract the data from the 11.5 column only of dmo:

```
timeValues = dmo.(':')('11.5')
timeValues =

   -0.0260
   -0.1290
    0.4670
    0.3840
   -0.1840
```

Use dot indexing to assign new data to a subset of the elements in dmo:

```
dmo.(1:2)(':') = 7
dmo =

                  0       9.5     11.5      13.5
    SS DNA          7        7         7         7
    YAL003W         7        7         7         7
    YAL012W     0.157    0.175     0.467    -0.379
    YAL026C     0.246    0.796     0.384     0.981
    YAL034C    -0.235    0.487    -0.184    -0.669
```

Use dot indexing to delete an entire variable from dmo:

```
dmo.YAL034C = []
dmo =

                  0      9.5     11.5     13.5
    SS DNA         7        7        7         7
    YAL003W        7        7        7         7
    YAL012W    0.157    0.175    0.467    -0.379
    YAL026C    0.246    0.796    0.384     0.981
```

Use dot indexing to delete two columns from dmo:

```
dmo.(':')(2:3)=[]

dmo =

                  0     13.5
    SS DNA         7         7
    YAL003W        7         7
    YAL012W    0.157    -0.379
    YAL026C    0.246     0.981
```

Representing Expression Data Values in ExptData Objects

In this section...

Overview of ExptData Objects

You can use an ExptData object to store expression values from a microarray experiment. An ExprData object stores the data values in one or more DataMatrix objects, each having the same row names (feature names) and column names (sample names). Each element (DataMatrix object) in the ExptData object has an element name.

The following illustrates a small DataMatrix object containing expression values from three samples (columns) and seven features (rows):

```
                  A         B         C
100001_at      2.26     20.14     31.66
100002_at    158.86    236.25    206.27
100003_at     68.11    105.45     82.92
100004_at     74.32     96.68     84.87
100005_at     75.05     53.17     57.94
100006_at     80.36     42.89     77.21
100007_at    216.64    191.32    219.48
```

An ExptData object lets you store, manage, and subset the data values from a microarray experiment. An ExptData object includes properties and methods that let you access, retrieve, and change data values from a microarray experiment. These properties and methods are useful to view and analyze the data. For a list of the properties and methods, see ExptData class.

Constructing ExptData Objects

The mouseExprsData.txt file used in this example contains data from Hovatta et al., 2005.

1 Import the bioma.data package so that the DataMatrix and ExptData constructor functions are available.

```
import bioma.data.*
```

2 Use the DataMatrix constructor function to create a DataMatrix object from the gene expression data in the mouseExprsData.txt file. This file contains a table of expression values and metadata (sample and feature names) from a microarray experiment done using the Affymetrix MGU74Av2 GeneChip array. There are 26 sample names (A through Z), and 500 feature names (probe set names).

```
dmObj = DataMatrix('File', 'mouseExprsData.txt');
```

3 Use the ExptData constructor function to create an ExptData object from the DataMatrix object.

```
EDObj = ExptData(dmObj);
```

4 Display information about the ExptData object, EDObj.

```
EDObj

Experiment Data:
  500 features,  26 samples
  1 elements
  Element names: Elmt1
```

Note For complete information on constructing ExptData objects, see ExptData class.

Using Properties of an ExptData Object

To access properties of an ExptData object, use the following syntax:

```
objectname.propertyname
```

For example, to determine the number of elements (DataMatrix objects) in an ExptData object:

```
EDObj.NElements

ans =

     1
```

To set properties of an ExptData object, use the following syntax:

```
objectname.propertyname = propertyvalue
```

For example, to set the `Name` property of an ExptData object:

```
EDObj.Name = 'MyExptDataObject'
```

Note Property names are case sensitive. For a list and description of all properties of an ExptData object, see ExptData class.

Using Methods of an ExptData Object

To use methods of an ExptData object, use either of the following syntaxes:

```
objectname.methodname
```

or

```
methodname(objectname)
```

For example, to retrieve the sample names from an ExptData object:

```
EDObj.sampleNames

Columns 1 through 9

    'A'    'B'    'C'    'D'    'E'    'F'    'G'    'H'    'I'  ...
```

To return the size of an ExptData object:

```
size(EDObj)
```

```
ans =

   500    26
```

Note For a complete list of methods of an ExptData object, see ExptData class.

References

[1] Hovatta, I., Tennant, R S., Helton, R., et al. (2005). Glyoxalase 1 and glutathione reductase 1 regulate anxiety in mice. Nature *438*, 662–666.

Representing Sample and Feature Metadata in MetaData Objects

In this section...

Overview of MetaData Objects

You can store either sample or feature metadata from a microarray gene expression experiment in a MetaData object. The metadata consists of variable names, for example, related to either samples or microarray features, along with descriptions and values for the variables.

A MetaData object stores the metadata in two dataset arrays:

- **Values dataset array** — A dataset array containing the measured value of each variable per sample or feature. In this dataset array, the columns correspond to variables and rows correspond to either samples or features. The number and names of the columns in this dataset array must match the number and names of the rows in the Descriptions dataset array. If this dataset array contains *sample* metadata, then the number and names of the rows (samples) must match the number and names of the columns in the DataMatrix objects in the same ExpressionSet object. If this dataset array contains *feature* metadata, then the number and names of the rows (features) must match the number and names of the rows in the DataMatrix objects in the same ExpressionSet object.
- **Descriptions dataset array** — A dataset array containing a list of the variable names and their descriptions. In this dataset array, each row corresponds to a variable. The row names are the variable names, and a column, named `VariableDescription`, contains a description of the variable. The number and names of the rows in the Descriptions dataset array must match the number and names of the columns in the Values dataset array.

The following illustrates a dataset array containing the measured value of each variable per sample or feature:

```
           Gender     Age   Type            Strain              Source
     A     'Male'     8     'Wild type'     '129S6/SvEvTac'     'amygdala'
     B     'Male'     8     'Wild type'     '129S6/SvEvTac'     'amygdala'
     C     'Male'     8     'Wild type'     '129S6/SvEvTac'     'amygdala'
     D     'Male'     8     'Wild type'     'A/J '              'amygdala'
     E     'Male'     8     'Wild type'     'A/J '              'amygdala'
     F     'Male'     8     'Wild type'     'C57BL/6J '         'amygdala'
```

The following illustrates a dataset array containing a list of the variable names and their descriptions:

```
                  VariableDescription
  id              'Sample  identifier'
  Gender          'Gender of the mouse in study'
  Age             'The number of weeks since mouse birth'
  Type            'Genetic characters'
  Strain          'The mouse strain'
  Source          'The tissue source for RNA collection'
```

A MetaData object lets you store, manage, and subset the metadata from a microarray experiment. A MetaData object includes properties and methods that let you access, retrieve, and change metadata from a microarray experiment. These properties and methods are useful to view and analyze the metadata. For a list of the properties and methods, see MetaData class

Constructing MetaData Objects

Constructing a MetaData Object from Two dataset Arrays

1. Import the `bioma.data` package so that the `MetaData` constructor function is available.

   ```
   import bioma.data.*
   ```

2. Load some sample data, which includes Fisher's iris data of 5 measurements on a sample of 150 irises.

```
load fisheriris
```

3 Create a dataset array from some of Fisher's iris data. The dataset array will contain 750 measured values, one for each of 150 samples (iris replicates) at five variables (species, SL, SW, PL, PW). In this dataset array, the rows correspond to samples, and the columns correspond to variables.

```
irisValues = dataset({nominal(species),'species'}, ...
                     {meas, 'SL', 'SW', 'PL', 'PW'});
```

4 Create another dataset array containing a list of the variable names and their descriptions. This dataset array will contain five rows, each corresponding to the five variables: species, SL, SW, PL, and PW. The first column will contain the variable name. The second column will have a column header of `VariableDescription` and contain a description of the variable.

```
% Create 5-by-1 cell array of description text for the variables
varDesc = {'Iris species', 'Sepal Length', 'Sepal Width', ...
           'Petal Length', 'Petal Width'}';
% Create the dataset array from the variable descriptions
irisVarDesc = dataset(varDesc, ...
              'ObsNames', {'species','SL','SW','PL','PW'}, ...
              'VarNames', {'VariableDescription'})

irisVarDesc =

               VariableDescription
    species    'Iris species'
    SL         'Sepal Length'
    SW         'Sepal Width'
    PL         'Petal Length'
    PW         'Petal Width'
```

5 Create a MetaData object from the two dataset arrays.

```
MDObj1 = MetaData(irisValues, irisVarDesc);
```

Constructing a MetaData Object from a Text File

1 Import the bioma.datapackage so that the MetaData constructor function is available.

```
import bioma.data.*
```

2 View the mouseSampleData.txt file included with the Bioinformatics Toolbox software.

Note that this text file contains two tables. One table contains 130 measured values, one for each of 26 samples (A through Z) at five variables (Gender, Age, Type, Strain, and Source). In this table, the rows correspond to samples, and the columns correspond to variables. The second table has lines prefaced by the # symbol. It contains five rows, each corresponding to the five variables: Gender, Age, Type, Strain, and Source. The first column contains the variable name. The second column has a column header of VariableDescription and contains a description of the variable.

```
# id: Sample  identifier
# Gender: Gender of the mouse in study
# Age: The number of weeks since mouse birth
# Type: Genetic characters
# Strain: The mouse strain
# Source: The tissue source for RNA collection
ID Gender Age Type Strain Source
A Male 8 Wild type 129S6/SvEvTac amygdala
B Male 8 Wild type 129S6/SvEvTac amygdala
C Male 8 Wild type 129S6/SvEvTac amygdala
D Male 8 Wild type A/J  amygdala
E Male 8 Wild type A/J  amygdala
F Male 8 Wild type C57BL/6J  amygdala
G Male 8 Wild type C57BL/6J amygdala
H Male 8 Wild type 129S6/SvEvTac cingulate cortex
I Male 8 Wild type 129S6/SvEvTac cingulate cortex
J Male 8 Wild type A/J cingulate cortex
K Male 8 Wild type A/J cingulate cortex
L Male 8 Wild type A/J cingulate cortex
M Male 8 Wild type C57BL/6J cingulate cortex
N Male 8 Wild type C57BL/6J cingulate cortex
```

```
O Male 8 Wild type 129S6/SvEvTac hippocampus
P Male 8 Wild type 129S6/SvEvTac hippocampus
Q Male 8 Wild type A/J hippocampus
R Male 8 Wild type A/J hippocampus
S Male 8 Wild type C57BL/6J hippocampus
T Male 8 Wild type C57BL/6J4 hippocampus
U Male 8 Wild type 129S6/SvEvTac hypothalamus
V Male 8 Wild type 129S6/SvEvTac hypothalamus
W Male 8 Wild type A/J hypothalamus
X Male 8 Wild type A/J hypothalamus
Y Male 8 Wild type C57BL/6J hypothalamus
Z Male 8 Wild type C57BL/6J hypothalamus
```

3 Create a MetaData object from the metadata in the `mouseSampleData.txt` file.

```
MDObj2 = MetaData('File', 'mouseSampleData.txt', 'VarDescChar', '#')

Sample Names:
    A, B, ...,Z (26 total)
Variable Names and Meta Information:

              VariableDescription
    Gender    ' Gender of the mouse in study'
    Age       ' The number of weeks since mouse birth'
    Type      ' Genetic characters'
    Strain    ' The mouse strain'
    Source    ' The tissue source for RNA collection'
```

For complete information on constructing MetaData objects, see MetaData class.

Using Properties of a MetaData Object

To access properties of a MetaData object, use the following syntax:

```
objectname.propertyname
```

For example, to determine the number of variables in a MetaData object:

```
MDObj2.NVariables
```

```
ans =

     5
```

To set properties of a MetaData object, use the following syntax:

```
objectname.propertyname = propertyvalue
```

For example, to set the Description property of a MetaData object:

```
MDObj1.Description = 'This is my MetaData object for my sample metadata'
```

Note Property names are case sensitive. For a list and description of all properties of a MetaData object, see MetaData class.

Using Methods of a MetaData Object

To use methods of a MetaData object, use either of the following syntaxes:

```
objectname.methodname
```

or

```
methodname(objectname)
```

For example, to access the dataset array in a MetaData object that contains the variable values:

```
MDObj2.variableValues;
```

To access the dataset array of a MetaData object that contains the variable descriptions:

```
variableDesc(MDObj2)

ans =

              VariableDescription
    Gender    ' Gender of the mouse in study'
    Age       ' The number of weeks since mouse birth'
    Type       ' Genetic characters'
    Strain     ' The mouse strain'
    Source     ' The tissue source for RNA collection'
```

Note For a complete list of methods of a MetaData object, see MetaData class.

Representing Experiment Information in a MIAME Object

In this section...

Overview of MIAME Objects

You can store information about experimental methods and conditions from a microarray gene expression experiment in a MIAME object. It loosely follows the Minimum Information About a Microarray Experiment (MIAME) specification. It can include information about:

- Experiment design
- Microarrays used
- Samples used
- Sample preparation and labeling
- Hybridization procedures and parameters
- Normalization controls
- Preprocessing information
- Data processing specifications

A MIAME object includes properties and methods that let you access, retrieve, and change experiment information related to a microarray experiment. These properties and methods are useful to view and analyze the information. For a list of the properties and methods, see MIAME class.

Constructing MIAME Objects

For complete information on constructing MIAME objects, see MIAME class.

Constructing a MIAME Object from a GEO Structure

1 Import the `bioma.data` package so that the `MIAME` constructor function is available.

```
import bioma.data.*
```

2 Use the `getgeodata` function to return a MATLAB structure containing Gene Expression Omnibus (GEO) Series data related to accession number GSE4616.

```
geoStruct = getgeodata('GSE4616')

geoStruct =

    Header: [1x1 struct]
      Data: [12488x12 bioma.data.DataMatrix]
```

3 Use the `MIAME` constructor function to create a MIAME object from the structure.

```
MIAMEObj1 = MIAME(geoStruct);
```

4 Display information about the MIAME object, `MIAMEObj`.

```
MIAMEObj1

MIAMEObj1 =

Experiment Description:
  Author name: Mika,,Silvennoinen
Riikka,,Kivelˆ/
Maarit,,Lehti
Anna-Maria,,Touvras
Jyrki,,Komulainen
Veikko,,Vihko
Heikki,,Kainulainen
  Laboratory: LIKES - Research Center
  Contact information: Mika,,Silvennoinen
  URL:
  PubMedIDs: 17003243
  Abstract: A 90 word abstract is available. Use the Abstract property.
  Experiment Design: A 234 word summary is available. Use the ExptDesign property.
  Other notes:
    [1x80 char]
```

Constructing a MIAME Object from Properties

1 Import the `bioma.data` package so that the`MIAME` constructor function is available.

```
import bioma.data.*
```

2 Use the `MIAME` constructor function to create a MIAME object using individual properties.

```
MIAMEObj2 = MIAME('investigator', 'Jane Researcher',...
                  'lab', 'One Bioinformatics Laboratory',...
                  'contact', 'jresearcher@lab.not.exist',...
                  'url', 'www.lab.not.exist',...
                  'title', 'Normal vs. Diseased Experiment',...
                  'abstract', 'Example of using expression data',...
                  'other', {'Notes:Created from a text file.'});
```

3 Display information about the MIAME object, `MIAMEObj2`.

```
MIAMEObj2

MIAMEObj2 =

Experiment Description:
  Author name: Jane Researcher
  Laboratory: One Bioinformatics Laboratory
  Contact information: jresearcher@lab.not.exist
  URL: www.lab.not.exist
  PubMedIDs:
  Abstract: A 4 word abstract is available. Use the Abstract property.
  No experiment design summary available.
  Other notes:
    'Notes:Created from a text file.'
```

Using Properties of a MIAME Object

To access properties of a MIAME object, use the following syntax:

```
objectname.propertyname
```

For example, to retrieve the PubMed identifier of publications related to a MIAME object:

```
MIAMEObj1.PubMedID

ans =

17003243
```

To set properties of a MIAME object, use the following syntax:

```
objectname.propertyname = propertyvalue
```

For example, to set the `Laboratory` property of a MIAME object:

```
MIAMEObj1.Laboratory = 'XYZ Lab'
```

Note Property names are case sensitive. For a list and description of all properties of a MIAME object, see MIAME class.

Using Methods of a MIAME Object

To use methods of a MIAME object, use either of the following syntaxes:

```
objectname.methodname
```

or

```
methodname(objectname)
```

For example, to determine if a MIAME object is empty:

```
MIAMEObj1.isempty

ans =
     0
```

Note For a complete list of methods of a MIAME object, see MIAME class.

Representing All Data and Information in an ExpressionSet Object

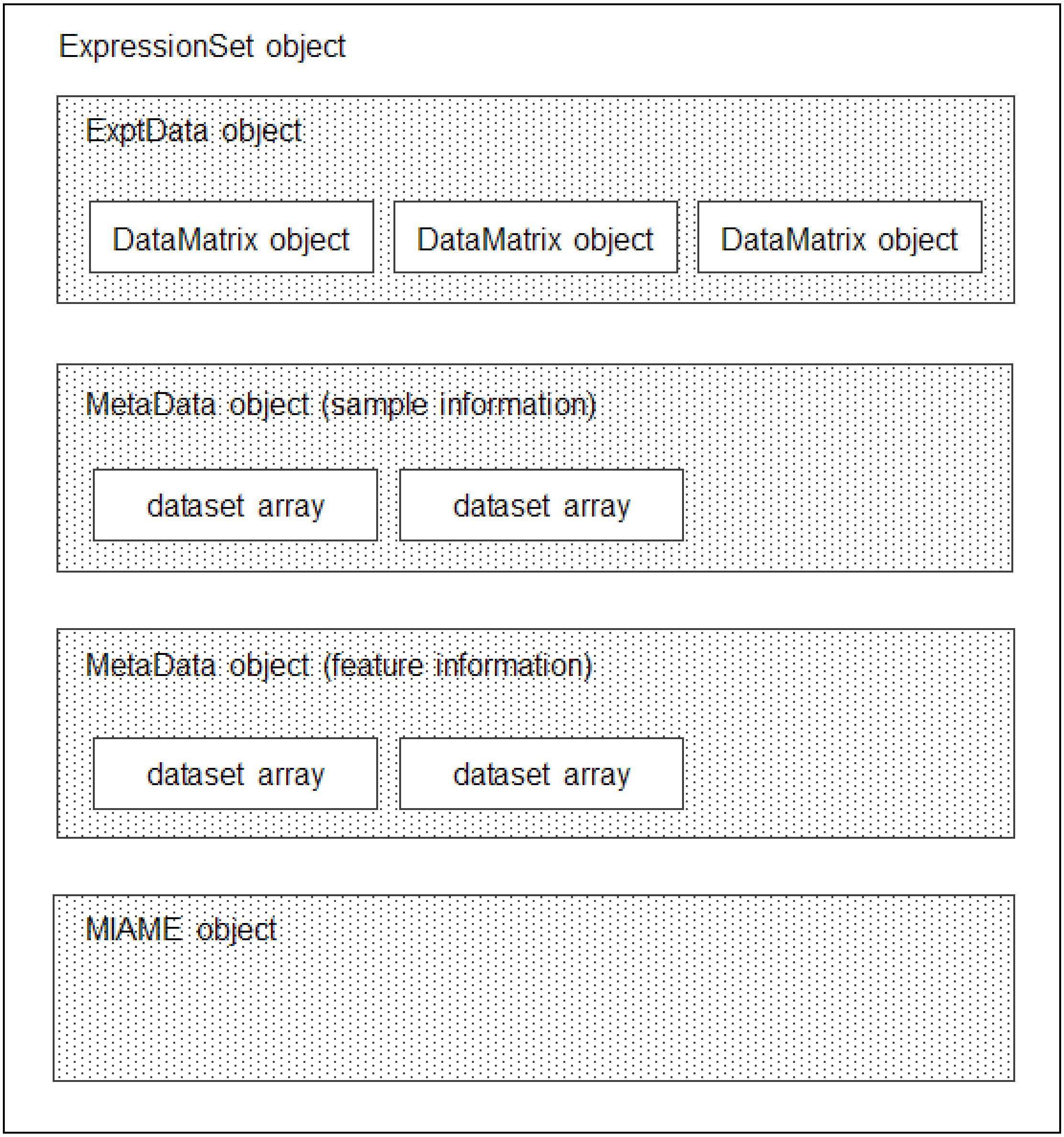
In this section...

Overview of ExpressionSet Objects

You can store all microarray experiment data and information in one object by assembling the following into an ExpressionSet object:

- One ExptData object containing expression values from a microarray experiment in one or more DataMatrix objects
- One MetaData object containing *sample* metadata in two dataset arrays
- One MetaData object containing *feature* metadata in two dataset arrays
- One MIAME object containing experiment descriptions

The following graphic illustrates a typical ExpressionSet object and its component objects.


ExpressionSet object
ExptData object
DataMatrix object
DataMatrix object
DataMatrix object
MetaData object (sample information)
dataset array
dataset array
MetaData object (feature information)
dataset array
dataset array
MIAME object

Each element (DataMatrix object) in the ExpressionSet object has an element name. Also, there is always one DataMatrix object whose element name is `Expressions`.

An ExpressionSet object lets you store, manage, and subset the data from a microarray gene expression experiment. An ExpressionSet object includes properties and methods that let you access, retrieve, and change data, metadata, and other information about the microarray experiment. These properties and methods are useful to view and analyze the data. For a list of the properties and methods, see ExpressionSet class.

Constructing ExpressionSet Objects

Note The following procedure assumes you have executed the example code in the previous sections:

- "Representing Expression Data Values in ExptData Objects" on page 4-11
- "Representing Sample and Feature Metadata in MetaData Objects" on page 4-15
- "Representing Experiment Information in a MIAME Object" on page 4-22

1. Import the `bioma` package so that the `ExpresssionSet` constructor function is available.

   ```
   import bioma.*
   ```

2. Construct an ExpressionSet object from `EDObj`, an ExptData object, `MDObj2`, a MetaData object containing sample variable information, and `MIAMEObj`, a MIAME object.

   ```
   ESObj = ExpressionSet(EDObj, 'SData', MDObj2, 'EInfo', MIAMEObj1);
   ```

3. Display information about the ExpressionSet object, `ESObj`.

   ```
   ESObj
   ```

```
ExpressionSet
Experiment Data: 500 features, 26 samples
  Element names: Expressions
Sample Data:
    Sample names:     A, B, ...,Z (26 total)
    Sample variable names and meta information:
        Gender:  Gender of the mouse in study
        Age:  The number of weeks since mouse birth
        Type:  Genetic characters
        Strain:  The mouse strain
        Source:  The tissue source for RNA collection
Feature Data: none
Experiment Information: use 'exptInfo(obj)'
```

For complete information on constructing ExpressionSet objects, see ExpressionSet class.

Using Properties of an ExpressionSet Object

To access properties of an ExpressionSet object, use the following syntax:

```
objectname.propertyname
```

For example, to determine the number of samples in an ExpressionSet object:

```
ESObj.NSamples

ans =

    26
```

Note Property names are case sensitive. For a list and description of all properties of an ExpressionSet object, see ExpressionSet class.

Using Methods of an ExpressionSet Object

To use methods of an ExpressionSet object, use either of the following syntaxes:

```
objectname.methodname
```

or

```
methodname(objectname)
```

For example, to retrieve the sample variable names from an ExpressionSet object:

```
ESObj.sampleVarNames

ans =

    'Gender'    'Age'    'Type'    'Strain'    'Source'
```

To retrieve the experiment information contained in an ExpressionSet object:

```
exptInfo(ESObj)

ans =

Experiment description
  Author name: Mika,,Silvennoinen
Riikka,,Kivelˆ/
Maarit,,Lehti
Anna-Maria,,Touvras
Jyrki,,Komulainen
Veikko,,Vihko
Heikki,,Kainulainen
  Laboratory: XYZ Lab
  Contact information: Mika,,Silvennoinen
  URL:
  PubMedIDs: 17003243
  Abstract: A 90 word abstract is available Use the Abstract property.
  Experiment Design: A 234 word summary is available Use the ExptDesign property.
  Other notes:
    [1x80 char]
```

Note For a complete list of methods of an ExpressionSet object, see ExpressionSet class.

Example: Visualizing Microarray Images

In this section...

Overview of the Mouse Example

This example looks at the various ways to visualize microarray data. The data comes from a pharmacological model of Parkinson's disease (PD) using a mouse brain. 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).

The microarray data used in this example is available in a Web supplement to the paper by Brown et al. and in the file `mouse_a1pd.gpr` included with the Bioinformatics Toolbox software.

```
http://labs.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 illustrates how to import data from the Web into the MATLAB environment, using data from a study about gene expression in mouse brains as an example. See "Overview of the Mouse Example" on page 4-33.

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

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

Information about the structure displays in the MATLAB Command Window:

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

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

```
pd.ColumnNames
```

The column names are shown below.

```
ans =
    'X'
    'Y'
    'Dia.'
    'F635 Median'
    'F635 Mean'
    'F635 SD'
    'B635 Median'
    'B635 Mean'
    'B635 SD'
    '% > B635+1SD'
    '% > B635+2SD'
    'F635 % Sat.'
    'F532 Median'
    'F532 Mean'
    'F532 SD'
    'B532 Median'
    'B532 Mean'
    'B532 SD'
    '% > B532+1SD'
    '% > B532+2SD'
    'F532 % Sat.'
    'Ratio of Medians'
    'Ratio of Means'
    'Median of Ratios'
    'Mean of Ratios'
    'Ratios SD'
    'Rgn Ratio'
    'Rgn 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

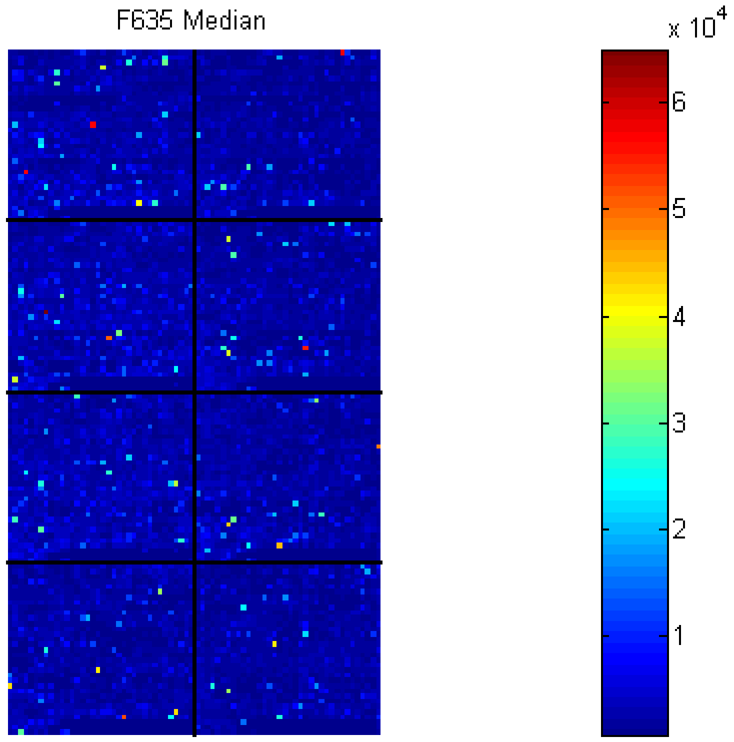
This procedure illustrates how to visualize microarray data by plotting image maps. The function `maimage` can take a microarray data structure and create a pseudocolor image of the data arranged in the same order as the spots on the array. In other words, `maimage` plots a spatial plot of the microarray.

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 4-34.

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')
   ```

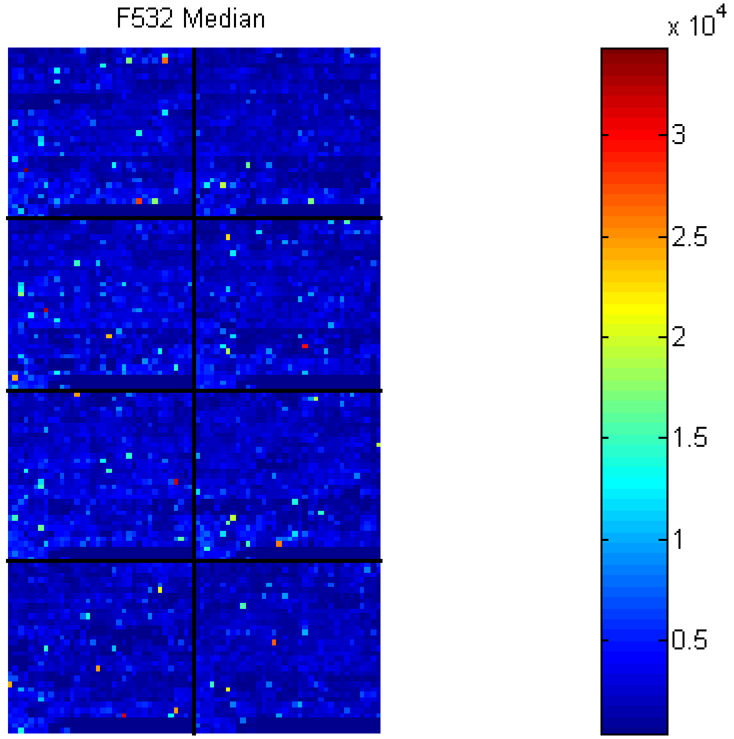
The MATLAB software plots an image showing the median pixel values for the foreground of the red (Cy5) channel.



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')
```

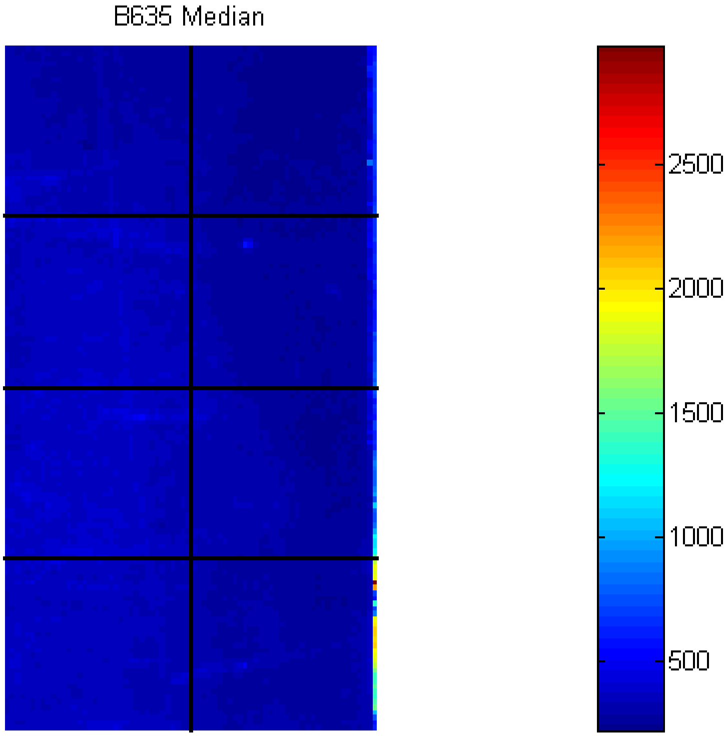
The MATLAB software plots an image showing the median pixel values of the foreground of the green (Cy3) channel.



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')
```

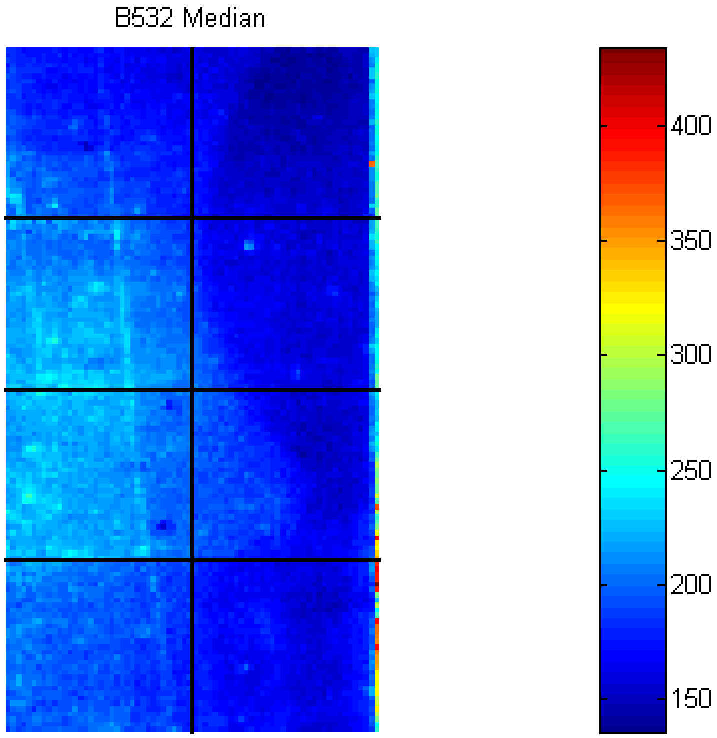
The MATLAB software plots an image for the background of the red channel. Notice the very high background levels down the right side of the array.



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')
```

The MATLAB software 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')
```

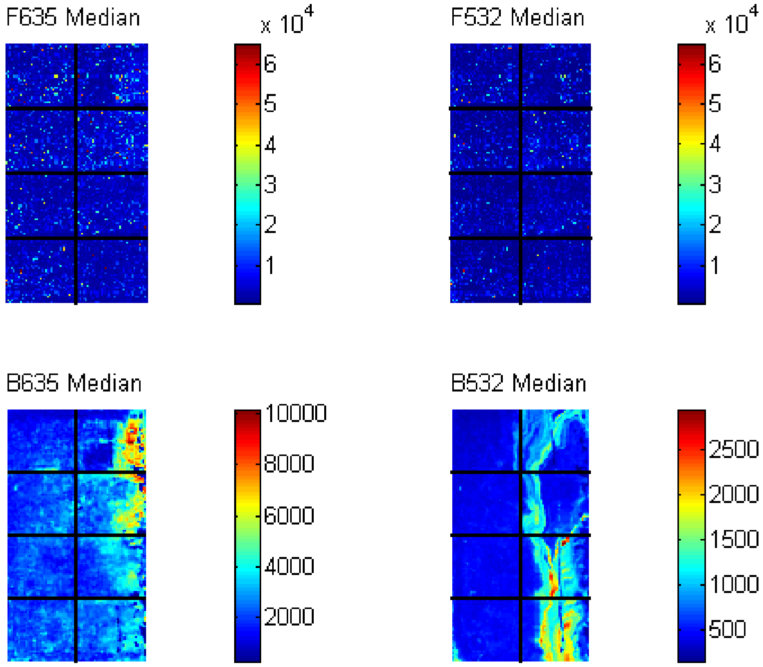
The MATLAB software creates a structure and displays information about the structure.

```
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')
```

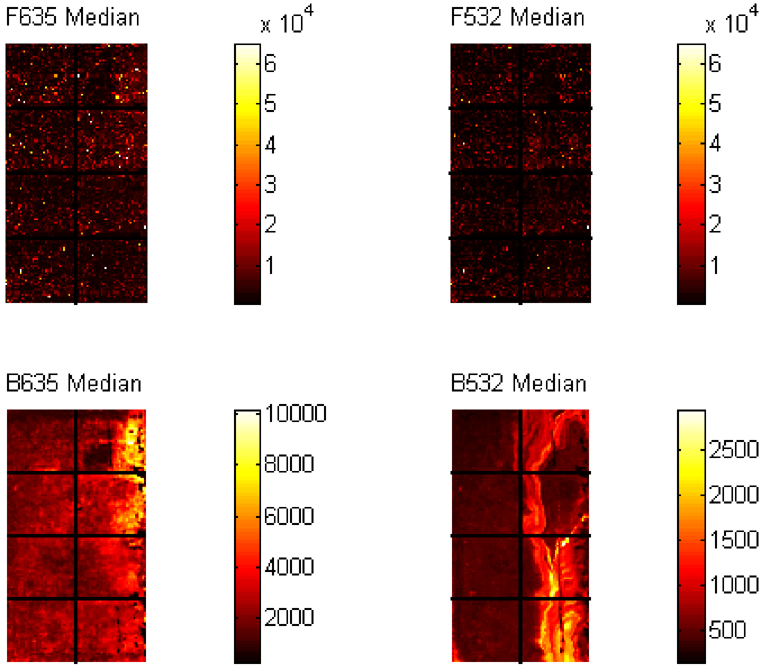
The MATLAB software plots the images.



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
```

The MATLAB software 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'))
```

The MATLAB software displays:

```
b532MedCol =
    16
```

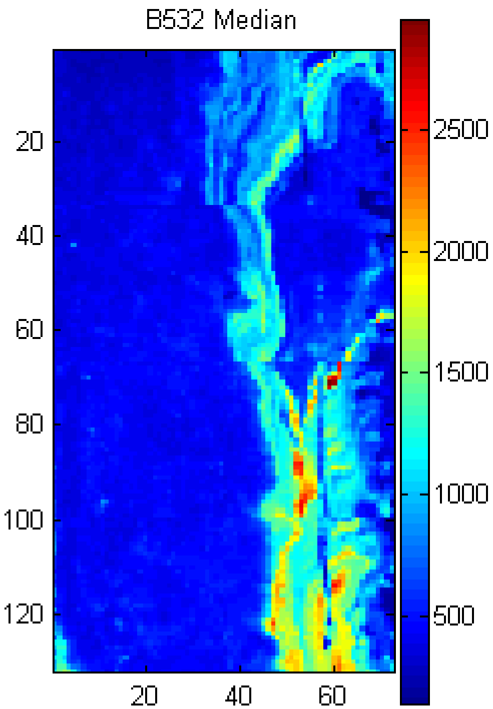
9 Extract that column from the field `Data`.

```
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')
```

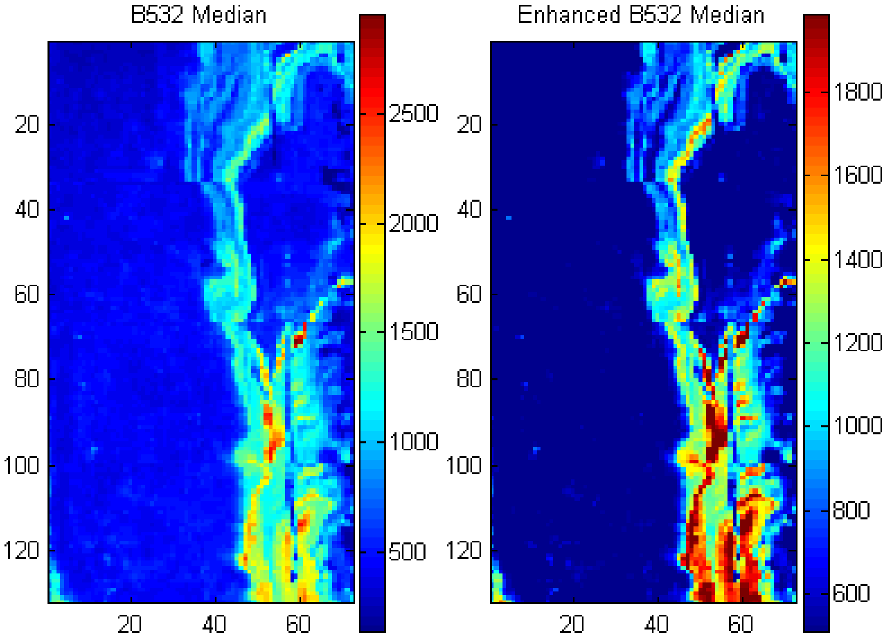
The MATLAB software plots the image.



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')
```

The MATLAB software plots the images.



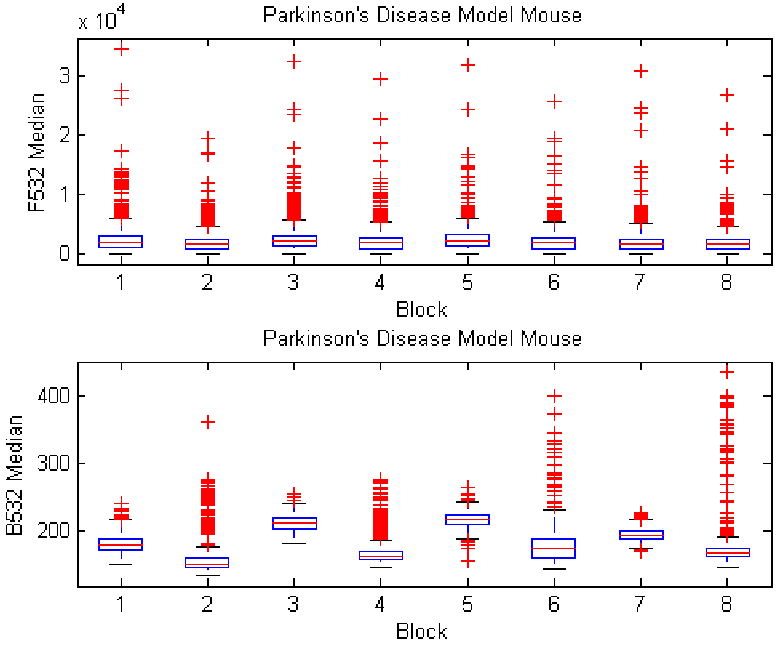
Statistics of the Microarrays

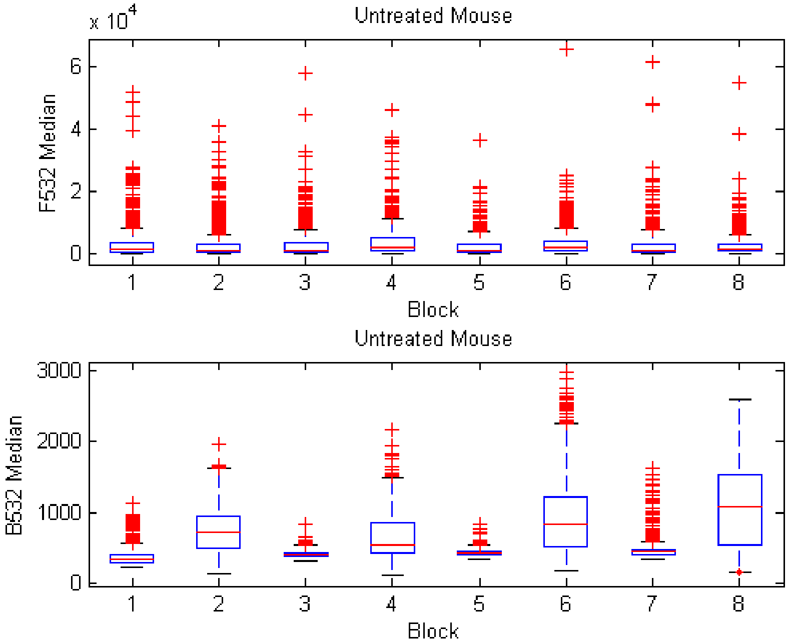
This procedure illustrates how to visualize distributions in microarray data. You can use the function maboxplot to look at the distribution of data in each of the blocks.

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')
```

The MATLAB software plots the images.


Parkinson's Disease Model Mouse
x 10^4
F532 Median
0
1
2
3
1
2
3
4
5
6
7
8
Block
Parkinson's Disease Model Mouse
B532 Median
200
300
400
1
2
3
4
5
6
7
8
Block



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

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

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

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

The MATLAB software displays:

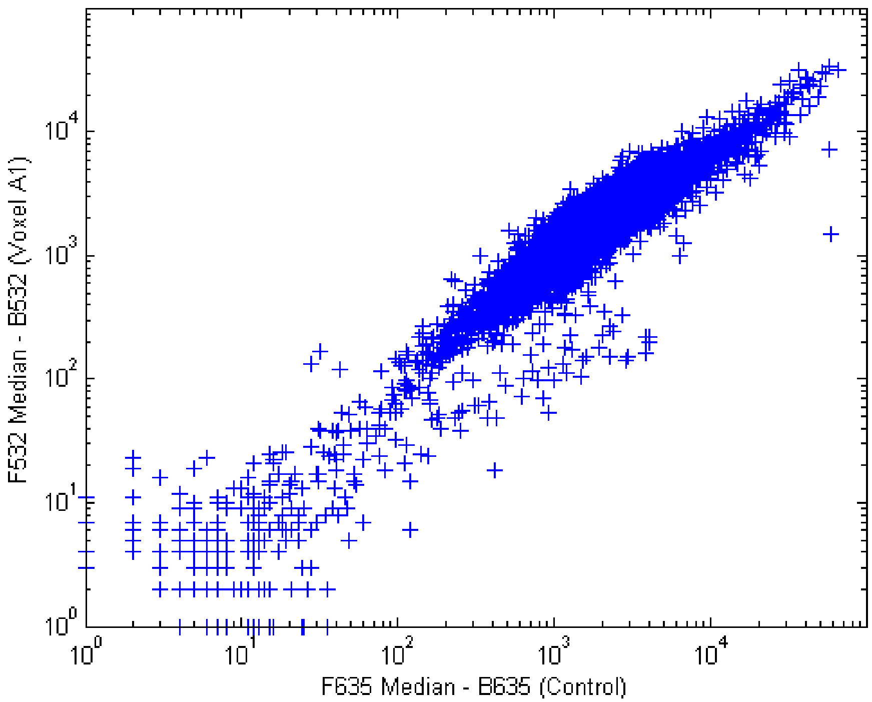
```
cy5DataCol =
    34

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)');
```

The MATLAB software displays the following messages and plots the images.

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



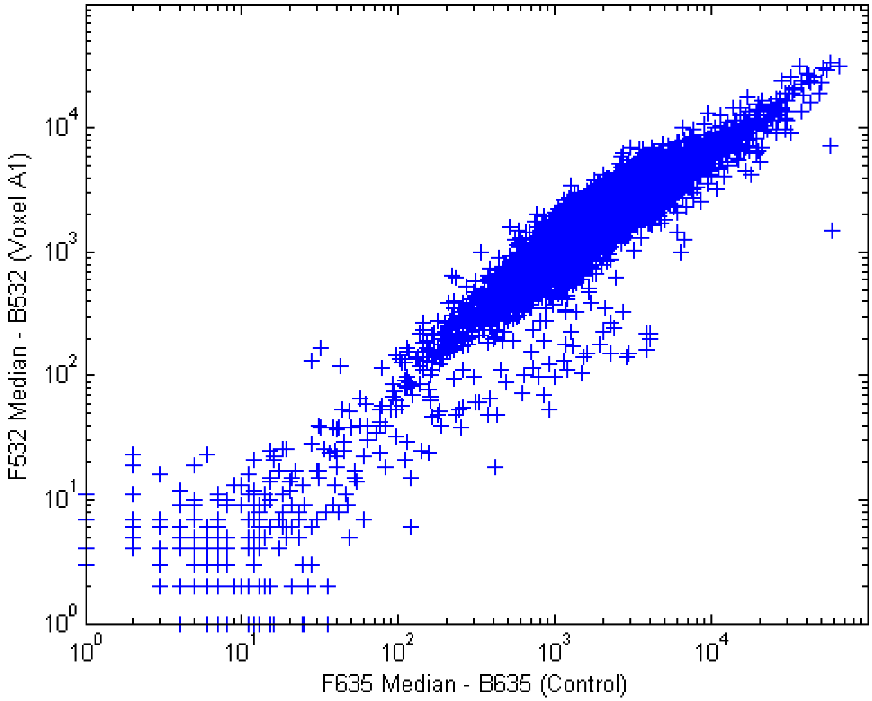
Notice that this function gives some warnings about negative and zero elements. This is because some of the values in the `'F635 Median - B635'` and `'F532 Median - B532'` columns are zero or even less than zero. Spots where this happened might be bad spots or spots that failed to hybridize. Points with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

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

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

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

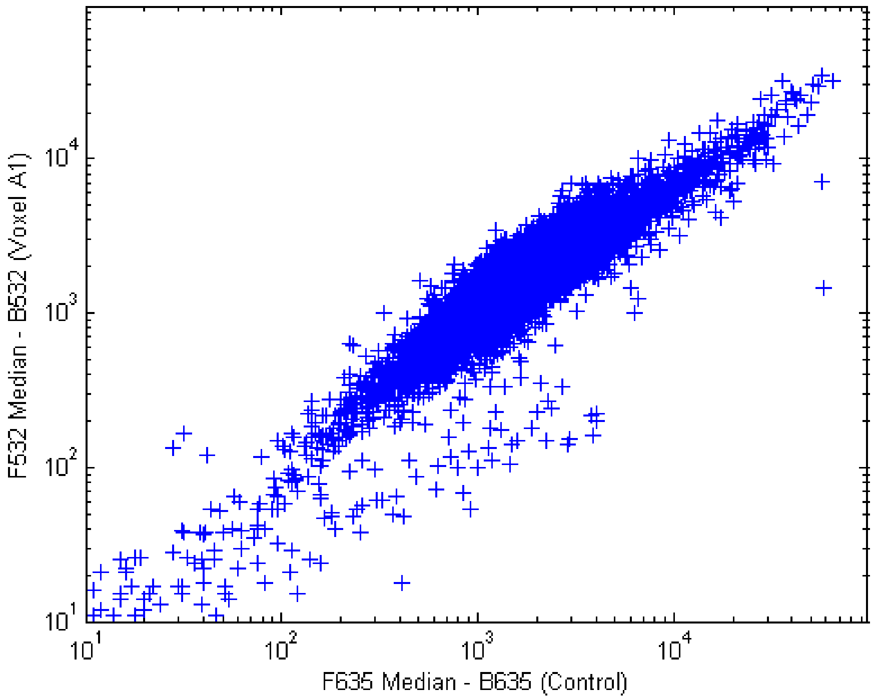
The MATLAB software plots the image.



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);
```

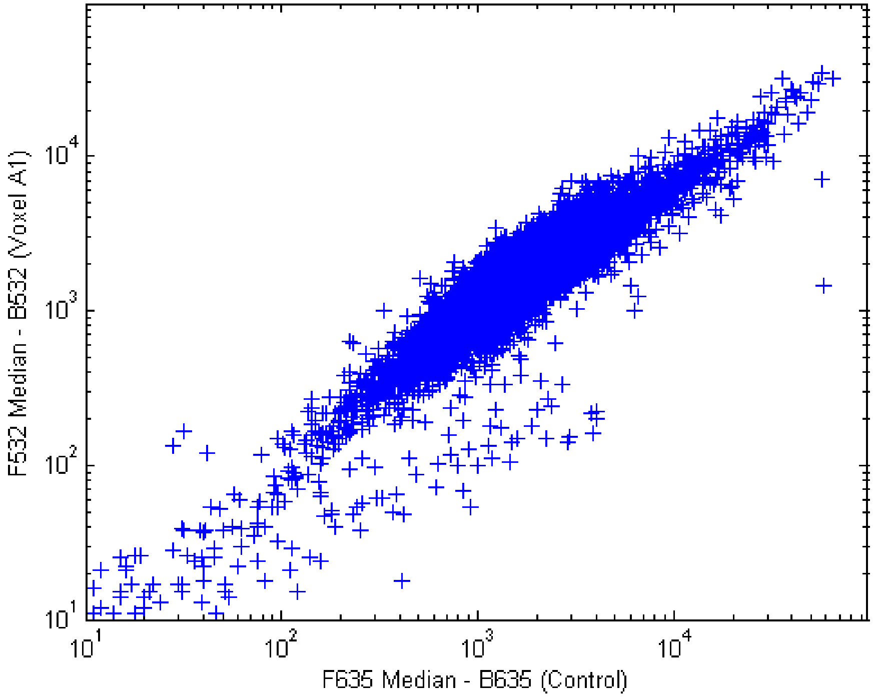
The MATLAB software plots the image.



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)');
```

The MATLAB software plots the image.

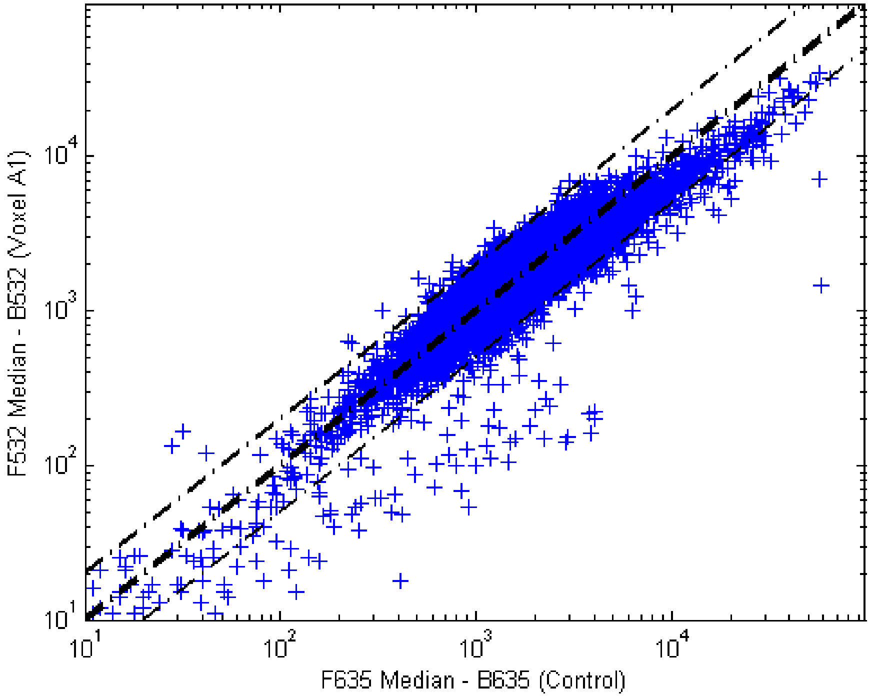


This plot shows the distribution of points but does not give any indication about which genes correspond to which points.

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)');
```

The MATLAB software 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.

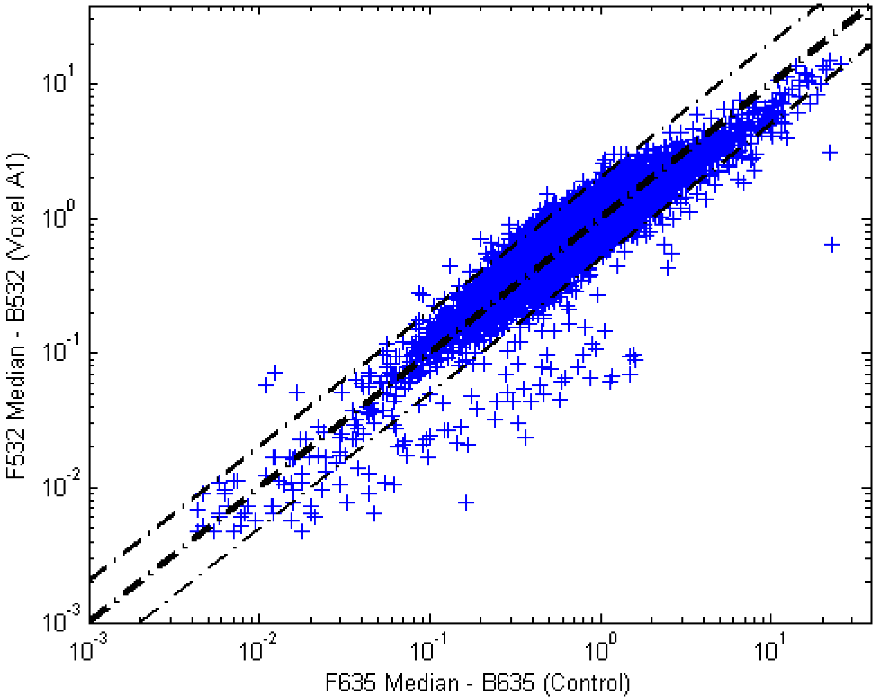
```
normcy5 = mameannorm(cy5Data);
normcy3 = mameannorm(cy3Data);
```

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)');
```

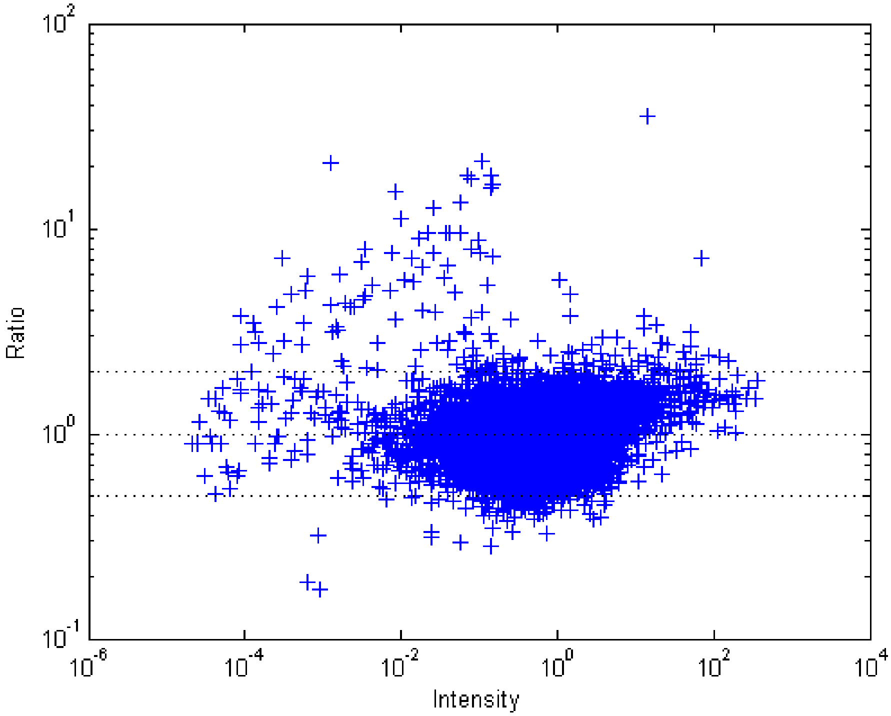
The MATLAB software plots the image.



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)
```

The MATLAB software plots the image.



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

Example: Analyzing Gene Expression Profiles

In this section...

Overview of the Yeast Example

This example demonstrates a number of ways to look for patterns in gene expression profiles, using gene expression data from yeast shifting from fermentation to respiration.

The microarray data for this example is from DeRisi, J.L., Iyer, V.R., and Brown, P.O. (Oct 24, 1997). Exploring the metabolic and genetic control of gene expression on a genomic scale. Science, *278 (5338)*, 680–686. PMID: 9381177.

The authors used DNA microarrays to study temporal gene expression of almost all genes in *Saccharomyces cerevisiae* during the metabolic shift from fermentation to respiration. Expression levels were measured at seven time points during the diauxic shift. The full data set can be downloaded from the Gene Expression Omnibus Web site at:

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

Exploring the Data Set

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

1 Load data into the MATLAB environment.

```
load yeastdata.mat
```

2 Get the size of the data by typing

```
numel(genes)
```

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

```
ans =
        6400
```

3 Access the entries using cell array indexing.

```
genes{15}
```

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

```
ans =
  YAL054C
```

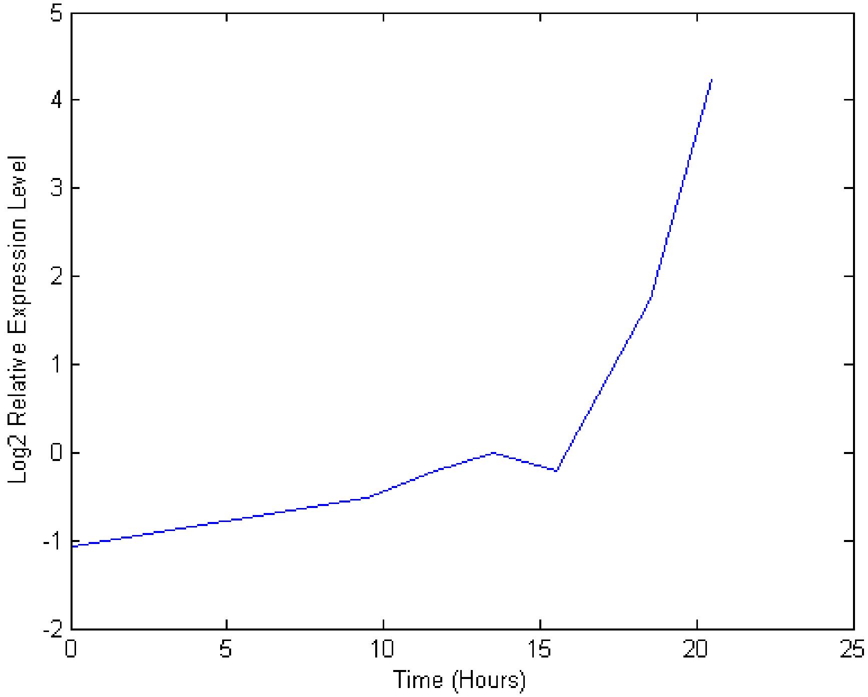
4 Use the function `web` to access information about this ORF in the Saccharomyces Genome Database (SGD).

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

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

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

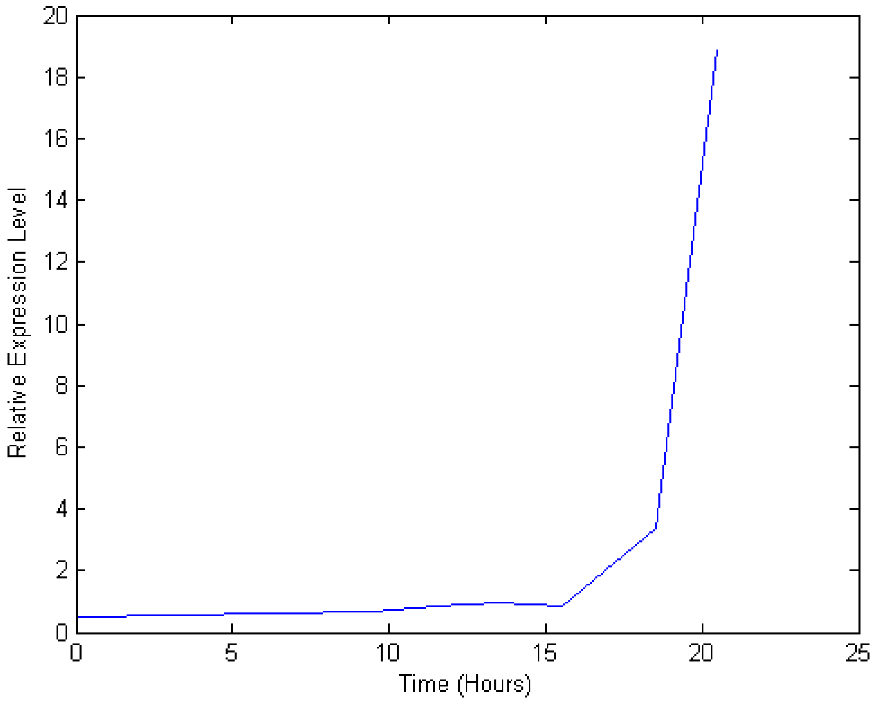
The MATLAB software plots the figure. The values are $\log_2$ ratios.



6 Plot the actual values.

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

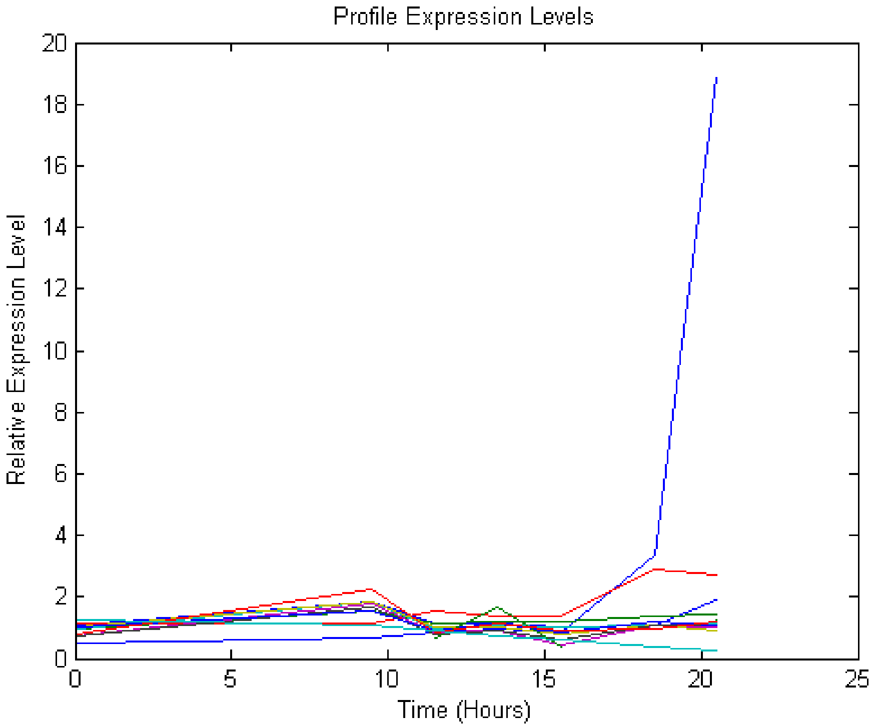
The MATLAB software plots the figure. The gene associated with this ORF, ACS1, appears to be strongly up-regulated during the diauxic shift.



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');
```

The MATLAB software plots the image.



Filtering Genes

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

1 If you look through the gene list you will see several spots marked as `'EMPTY'`. These are empty spots on the array, and while they might have data associated with them, for the purposes of this example, you can

consider these points to be noise. These points can be found using the strcmp function and removed from the data set with indexing commands.

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

The MATLAB software 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 the isnan function 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)
```

The MATLAB software 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 toolbox to remove genes with various types of profiles that do not provide useful information about genes affected by the metabolic change.

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

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

The MATLAB software 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)
```

The MATLAB software 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)
```

The MATLAB software 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 software.

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');
```

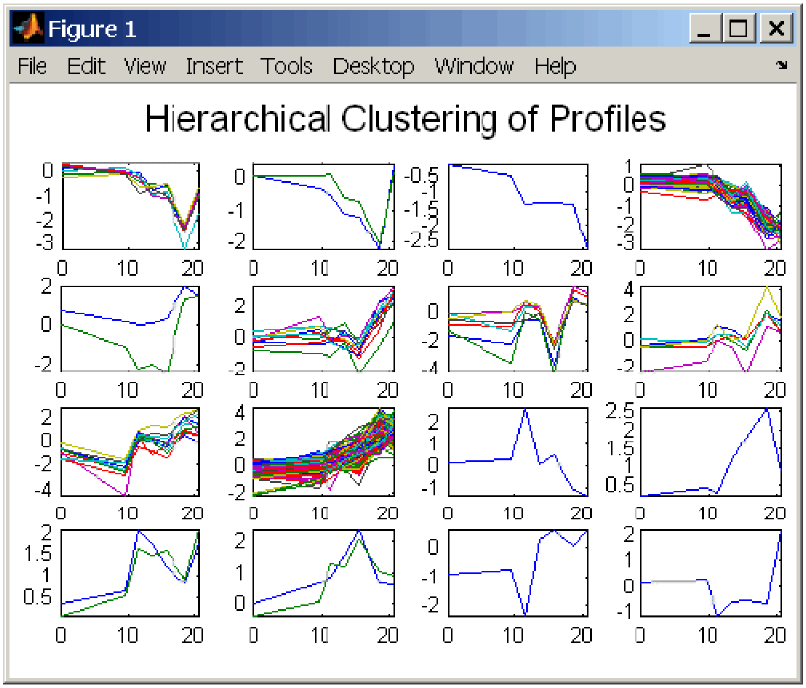
2 The function cluster calculates the clusters based on either a cutoff distance or a maximum number of clusters. In this case, the 'maxclust' option is used to identify 16 distinct clusters.

```
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');
```

The MATLAB software plots the images.

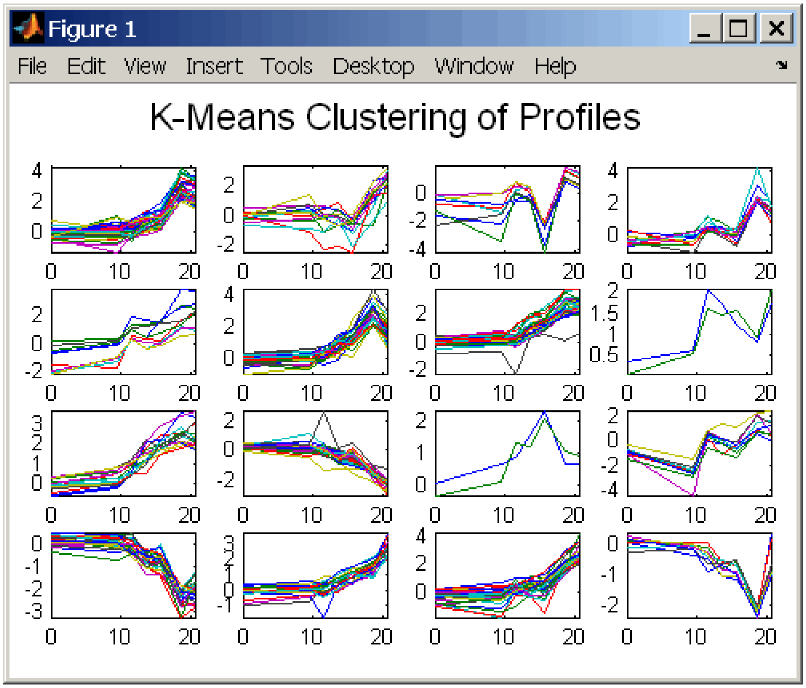


4 The Statistics Toolbox software also has a K-means clustering function. Again, 16 clusters are found, but because the algorithm is different these are not necessarily the same clusters as those found by hierarchical clustering.

```
[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');
```

The MATLAB software displays:

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



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

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

The MATLAB software plots the figure:

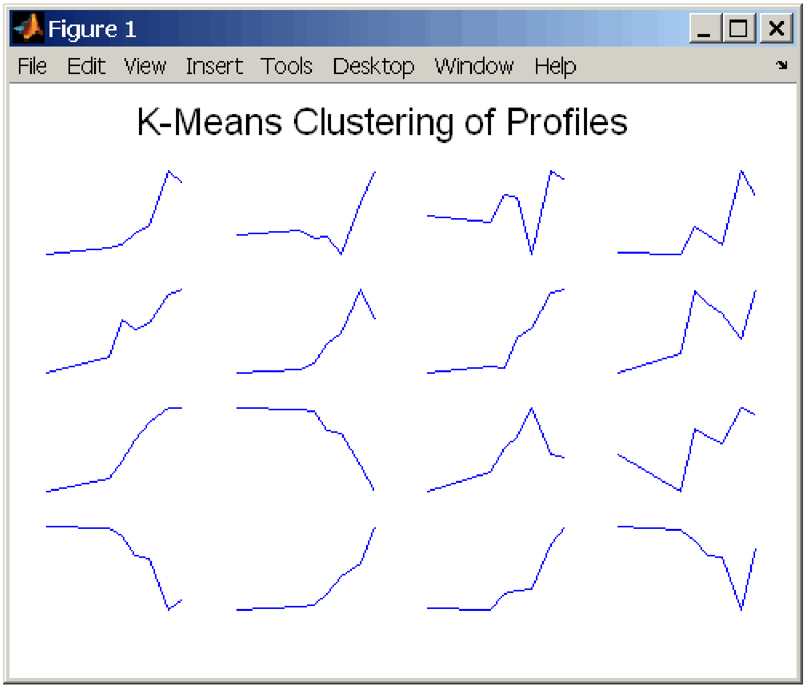
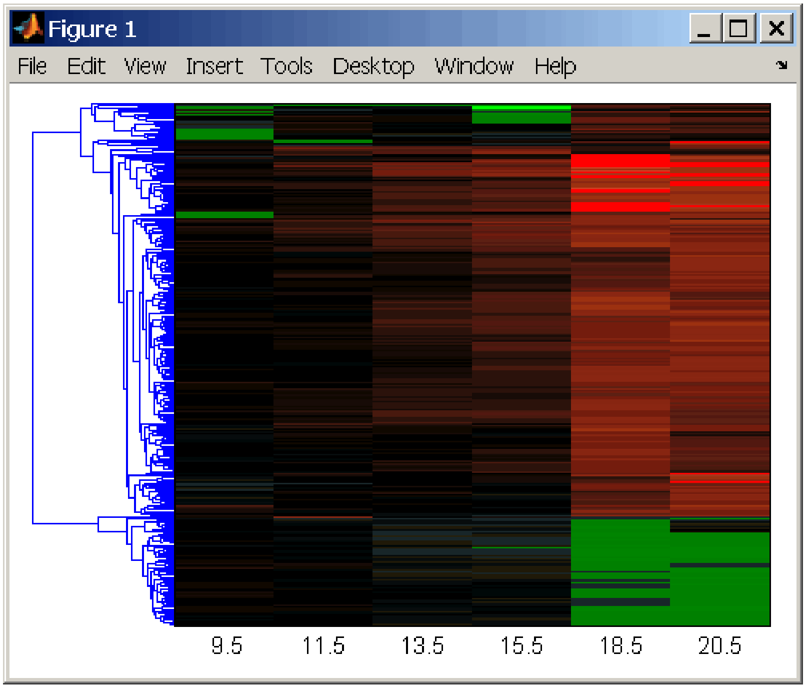


Figure 1
File Edit View Insert Tools Desktop Window Help
K-Means Clustering of Profiles

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))
```

The MATLAB software 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. You can also use PCA to find signals in noisy data.

1 Use the `princomp` function in the Statistics Toolbox software to calculate the principal components of a data set.

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

The MATLAB software 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)
```

The MATLAB software 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');
```

The MATLAB software plots the figure:

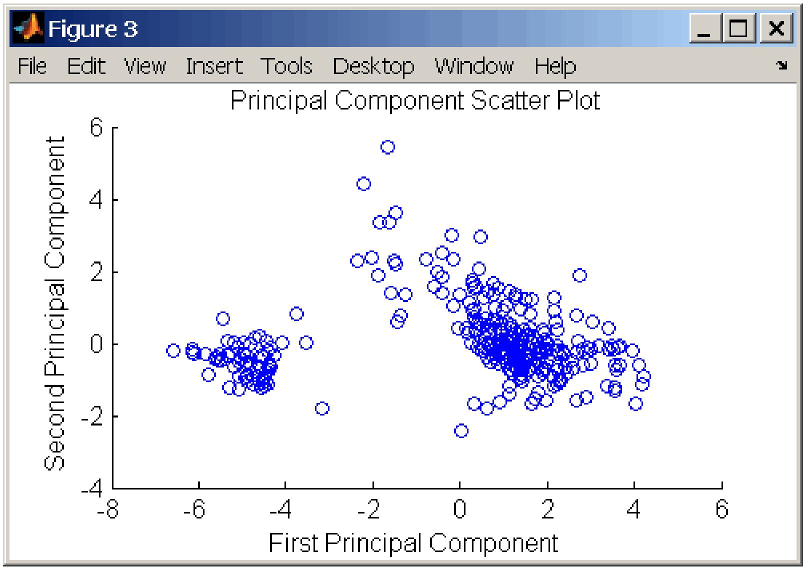


Figure 3
File Edit View Insert Tools Desktop Window Help
Principal Component Scatter Plot
Second Principal Component
6
4
2
0
-2
-4
-8
-6
-4
-2
0
2
4
6
First Principal Component

4 The gname function from the Statistics Toolbox software 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 gscatter function from the Statistics Toolbox software. 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.
```

The MATLAB software plots the figure:

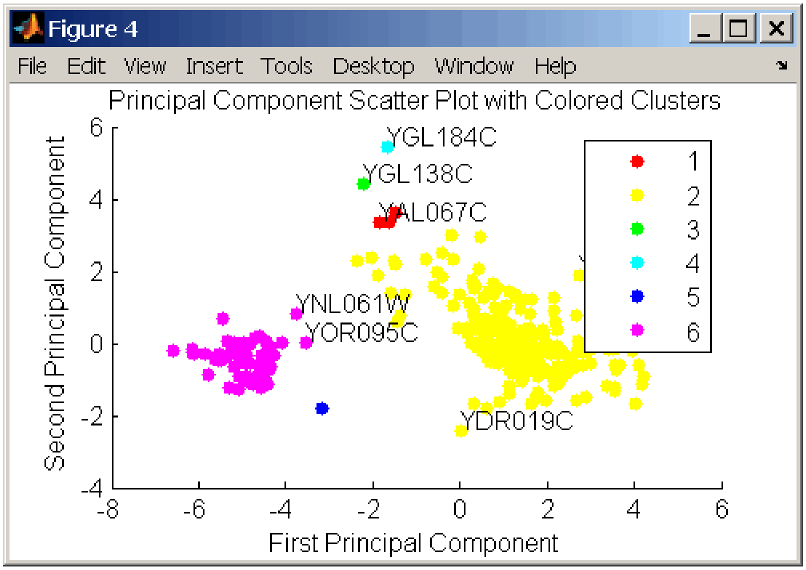


Figure 4
File Edit View Insert Tools Desktop Window Help
Principal Component Scatter Plot with Colored Clusters
Second Principal Component
6
4
2
0
-2
-4
-8
-6
-4
-2
0
2
4
6
First Principal Component
YGL184C
YGL138C
YAL067C
YNL061W
YOR095C
YDR019C
● 1
● 2
● 3
● 4
● 5
● 6

Detecting DNA Copy Number Alteration in Array-Based CGH Data

This example detects DNA copy number alterations in genome-wide array-based comparative genomic hybridization (CGH) data, using various functions of MATLAB and its toolboxes.

Introduction

Copy number changes or alterations is a form of genetic variation in the human genome [1]. DNA copy number alterations (CNAs) have been linked to the development and progression of cancer and many diseases.

DNA microarray based comparative genomic hybridization (CGH) is a technique allows simultaneous monitoring of copy number of thousands of genes throughout the genome [2,3]. In this technique, DNA fragments or "clones" from a test sample and a reference sample differentially labeled with dyes (typically, Cy3 and Cy5) are hybridized to mapped DNA microarrays and imaged. Copy number alterations are related to the Cy3 and Cy5 fluorescence intensity ratio of the targets hybridized to each probe on a microarray. Clones with normalized test intensities significantly greater than reference intensities indicate copy number gains in the test sample at those positions. Similarly, significantly lower intensities in the test sample are signs of copy number loss. BAC (bacterial artificial chromosome) clone based CGH arrays have a resolution in the order of one million base pairs (1Mb) [3]. Oligonucleotide and cDNA arrays provide a higher resolution of 50-100kb [2].

Array CGH log2-based intensity ratios provide useful information about genome-wide CNAs. In humans, the normal DNA copy number is two for all the autosomes. In an ideal situation, the normal clones would correspond to a log2 ratio of zero. The log2 intensity ratios of a single copy loss would be -1, and a single copy gain would be 0.58. The goal is to effectively identify locations of gains or losses of DNA copy number.

The data in this example is the Coriell cell line BAC array CGH data analyzed by Snijders et al.(2001). The Coriell cell line data is widely regarded as a "gold standard" data set. You can download this data of normalized log2-based intensity ratios and the supplemental table of known karyotypes from http://www.nature.com/ng/journal/v29/n3/suppinfo/ng754_S1.html. You will compare these cytogenically mapped alterations with the locations of gains or losses identified with various functions of MATLAB and its toolboxes.

For this example, the Coriell cell line data are provided in a MAT file. The data file `coriell_baccgh.mat` contains `coriell_data`, a structure containing of the normalized average of the log2-based test to reference intensity ratios of 15 fibroblast cell lines and their genomic positions. The BAC targets are ordered by genome position beginning at *1p* and ending at *Xq*.

```
load coriell_baccgh
coriell_data


coriell_data = 

              Sample: {1x15 cell}
          Chromosome: [2285x1 int8]
     GenomicPosition: [2285x1 int32]
           Log2Ratio: [2285x15 double]
             FISHMap: {2285x1 cell}
```

Visualizing the Genome Profile of the Array CGH Data Set

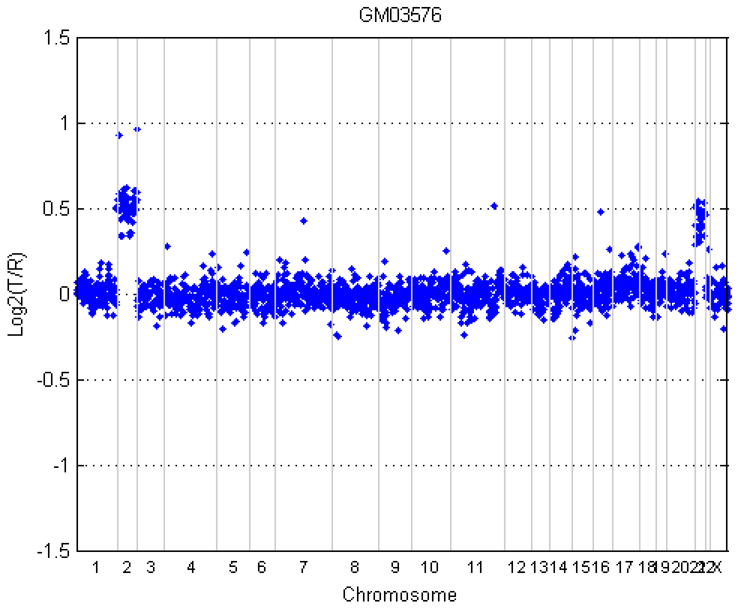
You can plot the genome wide log2-based test/reference intensity ratios of DNA clones. In this example, you will display the log2 intensity ratios for cell line GM03576 for chromosomes 1 through 23.

Find the sample index for the CM03576 cell line.

```
sample = find(strcmpi(coriell_data.Sample, 'GM03576'))


sample =

     8
```

To label chromosomes and draw the chromosome borders, you need to find the number of data points of in each chromosome.

```
chr_nums = zeros(1, 23);
chr_data_len = zeros(1,23);
for c = 1:23
    tmp = coriell_data.Chromosome == c;
    chr_nums(c) = find(tmp, 1, 'last');
    chr_data_len(c) = length(find(tmp));
end

% Draw a vertical bar at the end of a chromosome to indicate the border
x_vbar = repmat(chr_nums, 3, 1);
y_vbar = repmat([2;-2;NaN], 1, 23);

% Label the autosomes with their chromosome numbers, and the sex chromoso
% with X.
x_label = chr_nums - ceil(chr_data_len/2);
y_label = zeros(1, length(x_label)) - 1.6;
chr_labels=num2str((1:1:23)');
chr_labels = cellstr(chr_labels);
chr_labels{end} = 'X';

figure;hold on
h_ratio = plot(coriell_data.Log2Ratio(:,sample), '.');
h_vbar = line(x_vbar, y_vbar, 'color', [0.8 0.8 0.8]);
h_text = text(x_label, y_label, chr_labels,...
             'fontsize', 8, 'HorizontalAlignment', 'center');

h_axis = get(h_ratio, 'parent');
set(h_axis, 'xtick', [], 'ygrid', 'on', 'box', 'on',...
            'xlim', [0 chr_nums(23)], 'ylim', [-1.5 1.5])

title(coriell_data.Sample{sample})
xlabel({'', 'Chromosome'})
ylabel('Log2(T/R)')
hold off
```

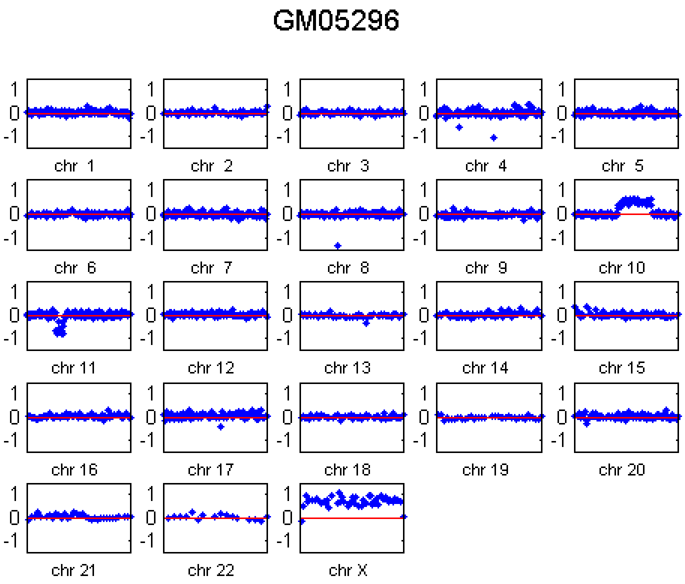


In the plot, borders between chromosomes are indicated by grey vertical bars. The plot indicates that the GM03576 cell line is trisomic for chromosomes 2 and 21 [3].

You can also plot the profile of each chromosome in a genome. In this example, you will display the log2 intensity ratios for each chromosome in cell line GM05296 individually.

```
sample = find(strcmpi(coriell_data.Sample, 'GM05296'));
figure;
for c = 1:23
    idx = coriell_data.Chromosome == c;
    chr_y = coriell_data.Log2Ratio(idx, sample);
    subplot(5,5,c);

    hp = plot(chr_y, '.');
    line([0, chr_data_len(c)], [0,0], 'color', 'r');

    h_axis = get(hp, 'Parent');
```

```
        set(h_axis, 'xtick', [], 'Box', 'on',...
                'xlim', [0 chr_data_len(c)], 'ylim', [-1.5 1.5])
        xlabel(['chr ' chr_labels{c}], 'FontSize', 8)
    end
    suptitle('GM05296');
```



The plot indicates the GM05296 cell line has a partial trisomy at chromosome 10 and a partial monosomy at chromosome 11.

Observe that the gains and losses of copy number are discrete. These alterations occur in contiguous regions of a chromosome that cover several clones to entitle chromosome.

The array-based CGH data can be quite noisy. Therefore, accurate identification of chromosome regions of equal copy number that accounts for the noise in the data requires robust computational methods. In the rest of this example, you will work with the data of chromosomes 9, 10 and 11 of the GM05296 cell line.

Initialize a structure array for the data of these three chromosomes.

```
GM05296_Data = struct('Chromosome', {9 10 11},...
                      'GenomicPosition', {[], [], []},...
                      'Log2Ratio', {[], [], []},...
                      'SmoothedRatio', {[], [], []},...
                      'DiffRatio', {[], [], []},...
                      'SegIndex', {[], [], []});
```

Filtering and Smoothing Data

A simple approach to perform high-level smoothing is to use a median filter. The median filter removes outliers while preserving sustained changes in the input data. The median filter function, medfilt1, is available with Signal Processing Toolbox™.

```
for iloop = 1:length(GM05296_Data)
    idx = coriell_data.Chromosome == GM05296_Data(iloop).Chromosome;
    chr_x = coriell_data.GenomicPosition(idx);
    chr_y = coriell_data.Log2Ratio(idx, sample);

    % Remove NaN data points
    idx = ~isnan(chr_y);
    GM05296_Data(iloop).GenomicPosition = chr_x(idx);
    GM05296_Data(iloop).Log2Ratio = chr_y(idx);

    % Apply a median filter
    GM05296_Data(iloop).SmoothedRatio = medfilt1(GM05296_Data(iloop).Log2

    % Find the derivative of the smoothed ratio
    GM05296_Data(iloop).DiffRatio = diff([0; GM05296_Data(iloop).Smoothed
end
```

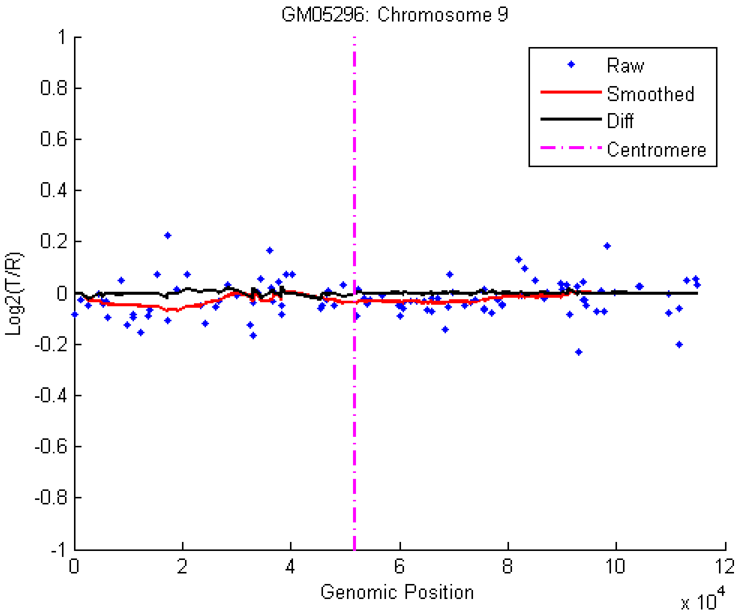
To better visualize and later validate the locations of copy number changes, we need cytoband information. Read the human cytoband information from the hs_cytoBand.txt data file using the cytobandread function. It returns a structure of human cytoband information [4].

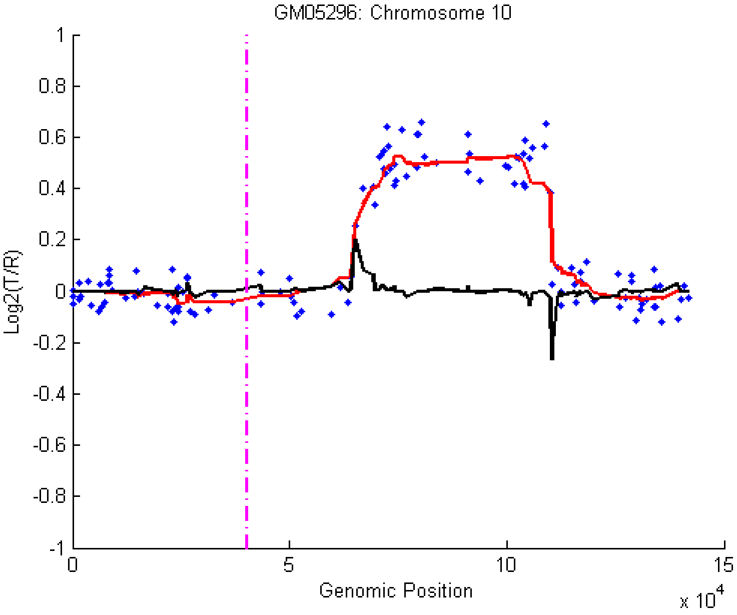
```
hs_cytobands = cytobandread('hs_cytoBand.txt')

% Find the centromere positions for the chromosomes.
```

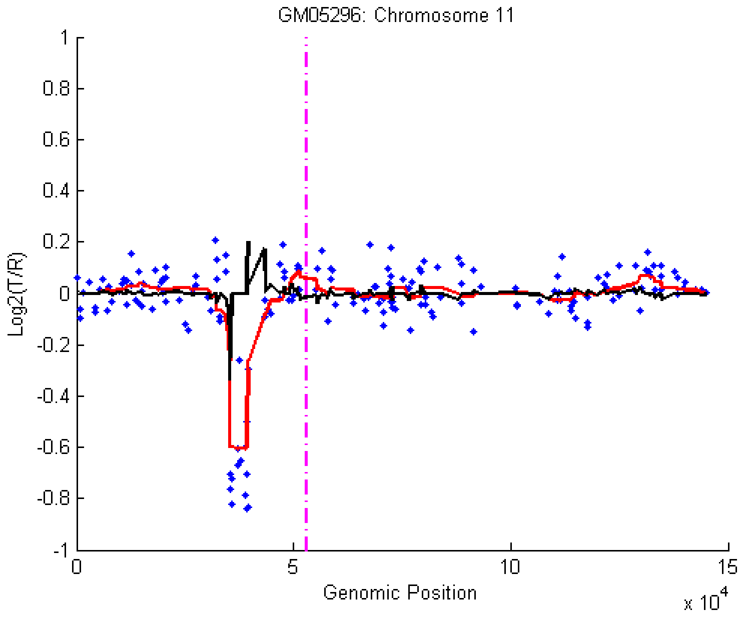
```
acen_idx = strcmpi(hs_cytobands.GieStains, 'acen');
acen_ends = hs_cytobands.BandEndBPs(acen_idx);

% Convert the cytoband data from bp to kilo bp because the genomic
% positions in Coriell Cell Line data set are in kilo base pairs.
acen_pos = acen_ends(1:2:end)/1000;


hs_cytobands =

     ChromLabels: {862x1 cell}
    BandStartBPs: [862x1 int32]
      BandEndBPs: [862x1 int32]
      BandLabels: {862x1 cell}
       GieStains: {862x1 cell}
```

You can inspect the data by plotting the log2-based ratios, the smoothed ratios and the derivative of the smoothed ratios together. You can also display the centromere position of a chromosome in the data plots. The magenta vertical bar marks the centromere of the chromosome.

```
for iloop = 1:length(GM05296_Data)
    chr = GM05296_Data(iloop).Chromosome;
    chr_x = GM05296_Data(iloop).GenomicPosition;
    figure; hold on
    plot(chr_x, GM05296_Data(iloop).Log2Ratio, '.');
    line(chr_x, GM05296_Data(iloop).SmoothedRatio,...
                'Color', 'r', 'LineWidth', 2);
    line(chr_x, GM05296_Data(iloop).DiffRatio,...
                'Color', 'k', 'LineWidth', 2);
    line([acen_pos(chr), acen_pos(chr)], [-1, 1],...
                'Color', 'm', 'LineWidth', 2, 'LineStyle', '-.');
    if iloop == 1
        legend('Raw','Smoothed','Diff', 'Centromere');
    end
    ylim([-1, 1])
    xlabel('Genomic Position')
    ylabel('Log2(T/R)')
    title(sprintf('GM05296: Chromosome %d ', chr))
    hold off
end
```


GM05296: Chromosome 9
Raw
Smoothed
Diff
Centromere
Log2(T/R)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
2
4
6
8
10
12
Genomic Position
x 10^4


GM05296: Chromosome 10
Log2(T/R)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
5
10
15
Genomic Position
x 10^4



Detecting Change-Points

The derivatives of the smoothed ratio over a certain threshold usually indicate substantial changes with large peaks, and provide the estimate of the change-point indices. For this example you will select a threshold of 0.1.

```
thrd = 0.1;

for iloop = 1:length(GM05296_Data)
    idx = find(abs(GM05296_Data(iloop).DiffRatio) > thrd );
    N = numel(GM05296_Data(iloop).SmoothedRatio);
    GM05296_Data(iloop).SegIndex = [1;idx;N];

    % Number of possible segments found
    disp(sprintf('%d segments initially found on Chromosome %d',...
                 numel(GM05296_Data(iloop).SegIndex) - 1,...
                 GM05296_Data(iloop).Chromosome));
end
```

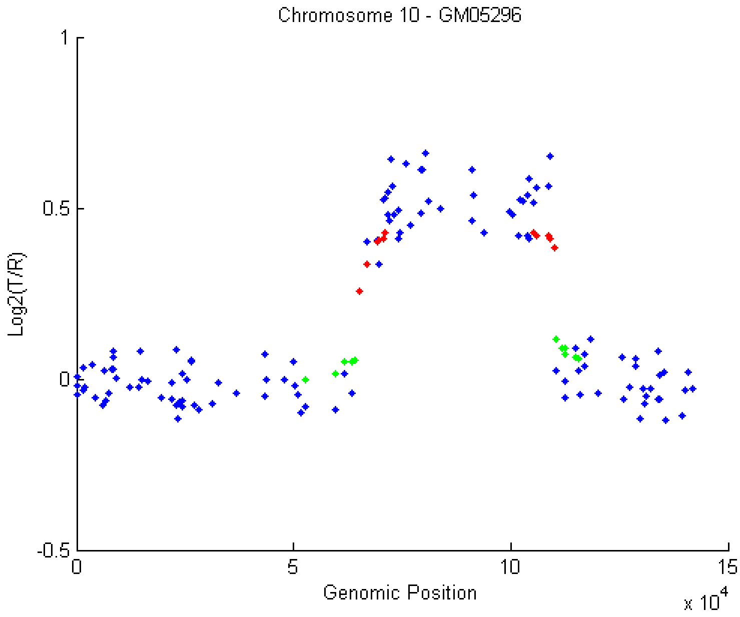
```
1 segments initially found on Chromosome 9
3 segments initially found on Chromosome 10
5 segments initially found on Chromosome 11
```

Optimizing Change-Points by GM Clustering

Gaussian Mixture (GM) or Expectation-Maximization (EM) clustering can provide fine adjustments to the change-point indices [5]. The convergence to statistically optimal change-point indices can be facilitated by surrounding each index with equal-length set of adjacent indices. Thus each edge is associated with left and right distributions. The GM clustering learns the maximum-likelihood parameters of the two distributions. It then optimally adjusts the indices given the learned parameters.

You can set the length for the set of adjacent positions distributed around the change-point indices. For this example, you will select a length of 5. You can also inspect each change-point by plotting its GM clusters. In this example, you will plot the GM clusters for the Chromosome 10 data.

```
len = 5;
for iloop = 1:length(GM05296_Data)
    seg_num = numel(GM05296_Data(iloop).SegIndex) - 1;
    if seg_num > 1
        % Plot the data points in chromosome 10 data
        if GM05296_Data(iloop).Chromosome == 10
            figure; hold on;
            plot(GM05296_Data(iloop).GenomicPosition,...
                 GM05296_Data(iloop).Log2Ratio, '.')
            ylim([-0.5, 1])
            xlabel('Genomic Position')
            ylabel('Log2(T/R)')
            title(sprintf('Chromosome %d - GM05296', GM05296_Data(iloop)
        end

        segidx = GM05296_Data(iloop).SegIndex;
        segidx_emadj = GM05296_Data(iloop).SegIndex;

        for jloop = 2:seg_num
            ileft = min(segidx(jloop) - len, segidx(jloop));
            iright = max(segidx(jloop) + len, segidx(jloop));
            gmx = GM05296_Data(iloop).GenomicPosition(ileft:iright);
            gmy = GM05296_Data(iloop).SmoothedRatio(ileft:iright);

            % Select initial guess for the of cluster index for each poi
            gmpart = (gmy > (min(gmy) + range(gmy)/2)) + 1;

            % Create a Gaussian mixture model object
            gm = gmdistribution.fit(gmy, 2, 'start', gmpart);
            gmid = gm.cluster(gmy);

            segidx_emadj(jloop) = find(abs(diff(gmid))==1) + ileft;

          % Plot GM clusters for the change-points in chromosome 10 data
            if GM05296_Data(iloop).Chromosome == 10
                plot(gmx(gmid==1),gmy(gmid==1), 'g.',...
                     gmx(gmid==2), gmy(gmid==2), 'r.')
            end
        end

        % Remove repeat indices
        zeroidx = [diff(segidx_emadj) == 0; 0];
        GM05296_Data(iloop).SegIndex = segidx_emadj(~zeroidx);
    end

    % Number of possible segments found
    disp(sprintf('%d segments found on Chromosome %d after GM clustering
                 numel(GM05296_Data(iloop).SegIndex) - 1,...
                 GM05296_Data(iloop).Chromosome));
end
hold off;

1 segments found on Chromosome 9 after GM clustering adjustment.
3 segments found on Chromosome 10 after GM clustering adjustment.
3 segments found on Chromosome 11 after GM clustering adjustment.
```



Testing Change-Point Significance

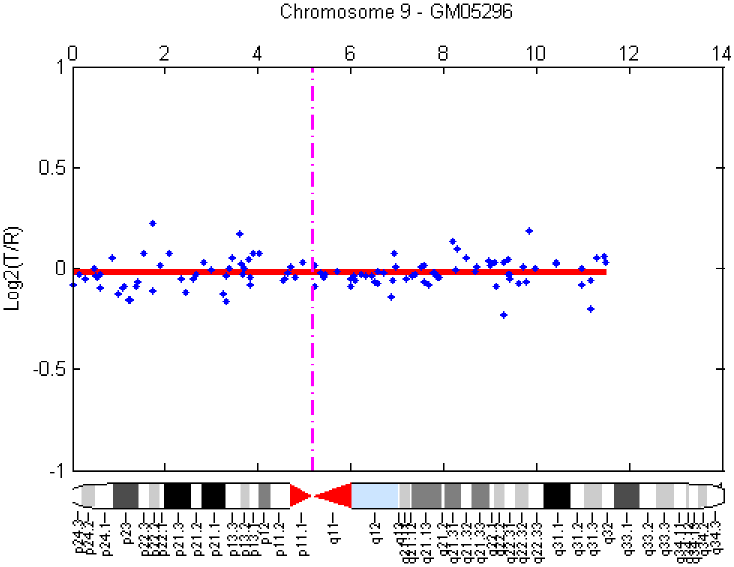
Once you determine the optimal change-point indices, you also need to determine if each segment represents a statistically significant changes in DNA copy number. You will perform permutation t-tests to assess the significance of the segments identified. A segment includes all the data points from one change-point to the next change-point or the chromosome end. In this example, you will perform 10,000 permutations of the data points on two consecutive segments along the chromosome at the significance level of 0.01.

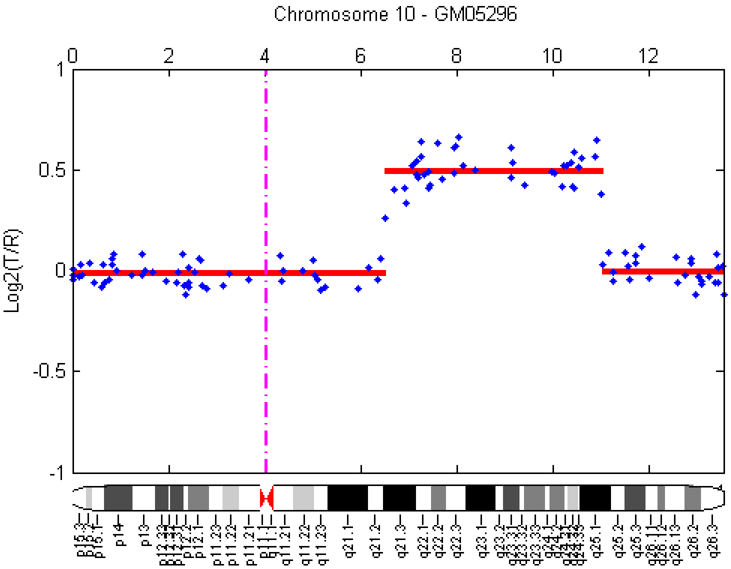
```
alpha = 0.01;
for iloop = 1:length(GM05296_Data)
    seg_num = numel(GM05296_Data(iloop).SegIndex) - 1;
    seg_index = GM05296_Data(iloop).SegIndex;
    if seg_num > 1
        ppvals = zeros(seg_num+1, 1);

        for sloop =  1:seg_num-1
            seg1idx = seg_index(sloop):seg_index(sloop+1)-1;
            if sloop== seg_num-1
                seg2idx = seg_index(sloop+1):(seg_index(sloop+2));
            else
                seg2idx = seg_index(sloop+1):(seg_index(sloop+2)-1);
            end

            seg1 = GM05296_Data(iloop).SmoothedRatio(seg1idx);
            seg2 = GM05296_Data(iloop).SmoothedRatio(seg2idx);

            n1 = numel(seg1);
            n2 = numel(seg2);
            N = n1+n2;
            segs = [seg1;seg2];

            % Compute observed t statistics
            t_obs = mean(seg1) - mean(seg2);

            % Permutation test
            iter = 10000;
            t_perm = zeros(iter,1);
            for i = 1:iter
                randseg = segs(randperm(N));
                t_perm(i) = abs(mean(randseg(1:n1)) - mean(randseg(n1+1:N
            end
            ppvals(sloop+1) = sum(t_perm >= abs(t_obs))/iter;
        end

        sigidx = ppvals < alpha;
        GM05296_Data(iloop).SegIndex = seg_index(sigidx);
    end

    % Number segments after significance tests
    disp(sprintf('%d segments found on Chromosome %d after significance t
        numel(GM05296_Data(iloop).SegIndex) - 1, GM05296_Data(iloop).Chro
end

1 segments found on Chromosome 9 after significance tests.
3 segments found on Chromosome 10 after significance tests.
3 segments found on Chromosome 11 after significance tests.
```

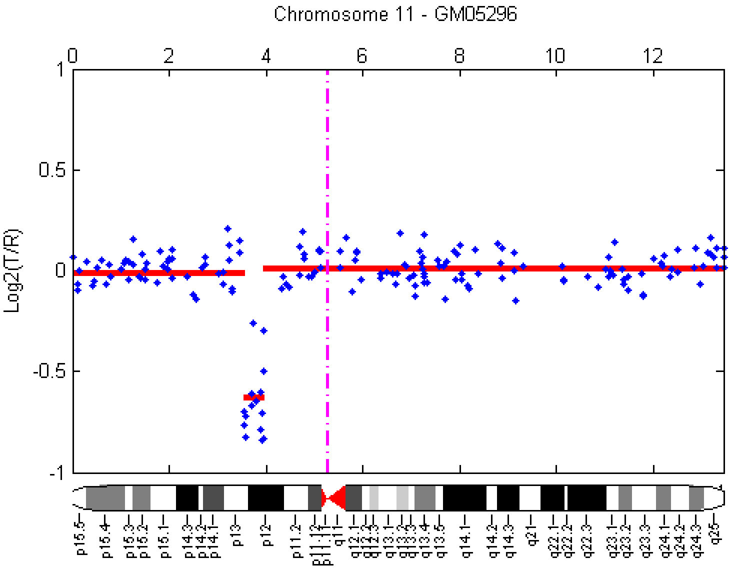
Assessing Copy Number Alterations

Cytogenetic study indicates cell line GM05296 has a trisomy at *10q21-10q24* and a monosomy at *11p12-11p13* [3]. Plot the segment means of the three chromosomes over the original data with bold red lines, and add the chromosome ideograms to the plots using the `chromosomeplot` function. Note that the genomic positions in the Coriell cell line data set are in kilo base pairs. Therefore, you will need to convert cytoband data from bp to kilo bp when adding the ideograms to the plot.

```
for iloop = 1:length(GM05296_Data)
    figure;
    seg_num = numel(GM05296_Data(iloop).SegIndex) - 1;
    seg_mean = ones(seg_num,1);
    chr_num = GM05296_Data(iloop).Chromosome;
    for jloop = 2:seg_num+1
        idx = GM05296_Data(iloop).SegIndex(jloop-1):GM05296_Data(iloop).S
        seg_mean(idx) = mean(GM05296_Data(iloop).Log2Ratio(idx));
        line(GM05296_Data(iloop).GenomicPosition(idx), seg_mean(idx),...
            'color', 'r', 'linewidth', 3);
    end
    line(GM05296_Data(iloop).GenomicPosition, GM05296_Data(iloop).Log2Rat
        'linestyle', 'none', 'Marker', '.');
    line([acen_pos(chr_num), acen_pos(chr_num)], [-1, 1],...
        'linewidth', 2,...
        'color', 'm',...
        'linestyle', '-.');

    ylabel('Log2(T/R)')
    set(gca, 'Box', 'on', 'ylim', [-1, 1])
    title(sprintf('Chromosome %d - GM05296', chr_num));
    chromosomeplot(hs_cytobands, chr_num, 'addtoplot', gca,  'unit', 2)

end
```


Chromosome 9 - GM05296
0
2
4
6
8
10
12
14
1
0.5
0
-0.5
-1
Log2(T/R)
p24.3
p24.2
p24.1
p23
p22.3
p22.2
p22.1
p21.3
p21.2
p21.1
p13.3
p13.2
p13.1
p12
p11.2
p11.1
q11
q12
q13
q21.11
q21.12
q21.13
q21.2
q21.31
q21.32
q21.33
q22.1
q22.2
q22.31
q22.32
q22.33
q31.1
q31.2
q31.3
q32
q33.1
q33.2
q33.3
q34.11
q34.12
q34.13
q34.2
q34.3


Chromosome 10 - GM05296
0
2
4
6
8
10
12
1
0.5
0
-0.5
-1
Log2(T/R)
p15.3
p15.2
p15.1
p14
p13
p12.33
p12.32
p12.31
p12.2
p12.1
p11.23
p11.22
p11.21
p11.1
q11.1
q11.21
q11.22
q11.23
q21.1
q21.2
q21.3
q22.1
q22.2
q22.3
q23.1
q23.2
q23.31
q23.32
q23.33
q24.1
q24.2
q24.31
q24.32
q24.33
q25.1
q25.2
q25.3
q26.11
q26.12
q26.13
q26.2
q26.3



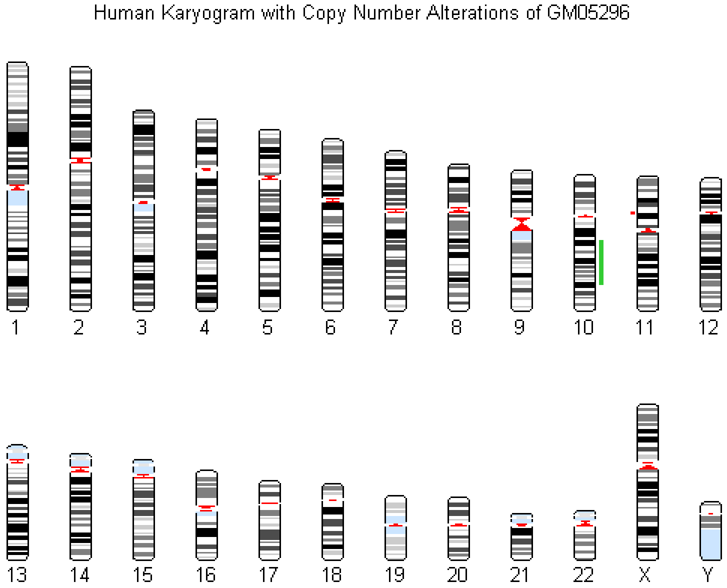
As shown in the plots, no copy number alterations were found on chromosome 9, there is copy number gain span from *10q21* to *10q24*, and a copy number loss region from *11p12* to *11p13*. The CNAs found match the known results in cell line GM05296 determined by cytogenetic analysis.

You can also display the CNAs of the GM05296 cell line align to the chromosome ideogram summary view using the `chromosomeplot` function. Determine the genomic positions for the CNAs on chromosomes 10 and 11.

```
chr10_idx = GM05296_Data(2).SegIndex(2):GM05296_Data(2).SegIndex(3)-1;
chr10_cna_start = GM05296_Data(2).GenomicPosition(chr10_idx(1))*1000;
chr10_cna_end   = GM05296_Data(2).GenomicPosition(chr10_idx(end))*1000;

chr11_idx = GM05296_Data(3).SegIndex(2):GM05296_Data(3).SegIndex(3)-1;
chr11_cna_start = GM05296_Data(3).GenomicPosition(chr11_idx(1))*1000;
chr11_cna_end = GM05296_Data(3).GenomicPosition(chr11_idx(end))*1000;
```

Create a structure containing the copy number alteration data from the GM05296 cell line data according to the input requirements of the `chromosomeplot` function.

```
cna_struct = struct('Chromosome', [10 11],...
                    'CNVType', [2 1],...
                    'Start', [chr10_cna_start, chr11_cna_start],...
                    'End',   [chr10_cna_end, chr11_cna_end])


cna_struct = 

    Chromosome: [10 11]
       CNVType: [2 1]
         Start: [65000000 35416000]
           End: [110000000 39389000]


chromosomeplot(hs_cytobands, 'cnv', cna_struct, 'unit', 2)
title('Human Karyogram with Copy Number Alterations of GM05296')
```



This example shows how MATLAB and its toolboxes provide tools for the analysis and visualization of copy-number alterations in array-based CGH data.

References

[1] Redon, R., Ishikawa, S., Fitch, K.R., et al. (2006). Global variation in copy number in the human genome. Nature 444, 444-454.

[2] Pinkel, D., Segraves, R., Sudar, D., Clark, S., Poole, I., Kowbel, D., Collins, C. Kuo, W.L., Chen, C., Zhai, Y., et al. (1998). High resolution analysis of DNA copy number variations using comparative genomic hybridization to microarrays. Nat. Genet. 20, 207-211.

[3] Snijders, A.M., Nowak, N., Segraves, R., Blackwood, S., Brown, N., Conroy, J., Hamilton, G., Hindle, A.K., Huey, B., Kimura, K., et al. (2001). Assembly of microarrays for genome-wide measurement of DNA copy number", Nat. Genet. 29, 263-264.

[4] Human Genome NCBI Build 36.

[5] Myers, C.L., Dunham, M.J., Kung, S.Y., and Troyanskaya, O.G. (2004). Accurate detection of aneuploidies in array CGH and gene expression microarray data. Bioinformatics 20, 18, 3533-3543.

Suggest an enhancement for this example.

Exploring Gene Expression Data

This example identifies differentially expressed genes, then uses Gene Ontology to determine their biological functions. It uses MATLAB and Bioinformatics Toolbox™ functions.

Introduction

Microarrays contain oligonucleotide or cDNA probes for comparing the expression profile of genes on a genomic scale. Determining if changes in gene expression are statistically significant between different conditions, e.g. two different tumor types, and determining the biological function of the differentially expressed genes, are important aims in a microarray experiment.

A publicly available dataset containing gene expression data of 42 tumor tissues of the embryonal central nervous system (CNS, Pomeroy et al. 2002) is used for this example. The samples were hybridized on Affymetrix HuGeneFL GeneChip arrays.

The CNS dataset (CEL files) is available at the CNS experiment web site. The 42 tumor tissue samples include 10 medulloblastomas, 10 rhabdoid, 10 malignant glioma, 8 supratentorial PNETS, and 4 normal human cerebella. The CNS raw dataset was preprocessed with the Robust Multi-array Average (RMA) and GC Robust Multi-array Average (GCRMA) procedures. For further information on Affymetrix oligonucleotide microarray preprocessing, see Preprocessing Affymetrix Microarray Data at the Probe Level.

You will use the t-test and false discovery rate to detect differentially expressed genes between two of the tumor types. Additionally, you will look at Gene Ontology terms related to the significantly up-regulated genes.

Loading the Expression Data

Load the MAT file cnsexpressiondata containing three DataMatrix objects. Gene expression values preprocessed by RMA and GCRMA (MLE and EB) procedures are stored in the DataMatrix objects expr_cns_rma, expr_cns_gcrma_mle, and expr_cns_gcrma_eb respectively.

```
load cnsexpressiondata
```

In each DataMatrix object, each row corresponds to a probe set on the HuGeneFl array, and each column corresponds to a sample. The row names are the probe set IDs and column names are the sample names. The DataMatrix object `expr_cns_gcrma_eb` will be used in this example. You can use either of the other two expression variables as well.

You can get the properties of the DataMatrix object `expr_cns_gcrma_eb` using the `get` command.

```
get(expr_cns_gcrma_eb)

            Name: ''
        RowNames: {7129x1 cell}
        ColNames: {1x42 cell}
           NRows: 7129
           NCols: 42
           NDims: 2
    ElementClass: 'single'
```

Determine the number of genes and number of samples in the DataMatrix object `expr_cns_gcrma_eb`.

```
[nGenes, nSamples] = size(expr_cns_gcrma_eb)


nGenes =

        7129


nSamples =

    42
```

You can use gene symbols instead of the probe set IDs to label the expression values. The gene symbols for the HuGeneFl array are provided in a MAT file containing a `Map` object.

```
load HuGeneFL_GeneSymbol_Map;
```

Create a cell array of gene symbols for the expression values from the `hu6800GeneSymbolMap` object.

```
huGenes = values(hu6800GeneSymbolMap, expr_cns_gcrma_eb.RowNames);
```

Set the row names of the `exprs_cns_gcrma_eb` to gene symbols using the `rownames` method of the DataMatrix object.

```
expr_cns_gcrma_eb = rownames(expr_cns_gcrma_eb, ':', huGenes);
```

Filtering the Expression Data

Remove gene expression data with empty gene symbols. In the example, the empty symbols are labeled as `'---'`.

```
expr_cns_gcrma_eb('---', :) = [];
```

Many of the genes in this study are not expressed, or have only small variability across the samples. Remove these genes using non-specific filtering.

Use `genelowvalfilter` to filter out genes with very low absolute expression values.

```
[mask, expr_cns_gcrma_eb] = genelowvalfilter(expr_cns_gcrma_eb);
```

Use `genevarfilter` to filter out genes with a small variance across samples.

```
[mask, expr_cns_gcrma_eb] = genevarfilter(expr_cns_gcrma_eb);
```

Determine the number of genes after filtering.

```
nGenes = expr_cns_gcrma_eb.NRows


nGenes =

        5669
```

Identifying Differential Gene Expression

You can now compare the gene expression values between two groups of data: CNS medulloblastomas (MD) and non-neuronal origin malignant gliomas (Mglio) tumor.

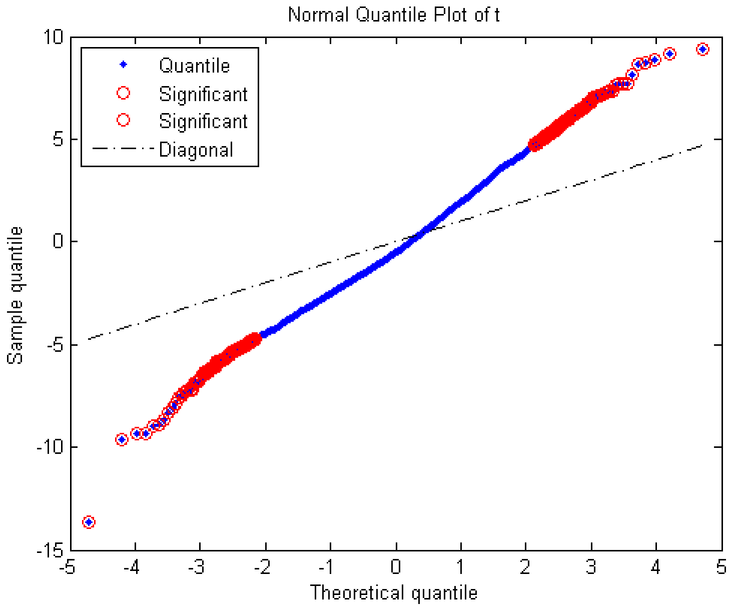
From the expression data of all 42 samples, extract the data of the 10 MD samples and the 10 Mglio samples.

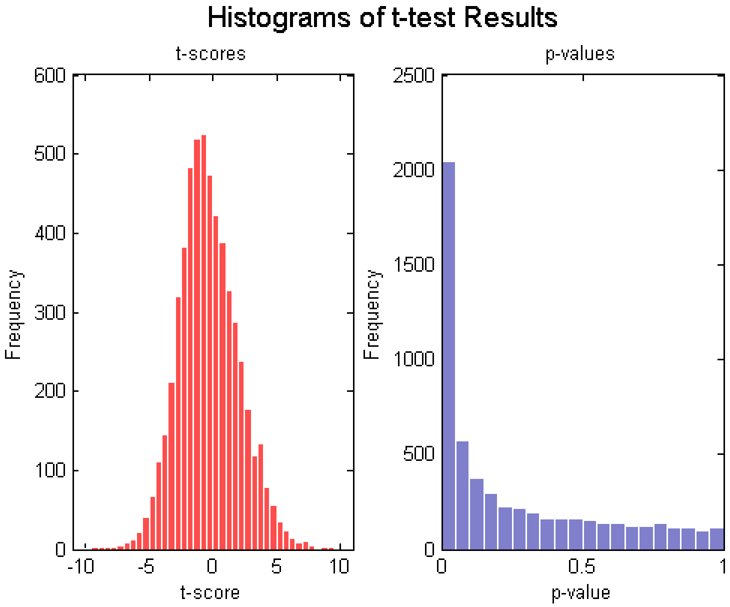
```
MDs = strncmp(expr_cns_gcrma_eb.ColNames,'Brain_MD', 8);
Mglios = strncmp(expr_cns_gcrma_eb.ColNames,'Brain_MGlio', 11);

MDData = expr_cns_gcrma_eb(:, MDs);
get(MDData)

            Name: ''
        RowNames: {5669x1 cell}
        ColNames: {1x10 cell}
           NRows: 5669
           NCols: 10
           NDims: 2
    ElementClass: 'single'


MglioData = expr_cns_gcrma_eb(:, Mglios);
get(MglioData)

            Name: ''
        RowNames: {5669x1 cell}
        ColNames: {1x10 cell}
           NRows: 5669
           NCols: 10
           NDims: 2
    ElementClass: 'single'
```

A standard statistical test for detecting significant changes between the measurement of a variable in two groups is the t-test. Conduct a t-test for each gene to identify significant changes in expression values between the

MD samples and Mglio samples. You can inspect the test results from the normal quantile plot of t-scores and the histograms of t-scores and *p-values* of the t-tests.

```
[pvalues, tscores] = mattest(MDData, MglioData,...
                              'Showhist', true', 'Showplot', true);
```





In any test situation, two types of errors can occur, a false positive by declaring that a gene is differentially expressed when it is not, and a false negative when the test fails to identify a truly differentially expressed gene. In multiple hypothesis testing, which simultaneously tests the null hypothesis of thousands of genes using microarray expression data, each test has a specific false positive rate, or a false discovery rate (FDR). False discovery rate is defined as the expected ratio of the number of false positives to the total number of positive calls in a differential expression analysis between two groups of samples (Storey et al., 2003).

In this example, you will compute the FDR using the Storey-Tibshirani procedure (Storey et al., 2003). The procedure also computes the q-value of a test, which measures the minimum FDR that occurs when calling the test significant. The estimation of FDR depends on the truly null distribution of the multiple tests, which is unknown. Permutation methods can be used to estimate the truly null distribution of the test statistics by permuting the columns of the gene expression data matrix (Storey et al., 2003, Dudoit et al., 2003). Depending on the sample size, it may not be feasible to consider all possible permutations. Usually a random subset of permutations are considered in the case of large sample size. Use the nchoosek function in Statistics Toolbox™ to find out the number of all possible permutations of the samples in this example.

```
all_possible_perms = nchoosek(1:MDData.NCols+MglioData.NCols, MDData.NCol
size(all_possible_perms, 1)


ans =

      184756
```

Perform a permutation t-test using mattest and the PERMUTE option to compute the *p-values* of *10,000* permutations by permuting the columns of the gene expression data matrix of MDData and MglioData (Dudoit et al., 2003).

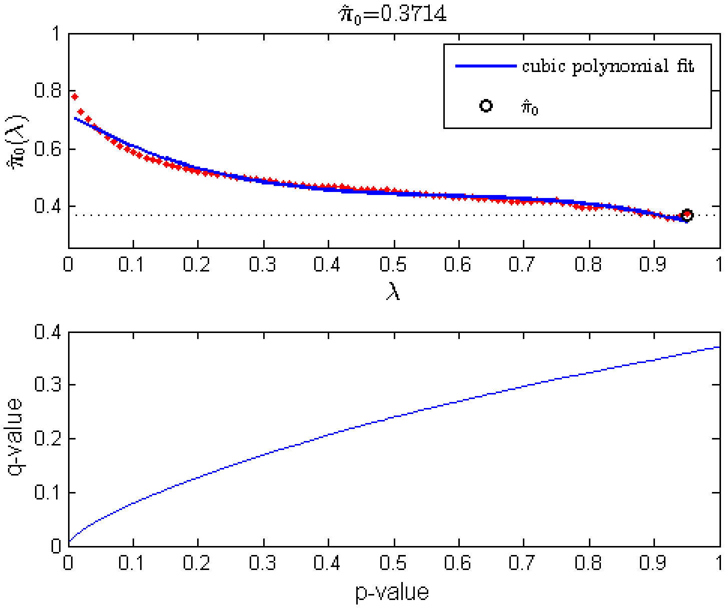
```
pvaluesCorr = mattest(MDData, MglioData, 'Permute', 10000);
```

Determine the number of genes considered to have statistical significance at the p-value cutoff of 0.05. Note: You may get a different number of genes due to the permutation test outcome.

```
cutoff = 0.05;
sum(pvaluesCorr < cutoff)


ans =

        2125
```

Estimate the FDR and q-values for each test using mafdr. The quantity *pi0* is the overall proportion of true null hypotheses in the study. It is estimated from the simulated null distribution via bootstrap or the cubic polynomial fit. Note: You can also manually set the value of lambda for estimating *pi0*.

```
figure;
[pFDR, qvalues] = mafdr(pvaluesCorr, 'showplot', true);
```



Determine the number of genes that have q-values less than the cutoff value. Note: You may get a different number of genes due to the permutation test and the bootstrap outcomes.

```
sum(qvalues < cutoff)


ans =

        2134
```

Many genes with low FDR implies that the two groups, MD and Mglio, are biologically distinct.

You can also empirically estimate the FDR adjusted *p-values* using the Benjamini-Hochberg (BH) procedure (Benjamini et al, 1995) by setting the `mafdr` input parameter `BHFDR` to true.

```
pvaluesBH = mafdr(pvaluesCorr, 'BHFDR', true);
sum(pvaluesBH < cutoff)


ans =

        1375
```

You can store the t-scores, *p-values*, pFDRs, q-values and BH FDR corrected *p-values* together as a DataMatrix object.

```
testResults = [tscores pvaluesCorr pFDR qvalues pvaluesBH];
```

Update the column name for BH FDR corrected *p-values* using the `colnames` method of DataMatrix object.

```
testResults = colnames(testResults, 5, {'FDR_BH'});
```

You can sort by *p-values* `pvaluesCorr` using the `sortrows` mathod.

```
testResults = sortrows(testResults, 2);
```

Display the first 23 genes in `testResults`. Note: Your results may be different from those shown below due to the permutation test and the bootstrap outcomes.

```
testResults(1:23, :)


ans =

                t-scores    p-values       FDR             q-values
    RAB31       -13.664     1.2256e-008    2.5806e-005      2.572e-005
    PLEC1       -9.6223     5.6687e-008    5.9681e-005      2.572e-005
    HNRPA1        9.359     7.5912e-008    5.3281e-005      2.572e-005
    FCGR2A      -9.3548     7.6023e-008    4.0019e-005      2.572e-005
    PLEC1       -9.3495     7.6167e-008    3.2076e-005      2.572e-005
    FBL          9.1518     1.1161e-007    3.9166e-005      2.572e-005
    KIAA0367     -8.996     1.2581e-007    3.7845e-005      2.572e-005
    ID2B        -8.9285      1.307e-007    3.4402e-005      2.572e-005
```

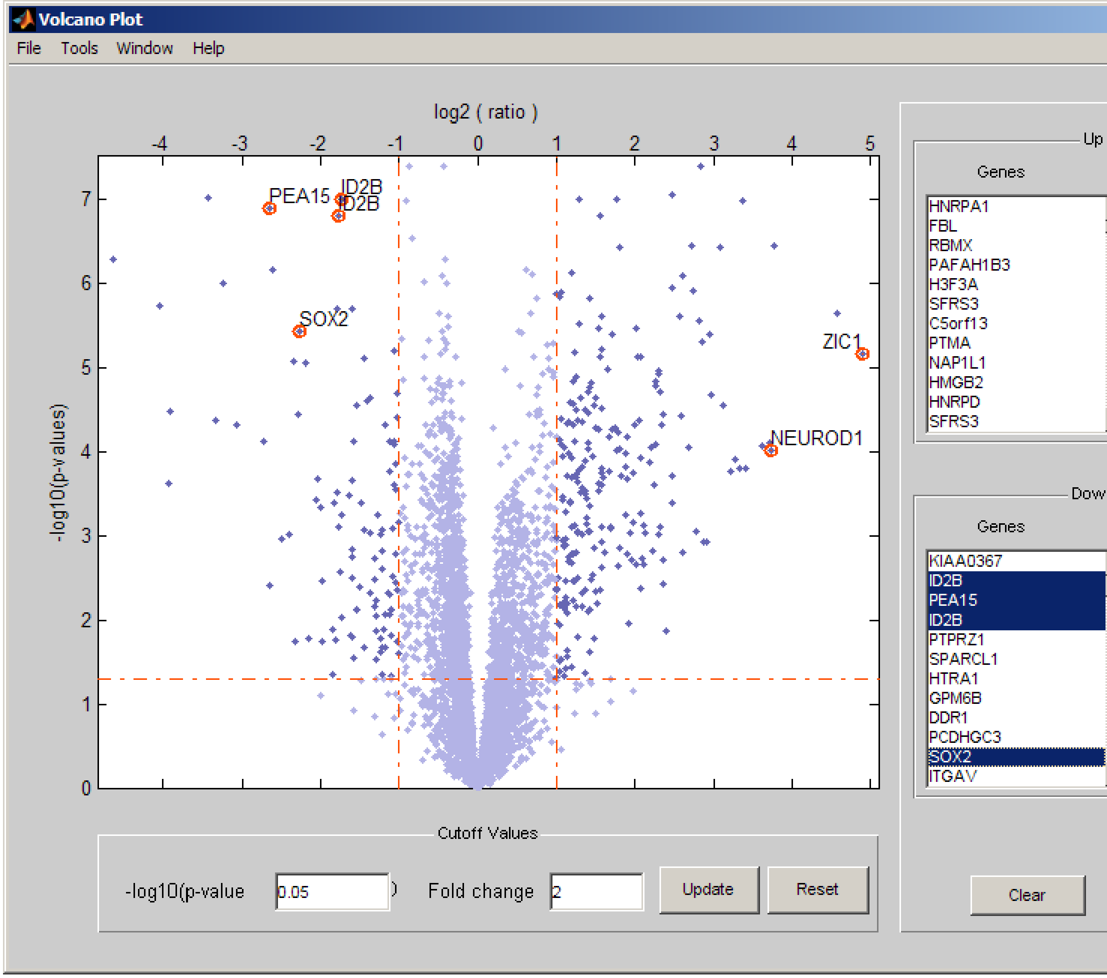
```
RBMX          8.8905     1.3305e-007    3.1128e-005      2.572e-005
PAFAH1B3      8.7561     1.4024e-007    2.9529e-005      2.572e-005
H3F3A         8.6512     1.4627e-007    2.7999e-005      2.572e-005
LRP1         -8.6465     1.4658e-007     2.572e-005      2.572e-005
PEA15        -8.3256     1.8957e-007    3.0704e-005     3.0704e-005
ID2B         -8.1183      2.281e-007    3.4307e-005     3.2051e-005
SFRS3         8.1166     2.2833e-007    3.2051e-005     3.2051e-005
HLA-DPA1     -7.8546     3.6101e-007    4.7509e-005     4.7509e-005
C5orf13       7.7195     4.8936e-007    6.0612e-005      5.827e-005
PTMA          7.7013     4.9812e-007     5.827e-005      5.827e-005
NAP1L1         7.674     5.3726e-007    5.9541e-005     5.8442e-005
HMGB2         7.6532      5.551e-007    5.8442e-005     5.8442e-005
ARAF         -7.5549     6.1454e-007    6.1618e-005      5.929e-005
PTPRZ1       -7.5352     6.1948e-007     5.929e-005      5.929e-005
SPARCL1      -7.3639     8.1963e-007    7.5036e-005     7.3354e-005


            FDR_BH
RAB31       6.9245e-005
PLEC1       6.9245e-005
HNRPA1      6.9245e-005
FCGR2A      6.9245e-005
PLEC1       6.9245e-005
FBL         6.9245e-005
KIAA0367    6.9245e-005
ID2B        6.9245e-005
RBMX        6.9245e-005
PAFAH1B3    6.9245e-005
H3F3A       6.9245e-005
LRP1        6.9245e-005
PEA15       8.2665e-005
ID2B        8.6292e-005
SFRS3       8.6292e-005
HLA-DPA1     0.00012791
C5orf13      0.00015688
PTMA         0.00015688
NAP1L1       0.00015734
HMGB2        0.00015734
ARAF         0.00015963
PTPRZ1       0.00015963
```

```
    SPARCL1      0.00019749
```

A gene is considered to be differentially expressed between the two groups of samples if it shows both statistical and biological significance. In this example, you will compare the gene expression ratio of MD over Mglio tumor samples. Therefore an up-regulated gene in this example has higher expression in MD and down-regulate gene has higher expression in Mglio.

Plot the –log10 of *p-values* against the biological effect in a volcano plot. Note: From the volcano plot UI, you can interactively change the p-value cutoff and fold change limit, and export differentially expressed genes.

```
diffStruct = mavolcanoplot(MDData, MglioData, pvaluesCorr)


diffStruct =

           Name: 'Differentially Expressed'
       PVCutoff: 0.0500
    FCThreshold: 2
     GeneLabels: {327x1 cell}
        PValues: [327x1 bioma.data.DataMatrix]
    FoldChanges: [327x1 bioma.data.DataMatrix]
```



-log10(p-value 0.05)   Fold change 2

Ctrl-click genes in the gene lists to label the genes in the plot. As seen in the volcano plot, genes specific for neuronal based cerebella granule cells, such as *ZIC* and *NEUROD*, are found in the up-regulated gene list, while genes typical of the astrocytic and oligodendrocytic lineage and cell differentiation, such as *SOX2*, *PEA15*, and *ID2B*, are found in the down-regulated list.

Determine the number of differentially expressed genes.

```
nDiffGenes = diffStruct.PValues.NRows


nDiffGenes =

   327
```

Get the list of up-regulated genes for MD compared to Mglio.

```
up_geneidx = find(diffStruct.FoldChanges > 0);
up_genes = rownames(diffStruct.FoldChanges, up_geneidx);
nUpGenes = length(up_geneidx)


nUpGenes =

   225
```

Determine the number of down-regulated genes for MD compared to Mglio.

```
nDownGenes = sum(diffStruct.FoldChanges < 0)


nDownGenes =

   102
```

Annotating Up-Regulated Genes Using Gene Ontology

Use Gene Ontology (GO) to annotate the differentially expressed genes. You can look at the up-regulated genes from the analysis above. Download the *Homo sapiens* annotations (`gene_association.goa_human.gz` file) from Gene Ontology Current Annotations, unzip, and store it in your the current directory.

Find the indices of the up-regulated genes for Gene Ontology analysis.

```
huGenes = rownames(expr_cns_gcrma_eb);
for i = 1:nUpGenes
    up_geneidx(i) = find(strncmpi(huGenes, up_genes{i}, length(up_genes{i
end
```

Load the Gene Ontology database into a MATLAB object using the `geneont` function.

```
GO = geneont('live',true);
```

Read the *Homo sapiens* gene annotation file. For this example, you will look only at genes that are related to molecular function, so you only need to read the information where the `Aspect` field is set to 'F'. The fields that are of interest are the gene symbol and associated ID. In GO Annotation files these have field names `DB_Object_Symbol` and `GOid` respectively.

```
HGann = goannotread('gene_association.goa_human',...
    'Aspect','F','Fields',{'DB_Object_Symbol','GOid'});
```

Create a map between annotated genes and GO terms.

```
HGmap = containers.Map();
for i=1:numel(HGann)
    key = HGann(i).DB_Object_Symbol;
    if isKey(HGmap,key)
        HGmap(key) = [HGmap(key) HGann(i).GOid];
    else
        HGmap(key) = HGann(i).GOid;
    end
end
```

Determine the number of *Homo sapiens* annotated genes related to molecular function.

```
HGmap.Count


ans =

                15561
```

Not all of the 5758 genes on the HuGeneFL chip are annotated. For every gene on the chip, see if it is annotated by comparing its gene symbol to the list of gene symbols from GO. Track the number of annotated genes and the number of up-regulated genes associated with each GO term. Note: You might get warnings about invalid or obsolete IDs due to the frequent update to the *Homo sapiens* gene annotation file.

```
m = GO.Terms(end).id;        % gets the last term id
chipgenesCount = zeros(m,1); % a vector of GO term counts for the entire
upgenesCount  = zeros(m,1);  % a vector of GO term counts for up-regulate
for i = 1:length(huGenes)
    if isKey(HGmap,huGenes{i})
        goid = getrelatives(GO,HGmap(huGenes{i}));
        % Update the tally
        chipgenesCount(goid) = chipgenesCount(goid) + 1;
        if (any(i == up_geneidx))
            upgenesCount(goid) = upgenesCount(goid) +1;
        end
    end
end
```

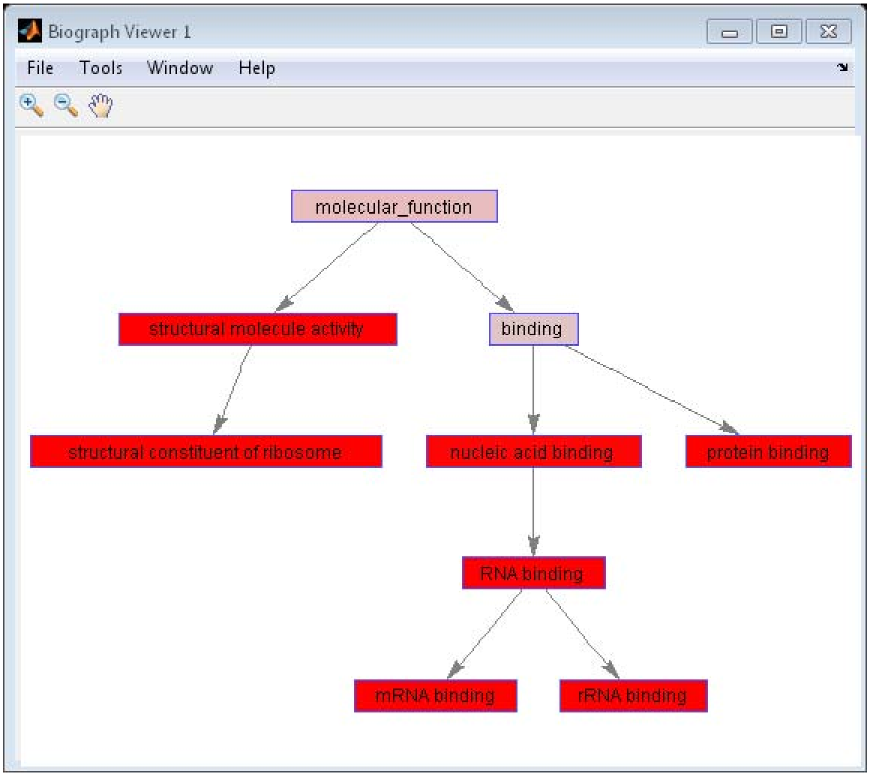
Determine the statistically significant GO terms using the hypergeometric probability distribution. For each GO term, a p-value is calculated representing the probability that the number of annotated genes associated with it could have been found by chance.

```
gopvalues = hygepdf(upgenesCount,max(chipgenesCount),...
                        max(upgenesCount),chipgenesCount);
[dummy, idx] = sort(gopvalues);

report = sprintf('GO Term     p-value     counts      definition\n');
for i = 1:10
    term = idx(i);
    report = sprintf('%s%s\t%-1.5f\t%3d / %3d\t%s...\n',...
             report, char(num2goid(term)), gopvalues(term),...
             upgenesCount(term), chipgenesCount(term),...
            GO(term).Term.definition(2:min(50,end)));
end
disp(report);
```

```
GO Term     p-value     counts      definition
GO:0005515 0.00000 119 / 2984 Interacting selectively with any protein or
GO:0003735 0.00000  56 / 151 The action of a molecule that contributes to
GO:0019843 0.00000  52 / 214 Interacting selectively with ribosomal RNA."
GO:0003723 0.00000  59 / 299 Interacting selectively with an RNA molecule
GO:0003729 0.00000  50 / 225 Interacting selectively with pre-messenger R
GO:0008312 0.00000  48 / 208 Interacting selectively with 7S RNA, the RNA
GO:0000049 0.00000  47 / 211 Interacting selectively with transfer RNA."
GO:0000498 0.00000  46 / 204 Interacting selectively with ribonucleic aci
GO:0030515 0.00000  46 / 204 Interacting selectively with small nucleolar
GO:0033204 0.00000  46 / 204 Interacting selectively with the RNA subunit
```

Inspect the significant GO terms and select the terms related to specific molecule functions to build a sub-ontology that includes the ancestors of the terms. Visualize this ontology using the biograph function. You can also color the graphs nodes. In this example, the red nodes are the most significant, while the blue nodes are the least significant gene ontology terms. Note: The GO terms returned may differ from those shown due to the frequent update to the *Homo sapiens* gene annotation file.

```
fcnAncestors = GO(getancestors(GO,idx(1:5)))
[cm acc rels] = getmatrix(fcnAncestors);
BG = biograph(cm,get(fcnAncestors.Terms,'name'))

for i=1:numel(acc)
    pval = gopvalues(acc(i));
    color = [(1-pval).^(1) pval.^(1/8) pval.^(1/8)];
    set(BG.Nodes(i),'Color',color);
end
view(BG)

Gene Ontology object with 9 Terms.
Biograph object with 9 nodes and 8 edges.
```



Finding the Differentially Expressed Genes in Pathways

You can query the pathway information of the differentially expressed genes from the KEGG pathway database through KEGG's SOAP Web Service (For more information, see Connecting to the KEGG API Web Service), or by simply passing the list of gene symbols to KEGG's Web query tool.

Following are a few pathway maps with the genes in the up-regulated gene list highlighted:

Cell Cycle

Cell Communication

Hedgehog Signaling pathway

mTor Signaling pathway

References

[1] Pomeroy, S.L., Tamayo, P., Gaasenbeek, M., Sturla, L.M., Angelo, M., McLaughlin, M.E., Kim, J.Y., Goumnerova, L.C., Black, P.M., Lau, C., Allen, J.C., Zagzag, D., Olson, J.M., Curran, T., Wetmore, C., Biegel, J.A., Poggio, T., Mukherjee, S., Rifkin, R., Califano, A., Stolovitzky, G., Louis, DN, Mesirov, J.P., Lander, E.S., and Golub, T.R. (2002). Prediction of central nervous system embryonal tumour outcome based on gene expression. Nature, 415(6870), 436-442.

[2] Storey, J.D., and Tibshirani, R. (2003). Statistical significance for genomewide studies. Proc.Nat.Acad.Sci., 100(16), 9440-9445.

[3] Dudoit, S., Shaffer, J.P., and Boldrick, J.C. (2003). Multiple hypothesis testing in microarray experiment. Statistical Science, 18, 71-103.

[4] Benjamini, Y., and Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. J. Royal Stat. Soc., B 57, 289-300.

Suggest an enhancement for this example.

Phylogenetic Analysis

Overview of Phylogenetic Analysis

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

Example: Building a Phylogenetic Tree

In this section...

Note For information on creating a phylogenetic tree with multiply aligned sequences, see the `phytree` function.

Overview of the Primate Example

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

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

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

Why Use Mitochondrial DNA Sequences for Phylogenetic Study?

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

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

Neanderthal DNA

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

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

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

References

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

Sajantila A., et al. (1995). Genes and languages in Europe: an analysis of mitochondrial lineages. Genome Research *5 (1)*, 42–52.

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

Jensen-Seaman, M., Kidd K. (2001). Mitochondrial DNA variation and biogeography of eastern gorillas. Molecular Ecology *10(9)*, 2241–2247.

Searching NCBI for Phylogenetic Data

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

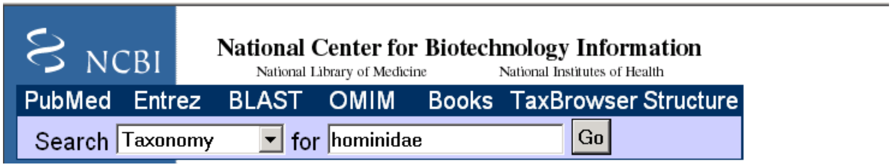
This procedure uses the family Hominidae (orangutans, chimpanzees, gorillas, and humans) as a taxonomy example for searching the NCBI Web site and locating mitochondrial D-loop sequences.

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

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

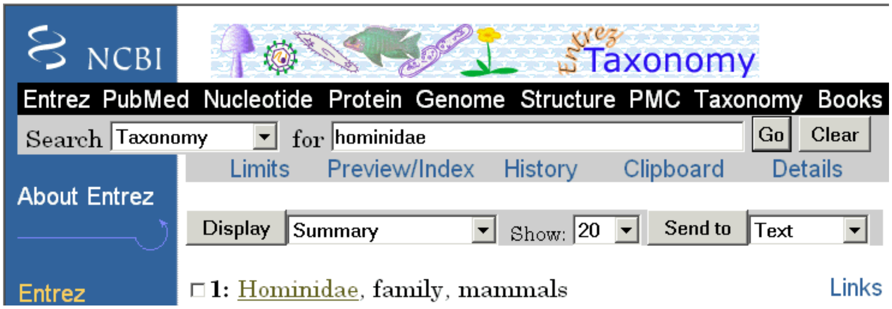
A separate browser window opens with the home page for the NCBI Web site.

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


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

Search Taxonomy for hominidae

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

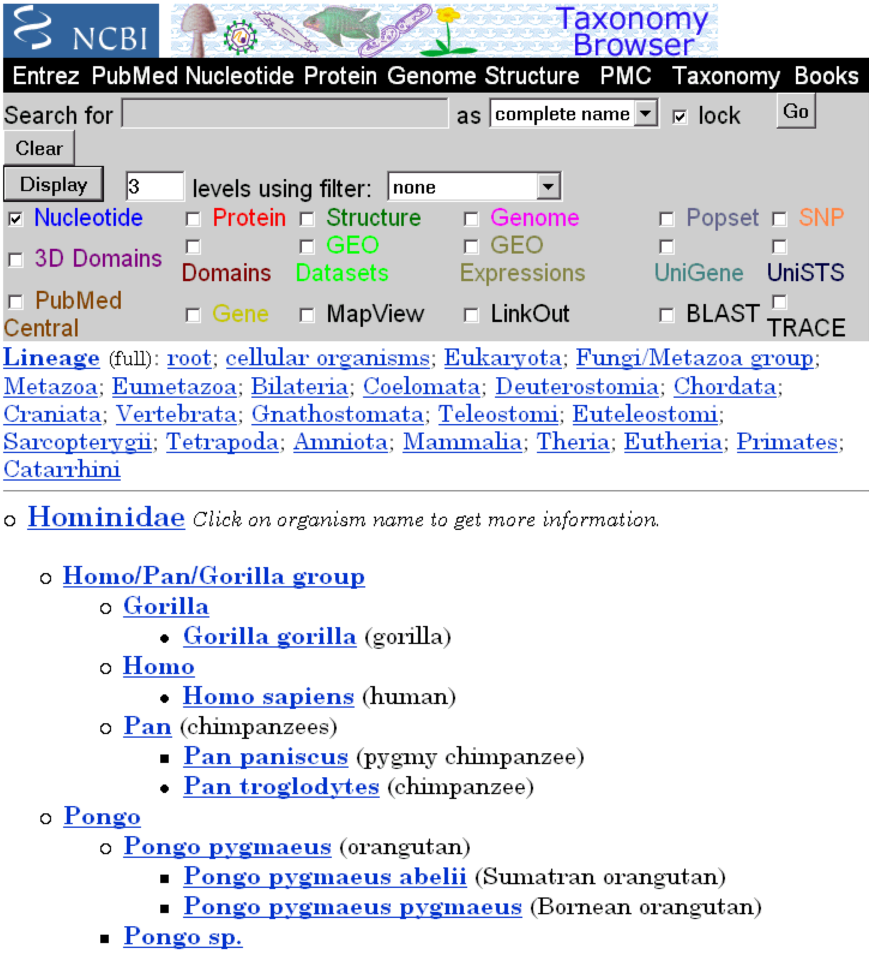


Search Taxonomy for hominidae

Display Summary Show: 20 Send to Text

□ 1: Hominidae, family, mammals

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


NCBI
Taxonomy Browser
Entrez PubMed Nucleotide Protein Genome Structure PMC Taxonomy Books

Search for [ ] as complete name ☑ lock Go

Clear

Display 3 levels using filter: none

☑ Nucleotide □ Protein □ Structure □ Genome □ Popset □ SNP
□ 3D Domains □ Domains □ GEO Datasets □ GEO Expressions □ UniGene □ UniSTS
□ PubMed Central □ Gene □ MapView □ LinkOut □ BLAST □ TRACE

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

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

Creating a Phylogenetic Tree for Five Species

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

After locating the GenBank accession codes for the sequences you are interested in studying, you can create a phylogenetic tree with the data. For information on locating accession codes, see “Searching NCBI for Phylogenetic Data” on page 5-5.

In the following example, you will use the Jukes-Cantor method to calculate distances between sequences, and the Unweighted Pair Group Method Average (UPGMA) method for linking the tree nodes.

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

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

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

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

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

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

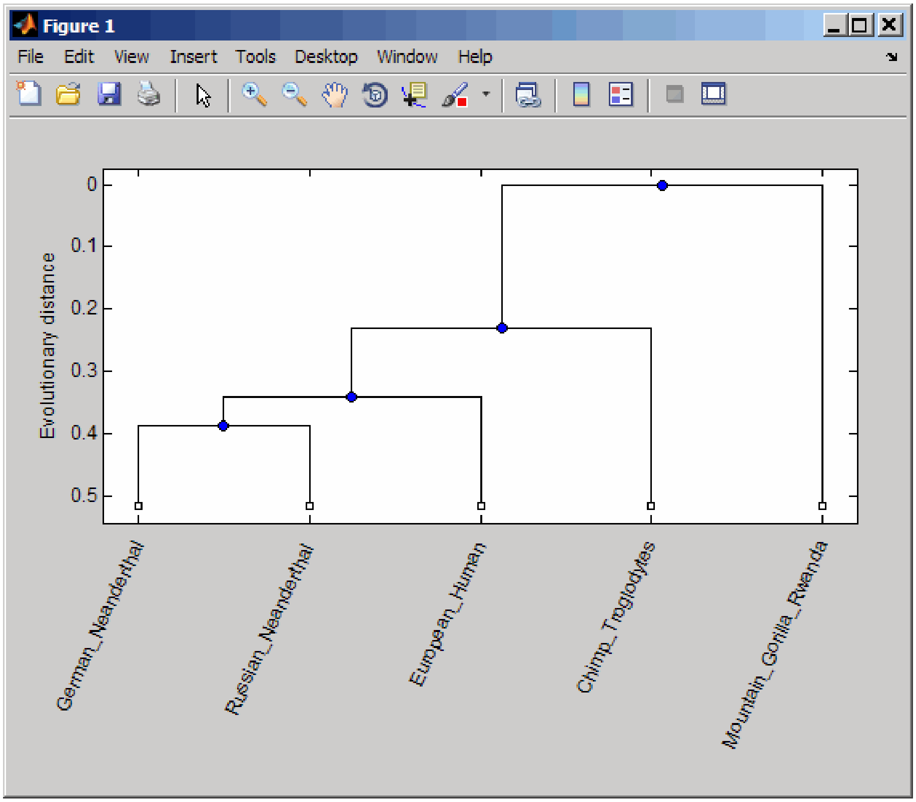
The MATLAB software displays information about the phytree object. The function `seqpdist` calculates the pairwise distances between pairs of sequences while the function `seqlinkage` uses the distances to build a hierarchical cluster tree. First, the most similar sequences are grouped together, and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

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

4 Draw a phylogenetic tree.

```
h = plot(tree,'orient','top');
ylabel('Evolutionary distance')
set(h.terminalNodeLabels,'Rotation',65)
```

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



Creating a Phylogenetic Tree for Twelve Species

Plotting a simple phylogenetic tree for five species seems to indicate a number of monophyletic groups (see "Creating a Phylogenetic Tree for Five Species" on page 5-6). After a preliminary analysis with five species, you can add more species to your phylogenetic tree. Adding more species to the data set will help you to confirm the observed monophyletic groups are valid.

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

```
data2 = {'Puti_Orangutan'          'AF451972';
         'Jari_Orangutan'          'AF451964';
         'Western_Lowland_Gorilla' 'AY079510';
         'Eastern_Lowland_Gorilla' 'AF050738';
```

```
          'Chimp_Schweinfurthii'    'AF176722';
          'Chimp_Vellerosus'        'AF315498';
          'Chimp_Verus'             'AF176731';
        };
```

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

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

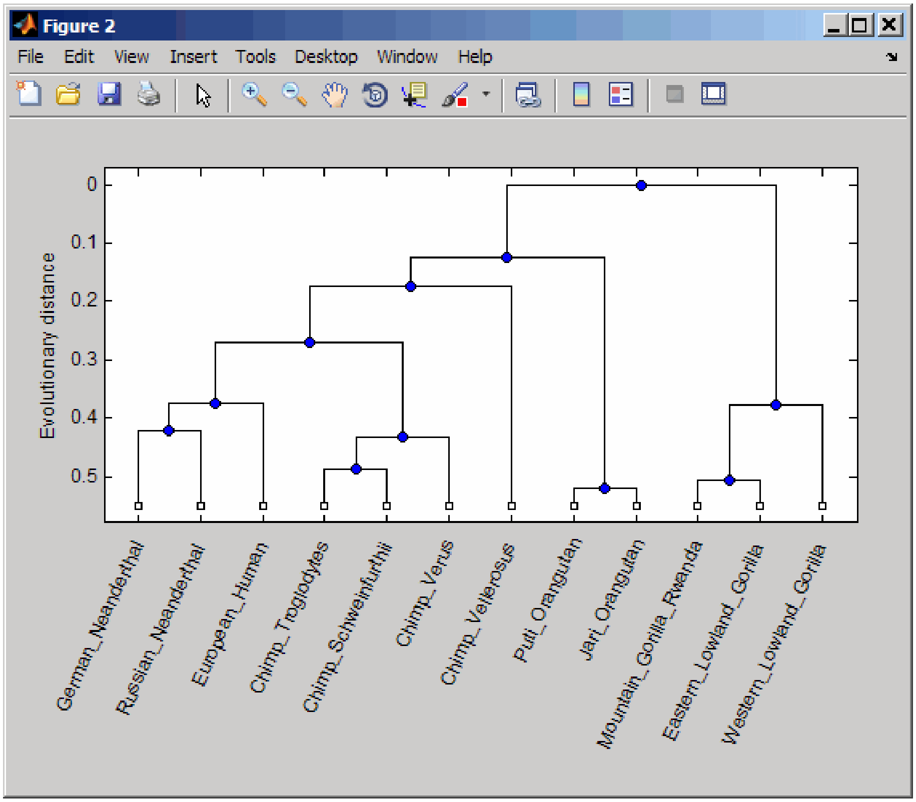
3 Calculate pairwise distances and the hierarchical linkage.

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

4 Draw a phylogenetic tree.

```
h = plot(tree,'orient','top');
ylabel('Evolutionary distance')
set(h.terminalNodeLabels,'Rotation',65)
```

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



Exploring the Phylogenetic Tree

After you create a phylogenetic tree, you can explore the tree using the MATLAB command line or the `phytreetool` GUI. This procedure uses the tree created in "Creating a Phylogenetic Tree for Twelve Species" on page 5-9 as an example.

1 List the members of a tree.

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

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

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

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

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

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

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

3 List the names of the closest species.

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

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

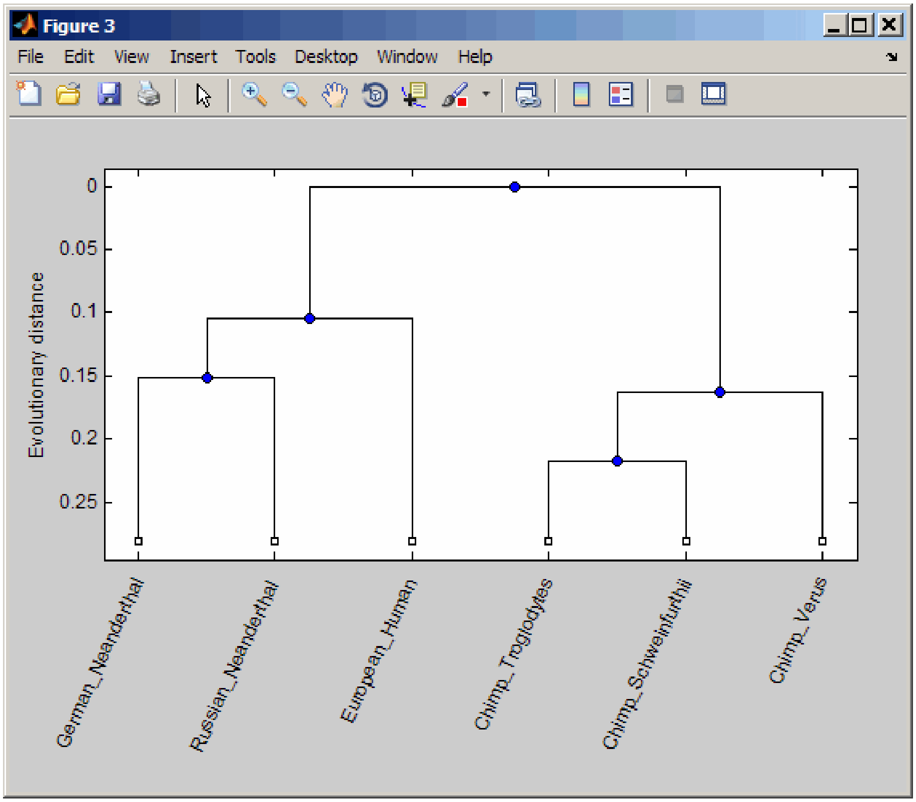
```
   subtree_names =

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

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

```
leaves_to_prune = ~h_leaves;
pruned_tree = prune(tree,leaves_to_prune)
h = plot(pruned_tree,'orient','top');
ylabel('Evolutionary distance')
set(h.terminalNodeLabels,'Rotation',65)
```

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

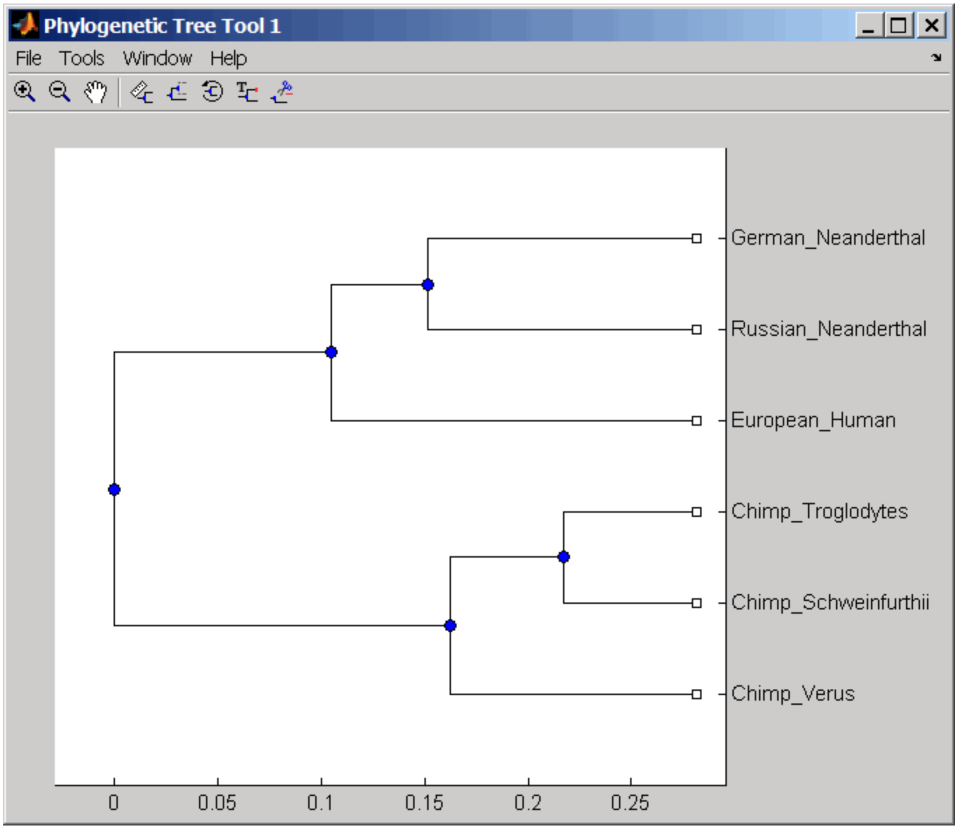
```
Phylogenetic tree object with 6 leaves (5 branches)
```



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

```
phytreetool(pruned_tree)
```

The Phylogenetic Tree Tool window opens and draws the tree.



You can interactively change the appearance of the tree within the tool window. For information on using this GUI, see "Phylogenetic Tree Tool Reference" on page 5-16.

Phylogenetic Tree Tool Reference

In this section...

Overview of the Phylogenetic Tree Tool

The Phylogenetic Tree Tool is an interactive graphical user interface (GUI) that allows you to view, edit, format, and explore phylogenetic tree data. With this GUI you can prune, reorder, rename branches, and explore distances. You can also open or save Newick-formatted files. The following sections give a description of menu commands and features for creating publishable tree figures.

Opening the Phylogenetic Tree Tool

This section illustrates how to draw a phylogenetic tree from data in a phytree object or a previously saved file.

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

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

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

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

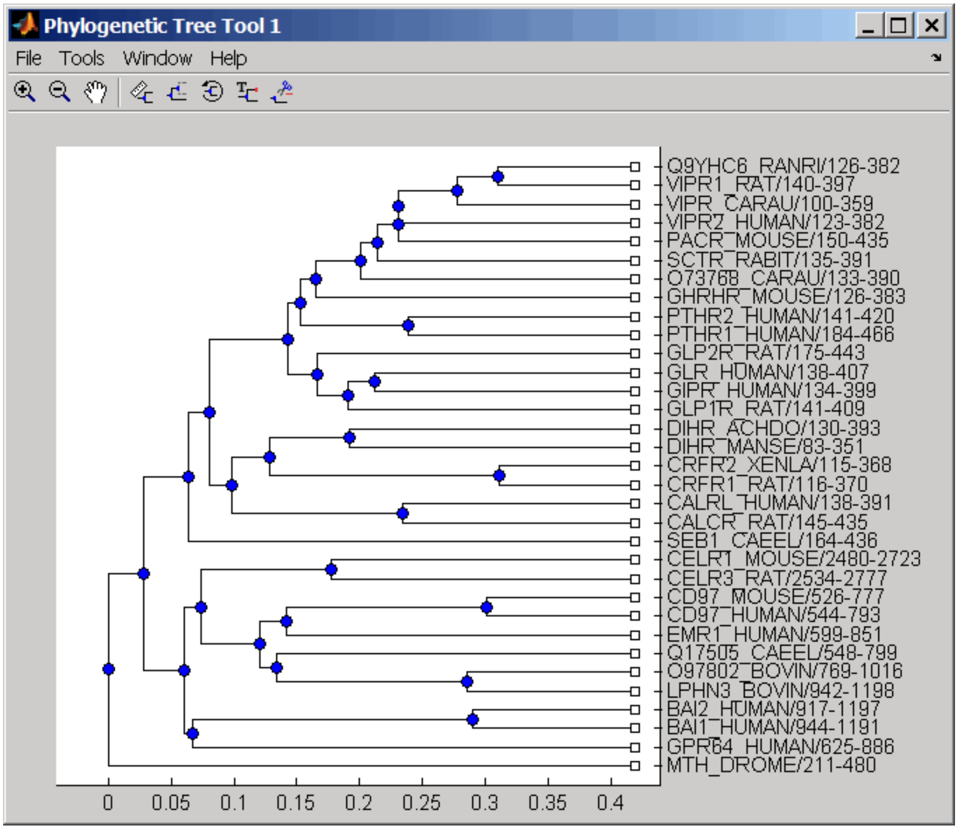
The MATLAB software creates a phytree object.

```
Phylogenetic tree object with 33 leaves (32 branches)
```

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

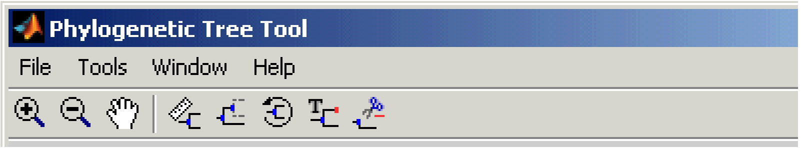
```
phytreetool(tr)
```

The Phylogenetic Tree Tool window opens.



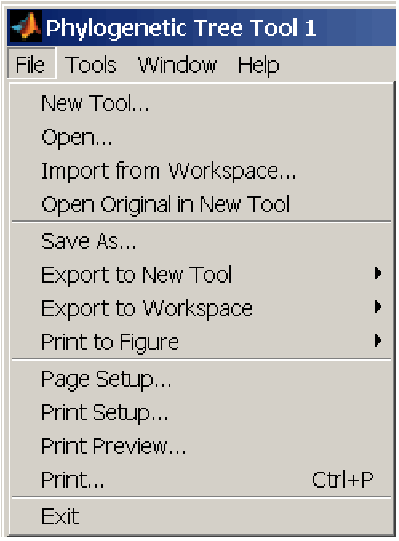
Alternately, if you do not give the `phytreetool` function an argument, the Select Phylogenetic Tree dialog box opens. Select a Newick-formatted file and then click **Open**.

3 Select a command from the menu or toolbar.



File Menu

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

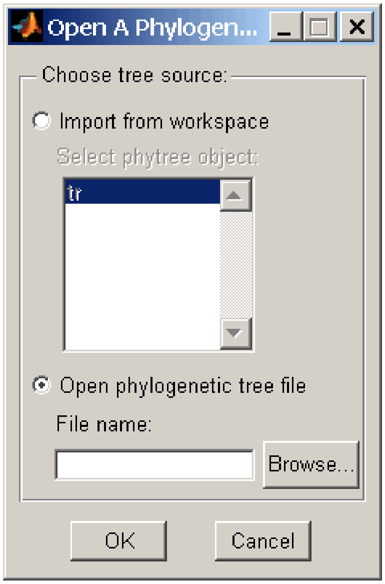


New Tool Command

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

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

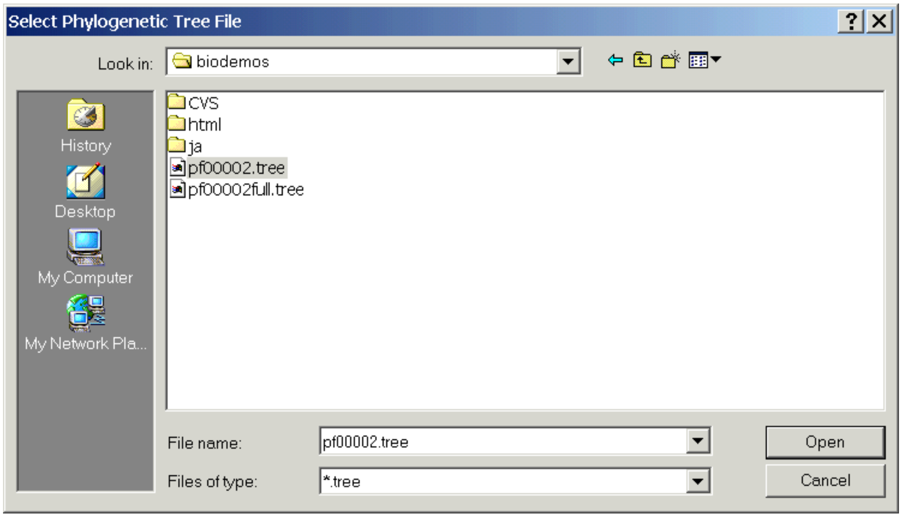
The Open A Phylogenetic Tree dialog box opens.



2 Choose the source for a tree.

- MATLAB Workspace — Select the **Import from Workspace** options, and then select a phytree object from the list.

- File — Select the **Open phylogenetic tree file** option, click the **Browse** button, select a directory, select a file with the extension `.tree`, and then click **Open**. The toolbox uses the file extension `.tree` for Newick-formatted files, but you can use any Newick-formatted file with any extension.



Select Phylogenetic Tree File

Look in: biodemos

CVS
html
ja
pf00002.tree
pf00002full.tree

File name: pf00002.tree

Files of type: *.tree

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

Open Command

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

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

   The Select Phylogenetic Tree File dialog box opens.

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

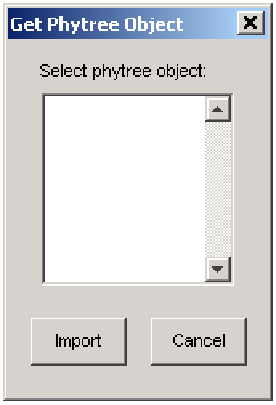
   The MATLAB software replaces the current tree data with data from the selected file.

Import from Workspace Command

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

1 From the **File** menu, select **Import from Workspace**.

   The Get Phytree Object dialog box opens.



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

3 Click the **Import** button.

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

Open Original in New Tool

There may be times when you make changes that you would like to undo. `Phytreetool` does not have an undo command, but you can get back to the original tree you started viewing with the **Open Original in New Tool** command.

From the **File** menu, select **Open Original in New Tool**.

A new Phylogenetic Tree Tool window opens with the original tree.

Save As Command

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

1 From the **File** menu, select **Save As**.

The Save Phylogenetic tree as dialog box opens.

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

3 Click **Save**.

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

Export to New Tool Command

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

1 Select **File > Export to New Tool**, and then select either **With Hidden Nodes** or **Only Displayed**.

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

2 Use the new figure to continue your analysis.

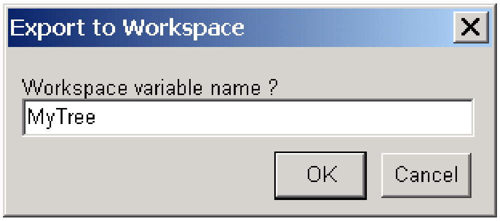
Export to Workspace Command

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

1 Select **File > Export to Workspace**, and then select either **With Hidden Nodes** or **Only Displayed**.

The Export to Workspace dialog box opens.

2 In the **Workspace variable name** box, enter the name for your phylogenetic tree data. For example, enter `MyTree`.



Export to Workspace

Workspace variable name ?

MyTree

3 Click **OK**.

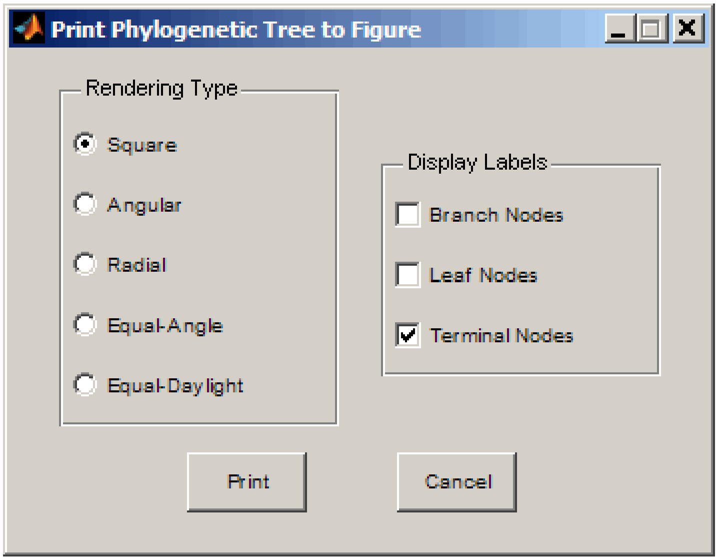
The `phytreetool` creates a phytree object in the MATLAB Workspace.

Print to Figure Command

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

1 From the **File** menu, select **Print to Figure**, and then select either **With Hidden Nodes** or **Only Displayed**.

The Publish Phylogenetic Tree to Figure dialog box opens.



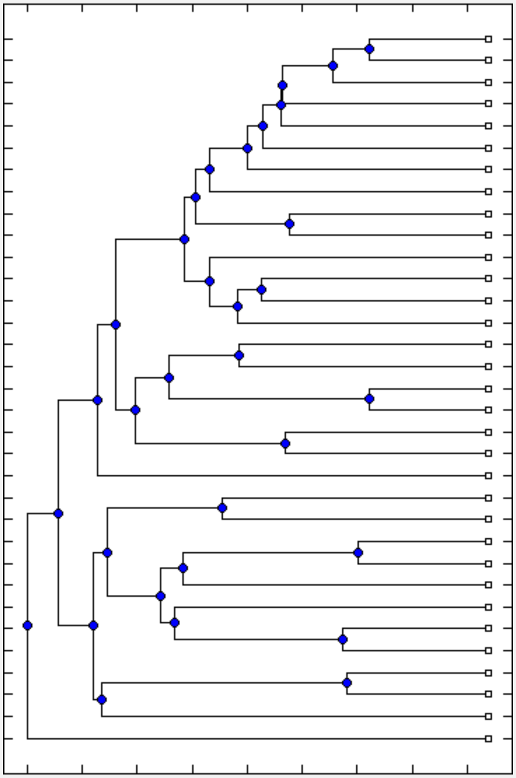
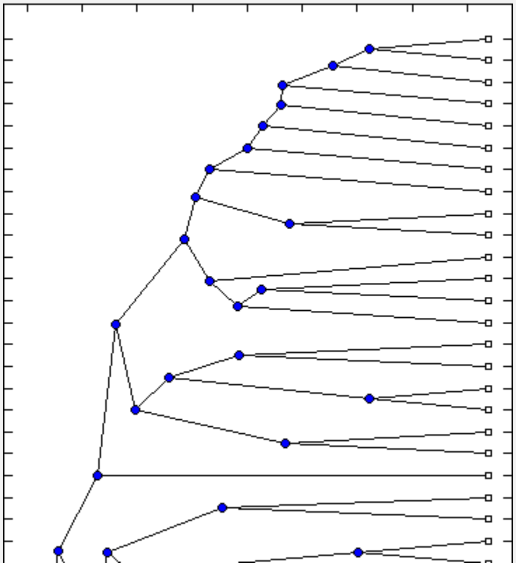
Rendering Type

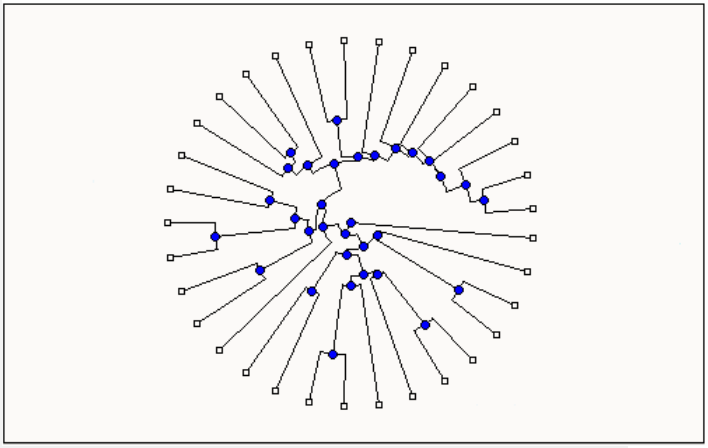
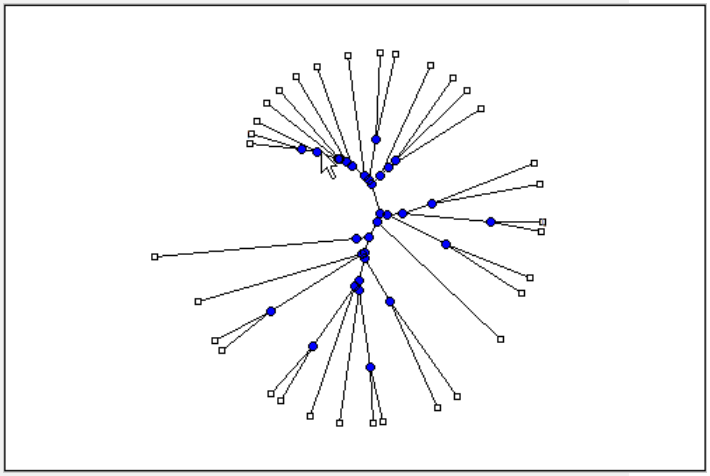
● Square
○ Angular
○ Radial
○ Equal-Angle
○ Equal-Daylight

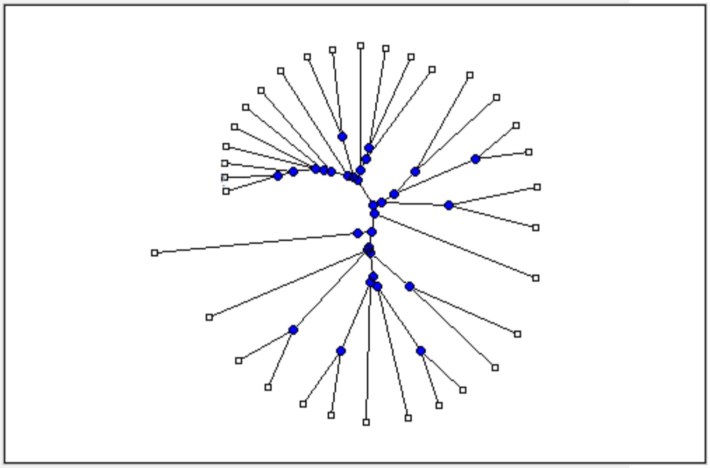
Display Labels

□ Branch Nodes
□ Leaf Nodes
☑ Terminal Nodes

2 Select one of the **Rendering Types**.

Rendering Type	Description
'square' (default)	
'angular'	

Rendering Type	Description
'radial'	
'equalangle'	

Rendering Type	Description
	Tip This rendering type hides the significance of the root node and emphasizes clusters, thereby making it useful for visually assessing clusters and detecting outliers.
`'equaldaylight'`	 **Tip** This rendering type hides the significance of the root node and emphasizes clusters, thereby making it useful for visually assessing clusters and detecting outliers.

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

- **Branch Nodes** — Display branch node names on the figure.
- **Leaf Nodes** — Display leaf node names on the figure.
- **Terminal Nodes** — Display terminal node names on the right border.

4 Click the **Print** button.

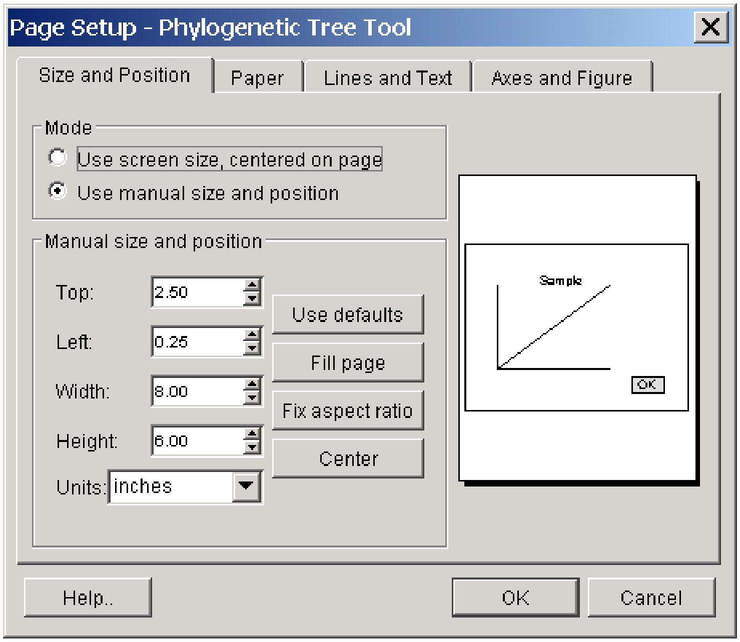
A new Figure window opens with the characteristics you selected.

Page Setup Command

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

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

The Page Setup - Phylogenetic Tree Tool dialog box opens, which you can use to select page formatting options.



Mode

○ Use screen size, centered on page

● Use manual size and position

Manual size and position

Top: 2.50

Left: 0.25

Width: 8.00

Height: 6.00

Units: inches

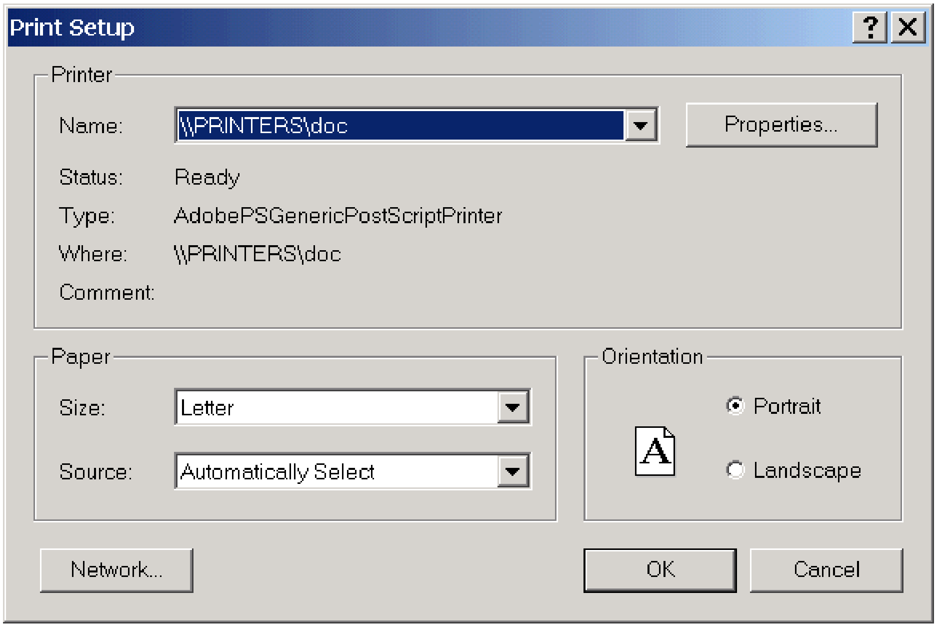
2 Select the page formatting options and values you want, and then click **OK**.

Print Setup Command

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

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

The Print Setup dialog box opens.



Name: \\PRINTERS\doc

Status: Ready

Type: AdobePSGenericPostScriptPrinter

Where: \\PRINTERS\doc

Comment:

Size: Letter

Source: Automatically Select

● Portrait

○ Landscape

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

Print Preview Command

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

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

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

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

Print Command

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

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

The Print dialog box opens.

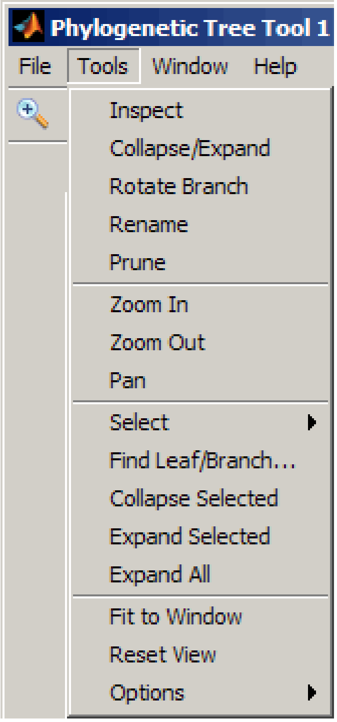
2 From the **Name** list, select a printer, and then click **OK**.

Tools Menu

Use the **Tools** menu to:

- Explore branch paths
- Rotate branches
- Find, rename, hide, and prune branches and leaves.

The **Tools** menu and toolbar contain most of the commands specific to trees and phylogenetic analysis. Use these commands and modes to edit and format your tree interactively. The **Tools** menu commands are:



Inspect Mode

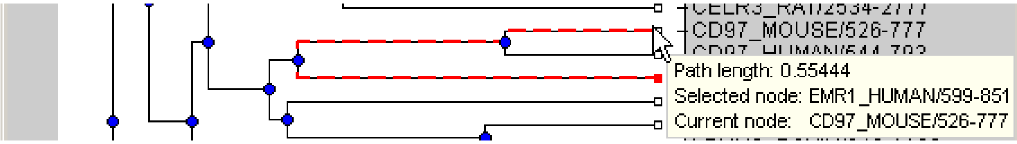
Viewing a phylogenetic tree in the Phylogenetic Tree Tool window provides a rough idea of how closely related two sequences are. However, to see exactly how closely related two sequences are, measure the distance of the path between them. Use the **Inspect** command to display and measure the path between two sequences.

1 Select **Tools > Inspect**, or from the toolbar, click the Inspect Tool Mode icon .

   The Phylogenetic Tree Tool is set to inspect mode.

2 Click a branch or leaf node (selected node), and then hover your cursor over another branch or leaf node (current node).

The tool highlights the path between the two nodes and displays the path length in the pop-up window. The path length is the patristic distance calculated by the `seqpdist` function.



Path length: 0.55444
Selected node: EMR1_HUMAN/599-851
Current node: CD97_MOUSE/526-777

Collapse and Expand Branch Mode

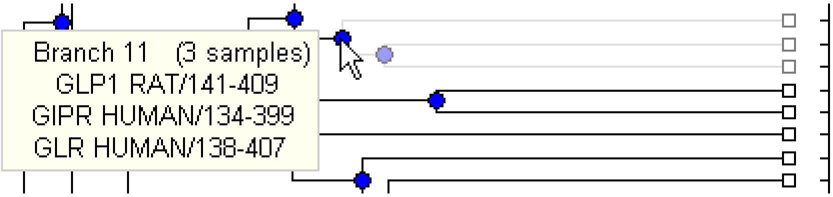
Some trees have thousands of leaf and branch nodes. Displaying all the nodes can create an unreadable tree diagram. By collapsing some branches, you can better see the relationships between the remaining nodes.

1 Select **Tools > Collapse/Expand**, or from the toolbar, click the Collapse/Expand Brand Mode icon .

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

2 Point to a branch.

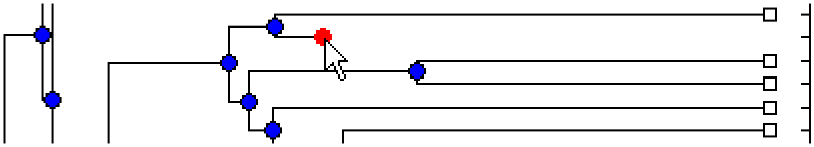
The paths, branch nodes, and leaf nodes below the selected branch appear in gray, indicating you selected them to collapse (hide from view).



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

3 Click the branch node.

The tool hides the display of paths, branch nodes, and leaf nodes below the selected branch. However, it does not remove the data.



4 To expand a collapsed branch, click it or select **Tools > Reset View**.

Tip After collapsing nodes, you can redraw the tree by selecting **Tools > Fit to Window**.

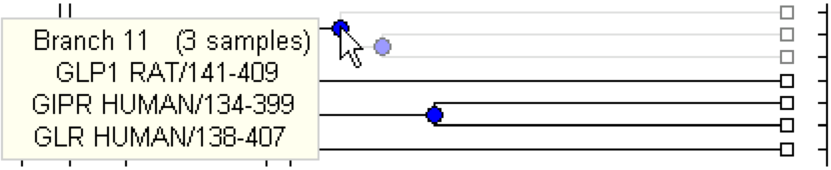
Rotate Branch Mode

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

1 Select **Tools > Rotate Branch**, or from the toolbar, click the Rotate Branch Mode icon .

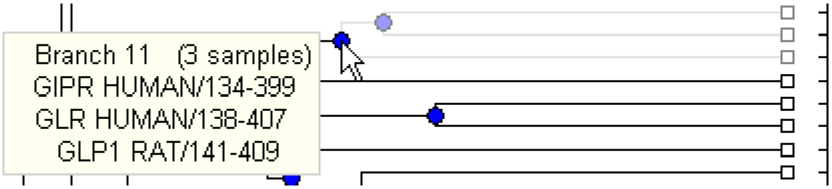
The Phylogenetic Tree Tool is set to rotate branch mode.

2 Point to a branch node.



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

3 Click the branch node.



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

The branch and leaf nodes below the selected branch node rotate 180 degrees around the branch node.

4 To undo the rotation, simply click the branch node again.

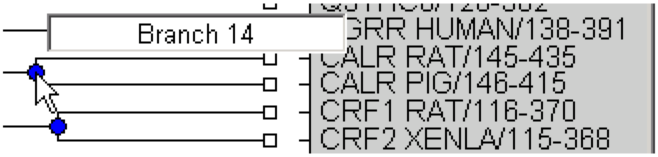
Rename Leaf or Branch Mode

The Phylogenetic Tree Tool takes the node names from the phytree object and creates numbered branch names starting with `Branch 1`. You can edit any of the leaf or branch names.

1 Select **Tools > Rename**, or from the toolbar, click the Rename Leaf/Branch Mode icon .

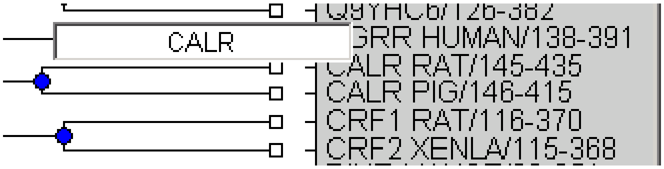
The Phylogenetic Tree Tool is set to rename mode.

2 Click a branch or leaf node.



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

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



4 To accept your changes and close the text box, click outside of the text box. To save your changes, select **File > Save As**.

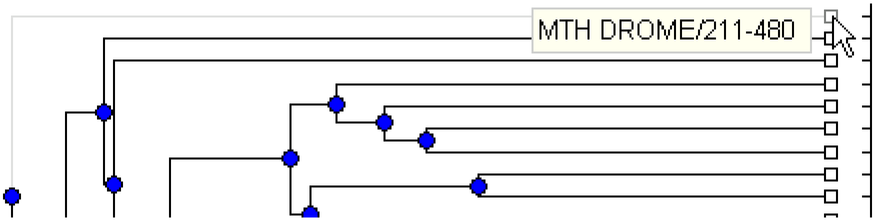
Prune (Delete) Leaf or Branch Mode

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

1 Select **Tools > Prune**, or from the toolbar, click the Prune (delete) Leaf/Branch Mode icon .

The Phylogenetic Tree Tool is set to prune mode.

2 Point to a branch or leaf node.



For a leaf node, the branch line connected to the leaf appears in gray. For a branch node, the branch lines below the node appear in gray.

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

---

3 Click the branch or leaf node.

The tool removes the branch from the figure and rearranges the other nodes to balance the tree structure. It does not recalculate the phylogeny.

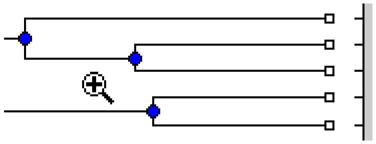
Tip After pruning nodes, you can redraw the tree by selecting **Tools > Fit to Window**.

Zoom In, Zoom Out, and Pan Commands

The Zoom and Pan commands are the standard controls for resizing and moving the screen in any MATLAB Figure window.

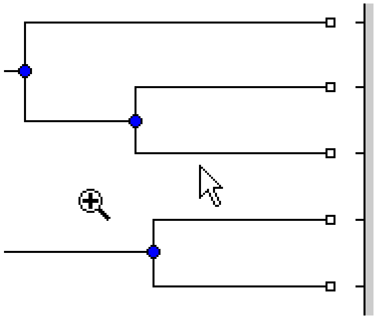
1 Select **Tools > Zoom In**, or from the toolbar, click the Zoom In icon .

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



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

The tree diagram doubles its size.



3 From the toolbar click the Pan icon ✋.

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

Tip After zooming and panning, you can reset the tree to its original view, by selecting **Tools > Reset View**.

Select Submenu

Select a single branch or leaf node by clicking it. Select multiple branch or leaf nodes by **Shift**-clicking the nodes, or click-dragging to draw a box around nodes.

Use the **Select** submenu to select specific branch and leaf nodes based on different criteria.

- **Select By Distance** — Displays a slider bar at the top of the window, which you slide to specify a distance threshold. Nodes whose distance from the selected node are below this threshold appear in red. Nodes whose distance from the selected node are above this threshold appear in blue.
- **Select Common Ancestor** — For all selected nodes, highlights the closest common ancestor branch node in red.
- **Select Leaves** — If one or more nodes are selected, highlights the nodes that are leaf nodes in red. If no nodes are selected, highlights all leaf nodes in red
- **Propogate Selection** — For all selected nodes, highlights the descendant nodes in red.
- **Swap Selection** — Clears all selected nodes and selects all deselected nodes.

After selecting nodes using one of the previous commands, hide and show the nodes using the following commands:

- **Collapse Selected**
- **Expand Selected**
- **Expand All**

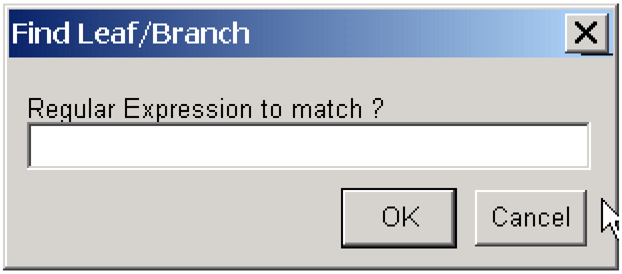
Clear all selected nodes by clicking anywhere else in the Phylogenetic Tree Tool window.

Find Leaf or Branch Command

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

1. Select **Tools > Find Leaf/Branch**.

   The Find Leaf/Branch dialog box opens.



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

3 Click **OK**.

The branch or leaf nodes that match the expression appear in red.

After selecting nodes using the **Find Leaf/Branch** command, you can hide and show the nodes using the following commands:

- **Collapse Selected**
- **Expand Selected**
- **Expand All**

Collapse Selected, Expand Selected, and Expand All Commands

When you select nodes, either manually or using the previous commands, you can then collapse them by selecting **Tools > Collapse Selected**.

The data for branches and leaves that you hide using the **Collapse/Expand** or **Collapse Selected** command are not removed from the tree. You can display selected or all hidden data using the **Expand Selected** or **Expand All** command.

Fit to Window Command

After you hide nodes with the collapse commands, or delete nodes with the **Prune** command, there can be extra space in the tree diagram. Use the **Fit**

to Window command to redraw the tree diagram to fill the entire Figure window.

Select **Tools > Fit to Window**.

Reset View Command

Use the **Reset View** command to remove formatting changes such as collapsed branches and zooms.

Select **Tools > Reset View**.

Options Submenu

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

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

Window Menu

This section illustrates how to switch to any open window.

The **Window** menu is standard on MATLAB interfaces and Figure windows. Use this menu to select any opened window.

Help Menu

This section illustrates how to select quick links to the Bioinformatics Toolbox documentation for phylogenetic analysis functions, tutorials, and the `phytreetool` reference

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

Examples

Use this list to find examples in the documentation.

Introduction

Sequence Analysis

High-Throughput Sequencing

Microarray Analysis

"Exploring Gene Expression Data" on page 4-93

Phylogenetic Analysis

"Example: Building a Phylogenetic Tree" on page 5-3

Index