

Bioinformatics Toolbox™

User's Guide



MATLAB®

R2020a



How to Contact MathWorks



Latest news: www.mathworks.com
Sales and services: www.mathworks.com/sales_and_services
User community: www.mathworks.com/matlabcentral
Technical support: www.mathworks.com/support/contact_us



Phone: 508-647-7000



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

Bioinformatics Toolbox™ User's Guide

© COPYRIGHT 2003–2020 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

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)
March 2012	Online only	Revised for Version 4.1 (Release 2012a)
September 2012	Online only	Revised for Version 4.2 (Release 2012b)
March 2013	Online only	Revised for Version 4.3 (Release 2013a)
September 2013	Online only	Revised for Version 4.3.1 (Release 2013b)
March 2014	Online only	Revised for Version 4.4 (Release 2014a)
October 2014	Online only	Revised for Version 4.5 (Release 2014b)
March 2015	Online only	Revised for Version 4.5.1 (Release 2015a)
September 2015	Online only	Revised for Version 4.5.2 (Release 2015b)
March 2016	Online only	Revised for Version 4.6 (Release 2016a)
September 2016	Online only	Updated for Version 4.7 (Release 2016b)
March 2017	Online only	Updated for Version 4.8 (Release 2017a)
September 2017	Online only	Updated for Version 4.9 (Release 2017b)
March 2018	Online only	Updated for Version 4.10 (Release 2018a)
September 2018	Online only	Updated for Version 4.11 (Release 2018b)
March 2019	Online only	Updated for Version 4.12 (Release 2019a)
September 2019	Online only	Updated for Version 4.13 (Release 2019b)
March 2020	Online only	Updated for Version 4.14 (Release 2020a)

1

Getting Started

Bioinformatics Toolbox Product Description	1-2
Key Features	1-2
Product Overview	1-3
Features	1-3
Expected Users	1-4
Data Formats and Databases	1-5
Sequence Alignments	1-7
Sequence Utilities and Statistics	1-8
Protein Property Analysis	1-9
Phylogenetic Analysis	1-10
Microarray Data Analysis Tools	1-11
Microarray Data Storage	1-12
Mass Spectrometry Data Analysis	1-13
Graph Theory Functions	1-15
Graph Visualization	1-16
Statistical Learning and Visualization	1-17
Prototyping and Development Environment	1-18
Data Visualization	1-19
Exchange Bioinformatics Data Between Excel and MATLAB	1-20
Using Excel and MATLAB Together	1-20
About the Example	1-20
Before Running the Example	1-20
Running the Example for the Entire Data Set	1-21
Editing Formulas to Run the Example on a Subset of the Data	1-22
Using the Spreadsheet Link product to Interact With the Data in MATLAB	1-23

Get Information from Web Database	1-26
What Are get Functions?	1-26
Creating the getpubmed Function	1-26

High-Throughput Sequence Analysis

2

Work with Next-Generation Sequencing Data	2-2
Overview	2-2
What Files Can You Access?	2-2
Before You Begin	2-3
Create a BioIndexedFile Object to Access Your Source File	2-3
Determine the Number of Entries Indexed By a BioIndexedFile Object ...	2-3
Retrieve Entries from Your Source File	2-4
Read Entries from Your Source File	2-4
 Manage Sequence Read Data in Objects	 2-6
Overview	2-6
Represent Sequence and Quality Data in a BioRead Object	2-7
Represent Sequence, Quality, and Alignment/Mapping Data in a BioMap Object	2-8
Retrieve Information from a BioRead or BioMap Object	2-10
Set Information in a BioRead or BioMap Object	2-12
Determine Coverage of a Reference Sequence	2-12
Construct Sequence Alignments to a Reference Sequence	2-13
Filter Read Sequences Using SAM Flags	2-14
 Store and Manage Feature Annotations in Objects	 2-16
Represent Feature Annotations in a GFFAnnotation or GTFAnnotation Object	2-16
Construct an Annotation Object	2-16
Retrieve General Information from an Annotation Object	2-16
Access Data in an Annotation Object	2-17
Use Feature Annotations with Sequence Read Data	2-18
 Visualize and Investigate Sequence Read Alignments	 2-21
When to Use the NGS Browser to Visualize and Investigate Data	2-21
Open the NGS Browser	2-21
Import Data into the NGS Browser	2-23
Zoom and Pan to a Specific Region of the Alignment	2-25
View Coverage of the Reference Sequence	2-25
View the Pileup View of Short Reads	2-26
Compare Alignments of Multiple Data Sets	2-26
View Location, Quality Scores, and Mapping Information	2-27
Flag Reads	2-28
Evaluate and Flag Mismatches	2-28
View Insertions and Deletions	2-29
View Feature Annotations	2-29
Print and Export the Browser Image	2-30
 Count Features from NGS Reads	 2-31

Identifying Differentially Expressed Genes from RNA-Seq Data	2-41
Visualize NGS Data Using Genomics Viewer App	2-69
Open the App	2-69
Add Tracks by Importing Data	2-69
Visualize Single Nucleotide Variation in Cytochrome P450	2-70

Sequence Analysis

3

Exploring a Nucleotide Sequence Using Command Line	3-2
Overview of Example	3-2
Searching the Web for Sequence Information	3-2
Reading Sequence Information from the Web	3-4
Determining Nucleotide Composition	3-5
Determining Codon Composition	3-8
Open Reading Frames	3-11
Amino Acid Conversion and Composition	3-13
Exploring a Nucleotide Sequence Using the Sequence Viewer App	3-15
Overview of the Sequence Viewer	3-15
Importing a Sequence into the Sequence Viewer	3-15
Viewing Nucleotide Sequence Information	3-17
Searching for Words	3-19
Exploring Open Reading Frames	3-22
Closing the Sequence Viewer	3-25
Explore a Protein Sequence Using the Sequence Viewer App	3-26
Overview of the Sequence Viewer	3-26
Viewing Amino Acid Sequence Statistics	3-26
Closing the Sequence Viewer	3-28
References	3-29
Compare Sequences Using Sequence Alignment Algorithms	3-30
Overview of Example	3-30
Find a Model Organism to Study	3-30
Retrieve Sequence Information from a Public Database	3-31
Search a Public Database for Related Genes	3-33
Locate Protein Coding Sequences	3-34
Compare Amino Acid Sequences	3-36
View and Align Multiple Sequences	3-43
Overview of the Sequence Alignment App	3-43
Visualize Multiple Sequence Alignment	3-43
Adjust Sequence Alignments Manually	3-44
Rearrange Rows	3-52
Generate Phylogenetic Tree from Aligned Sequences	3-54

Managing Gene Expression Data in Objects	4-2
Representing Expression Data Values in DataMatrix Objects	4-5
Overview of DataMatrix Objects	4-5
Constructing DataMatrix Objects	4-5
Getting and Setting Properties of a DataMatrix Object	4-6
Accessing Data in DataMatrix Objects	4-6
Representing Expression Data Values in ExptData Objects	4-9
Overview of ExptData Objects	4-9
Constructing ExptData Objects	4-9
Using Properties of an ExptData Object	4-10
Using Methods of an ExptData Object	4-10
References	4-11
Representing Sample and Feature Metadata in MetaData Objects	4-12
Overview of MetaData Objects	4-12
Constructing MetaData Objects	4-13
Using Properties of a MetaData Object	4-15
Using Methods of a MetaData Object	4-15
Representing Experiment Information in a MIAME Object	4-16
Overview of MIAME Objects	4-16
Constructing MIAME Objects	4-16
Using Properties of a MIAME Object	4-17
Using Methods of a MIAME Object	4-18
Representing All Data in an ExpressionSet Object	4-19
Overview of ExpressionSet Objects	4-19
Constructing ExpressionSet Objects	4-20
Using Properties of an ExpressionSet Object	4-21
Using Methods of an ExpressionSet Object	4-21
Visualizing Microarray Images	4-23
Overview of the Mouse Example	4-23
Exploring the Microarray Data Set	4-23
Spatial Images of Microarray Data	4-25
Statistics of the Microarrays	4-33
Scatter Plots of Microarray Data	4-34

Using the Phylogenetic Tree App	5-2
Overview of the Phylogenetic Tree App	5-2
Opening the Phylogenetic Tree App	5-2
File Menu	5-3
Tools Menu	5-11

Window Menu	5-17
Help Menu	5-18

Getting Started

- “Bioinformatics Toolbox Product Description” on page 1-2
- “Product Overview” on page 1-3
- “Data Formats and Databases” on page 1-5
- “Sequence Alignments” on page 1-7
- “Sequence Utilities and Statistics” on page 1-8
- “Protein Property Analysis” on page 1-9
- “Phylogenetic Analysis” on page 1-10
- “Microarray Data Analysis Tools” on page 1-11
- “Microarray Data Storage” on page 1-12
- “Mass Spectrometry Data Analysis” on page 1-13
- “Graph Theory Functions” on page 1-15
- “Graph Visualization” on page 1-16
- “Statistical Learning and Visualization” on page 1-17
- “Prototyping and Development Environment” on page 1-18
- “Data Visualization” on page 1-19
- “Exchange Bioinformatics Data Between Excel and MATLAB” on page 1-20
- “Get Information from Web Database” on page 1-26

Bioinformatics Toolbox Product Description

Read, analyze, and visualize genomic and proteomic data

Bioinformatics Toolbox provides algorithms and apps 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.

Key Features

- Next Generation Sequencing analysis and browser
- Sequence analysis and visualization, including pairwise and multiple sequence alignment and peak detection
- Microarray data analysis, including reading, filtering, normalizing, and visualization
- Mass spectrometry analysis, including preprocessing, classification, and marker identification
- Phylogenetic tree analysis
- Graph theory functions, including interaction maps, hierarchy plots, and pathways
- Data import from genomic, proteomic, and gene expression files, including SAM, FASTA, CEL, and CDF, and from databases such as NCBI and GenBank

Product Overview

Features

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 1-18.
- **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.

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.

Data Formats and Databases

The Bioinformatics Toolbox lets you access many of the databases on the web and other online data repositories. It lets you 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`). Select sections of the ontology with methods for the `geneont` object (`getancestors (geneont)`, `getdescendants (geneont)`, `getmatrix (geneont)`, `getrelatives (geneont)`), 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.

See Also

More About

- “High-Throughput Sequencing”
- “Microarray Analysis”
- “Sequence Analysis”
- “Structural Analysis”
- “Mass Spectrometry and Bioanalytics”

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 (`seqalignviewer`) 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`).

See Also

More About

- “Sequence Utilities and Statistics” on page 1-8
- “Sequence Analysis”
- “Data Formats and Databases” on page 1-5

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 (`seqviewer`).

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`).

See Also

More About

- “Sequence Alignments” on page 1-7
- “Sequence Analysis”
- “Protein and Amino Acid Sequence Analysis”
- “Data Formats and Databases” on page 1-5

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 (`aaccount`) and get information about character codes (`aminolookup`).

See Also

More About

- “Protein and Amino Acid Sequence Analysis”
- “Structural Analysis”

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. The Bioinformatics Toolbox provides the following data structure and functions for phylogenetic analysis.

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`, `dndsm1`), 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 (`phytreeviewer` 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 `phytreeviewer` user interface 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`).

See Also

More About

- “Sequence Utilities and Statistics” on page 1-8
- “Sequence Analysis”

Microarray Data Analysis Tools

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 (`mimage`, `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.

See Also

More About

- “Microarray Data Storage” on page 1-12
- “Microarray Analysis”

Microarray Data Storage

The Bioinformatics 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.

See Also

More About

- “Microarray Data Analysis Tools” on page 1-11
- “Microarray Analysis”

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.

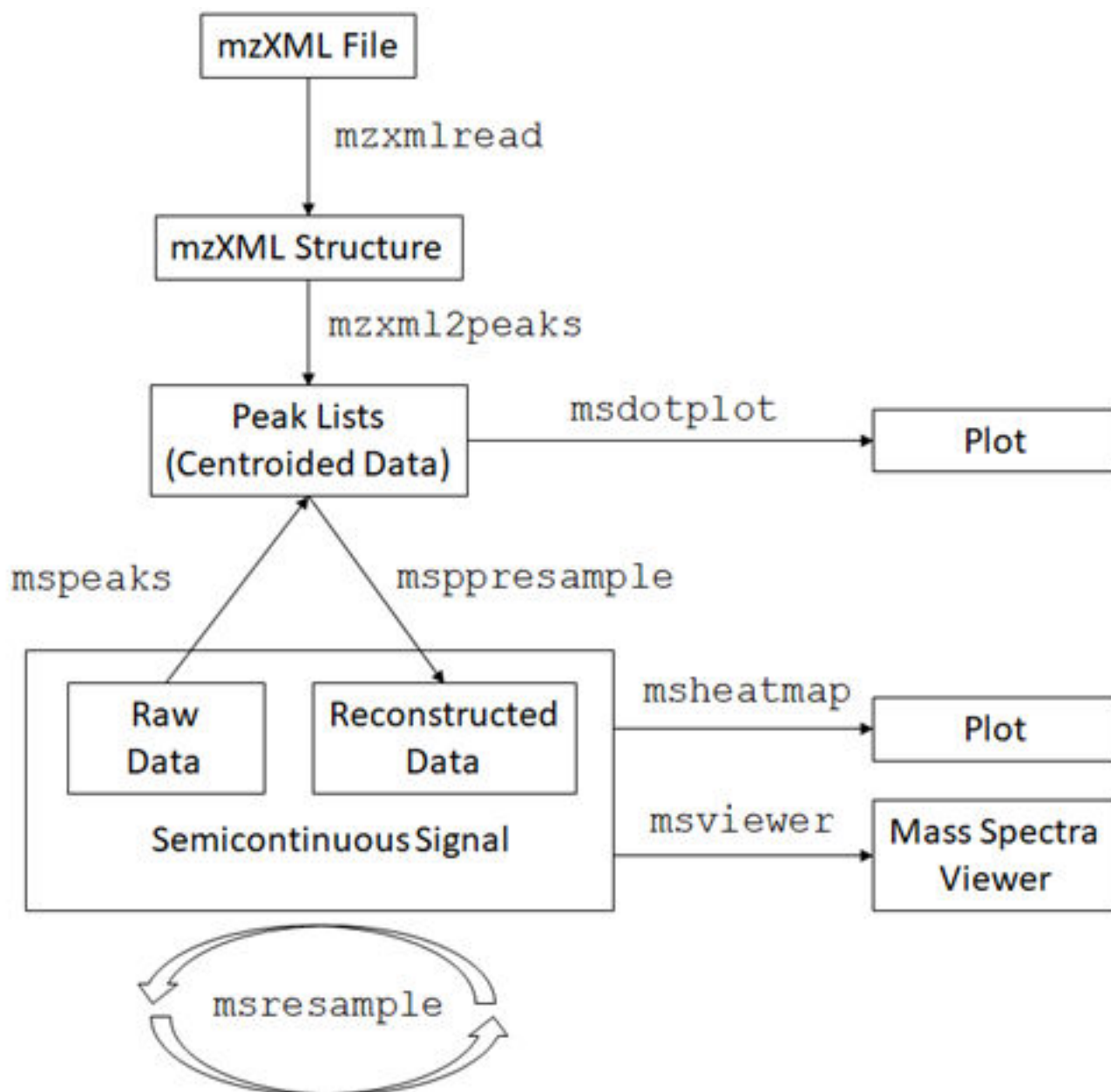
Reading raw data — Load raw mass/charge and ion intensity data from comma-separated-value (CSV) files, or read a JCAMP-DX-formatted file with mass spectrometry data (`jcampread`) into the MATLAB environment.

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.



See Also

More About

- “Mass Spectrometry and Bioanalytics”
- “Data Formats and Databases” on page 1-5

Graph Theory Functions

Graph theory functions in the Bioinformatics 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`.

See Also

More About

- “Graph Visualization” on page 1-16
- “Network Analysis and Visualization”

Graph Visualization

The Bioinformatics 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`).

See Also

More About

- “Graph Theory Functions” on page 1-15
- “Network Analysis and Visualization”

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 and Machine Learning Toolbox™ software (`classify`, `kmeans`, `fitctree`, and `fitrtree`).

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

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®.

Exchange Bioinformatics Data Between Excel and MATLAB

In this section...

“Using Excel and MATLAB Together” on page 1-20

“About the Example” on page 1-20

“Before Running the Example” on page 1-20

“Running the Example for the Entire Data Set” on page 1-21

“Editing Formulas to Run the Example on a Subset of the Data” on page 1-22

“Using the Spreadsheet Link product to Interact With the Data in MATLAB” on page 1-23

Using Excel and MATLAB Together

If you have bioinformatics data in an Excel (2007 or newer) spreadsheet, use Spreadsheet Link 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 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 “Gene Expression Profile Analysis”.

Before Running the Example

- 1 If not already done, modify your system path to include the MATLAB root folder as described in the Spreadsheet Link documentation.
- 2 If not already done, enable the Spreadsheet Link Add-In as described in “Add-In Setup” (Spreadsheet Link).
- 3 Close MATLAB and Excel if they are open.
- 4 Start Excel. MATLAB and Spreadsheet Link 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. If you encounter the “Can't find project or library” error, you might need to update the references in the Visual Basic software. Open Visual Basic by clicking the **Developer**

tab and selecting **Visual Basic**. Then select **Tools > References > SpreadsheetLink**. If the **MISSING: exclink2007.xlam** check box is selected, clear it.

Running the Example for the Entire Data Set

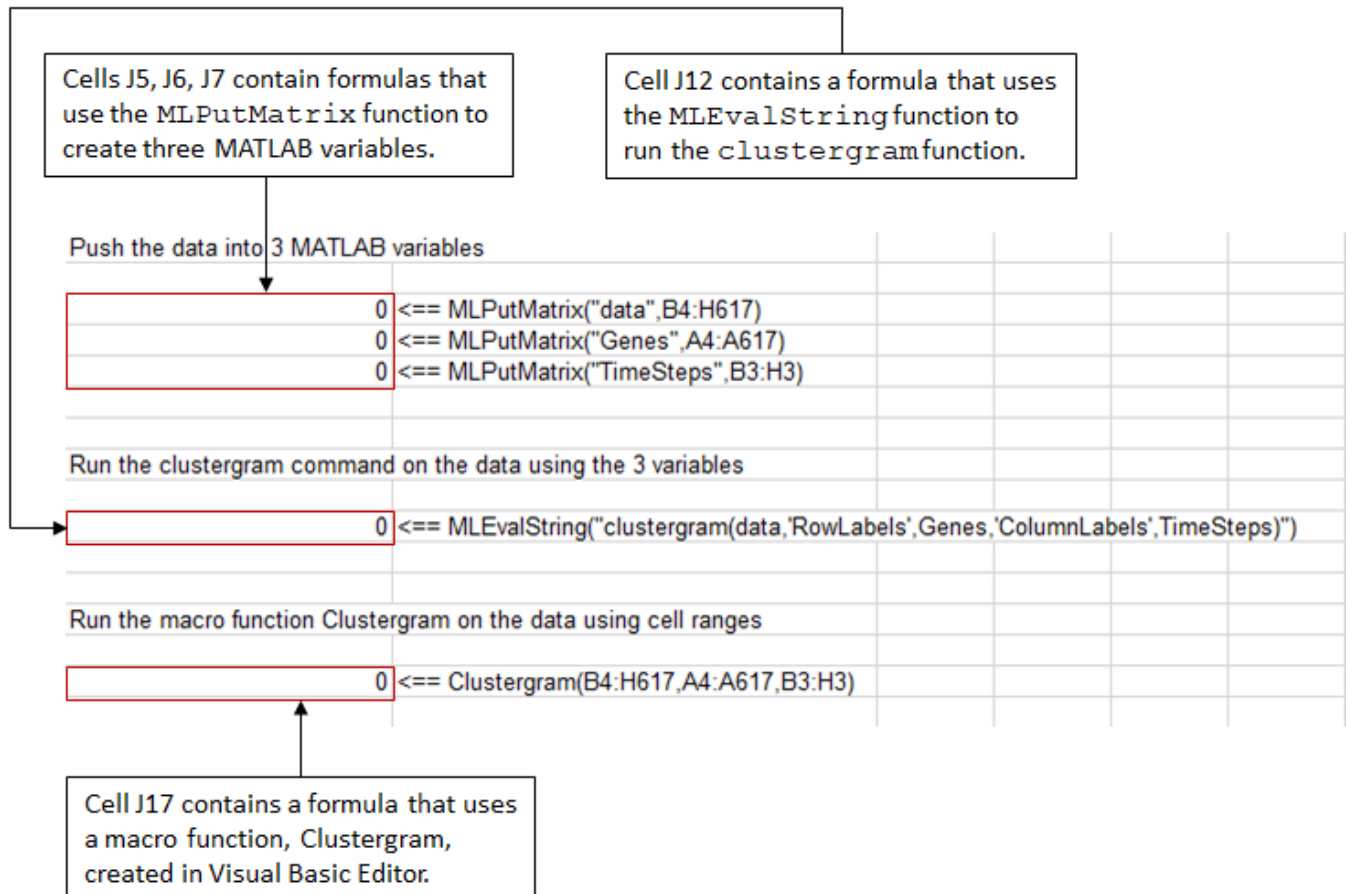
- 1 In the provided Excel file, note that columns A through H contain data from DeRisi et al. Also note that cells J5, J6, J7, and J12 contain formulas using Spreadsheet Link functions `MLPutMatrix` and `MLEvalString`.

Tip To view a cell's formula, select the cell, and then view the formula in the formula bar

 _____ 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 function `MLPutMatrix`, which creates a MATLAB variable from the data in the spreadsheet. Cell J12 contains a formula using the Spreadsheet Link function `MLEvalString`, which runs the Bioinformatics Toolbox `clustergram` function using the three variables as input. For more information on adding formulas using Spreadsheet Link functions, see "Create Diagonal Matrix Using Worksheet Cells" (Spreadsheet Link).



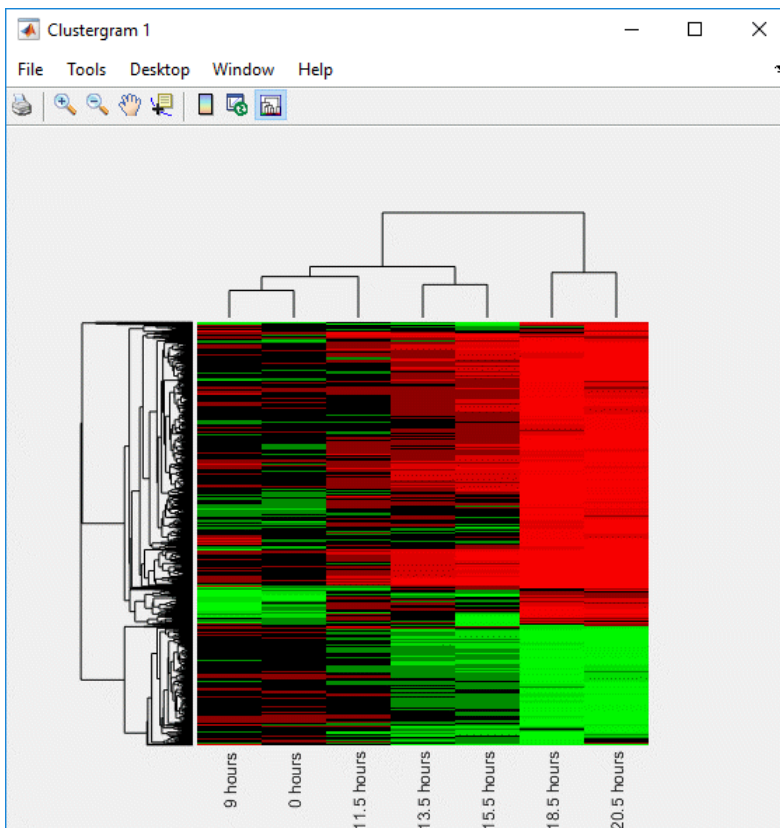
- Note that cell J17 contains a formula using a macro function Clustergram, which was created in the Visual Basic® Editor. Running this macro does the same as the formulas in cells J5, J6, J7, and J12. Optionally, view the Clustergram macro function by clicking the **Developer** tab, and then

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 “Create Diagonal Matrix Using VBA Macro” (Spreadsheet Link).

- Execute the formula in cell J17 to analyze and visualize the data:
 - Select cell **J17**.
 - Press **F2**.
 - 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

- 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**.

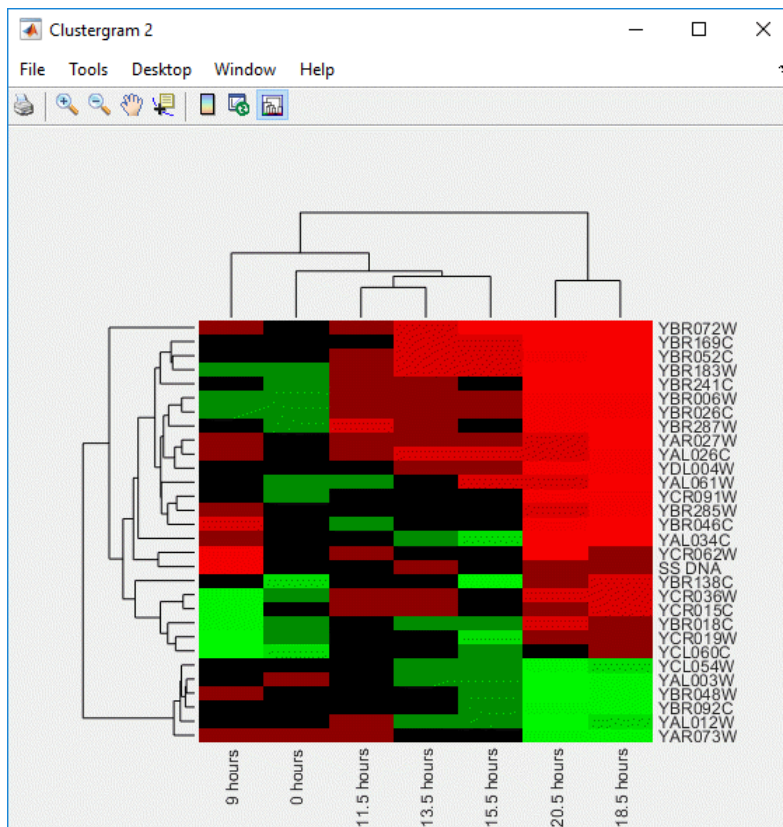
```
=MLPutMatrix("data",B4:H33)
```

- 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)
```

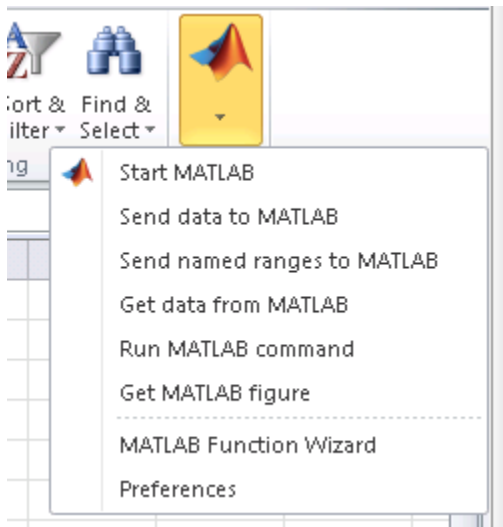
- 2 Run the formulas in cells J5, J6, J7, and J12 to analyze and visualize a subset of the data:

- 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 product 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.

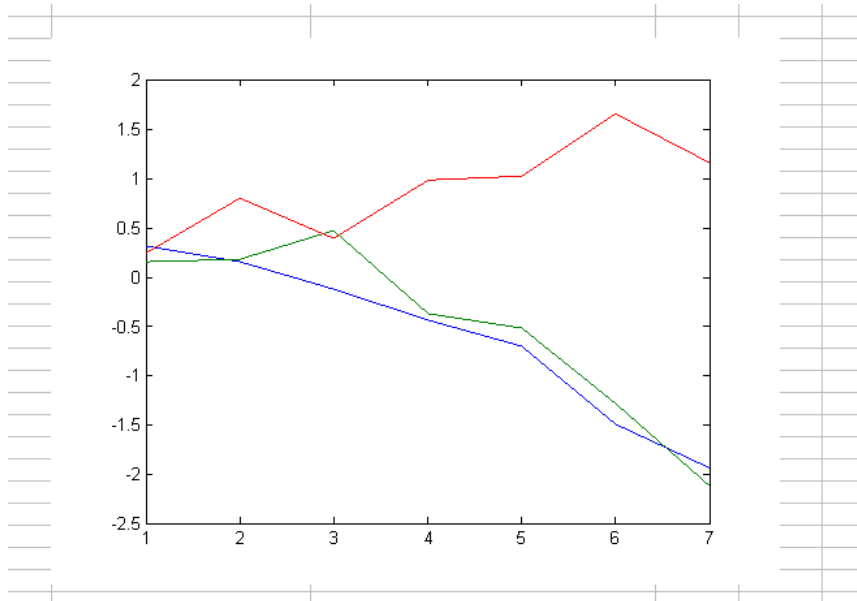
- 4 From the MATLAB group, select **Run MATLAB command**.
- 5 Type **plot(YAGenes')** for the command, and then click **OK**.

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.



Get Information from Web Database

In this section...

“What Are get Functions?” on page 1-26

“Creating the getpubmed Function” on page 1-26

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.

The screenshot shows the NCBI PubMed website. At the top, there is a navigation bar with tabs for 'All Databases', 'PubMed', 'Nucleotide', 'Protein', 'Genome', 'Structure', 'OMIM', 'PMC', 'Journals', and 'Books'. Below this is a search bar with 'PubMed' selected in the dropdown and a search input field. To the right of the search bar are buttons for 'Go', 'Clear', and 'Advanced Search (beta)'. Below the search bar are buttons for 'Limits', 'Preview/Index', 'History', 'Clipboard', and 'Details'. On the left side, there is a sidebar with links to 'About Entrez Text Version', 'Entrez PubMed Overview', 'Help | FAQ', 'Tutorials', 'New/Noteworthy', 'E-Utilities', 'PubMed Services', 'Journals Database', 'MeSH Database', 'Single Citation Matcher', 'Batch Citation Matcher', 'Clinical Queries', 'Special Queries', 'LinkOut', and 'My NCBI'. In the main content area, there is a message: 'To get started with PubMed, enter one or more search terms. Search terms may be topics, authors or journals.' Below this is a highlighted box with a 'My NCBI' logo and the text: 'Set up an automated PubMed update in fewer than five minutes.' followed by a list of steps: 1. Create a My NCBI account. 2. Save your search. 3. Your PubMed updates can be e-mailed directly to you. Below the list is a link to 'My NCBI Help' and a paragraph of text: 'Read the My NCBI Help material to explore other options, such as automated updates of other databases, setting search filters, and highlighting search terms.' At the bottom of the page, there is a paragraph: '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-value pairs that let the user of the function limit the search by publication date and limit the number of records returned. Below is the step-by-step guide to create the function from the beginning. To see the completed m-file, type `edit getpubmed.m`.

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(message('bioinfo:getpubmed:NotEnoughInputArguments'));
end
```

4 Create variables for the two property name-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-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 = 'https://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 file 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.

High-Throughput Sequence Analysis

- “Work with Next-Generation Sequencing Data” on page 2-2
- “Manage Sequence Read Data in Objects” on page 2-6
- “Store and Manage Feature Annotations in Objects” on page 2-16
- “Visualize and Investigate Sequence Read Alignments” on page 2-21
- “Count Features from NGS Reads” on page 2-31
- “Identifying Differentially Expressed Genes from RNA-Seq Data” on page 2-41
- “Visualize NGS Data Using Genomics Viewer App” on page 2-69

Work with Next-Generation Sequencing Data

In this section...

“Overview” on page 2-2

“What Files Can You Access?” on page 2-2

“Before You Begin” on page 2-3

“Create a BioIndexedFile Object to Access Your Source File” on page 2-3

“Determine the Number of Entries Indexed By a BioIndexedFile Object” on page 2-3

“Retrieve Entries from Your Source File” on page 2-4

“Read Entries from Your Source File” on page 2-4

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 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.

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 vector, 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. .

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.

Create 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.
- 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.

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

Determine 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.

Retrieve Entries from Your Source File

Retrieve entries from your source file using either:

- The index of the entry
- The entry key

Retrieve 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]);
```

Retrieve 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 character vector 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.

Read 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.

Set the Interpreter Property

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

If you constructed a <code>BioIndexedFile</code> object from ...	The <code>Interpreter</code> 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.

If you constructed a <code>BioIndexedFile</code> object from ...	The Interpreter property ...
A source file with a table, multi-row table, or flat format	By default is <code>[]</code> , 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 character vector 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.

Read 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 character vectors and extracts those 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'
```

Manage Sequence Read Data in Objects

In this section...

“Overview” on page 2-6

“Represent Sequence and Quality Data in a BioRead Object” on page 2-7

“Represent Sequence, Quality, and Alignment/Mapping Data in a BioMap Object” on page 2-8

“Retrieve Information from a BioRead or BioMap Object” on page 2-10

“Set Information in a BioRead or BioMap Object” on page 2-12

“Determine Coverage of a Reference Sequence” on page 2-12

“Construct Sequence Alignments to a Reference Sequence” on page 2-13

“Filter Read Sequences Using SAM Flags” on page 2-14

Overview

High-throughput sequencing instruments produce large amounts of sequence read 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 sequence read data.

Object	Contains This Information	Construct from One of These
BioRead	<ul style="list-style-type: none"> Sequence headers Read sequences Sequence qualities (base calling) 	<ul style="list-style-type: none"> FASTQ file SAM file FASTQ structure (created using the <code>fastqread</code> function) SAM structure (created using the <code>samread</code> function) Cell arrays containing header, sequence, and quality information (created using the <code>fastqread</code> function)
BioMap	<ul style="list-style-type: none"> Sequence headers Read sequences Sequence qualities (base calling) Sequence alignment and mapping information (relative to a single reference sequence), including mapping quality 	<ul style="list-style-type: none"> SAM file BAM file SAM structure (created using the <code>samread</code> function) BAM structure (created using the <code>bamread</code> function) Cell arrays containing header, sequence, quality, and mapping/alignment information (created using the <code>samread</code> or <code>bamread</code> function)

Represent Sequence and Quality Data in a BioRead Object

Prerequisites

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

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.

Construct 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 with properties:
```

```
    Quality: [50x1 File indexed property]
    Sequence: [50x1 File indexed property]
    Header: [50x1 File indexed property]
    NSeqs: 50
    Name: 'MyObject'
```

The constructor function constructs 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.

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

Prerequisites

A `BioMap` object represents a collection of sequence reads 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.

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 stays in 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.

Construct 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 If you do not know the number and names of the reference sequences in your source file, determine them 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 with properties:
```

```
SequenceDictionary: 'seq1'
  Reference: [1501x1 File indexed property]
  Signature: [1501x1 File indexed property]
  Start: [1501x1 File indexed property]
MappingQuality: [1501x1 File indexed property]
  Flag: [1501x1 File indexed property]
  MatePosition: [1501x1 File indexed property]
  Quality: [1501x1 File indexed property]
  Sequence: [1501x1 File indexed property]
  Header: [1501x1 File indexed property]
  NSeqs: 1501
  Name: 'MyObject'
```

The constructor function constructs a `BioMap` object and, if index files do not already exist, it also creates one or two index files:

- If constructing from a SAM-formatted file, it creates one index file that has the same file name as the source file, but with an `.IDX` extension. This index file, by default, is stored in the same location as the source file.
- If constructing from a BAM-formatted file, it creates two index files that have the same file name as the source file, but one with a `.BAI` extension and one with a `.LINEARINDEX` extension. These index files, by default, are stored in the same location as the source file.

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

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

Construct 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 with properties:

```
SequenceDictionary: {'seq1'}
Reference: {1501x1 cell}
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: ''
```

Retrieve Information from a BioRead or BioMap Object

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

Retrieve a Property from a BioRead or BioMap Object

You can retrieve a specific property from elements in a `BioRead` or `BioMap` object.

For example, to retrieve all headers from a `BioRead` object, use the `Header` property as follows:

```
allHeaders = BRObj1.Header;
```

This syntax returns a cell array containing the headers for all elements in the `BioRead` object.

Similarly, to retrieve all start positions of aligned read sequences from a `BioMap` object, use the `Start` property of the object:

```
allStarts = BMObj1.Start;
```

This 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.

Retrieve Multiple Properties from a BioRead or BioMap Object

You can retrieve multiple properties from a `BioRead` or `BioMap` object in a single command using the `get` method. For example, to retrieve both start positions and headers information of a `BioMap` object, use the `get` method as follows:

```
multiProp = get(BMobj1, {'Start', 'Header'});
```

This syntax returns a cell array containing all start positions and headers information of 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.

Retrieve 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]);
```

This 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);
```

This syntax returns a tab-delimited character vector 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 character vector) 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.

Set Information in a `BioRead` or `BioMap` Object

Prerequisites

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 “Construct a `BioMap` Object from a SAM or BAM Structure” on page 2-9.
- Construct the object from a source file using the `InMemory` name-value pair argument.

Provide Custom Headers for Sequences

First, create an object with the data in memory:

```
BRObj1 = BioRead('SRR005164_1_50.fastq','InMemory',true);
```

To provide custom headers for sequences of interest (in this case sequences 1 to 5), do the following:

```
BRObj1.Header(1:5) = {'H1', 'H2', 'H3', 'H4', 'H5'};
```

Alternatively, you can use the `setHeader` method:

```
BRObj1 = setHeader(BRObj1, {'H1', 'H2', 'H3', 'H4', 'H5'}, [1:5]);
```

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

Determine 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

```

Construct 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'
```

Filter 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 SAM Format Specification. The `filterByFlag` method lets you filter the read sequences in a `BioMap` object by using these flags.

Filter 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);
```

Filter 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);
```

Store and Manage Feature Annotations in Objects

In this section...

“Represent Feature Annotations in a GFFAnnotation or GTFAnnotation Object” on page 2-16

“Construct an Annotation Object” on page 2-16

“Retrieve General Information from an Annotation Object” on page 2-16

“Access Data in an Annotation Object” on page 2-17

“Use Feature Annotations with Sequence Read Data” on page 2-18

Represent 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.

Construct 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')
```

```
GFFAnnotObj =
```

```
    GFFAnnotation with properties:
```

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

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

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

```
GTFAnnotObj =
```

```
    GTFAnnotation with properties:
```

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

Retrieve 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.FieldNameNames
```

```
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
```

Access Data in an Annotation Object

Create 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
```

Access 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
```

Use Feature Annotations with Sequence Read Data

Investigate the results of HTS sequencing experiments by using `GFFAnnotation` and `GTFAnnotation` objects with `BioMap` objects. For example, you can:

- Determine counts of sequence reads 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.

Determine 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.

Filter 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.

Extract 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];
```

Determine Counts of Sequence Reads Aligned to Annotations

Construct a BioMap object from a BAM-formatted file containing sequence read 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
```

Visualize and Investigate Sequence Read Alignments

In this section...

“When to Use the NGS Browser to Visualize and Investigate Data” on page 2-21
 “Open the NGS Browser” on page 2-21
 “Import Data into the NGS Browser” on page 2-23
 “Zoom and Pan to a Specific Region of the Alignment” on page 2-25
 “View Coverage of the Reference Sequence” on page 2-25
 “View the Pileup View of Short Reads” on page 2-26
 “Compare Alignments of Multiple Data Sets” on page 2-26
 “View Location, Quality Scores, and Mapping Information” on page 2-27
 “Flag Reads” on page 2-28
 “Evaluate and Flag Mismatches” on page 2-28
 “View Insertions and Deletions” on page 2-29
 “View Feature Annotations” on page 2-29
 “Print and Export the Browser Image” on page 2-30

When to Use the NGS Browser to Visualize and Investigate Data

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

- Visualize sequence reads 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.

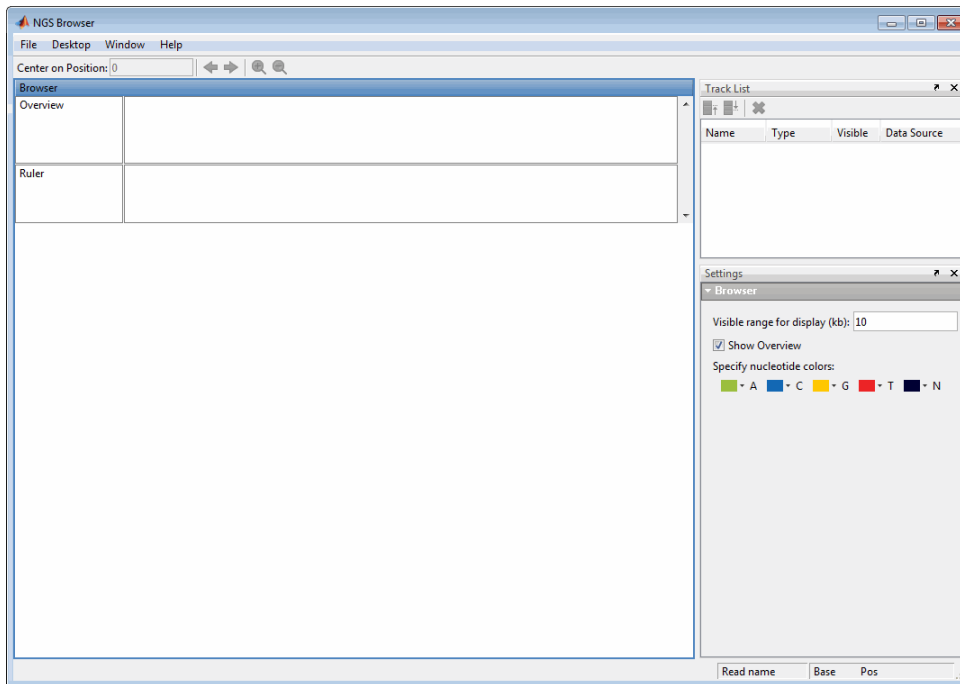
You can visualize and investigate the aligned data before, during, or after any preprocessing (filtering, quality recalibration) or analysis steps you perform on the aligned data.

Open the NGS Browser

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

```
ngsbrowser
```

Alternatively, click the **NGS Browser** on the **Apps** tab.



Import 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

Import 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.

Import Sequence Read Alignment Data

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

- BioMap object

Tip Construct a BioMap object on page 2-8 from a SAM- or BAM-formatted file to investigate, subset, and filter on page 2-14 the data before importing it into the NGS Browser.

- SAM- or BAM-formatted file

Note Your SAM- or BAM-formatted file must:

- Have reads ordered by start position in the reference sequence.
 - Have an IDX index file (for a SAM-formatted file) or BAI and LINEARINDEX index files (for a BAM-formatted file) stored in the same location as your source file. Otherwise, the source file must be stored in a location to which you have write access, because MATLAB needs to create and store index files in this location.
-

Tip Try 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 index files (IDX or BAI and LINEARINDEX) 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, you cannot import data from the source file directly into the browser. Instead, 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 sequence 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** If you select a file containing multiple reference sequences, in the Select Reference dialog box, select a reference or scan the file for available references and their mapped reads counts. Click **OK**.
- 4** Repeat the previous steps to import additional data sets.


Import 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.


Zoom and Pan 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:

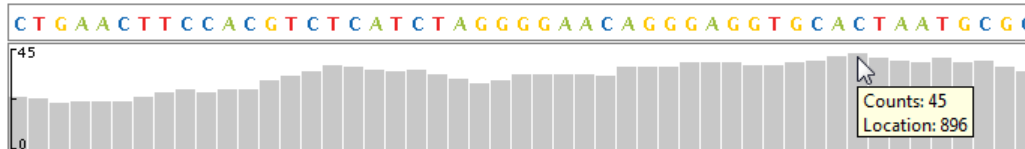
Use the  toolbar buttons, or click-drag the rubberband in the Overview area.



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

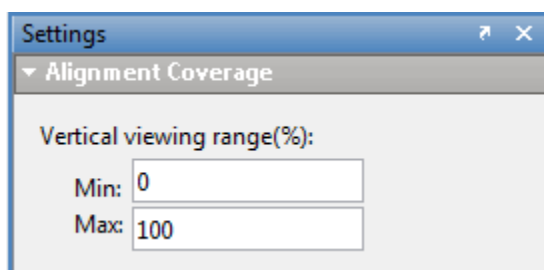
View Coverage of the Reference Sequence

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.



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

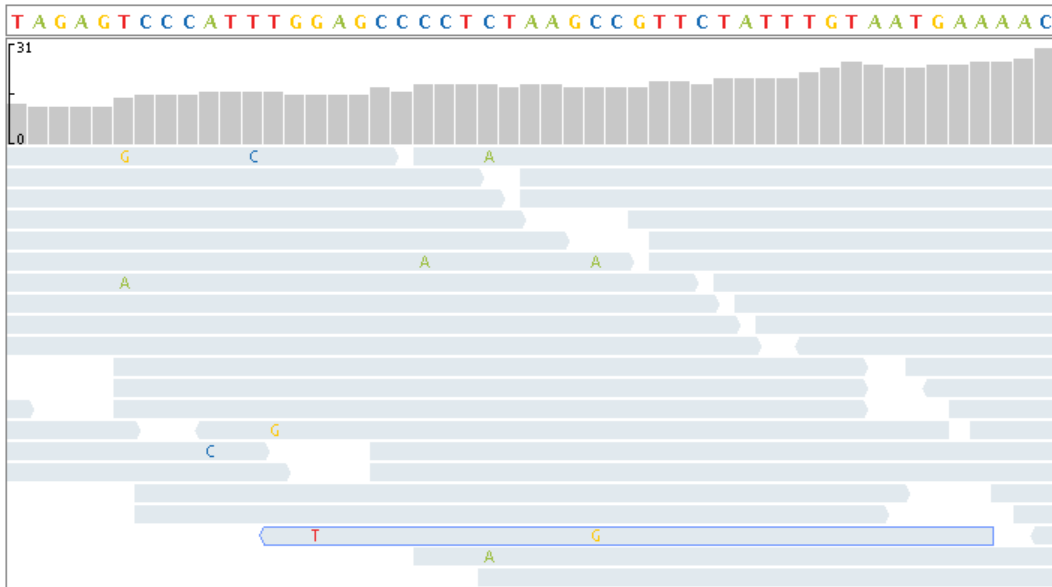
To change the percent coverage displayed, click anywhere in the alignment track, and then edit the Alignment Coverage settings.



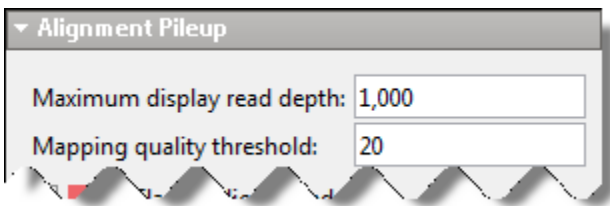
Tip Set **Max** to a value greater than 100, if needed, when comparing the coverage of multiple tracks of reads.

View the Pileup View of Short Reads

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



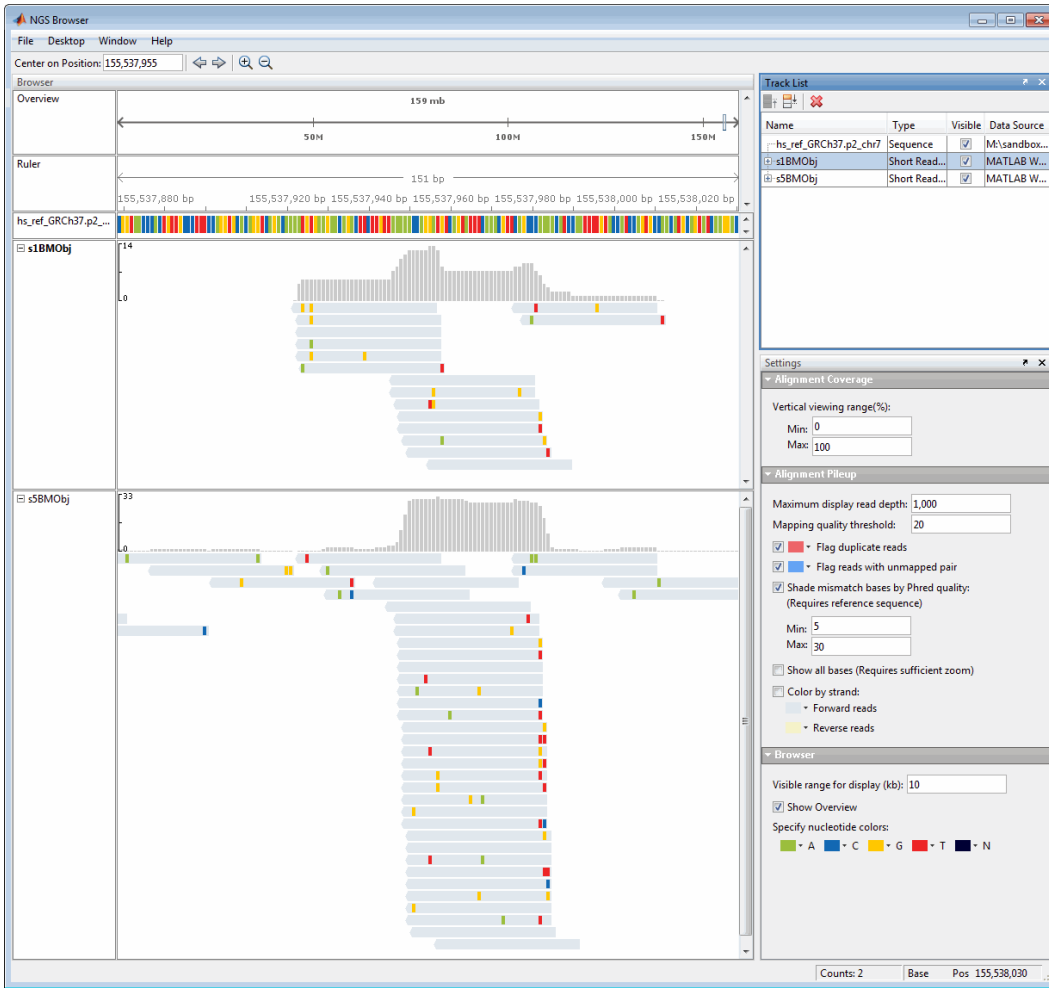
Limit the depth of the reads displayed in the pileup view by setting the **Maximum display read depth** in the Alignment Pileup settings.



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

Compare 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.



View 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
    
```

Flag Reads

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

▼ Alignment Pileup

Maximum display read depth:

Mapping quality threshold:

■ ▼ Flag duplicate reads

■ ▼ Flag reads with unmapped pair

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

Min:

Max:

Show all bases (Requires sufficient zoom)

Color by strand:

■ ▼ Forward reads

■ ▼ Reverse reads

Flag 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.

Flag Duplicate Reads

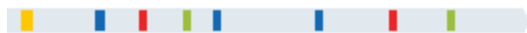
Select **Flag duplicate reads** and select an outline color.

Flag Reads with Unmapped Pairs

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

Evaluate and Flag Mismatches

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

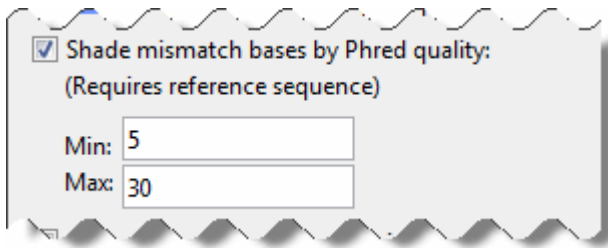


Zoomed out view of read – Mismatches display as bars



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.

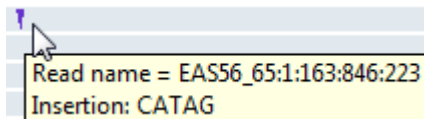


The mismatch blocks or letters display in:

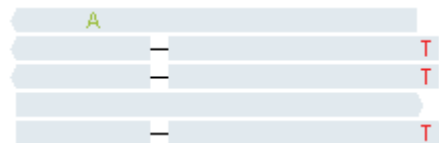
- Light shade — Mismatch bases with Phred scores below the minimum
- Graduation of medium shades — Mismatch bases with Phred scores within the minimum to maximum range
- Dark shade — Mismatch bases with Phred scores above the maximum

View Insertions and Deletions

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

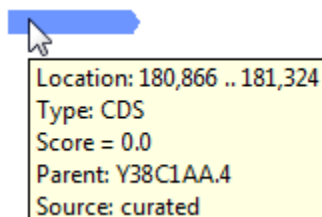


The NGS Browser designates deletions with dashes.



View 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.



Print and Export the Browser Image

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

Count Features from NGS Reads

This example shows how to count features from paired-end sequencing reads after aligning them to the whole human genome curated by the Genome Reference Consortium. This example uses Genome Reference Consortium Human Build 38 patch release 12 (GRCh38.p12) as the human genome reference.

Prerequisites and Data Set

This example works on the UNIX® and Mac platforms only. Download the Bioinformatics Toolbox™ Interface for Bowtie Aligner support package from the Add-On Explorer.

This example assumes you have:

- Downloaded and extracted the RefSeq assembly from Genome Reference Consortium Human Build 38 patch release 12 (GRCh38.p12).
- Downloaded and organized some paired-end reads data. This example uses the exome sequencing data from the 1000 genomes project. Paired-end reads are indicated by '_1' and '_2' in the filenames following the accession number. Here is one possibility for how the data can be organized in folders:

```
sequence/
+--HG00096/
|  +--SRR077487_1.filt.fastq
|  +--SRR077487_2.filt.fastq
|  +--SRR081241_1.filt.fastq
|  +--SRR081241_2.filt.fastq
|
+--HG00097
   +-- SRR765989_1.filt.fastq
   +-- SRR765989_2.filt.fastq
```

Build Index

Construct an index for aligning reads to the reference using `bowtie2build`. The file `GCF_000001405.38_GRCh38.p12_genomic.fna` contains the human reference genome in the FASTA format. `bowtieIdx` is the base name of the reference index files. The `'--threads 8'` option specifies the number of parallel threads to build index files faster.

```
bowtieIdx = 'GCF_000001405.38_GRCh38.p12_genomic.index';
buildFlag = bowtie2build('GCF_000001405.38_GRCh38.p12_genomic.fna',...
                        bowtieIdx,'--threads 8');
```

Align Reads to Reference

Use the helper function `alignAllReads` to align each set of paired-end reads to the reference. The function produces a SAM file in the current folder for each sample in the 'sequence' folder.

```
addpath(fullfile(matlabroot,'examples','bioinfo','main')); % Make sure the supporting files are of
type alignAllReads
```

```
function samFiles = alignAllReads(indexBaseName, inputDir, outputDir)
    %ALIGNALLREADS Helper function for CountFeaturesExample
    %
    % SAMFILES = alignAllReads(INDEXBASENAME, INPUTDIR, OUTPUTDIR) aligns
```

```
% paired-end sequencing reads using Bowtie2 to a prebuilt index,
% producing SAM-formatted alignment files. INDEXBASENAME
% is a character vector specifying the prefix of the index files
% created with BOWTIE2BUILD. INPUTDIR contains subdirectories for each
% sample containing paired-end reads in FASTQ format:
%
% INPUTDIR/
% +-- HG00096/
% |   +-- SRR077487_1.filt.fastq
% |   +-- SRR077487_2.filt.fastq
% |   +-- SRR081241_1.filt.fastq
% |   +-- SRR081241_2.filt.fastq
% |
% +-- HG00097/
%   +-- SRR765989_1.filt.fastq
%   +-- SRR765989_2.filt.fastq
%
% NOTE: each mate is distinguished by the '_1' or '_2' after the
% accession number in the filename.
%
% SAMFILES is a cell array of the resulting SAM-formatted files created
% with Bowtie2, and are placed in OUTPUTDIR.

% Copyright 2018 The MathWorks, Inc.

% Use dir() to identify sample names (subfolders of inputDir).
d = dir(inputDir);
samples = {d(3:end).name}; % skip special '.' & '..' folders
samFiles = strcat(samples, '.sam');

for s=1:numel(samples)
    % Identify mate pairs of reads for each sample
    sampleReadsPath = fullfile(inputDir, samples{s});
    reads1 = dir([sampleReadsPath '/*_1*']);
    reads2 = dir([sampleReadsPath '/*_2*']);

    reads1 = fullfile(sampleReadsPath, {reads1(:).name});
    reads2 = fullfile(sampleReadsPath, {reads2(:).name});

    % Get full filename to SAM file
    samFiles{s} = fullfile(outputDir, samFiles{s});

    % Perform alignment, if file doesn't exist
    if ~exist(samFiles{s}, 'file')
        bowtie2(indexBaseName, reads1, reads2, samFiles{s}, '-p 2');
    end
end
end

samFiles = alignAllReads(bowtieIdx, 'sequence', '.');
```

Selectively Align to Gene of Interest

SAM files can be very large. Use BioMap to select only the data for the correct reference. For this example, consider APOE, which is a gene on Chromosome 19 linked to Alzheimer's disease. Create a set of smaller BAM files for APOE to improve performance.

```

apoeRef = 'NC_000019.10'; % Reference name for Chromosome 19 in HG38
bamFiles = strep(samFiles, '.sam', '.bam');
for i=1:numel(samFiles)
    if ~exist(bamFiles{i}, 'file')
        bm = BioMap(samFiles{i}, 'SelectReference', apoeRef);
        write(bm, bamFiles{i}, 'Format', 'bam');
    end
end

```

Summarize Read Counts

Use `featurecount` to compare the number of transcripts for each APOE variant using a GTF file. A full table of features is included in the GRCh38.p12 assembly in GFF format, which can be converted to GTF using `cuffgffread`. This example uses a simplified GTF based on APOE transcripts. `APOE_gene.gtf` is included with the software.

```

[FeatTable, Summary] = featurecount('APOE_gene.gtf', bamFiles, ...
                                   'Metafeature', 'transcript_id');

```

```

Processing GTF file APOE_gene.gtf ...
Processing BAM file ./HG00096.bam ...
Processing reference NC_000019.10 ...
10000 reads processed ...
20000 reads processed ...
30000 reads processed ...
40000 reads processed ...
50000 reads processed ...
60000 reads processed ...
70000 reads processed ...
80000 reads processed ...
90000 reads processed ...
100000 reads processed ...
110000 reads processed ...
120000 reads processed ...
130000 reads processed ...
140000 reads processed ...
150000 reads processed ...
160000 reads processed ...
170000 reads processed ...
180000 reads processed ...
190000 reads processed ...
200000 reads processed ...
210000 reads processed ...
220000 reads processed ...
230000 reads processed ...
240000 reads processed ...
250000 reads processed ...
260000 reads processed ...
270000 reads processed ...
280000 reads processed ...
290000 reads processed ...
300000 reads processed ...
310000 reads processed ...
320000 reads processed ...
330000 reads processed ...
340000 reads processed ...
350000 reads processed ...
360000 reads processed ...

```

370000 reads processed ...
380000 reads processed ...
390000 reads processed ...
400000 reads processed ...
410000 reads processed ...
420000 reads processed ...
430000 reads processed ...
440000 reads processed ...
450000 reads processed ...
460000 reads processed ...
470000 reads processed ...
480000 reads processed ...
490000 reads processed ...
500000 reads processed ...
510000 reads processed ...
520000 reads processed ...
530000 reads processed ...
540000 reads processed ...
550000 reads processed ...
560000 reads processed ...
570000 reads processed ...
580000 reads processed ...
590000 reads processed ...
600000 reads processed ...
610000 reads processed ...
620000 reads processed ...
630000 reads processed ...
640000 reads processed ...
650000 reads processed ...
660000 reads processed ...
670000 reads processed ...
680000 reads processed ...
690000 reads processed ...
700000 reads processed ...
710000 reads processed ...
720000 reads processed ...
730000 reads processed ...
740000 reads processed ...
750000 reads processed ...
760000 reads processed ...
770000 reads processed ...
780000 reads processed ...
790000 reads processed ...
800000 reads processed ...
810000 reads processed ...
820000 reads processed ...
830000 reads processed ...
840000 reads processed ...
850000 reads processed ...
860000 reads processed ...
870000 reads processed ...
880000 reads processed ...
890000 reads processed ...
900000 reads processed ...
910000 reads processed ...
920000 reads processed ...
930000 reads processed ...
940000 reads processed ...


```
950000 reads processed ...
960000 reads processed ...
970000 reads processed ...
980000 reads processed ...
990000 reads processed ...
1000000 reads processed ...
1010000 reads processed ...
1020000 reads processed ...
1030000 reads processed ...
1040000 reads processed ...
1050000 reads processed ...
1060000 reads processed ...
1070000 reads processed ...
1080000 reads processed ...
1090000 reads processed ...
1100000 reads processed ...
1110000 reads processed ...
1120000 reads processed ...
1130000 reads processed ...
1140000 reads processed ...
1150000 reads processed ...
1160000 reads processed ...
1170000 reads processed ...
1180000 reads processed ...
1190000 reads processed ...
1200000 reads processed ...
1210000 reads processed ...
1220000 reads processed ...
1230000 reads processed ...
1240000 reads processed ...
1250000 reads processed ...
1260000 reads processed ...
1270000 reads processed ...
1280000 reads processed ...
1290000 reads processed ...
1300000 reads processed ...
1310000 reads processed ...
1320000 reads processed ...
1330000 reads processed ...
1340000 reads processed ...
1350000 reads processed ...
1360000 reads processed ...
1370000 reads processed ...
1380000 reads processed ...
1390000 reads processed ...
1400000 reads processed ...
1410000 reads processed ...
1420000 reads processed ...
1430000 reads processed ...
1440000 reads processed ...
1450000 reads processed ...
1460000 reads processed ...
1470000 reads processed ...
1480000 reads processed ...
1490000 reads processed ...
1500000 reads processed ...
1510000 reads processed ...
1520000 reads processed ...
```

```
1530000 reads processed ...
1540000 reads processed ...
1550000 reads processed ...
1560000 reads processed ...
1570000 reads processed ...
1580000 reads processed ...
1590000 reads processed ...
1600000 reads processed ...
1610000 reads processed ...
1620000 reads processed ...
1630000 reads processed ...
1640000 reads processed ...
1650000 reads processed ...
1660000 reads processed ...
1670000 reads processed ...
1680000 reads processed ...
1690000 reads processed ...
1700000 reads processed ...
1710000 reads processed ...
1720000 reads processed ...
1730000 reads processed ...
1740000 reads processed ...
1750000 reads processed ...
1760000 reads processed ...
1770000 reads processed ...
1780000 reads processed ...
1790000 reads processed ...
1800000 reads processed ...
1810000 reads processed ...
1820000 reads processed ...
1830000 reads processed ...
1840000 reads processed ...
1850000 reads processed ...
1860000 reads processed ...
1870000 reads processed ...
1880000 reads processed ...
1890000 reads processed ...
1900000 reads processed ...
1910000 reads processed ...
1920000 reads processed ...
1930000 reads processed ...
1940000 reads processed ...
1950000 reads processed ...
1960000 reads processed ...
1970000 reads processed ...
1980000 reads processed ...
1990000 reads processed ...
2000000 reads processed ...
2010000 reads processed ...
2020000 reads processed ...
2030000 reads processed ...
2040000 reads processed ...
2050000 reads processed ...
2060000 reads processed ...
2070000 reads processed ...
2080000 reads processed ...
2090000 reads processed ...
2100000 reads processed ...
```

```
2110000 reads processed ...
2120000 reads processed ...
2130000 reads processed ...
2140000 reads processed ...
2150000 reads processed ...
2160000 reads processed ...
2170000 reads processed ...
2180000 reads processed ...
2190000 reads processed ...
2200000 reads processed ...
2210000 reads processed ...
2220000 reads processed ...
2230000 reads processed ...
2240000 reads processed ...
2250000 reads processed ...
2260000 reads processed ...
2270000 reads processed ...
2280000 reads processed ...
2290000 reads processed ...
2300000 reads processed ...
2310000 reads processed ...
2320000 reads processed ...
2330000 reads processed ...
2340000 reads processed ...
2350000 reads processed ...
2360000 reads processed ...
2370000 reads processed ...
2380000 reads processed ...
2390000 reads processed ...
2400000 reads processed ...
2410000 reads processed ...
2420000 reads processed ...
2430000 reads processed ...
2440000 reads processed ...
2450000 reads processed ...
2460000 reads processed ...
2470000 reads processed ...
2480000 reads processed ...
2490000 reads processed ...
2500000 reads processed ...
2510000 reads processed ...
2520000 reads processed ...
2530000 reads processed ...
2540000 reads processed ...
2550000 reads processed ...
2560000 reads processed ...
2570000 reads processed ...
2580000 reads processed ...
2590000 reads processed ...
2600000 reads processed ...
2610000 reads processed ...
2620000 reads processed ...
2630000 reads processed ...
2640000 reads processed ...
2650000 reads processed ...
2660000 reads processed ...
2670000 reads processed ...
2680000 reads processed ...
```

```
2690000 reads processed ...
2700000 reads processed ...
2710000 reads processed ...
2720000 reads processed ...
2730000 reads processed ...
2740000 reads processed ...
2750000 reads processed ...
2760000 reads processed ...
2770000 reads processed ...
2780000 reads processed ...
2790000 reads processed ...
2800000 reads processed ...
2810000 reads processed ...
2820000 reads processed ...
2830000 reads processed ...
2840000 reads processed ...
2850000 reads processed ...
2860000 reads processed ...
2870000 reads processed ...
2880000 reads processed ...
2890000 reads processed ...
2900000 reads processed ...
2910000 reads processed ...
2920000 reads processed ...
2930000 reads processed ...
2940000 reads processed ...
2950000 reads processed ...
2960000 reads processed ...
2970000 reads processed ...
2980000 reads processed ...
2990000 reads processed ...
3000000 reads processed ...
3010000 reads processed ...
3020000 reads processed ...
3030000 reads processed ...
3040000 reads processed ...
3050000 reads processed ...
3060000 reads processed ...
3070000 reads processed ...
3080000 reads processed ...
3090000 reads processed ...
3100000 reads processed ...
3110000 reads processed ...
3120000 reads processed ...
3130000 reads processed ...
3140000 reads processed ...
3150000 reads processed ...
3160000 reads processed ...
3170000 reads processed ...
3180000 reads processed ...
3190000 reads processed ...
3200000 reads processed ...
3210000 reads processed ...
3220000 reads processed ...
3230000 reads processed ...
3240000 reads processed ...
3250000 reads processed ...
3260000 reads processed ...
```

```
3270000 reads processed ...
3280000 reads processed ...
3290000 reads processed ...
3300000 reads processed ...
3310000 reads processed ...
3320000 reads processed ...
3330000 reads processed ...
3340000 reads processed ...
3350000 reads processed ...
3360000 reads processed ...
3370000 reads processed ...
3380000 reads processed ...
3390000 reads processed ...
3400000 reads processed ...
3410000 reads processed ...
3420000 reads processed ...
3430000 reads processed ...
3440000 reads processed ...
3450000 reads processed ...
3460000 reads processed ...
3470000 reads processed ...
3480000 reads processed ...
3490000 reads processed ...
3500000 reads processed ...
3510000 reads processed ...
3520000 reads processed ...
3530000 reads processed ...
3540000 reads processed ...
3550000 reads processed ...
3560000 reads processed ...
3570000 reads processed ...
3580000 reads processed ...
3590000 reads processed ...
3600000 reads processed ...
3610000 reads processed ...
3620000 reads processed ...
3630000 reads processed ...
3640000 reads processed ...
3650000 reads processed ...
3660000 reads processed ...
3670000 reads processed ...
3680000 reads processed ...
3690000 reads processed ...
3700000 reads processed ...
3710000 reads processed ...
3720000 reads processed ...
3730000 reads processed ...
3740000 reads processed ...
3750000 reads processed ...
3760000 reads processed ...
3770000 reads processed ...
3780000 reads processed ...
3790000 reads processed ...
3800000 reads processed ...
3810000 reads processed ...
3820000 reads processed ...
3830000 reads processed ...
3840000 reads processed ...
```

```
3850000 reads processed ...
3860000 reads processed ...
3870000 reads processed ...
3880000 reads processed ...
3890000 reads processed ...
3900000 reads processed ...
3910000 reads processed ...
3920000 reads processed ...
3930000 reads processed ...
3940000 reads processed ...
3950000 reads processed ...
3960000 reads processed ...
3970000 reads processed ...
Done.
```

See Also

BioMap | bowtie2 | bowtie2build | cuffgffread | cufflinks | featurecount

Identifying Differentially Expressed Genes from RNA-Seq Data

This example shows how to test RNA-Seq data for differentially expressed genes using a negative binomial model.

Introduction

A typical differential expression analysis of RNA-Seq data consists of normalizing the raw counts and performing statistical tests to reject or accept the null hypothesis that two groups of samples show no significant difference in gene expression. This example shows how to inspect the basic statistics of raw count data, how to determine size factors for count normalization and how to infer the most differentially expressed genes using a negative binomial model.

The dataset for this example comprises of RNA-Seq data obtained in the experiment described by Brooks et al. [1]. The authors investigated the effect of siRNA knock-down of pasilla, a gene known to play an important role in the regulation of splicing in *Drosophila melanogaster*. The dataset consists of 2 biological replicates of the control (untreated) samples and 2 biological replicates of the knock-down (treated) samples.

Inspecting Read Count Tables for Genomic Features

The starting point for this analysis of RNA-Seq data is a count matrix, where the rows correspond to genomic features of interest, the columns correspond to the given samples and the values represent the number of reads mapped to each feature in a given sample.

The included file `pasilla_count_noMM.mat` contains two tables with the count matrices at the gene level and at the exon level for each of the considered samples. You can obtain similar matrices using the function `featurecount`.

```
load pasilla_count_noMM.mat
% preview the table of read counts for genes
geneCountTable(1:10,:)
```

```
ans =
```

```
10x6 table
```

ID	Reference	untreated3	untreated4	treated2	treated3
{'FBgn0000003'}	{'3R'}	0	1	1	2
{'FBgn0000008'}	{'2R'}	142	117	138	132
{'FBgn0000014'}	{'3R'}	20	12	10	19
{'FBgn0000015'}	{'3R'}	2	4	0	1
{'FBgn0000017'}	{'3L'}	6591	5127	4809	6027
{'FBgn0000018'}	{'2L'}	469	530	492	574
{'FBgn0000024'}	{'3R'}	5	6	10	8
{'FBgn0000028'}	{'X' }	0	0	2	1
{'FBgn0000032'}	{'3R'}	1160	1143	1138	1415
{'FBgn0000036'}	{'3R'}	0	0	0	1

Note that when counting is performed without summarization, the individual features (exons in this case) are reported with their metafeature assignment (genes in this case) followed by the start and stop positions.

```
% preview the table of read counts for exons
exonCountTable(1:10,:)
```

```
ans =
```

```
10x6 table
```

ID	Reference	untreated3	untreated4	treated2	treated3
{'FBgn0000003_2648220_2648518' }	{'3R' }	0	0	0	0
{'FBgn0000008_18024938_18025756' }	{'2R' }	0	1	0	0
{'FBgn0000008_18050410_18051199' }	{'2R' }	13	9	14	14
{'FBgn0000008_18052282_18052494' }	{'2R' }	4	2	5	5
{'FBgn0000008_18056749_18058222' }	{'2R' }	32	27	26	26
{'FBgn0000008_18058283_18059490' }	{'2R' }	14	18	29	29
{'FBgn0000008_18059587_18059757' }	{'2R' }	1	4	3	3
{'FBgn0000008_18059821_18059938' }	{'2R' }	0	0	2	2
{'FBgn0000015_12758093_12760298' }	{'3R' }	1	2	0	0
{'FBgn0000017_16615461_16618374' }	{'3L' }	1807	1572	1557	1557

You can annotate and group the samples by creating a logical vector as follows:

```
samples = geneCountTable(:,3:end).Properties.VariableNames;
untreated = strcmp(samples,'untreated',length('untreated'))
treated = strcmp(samples,'treated',length('treated'))
```

```
untreated =
```

```
1x4 logical array
```

```
1 1 0 0
```

```
treated =
```

```
1x4 logical array
```

```
0 0 1 1
```

Plotting the Feature Assignments

The included file also contains a table `geneSummaryTable` with the summary of assigned and unassigned SAM entries. You can plot the basic distribution of the counting results by considering the number of reads that are assigned to the given genomic features (exons or genes for this example), as well as the number of reads that are unassigned (i.e. not overlapping any feature) or ambiguous (i.e. overlapping multiple features).

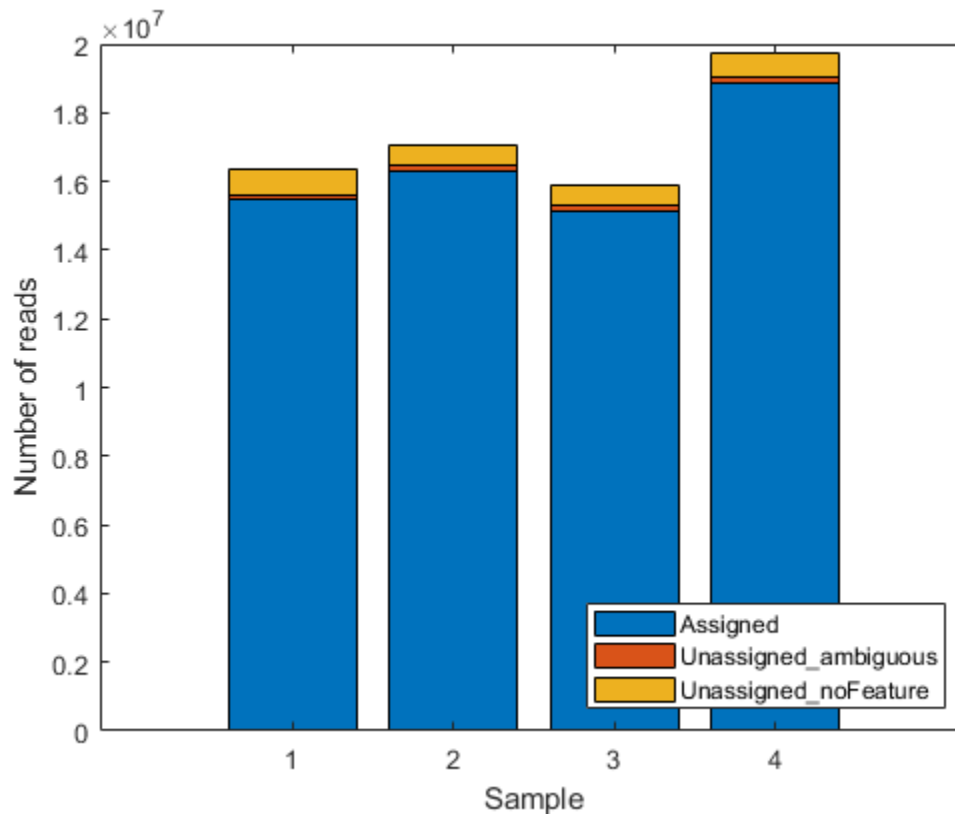
```
st = geneSummaryTable({'Assigned','Unassigned_ambiguous','Unassigned_noFeature'},:)
bar(table2array(st),'stacked');
legend(st.Properties.RowNames,'Interpreter','none','Location','southeast');
xlabel('Sample')
ylabel('Number of reads')
```



```
st =
```

```
3x4 table
```

	untreated3	untreated4	treated2	treated3
Assigned	1.5457e+07	1.6302e+07	1.5146e+07	1.8856e+07
Unassigned_ambiguous	1.5708e+05	1.6882e+05	1.6194e+05	1.9977e+05
Unassigned_noFeature	7.5455e+05	5.8309e+05	5.8756e+05	6.8356e+05



Note that a small fraction of the alignment records in the SAM files is not reported in the summary table. You can notice this in the difference between the total number of records in a SAM file and the total number of records processed during the counting procedure for that same SAM file. These unreported records correspond to the records mapped to reference sequences that are not annotated in the GTF file and therefore are not processed in the counting procedure. If the gene models account for all the reference sequences used during the read mapping step, then all records are reported in one of the categories of the summary table.

```
geneSummaryTable{'TotalEntries', :} - sum(geneSummaryTable{2:end, :})
```

```
ans =
```

89516 95885 98207 104629

Plotting Read Coverage Across a Given Chromosome

When read counting is performed without summarization using the function `featurecount`, the default IDs are composed by the attribute or metafeature (by default, `gene_id`) followed by the start and the stop positions of the feature (by default, `exon`). You can use the exon start positions to plot the read coverage across any chromosome in consideration, for example chromosome arm 2L.

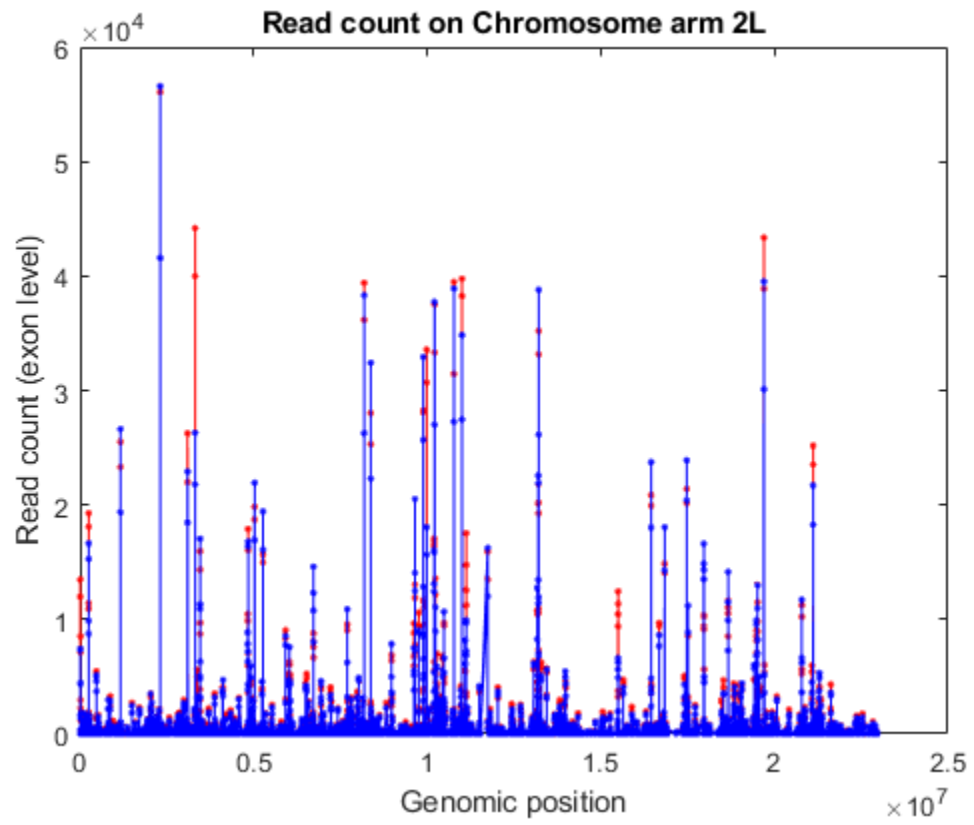
```
% consider chromosome arm 2L
chr2L = strcmp(exonCountTable.Reference, '2L');
exonCount = exonCountTable{:,3:end};

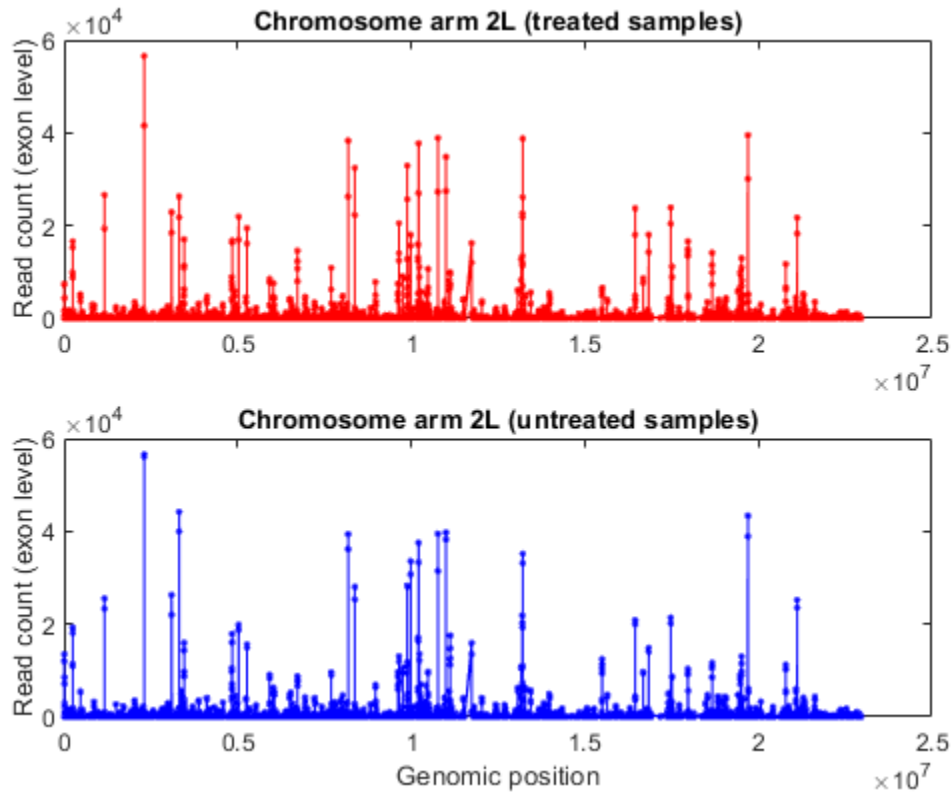
% retrieve exon start positions
exonStart = regexp(exonCountTable{chr2L,1}, '_(\d+)', 'tokens');
exonStart = [exonStart{:}];
exonStart = cellfun(@str2num, [exonStart{:}]);

% sort exon by start positions
[~,idx] = sort(exonStart);

% plot read coverage along the genomic coordinates
figure;
plot(exonStart(idx),exonCount(idx,treated),'.-r',...
     exonStart(idx),exonCount(idx,untreated),'.-b');
xlabel('Genomic position');
ylabel('Read count (exon level)');
title('Read count on Chromosome arm 2L');

% plot read coverage for each group separately
figure;
subplot(2,1,1);
plot(exonStart(idx),exonCount(idx,untreated),'.-r');
ylabel('Read count (exon level)');
title('Chromosome arm 2L (treated samples)');
subplot(2,1,2);
plot(exonStart(idx),exonCount(idx,treated),'.-b');
ylabel('Read count (exon level)');
xlabel('Genomic position');
title('Chromosome arm 2L (untreated samples)');
```





Alternatively, you can plot the read coverage considering the starting position of each gene in a given chromosome. The file `pasilla_geneLength.mat` contains a table with the start and stop position of each gene in the corresponding gene annotation file.

```
% load gene start and stop position information
load pasilla_geneLength
geneLength(1:10,:)
```

ans =

10x5 table

ID	Name	Reference	Start	Stop
{'FBgn0037213'}	{'CG12581'}	3R	380	10200
{'FBgn0000500'}	{'Dsk' }	3R	15388	16170
{'FBgn0053294'}	{'CR33294'}	3R	17136	21871
{'FBgn0037215'}	{'CG12582'}	3R	23029	30295
{'FBgn0037217'}	{'CG14636'}	3R	30207	41033
{'FBgn0037218'}	{'aux' }	3R	37505	53244
{'FBgn0051516'}	{'CG31516'}	3R	44179	45852
{'FBgn0261436'}	{'DhpD' }	3R	53106	54971
{'FBgn0037220'}	{'CG14641'}	3R	56475	58077
{'FBgn0015331'}	{'abs' }	3R	58765	60763

```

% consider chromosome 3 ('Reference' is a categorical variable)
chr3 = (geneLength.Reference == '3L') | (geneLength.Reference == '3R');
sum(chr3)

% consider the counts for genes in chromosome 3
counts = geneCountTable(:,3:end);
[~,j,k] = intersect(geneCountTable(:, 'ID'),geneLength{chr3,'ID'});
gstart = geneLength{k, 'Start'};
gcounts = counts(j,:);

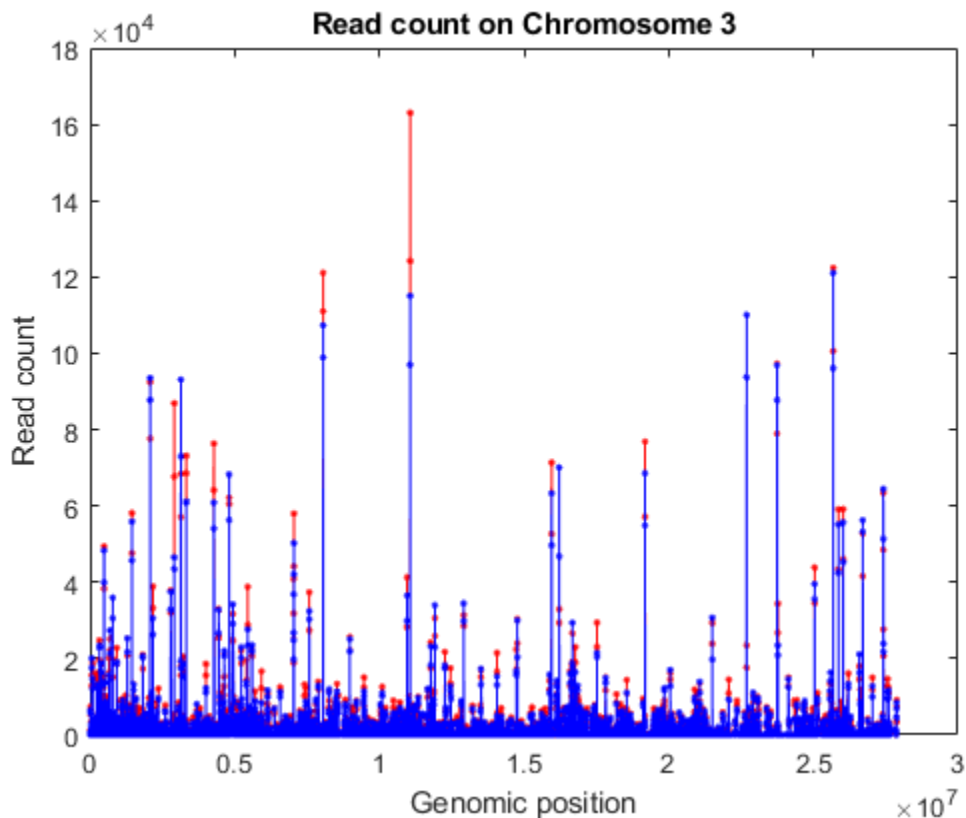
% sort according to ascending start position
 [~,idx] = sort(gstart);

% plot read coverage by genomic position
figure;
plot(gstart(idx), gcounts(idx,treated),'.-r',...
     gstart(idx), gcounts(idx,untreated),'.-b');
xlabel('Genomic position')
ylabel('Read count');
title('Read count on Chromosome 3');

ans =

```

```
6360
```



Normalizing Read Counts

The read count in RNA-Seq data has been found to be linearly related to the abundance of transcripts [2]. However, the read count for a given gene depends not only on the expression level of the gene, but also on the total number of reads sequenced and the length of the gene transcript. Therefore, in order to infer the expression level of a gene from the read count, we need to account for the sequencing depth and the gene transcript length. One common technique to normalize the read count is to use the RPKM (Read Per Kilobase Mapped) values, where the read count is normalized by the total number of reads yielded (in millions) and the length of each transcript (in kilobases). This normalization technique, however, is not always effective since few, very highly expressed genes can dominate the total lane count and skew the expression analysis.

A better normalization technique consists of computing the effective library size by considering a size factor for each sample. By dividing each sample's counts by the corresponding size factors, we bring all the count values to a common scale, making them comparable. Intuitively, if sample A is sequenced N times deeper than sample B, the read counts of non-differentially expressed genes are expected to be on average N times higher in sample A than in sample B, even if there is no difference in expression.

To estimate the size factors, take the median of the ratios of observed counts to those of a pseudo-reference sample, whose counts can be obtained by considering the geometric mean of each gene across all samples [3]. Then, to transform the observed counts to a common scale, divide the observed counts in each sample by the corresponding size factor.

```
% estimate pseudo-reference with geometric mean row by row
pseudoRefSample = geomean(counts,2);
nz = pseudoRefSample > 0;
ratios = bsxfun(@rdivide,counts(nz,:),pseudoRefSample(nz));
sizeFactors = median(ratios,1)
```

```
sizeFactors =
```

```
    0.9374    0.9725    0.9388    1.1789
```

```
% transform to common scale
```

```
normCounts = bsxfun(@rdivide,counts,sizeFactors);
normCounts(1:10,:)
```

```
ans =
```

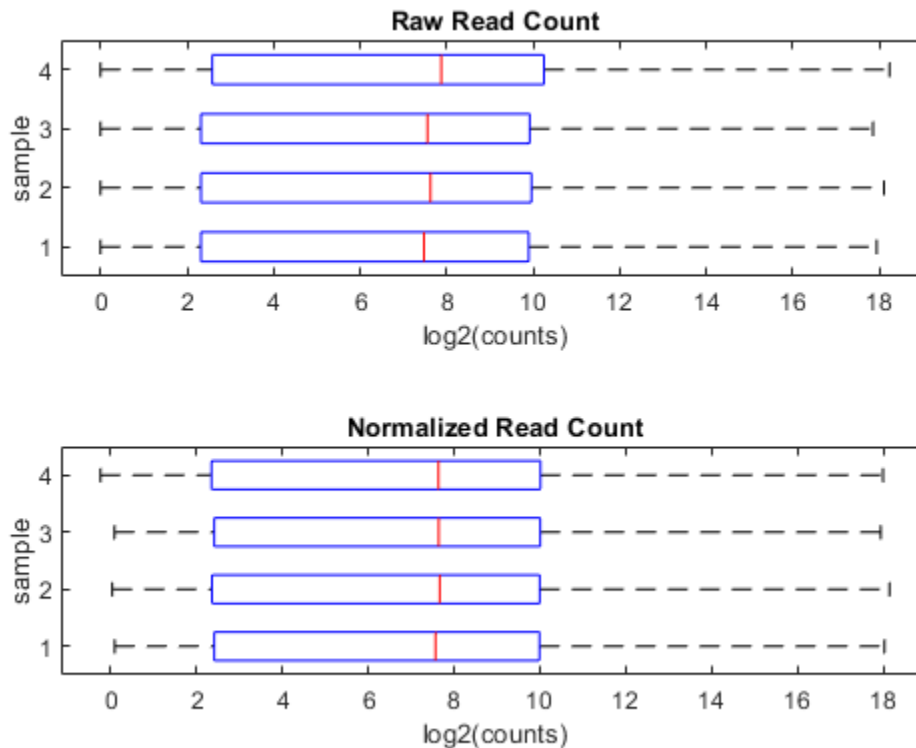
```
1.0e+03 *
      0      0.0010      0.0011      0.0017
  0.1515  0.1203  0.1470  0.1120
  0.0213  0.0123  0.0107  0.0161
  0.0021  0.0041         0  0.0008
  7.0315  5.2721  5.1225  5.1124
  0.5003  0.5450  0.5241  0.4869
  0.0053  0.0062  0.0107  0.0068
      0      0      0.0021  0.0008
  1.2375  1.1753  1.2122  1.2003
      0      0          0  0.0008
```

You can appreciate the effect of this normalization by using the function `boxplot` to represent statistical measures such as median, quartiles, minimum and maximum.

figure;

```
subplot(2,1,1)
maboxplot(log2(counts), 'title', 'Raw Read Count', 'orientation', 'horizontal')
ylabel('sample')
xlabel('log2(counts)')

subplot(2,1,2)
maboxplot(log2(normCounts), 'title', 'Normalized Read Count', 'orientation', 'horizontal')
ylabel('sample')
xlabel('log2(counts)')
```



Computing Mean, Dispersion and Fold Change

In order to better characterize the data, we consider the mean and the dispersion of the normalized counts. The variance of read counts is given by the sum of two terms: the variation across samples (raw variance) and the uncertainty of measuring the expression by counting reads (shot noise or Poisson). The raw variance term dominates for highly expressed genes, whereas the shot noise dominates for lowly expressed genes. You can plot the empirical dispersion values against the mean of the normalized counts in a log scale as shown below.

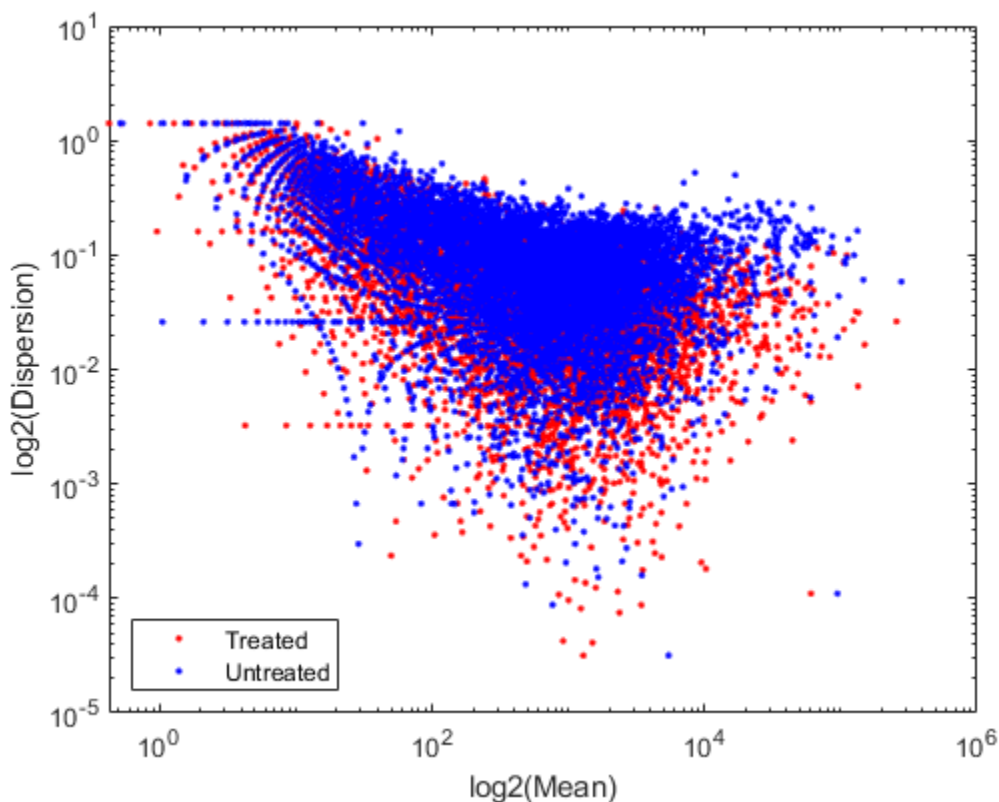
```
% consider the mean
meanTreated = mean(normCounts(:,treated),2);
meanUntreated = mean(normCounts(:,untreated),2);
```

```

% consider the dispersion
dispTreated = std(normCounts(:,treated),0,2) ./ meanTreated;
dispUntreated = std(normCounts(:,untreated),0,2) ./ meanUntreated;

% plot on a log-log scale
figure;
loglog(meanTreated,dispTreated,'r. ');
hold on;
loglog(meanUntreated,dispUntreated,'b. ');
xlabel('log2(Mean)');
ylabel('log2(Dispersion)');
legend('Treated','Untreated','Location','southwest');

```



Given the small number of replicates, it is not surprising to expect that the dispersion values scatter with some variance around the true value. Some of this variance reflects sampling variance and some reflects the true variability among the gene expressions of the samples.

You can look at the difference of the gene expression among two conditions, by calculating the fold change (FC) for each gene, i.e. the ratio between the counts in the treated group over the counts in the untreated group. Generally these ratios are considered in the log₂ scale, so that any change is symmetric with respect to zero (e.g. a ratio of 1/2 or 2/1 corresponds to -1 or +1 in the log scale).

```

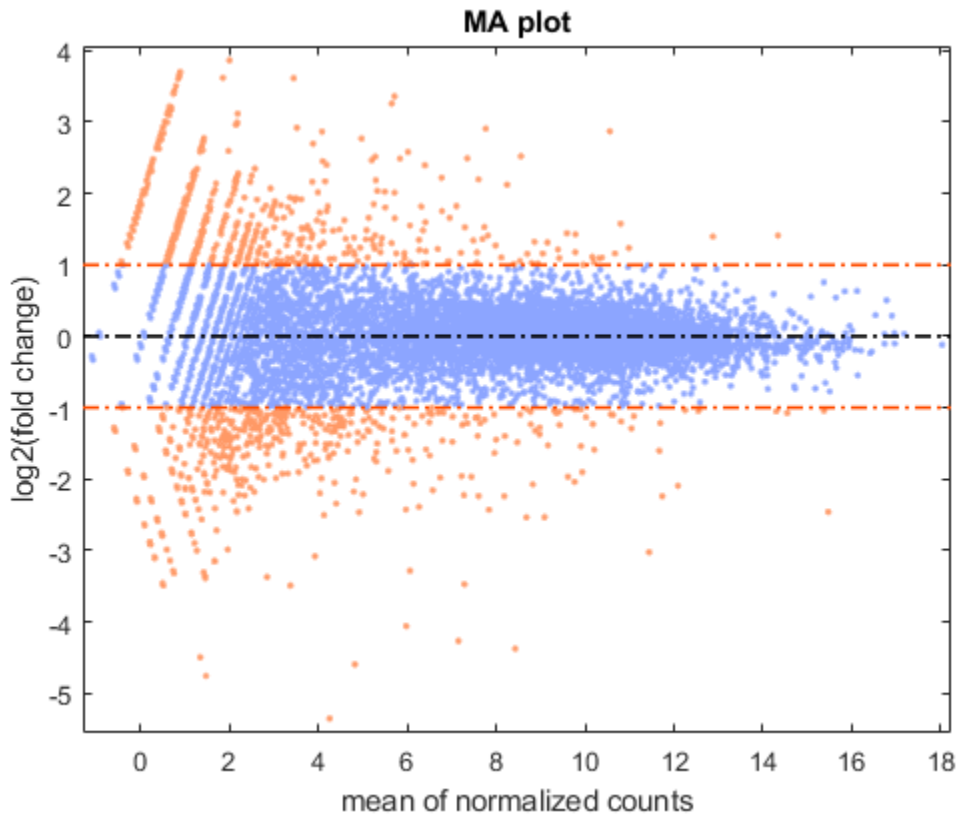
% compute the mean and the log2FC
meanBase = (meanTreated + meanUntreated) / 2;
foldChange = meanTreated ./ meanUntreated;
log2FC = log2(foldChange);

```



```
% plot mean vs. fold change (MA plot)
mairplot(meanTreated, meanUntreated, 'Type', 'MA', 'Plotonly', true);
set(get(gca, 'Xlabel'), 'String', 'mean of normalized counts')
set(get(gca, 'Ylabel'), 'String', 'log2(fold change)')
```

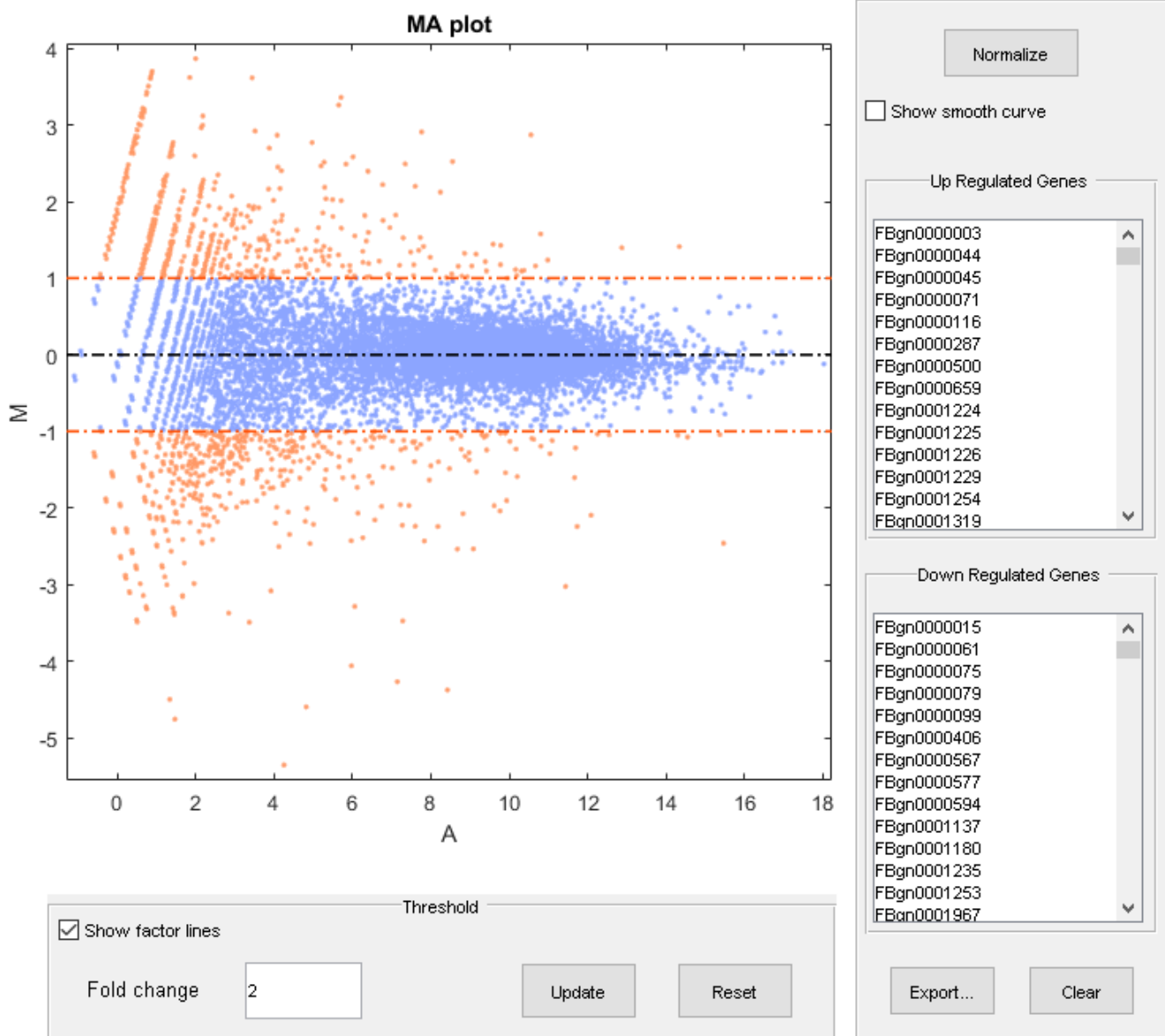
Warning: Zero values are ignored



It is possible to annotate the values in the plot with the corresponding gene names, interactively select genes, and export gene lists to the workspace by calling the `mairplot` function as illustrated below:

```
mairplot(meanTreated, meanUntreated, 'Labels', geneCountTable.ID, 'Type', 'MA');
```

Warning: Zero values are ignored



It is convenient to store the information about the mean value and fold change for each gene in a table. You can then access information about a given gene or a group of genes satisfying specific criteria by indexing the table by gene names.

```
% create table with statistics about each gene
geneTable = table(meanBase,meanTreated,meanUntreated, foldChange, log2FC) ;
geneTable.Properties.RowNames = geneCountTable.ID;

% summary
summary(geneTable)
```

Variables:

meanBase: 11609x1 double

Values:

```

Min      0.21206
Median   201.24
Max      2.6789e+05
    
```

meanTreated: 11609x1 double

Values:

```

Min      0
Median   201.54
Max      2.5676e+05
    
```

meanUntreated: 11609x1 double

Values:

```

Min      0
Median   199.44
Max      2.7903e+05
    
```

foldChange: 11609x1 double

Values:

```

Min      0
Median   0.99903
Max      Inf
    
```

log2FC: 11609x1 double

Values:

```

Min      -Inf
Median   -0.001406
Max      Inf
    
```

```

% preview
geneTable(1:10,:)
    
```

ans =

10x5 table

	meanBase	meanTreated	meanUntreated	foldChange	log2FC
FBgn0000003	0.9475	1.3808	0.51415	2.6857	1.4253
FBgn0000008	132.69	129.48	135.9	0.95277	-0.069799
FBgn0000014	15.111	13.384	16.838	0.79488	-0.33119

FBgn0000015	1.7738	0.42413	3.1234	0.13579	-2.8806
FBgn0000017	5634.6	5117.4	6151.8	0.83186	-0.26559
FBgn0000018	514.08	505.48	522.67	0.96711	-0.048243
FBgn0000024	7.2354	8.7189	5.752	1.5158	0.60009
FBgn0000028	0.74465	1.4893	0	Inf	Inf
FBgn0000032	1206.3	1206.2	1206.4	0.99983	-0.00025093
FBgn0000036	0.21206	0.42413	0	Inf	Inf

`% access information about a specific gene`

```
myGene = 'FBgn0261570';
geneTable(myGene, :)
geneTable(myGene, {'meanBase', 'log2FC'})
```

`% access information about a given gene list`

```
myGeneSet = {'FBgn0261570', 'FBgn0261573', 'FBgn0261575', 'FBgn0261560'};
geneTable(myGeneSet, :)
```

ans =

1x5 table

	meanBase	meanTreated	meanUntreated	foldChange	log2FC
FBgn0261570	4435.5	4939.1	3931.8	1.2562	0.32907

ans =

1x2 table

	meanBase	log2FC
FBgn0261570	4435.5	0.32907

ans =

4x5 table

	meanBase	meanTreated	meanUntreated	foldChange	log2FC
FBgn0261570	4435.5	4939.1	3931.8	1.2562	0.32907
FBgn0261573	2936.9	2954.8	2919.1	1.0122	0.01753
FBgn0261575	4.3776	5.6318	3.1234	1.8031	0.85047
FBgn0261560	2041.1	1494.3	2588	0.57738	-0.7924

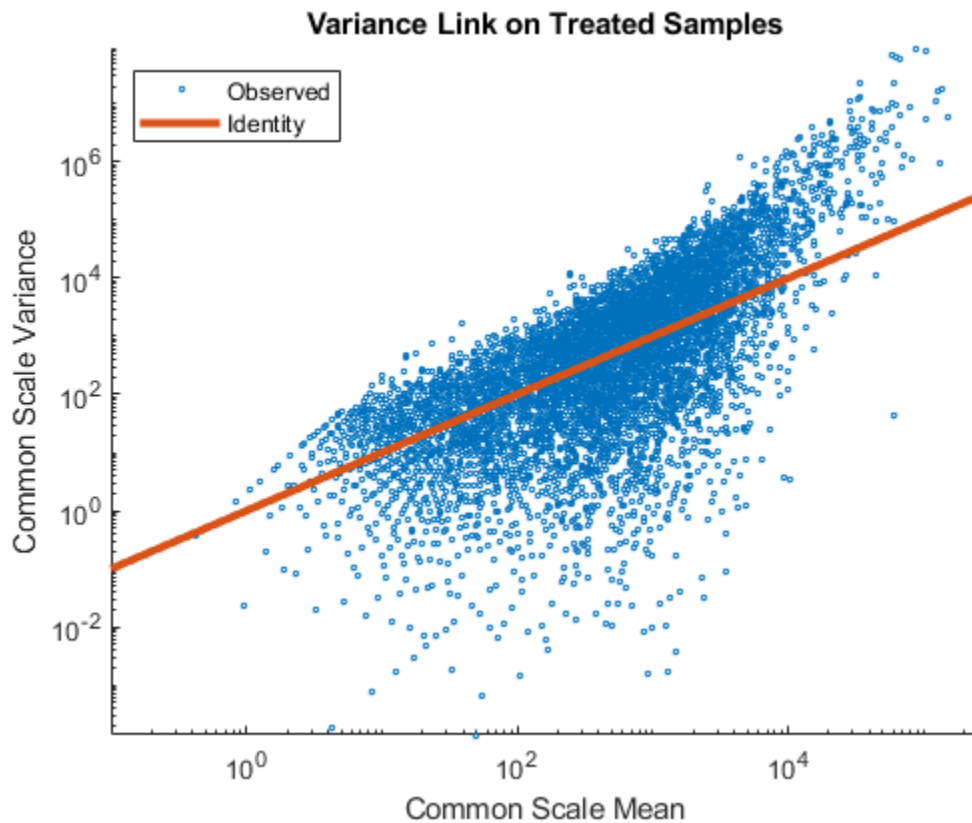
Inferring Differential Expression with a Negative Binomial Model

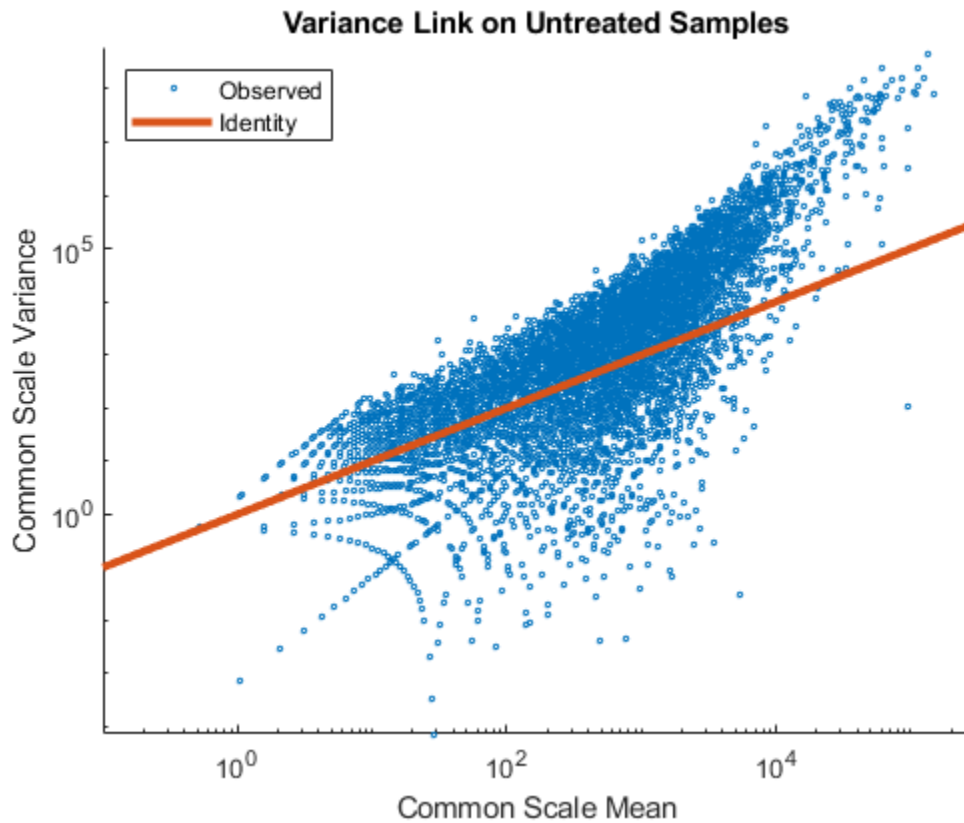
Determining whether the gene expressions in two conditions are statistically different consists of rejecting the null hypothesis that the two data samples come from distributions with equal means. This analysis assumes the read counts are modeled according to a negative binomial distribution (as

proposed in [3]). The function `nbintest` performs this type of hypothesis testing with three possible options to specify the type of linkage between the variance and the mean.

By specifying the link between variance and mean as an identity, we assume the variance is equal to the mean, and the counts are modeled by the Poisson distribution [4].

```
tIdentity = nbintest(counts(:,treated),counts(:,untreated),'VarianceLink','Identity');  
h = plotVarianceLink(tIdentity);  
  
% set custom title  
h(1).Title.String = 'Variance Link on Treated Samples';  
h(2).Title.String = 'Variance Link on Untreated Samples';
```

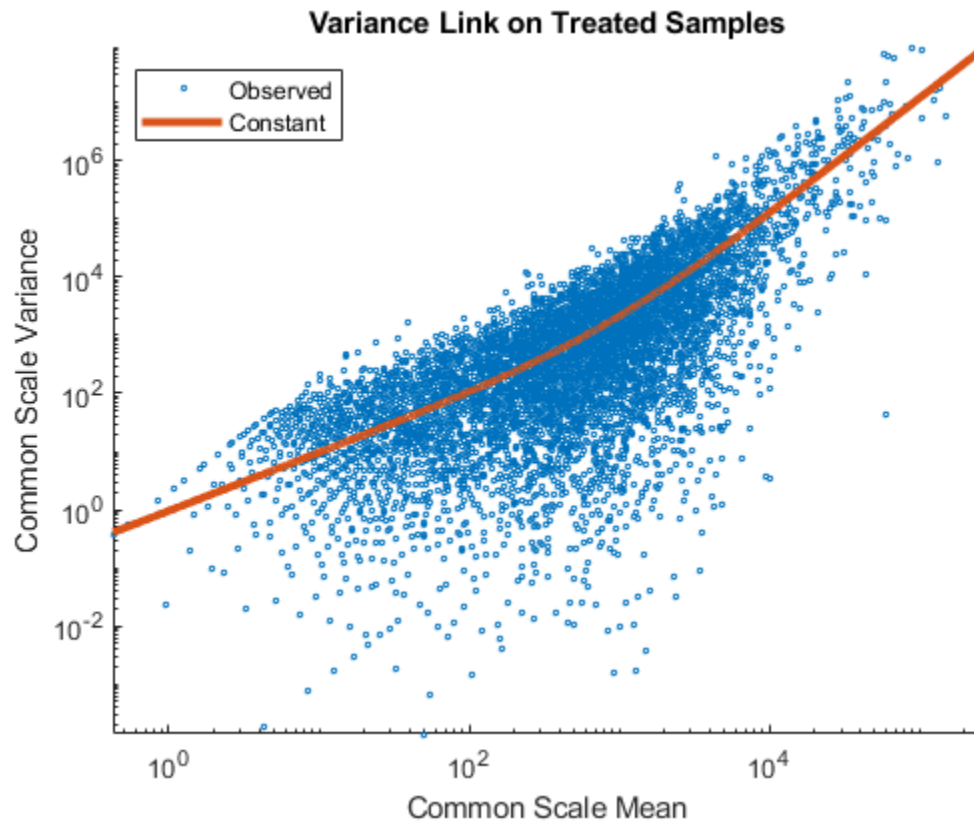


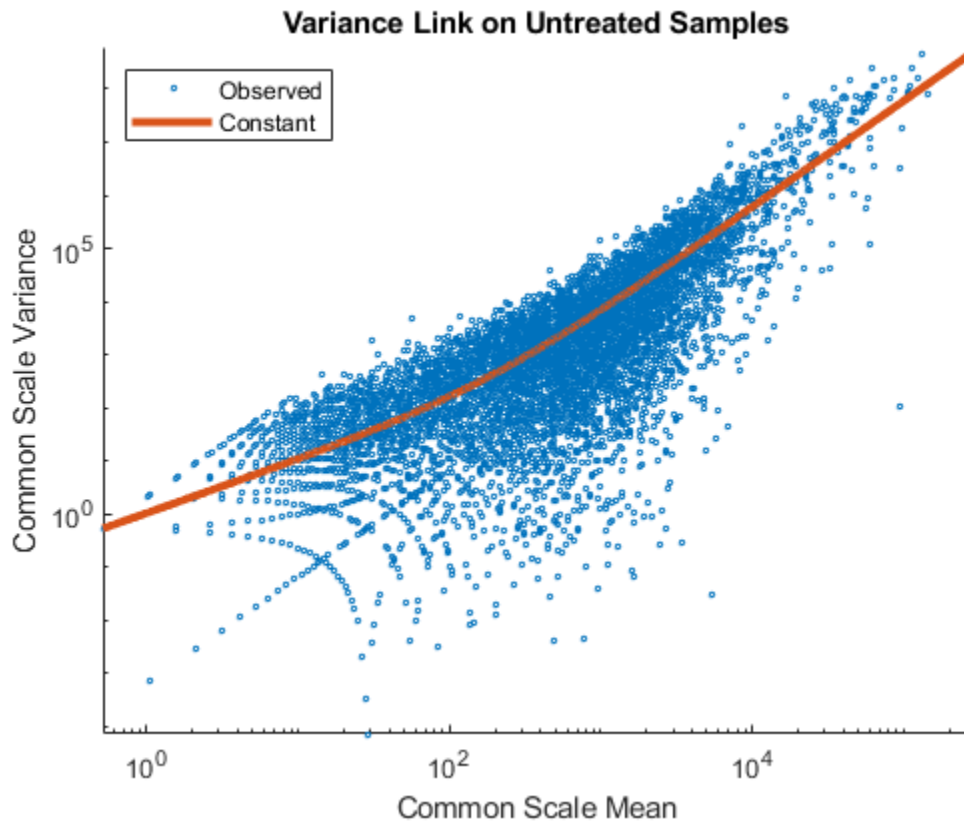


Alternatively, by specifying the variance as the sum of the shot noise term (i.e. mean) and a constant multiplied by the squared mean, the counts are modeled according to a distribution described in [5]. The constant term is estimated using all the rows in the data.

```
tConstant = nbintest(counts(:,treated),counts(:,untreated),'VarianceLink','Constant');  
h = plotVarianceLink(tConstant);
```

```
% set custom title  
h(1).Title.String = 'Variance Link on Treated Samples';  
h(2).Title.String = 'Variance Link on Untreated Samples';
```

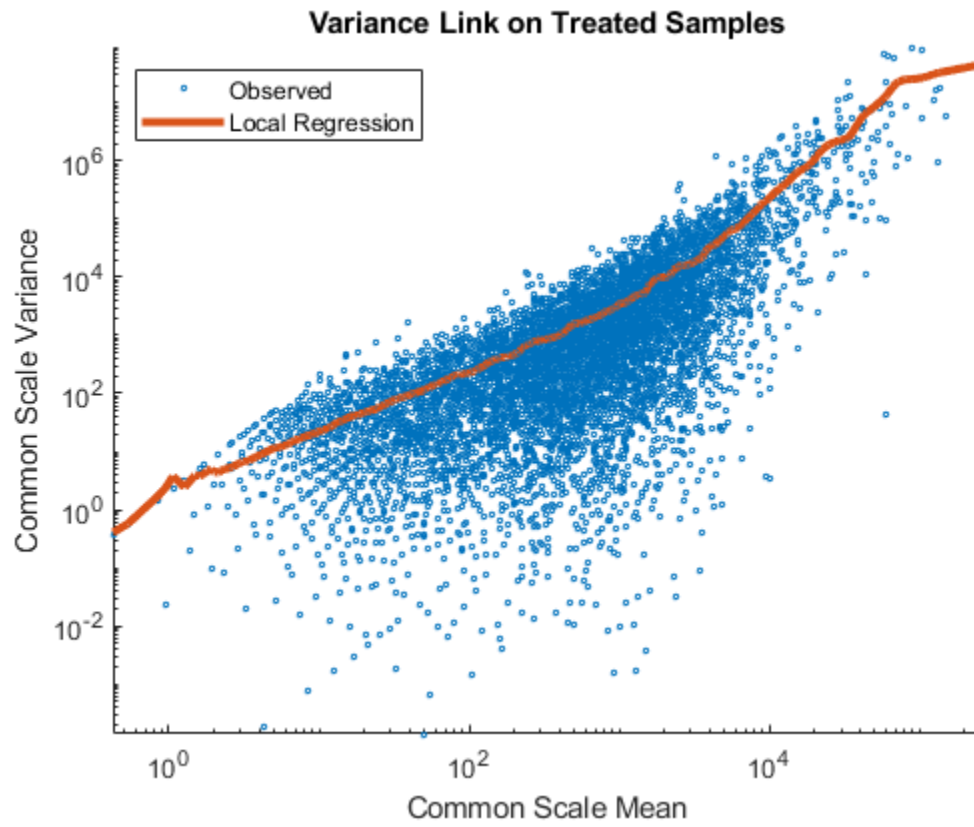


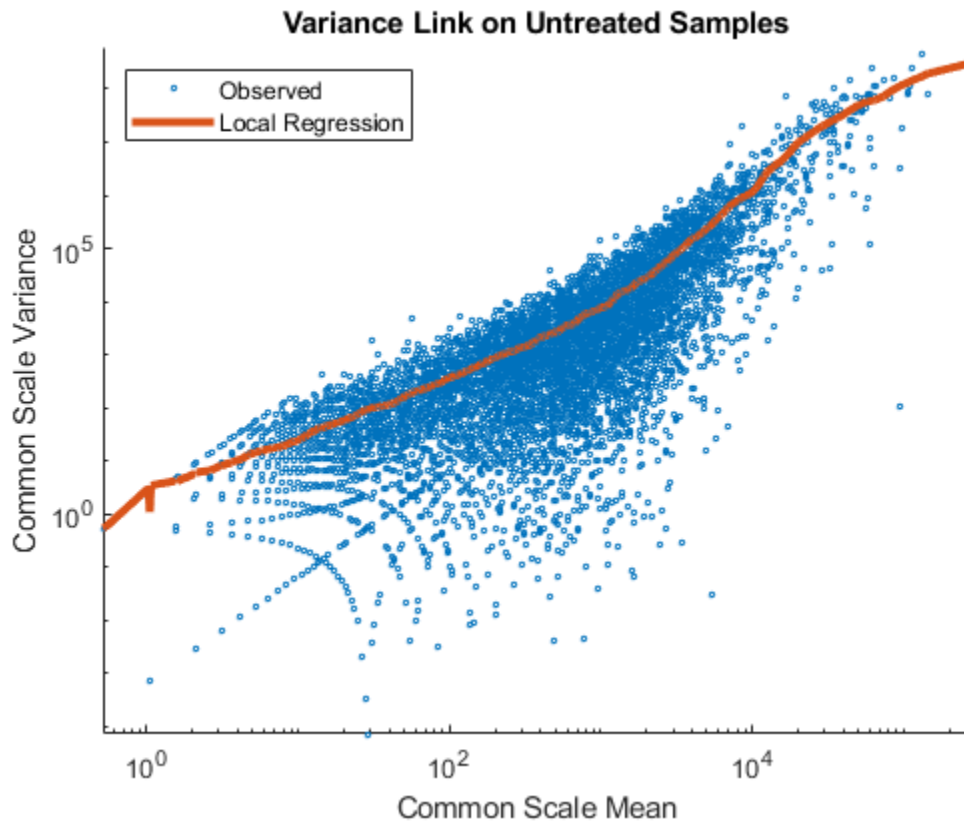


Finally, by considering the variance as the sum of the shot noise term (i.e. mean) and a locally regressed non-parametric smooth function of the mean, the counts are modeled according to the distribution proposed in [3].

```
tLocal = nbintest(counts(:,treated),counts(:,untreated),'VarianceLink','LocalRegression');
h = plotVarianceLink(tLocal);

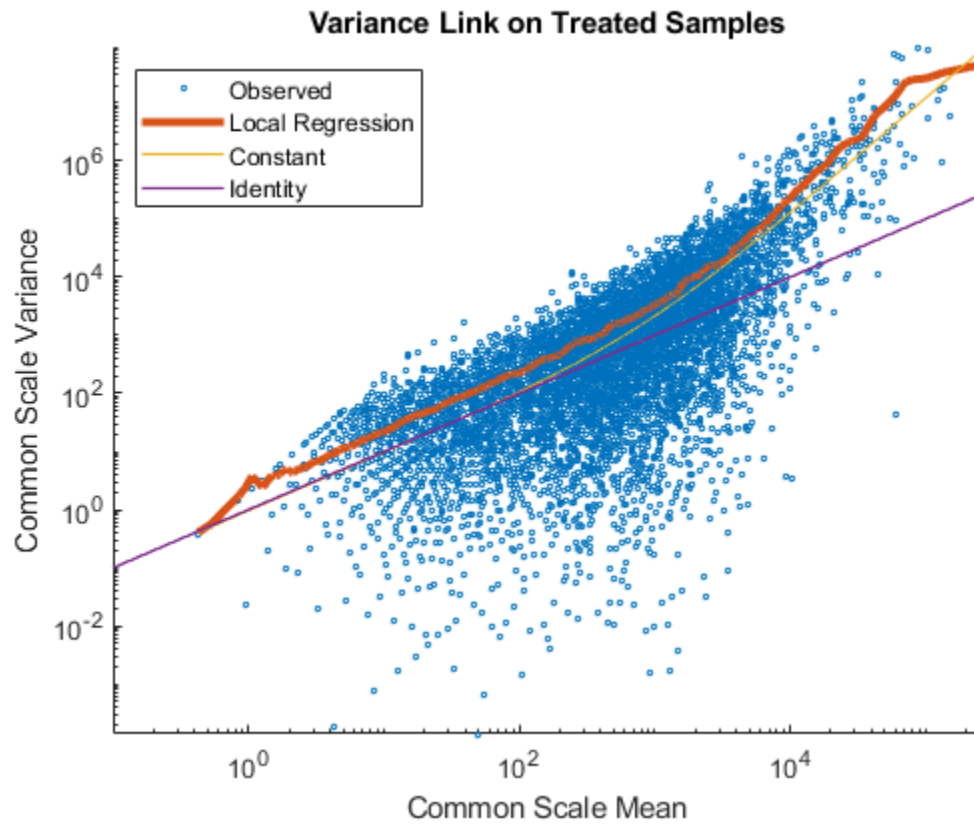
% set custom title
h(1).Title.String = 'Variance Link on Treated Samples';
h(2).Title.String = 'Variance Link on Untreated Samples';
```

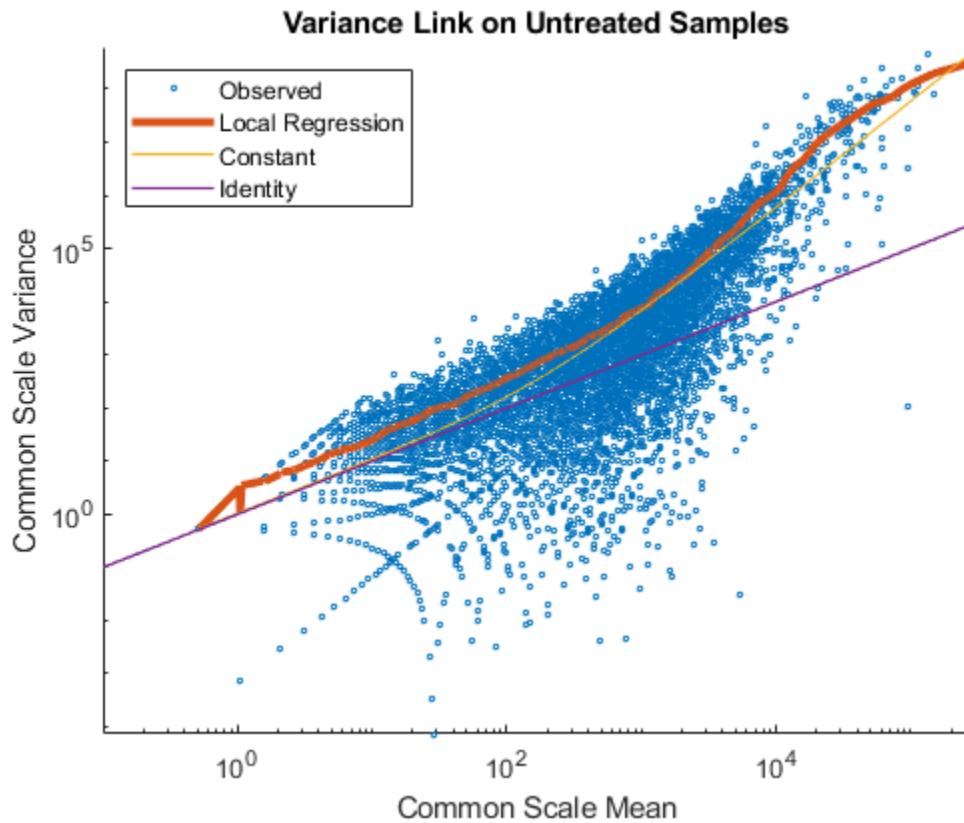





In order to evaluate which fit is the best for the data in consideration, you can compare the fitting curves in a single plot, as shown below.

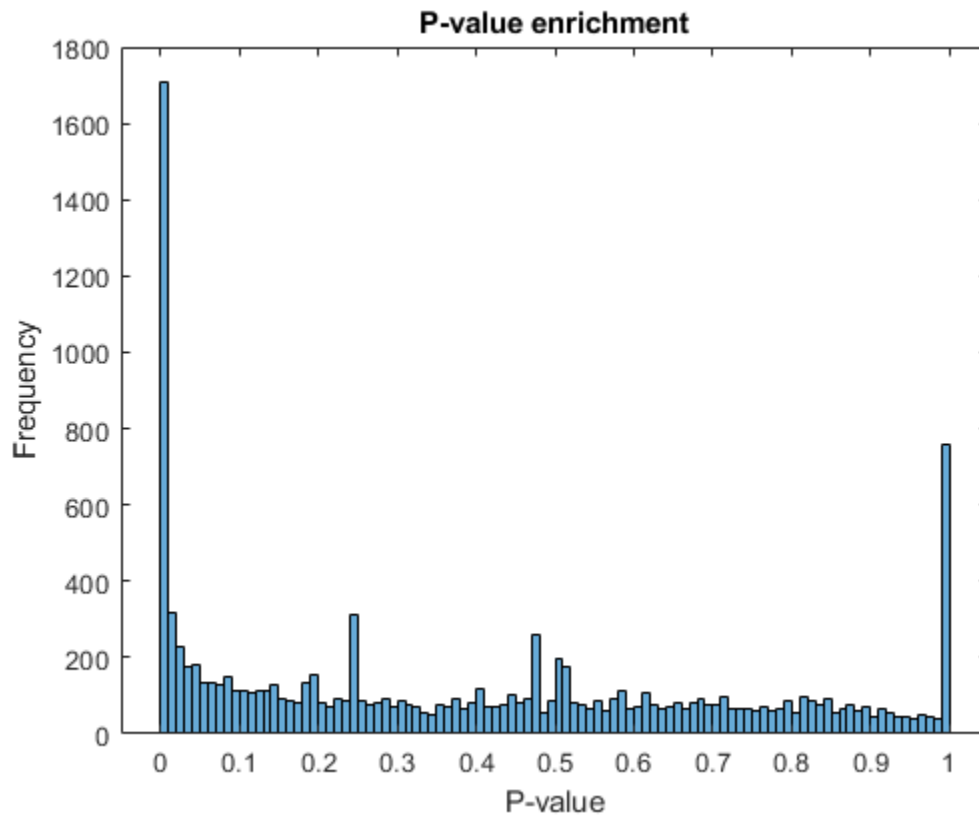
```
h = plotVarianceLink(tLocal, 'compare', true);  
  
% set custom title  
h(1).Title.String = 'Variance Link on Treated Samples';  
h(2).Title.String = 'Variance Link on Untreated Samples';
```





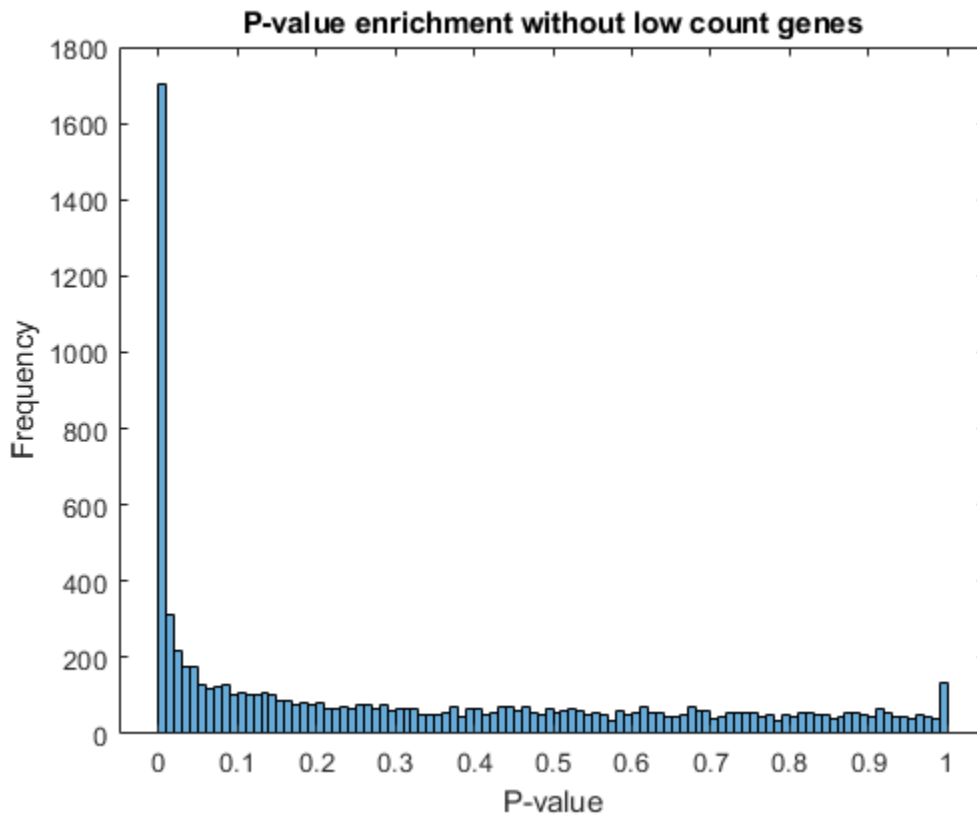
The output of `nbintest` includes a vector of P-values. A P-value indicates the probability that a change in expression as strong as the one observed (or even stronger) would occur under the null hypothesis, i.e. the conditions have no effect on gene expression. In the histogram of the P-values we observe an enrichment of low values (due to differentially expressed genes), whereas other values are uniformly spread (due to non-differentially expressed genes). The enrichment of values equal to 1 are due to genes with very low counts.

```
figure;
histogram(tLocal.pValue,100)
xlabel('P-value')
ylabel('Frequency')
title('P-value enrichment')
```



Filter out those genes with relatively low count to observe a more uniform spread of non-significant P-values across the range (0,1]. Note that this does not affect the distribution of significant P-values.

```
lowCountThreshold = 10;  
lowCountGenes = all(counts < lowCountThreshold, 2);  
histogram(tLocal.pValue(~lowCountGenes),100)  
xlabel('P-value')  
ylabel('Frequency')  
title('P-value enrichment without low count genes')
```



Multiple Testing and Adjusted P-values

Thresholding P-values to determine what fold changes are more significant than others is not appropriate for this type of data analysis, due to the multiple testing problem. While performing a large number of simultaneous tests, the probability of getting a significant result simply due to chance increases with the number of tests. In order to account for multiple testing, perform a correction (or adjustment) of the P-values so that the probability of observing at least one significant result due to chance remains below the desired significance level.

The Benjamini-Hochberg (BH) adjustment [6] is a statistical method that provides an adjusted P-value answering the following question: what would be the fraction of false positives if all the genes with adjusted P-values below a given threshold were considered significant? Set a threshold of 0.1 for the adjusted P-values, equivalent to consider a 10% false positives as acceptable, and identify the genes that are significantly expressed by considering all the genes with adjusted P-values below this threshold.

```
% compute the adjusted P-values (BH correction)
padj = mafdr(tLocal.pValue, 'BHfDR', true);

% add to the existing table
geneTable.pvalue = tLocal.pValue;
geneTable.padj = padj;

% create a table with significant genes
sig = geneTable.padj < 0.1;
geneTableSig = geneTable(sig, :);
```

```
geneTableSig = sortrows(geneTableSig, 'padj');
numberSigGenes = size(geneTableSig,1)
```

```
numberSigGenes =
```

```
1904
```

Identifying the Most Up-regulated and Down-regulated Genes

You can now identify the most up-regulated or down-regulated genes by considering an absolute fold change above a chosen cutoff. For example, a cutoff of 1 in log₂ scale yields the list of genes that are up-regulated with a 2 fold change.

```
% find up-regulated genes
up = geneTableSig.log2FC > 1;
upGenes = sortrows(geneTableSig(up,:), 'log2FC', 'descend');
numberSigGenesUp = sum(up)

% display the top 10 up-regulated genes
top10GenesUp = upGenes(1:10,:)

% find down-regulated genes
down = geneTableSig.log2FC < -1;
downGenes = sortrows(geneTableSig(down,:), 'log2FC', 'ascend');
numberSigGenesDown = sum(down)

% find top 10 down-regulated genes
top10GenesDown = downGenes(1:10,)
```

```
numberSigGenesUp =
```

```
129
```

```
top10GenesUp =
```

```
10x7 table
```

	meanBase	meanTreated	meanUntreated	foldChange	log2FC	pvalue
FBgn0030173	3.3979	6.7957	0	Inf	Inf	0.0063115
FBgn0036822	3.1364	6.2729	0	Inf	Inf	0.012203
FBgn0052548	8.158	15.269	1.0476	14.575	3.8654	0.00016945
FBgn0050495	6.8315	12.635	1.0283	12.287	3.6191	0.0018945
FBgn0063667	20.573	38.042	3.1042	12.255	3.6153	8.5037e-08
FBgn0033764	91.969	167.61	16.324	10.268	3.3601	1.8345e-21
FBgn0037290	85.845	155.46	16.228	9.5801	3.26	3.5583e-23
FBgn0033733	7.4634	13.384	1.5424	8.6773	3.1172	0.0027276
FBgn0037191	7.1766	12.753	1.6003	7.9694	2.9945	0.0047803
FBgn0033943	6.95	12.319	1.581	7.7921	2.962	0.0053633

```
numberSigGenesDown =
```

181

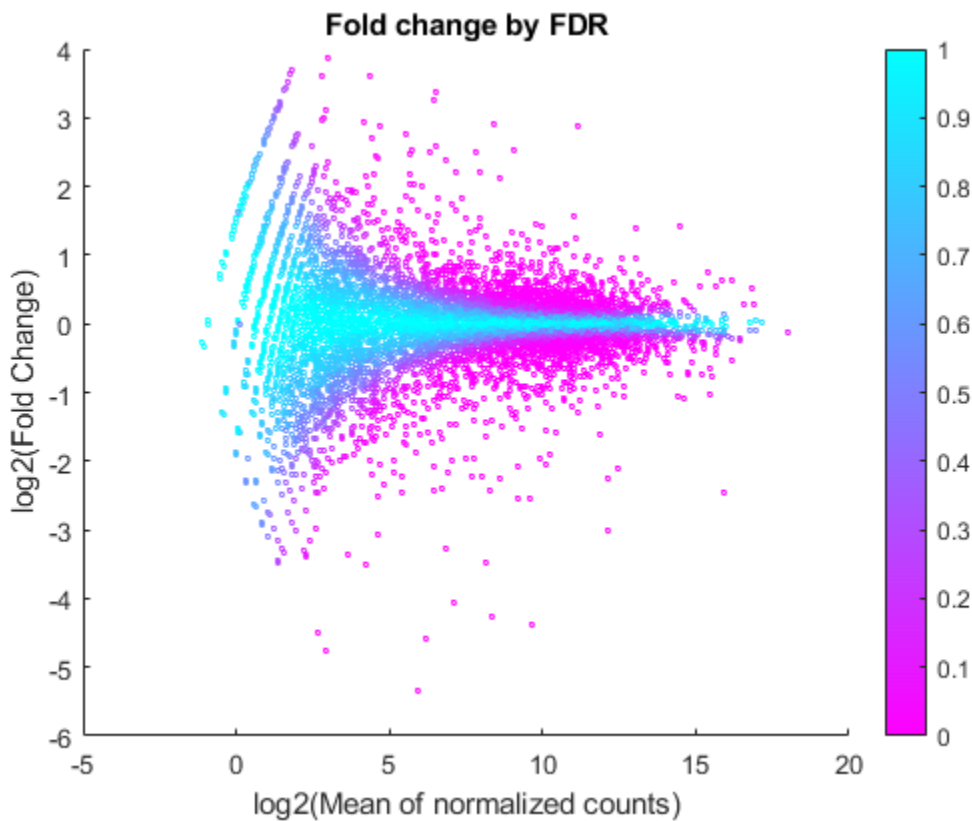
top10GenesDown =

10x7 table

	meanBase	meanTreated	meanUntreated	foldChange	log2FC	pvalue
FBgn0053498	15.469	0	30.938	0	-Inf	9.8404e-3
FBgn0259236	6.8092	0	13.618	0	-Inf	1.5526e-0
FBgn0052500	4.3703	0	8.7405	0	-Inf	0.0006678
FBgn0039331	3.6954	0	7.3908	0	-Inf	0.001955
FBgn0040697	3.419	0	6.8381	0	-Inf	0.002733
FBgn0034972	2.9145	0	5.8291	0	-Inf	0.006850
FBgn0040967	2.6382	0	5.2764	0	-Inf	0.009603
FBgn0031923	2.3715	0	4.7429	0	-Inf	0.01610
FBgn0085359	62.473	2.9786	121.97	0.024421	-5.3557	5.5813e-3
FBgn0004854	7.4674	0.53259	14.402	0.03698	-4.7571	8.1587e-0

A good visualization of the gene expressions and their significance is given by plotting the fold change versus the mean in log scale and coloring the data points according to the adjusted P-values.

```
figure
scatter(log2(geneTable.meanBase), geneTable.log2FC, 3, geneTable.padj, 'o')
colormap(flipud(cool(256)))
colorbar;
ylabel('log2(Fold Change)')
xlabel('log2(Mean of normalized counts)')
title('Fold change by FDR')
```

You can see here that for weakly expressed genes (i.e. those with low means), the FDR is generally high because low read counts are dominated by Poisson noise and consequently any biological variability is drowned in the uncertainties from the read counting.

References

- [1] Brooks et al. Conservation of an RNA regulatory map between Drosophila and mammals. *Genome Research* 2011. 21:193-202.
- [2] Mortazavi et al. Mapping and quantifying mammalian transcriptomes by RNA-Seq. *Nature Methods* 2008. 5:621-628.
- [3] Anders et al. Differential expression analysis for sequence count data. *Genome Biology* 2010. 11:R106.
- [4] Marioni et al. RNA-Seq: An assessment of technical reproducibility and comparison with gene expression arrays. *Genome Research* 2008. 18:1509-1517.
- [5] Robinson et al. Moderated statistical test for assessing differences in tag abundance. *Bioinformatics* 2007. 23(21):2881-2887.
- [6] Benjamini et al. Controlling the false discovery rate: a practical and powerful approach to multiple testing. 1995. *Journal of the Royal Statistical Society, Series B* 57 (1):289-300.

See Also

`featurecount` | `mairplot` | `nbintest` | `plotVarianceLink`

More About

- “High-Throughput Sequencing”

Visualize NGS Data Using Genomics Viewer App

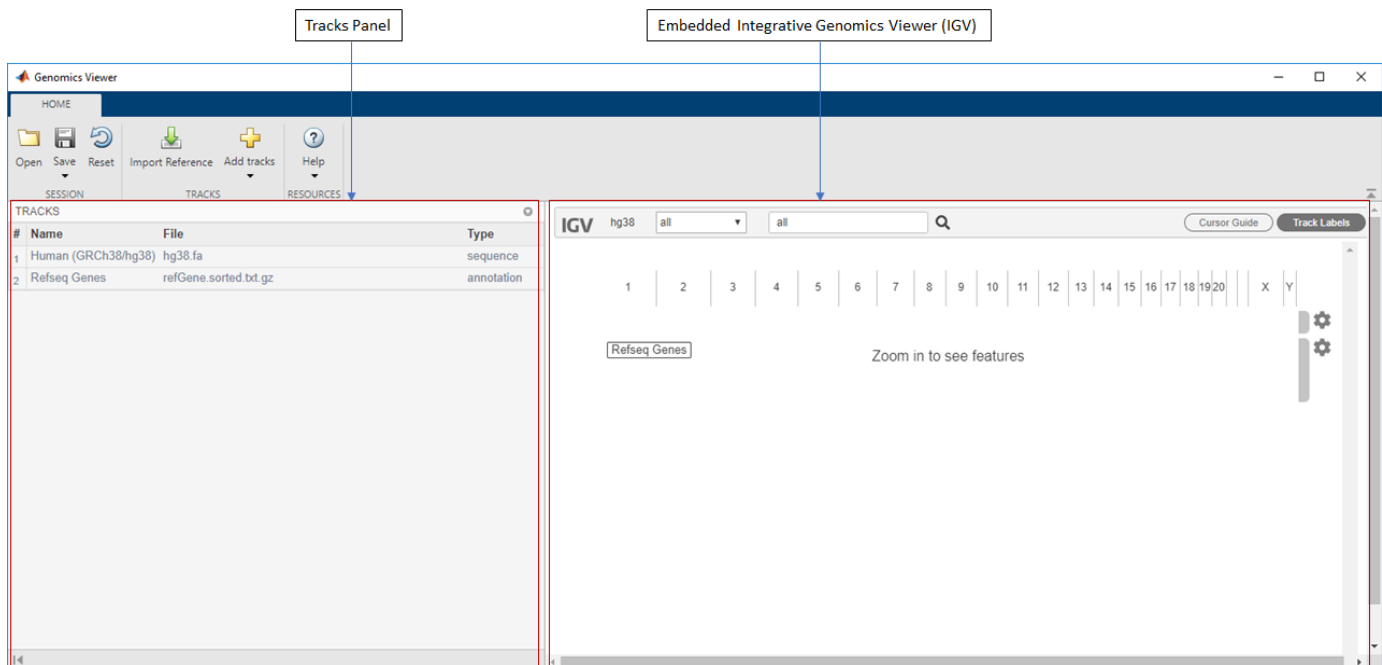
The **Genomics Viewer** app lets you view and explore integrated genomic data with an embedded version of the Integrative Genomics Viewer (IGV) [1][2]. The genomic data include NGS read alignments, genome variants, and segmented copy number data.

The first part of this example gives a brief overview of the app and supported file formats. The second part of the example explores a single nucleotide variation in the cytochrome p450 gene (CYP2C19).

Open the App

At the command line, type `genomicsViewer`. Alternatively, click the app icon on the **Apps** tab. The app requires an internet connection.

By default, the app loads Human (GRCh38/hg38) as the reference sequence and Refseq Genes as the annotation file. There are two main panels in the app. The left panel is the **Tracks** panel and the right panel is the embedded IGV web application. The **Track** panel is a *read-only* area displaying the track names, source file names, and track types. The **Track** panel updates accordingly as you configure the tracks in the embedded IGV app.



The **Reset** button restores the app to the default view with two tracks (HG38 with Refseq Genes) and removes any other existing tracks. Before resetting, you can save the current view as a session (.json) file and restore it later.

Add Tracks by Importing Data

Import Reference Sequence

You can import a single reference sequence. The reference sequence must be in a FASTA file. Select **Import Reference** on the **Home** tab. You can also import a corresponding cytoband file that contains

cytogenetic G-banding data. You can add local files or specify external URLs. The URL must start with either *https* or *gs*. Other file transfer protocols, such as *ftp*, are not supported.

Import Sequence Read Alignment Data

You can import multiple data sets of sequence read alignment data. The alignment data must be a BAM or CRAM file. It is not required that you have the corresponding index file (.BAI or .CRAI) in the same location as your BAM or CRAM file. However, the absence of the index file will make the app slower.

You can add read alignment files using **Add tracks from file** and **Add tracks from URL** options from the **Add tracks** button. If you are specifying a URL, the URL must start with either *https* or *gs*. Other file transfer protocols, such as *ftp*, are not supported.

Import Feature Annotations and Other Genomic Data

You can import multiple sets of feature annotations from several files that contain data for a single reference sequence. The supported annotation files are: .BED, .GFF, .GFF3, and .GTF.

You can also import structural variants (.VCF) and visualize genetic alterations, such as insertions and deletions.

You can view segmented copy number data (.SEG) and quantitative genomic data (.WIG, .BIGWIG, and .BEDGRAPH), such as ChIP peaks and alignment coverage.

You can add annotation and genomic data files using **Add tracks from file** and **Add tracks from URL** options from the **Add tracks** button. If you are specifying a URL, the URL must start with either *https* or *gs*. Other file transfer protocols, such as *FTP*, are not supported.

Visualize Single Nucleotide Variation in Cytochrome P450

The *CYP2C19* gene is a member of the cytochrome P450 gene family. Enzymes produced from cytochrome P450 genes are involved in the metabolism of various molecules and chemicals within cells. The *CYP2C19* enzyme plays a role in the metabolizing of at least 10 percent of commonly prescribed drugs [3]. Polymorphisms in the cytochrome p450 family may cause adverse drug responses in individuals. One example of single nucleotide variation is *rs4986893* at position *chr10:94,780,653* where G is replaced by A. This allelic variant is also known as *CYP2C19*3*. The following steps show how to visualize such variation in the app using both low coverage and high coverage data.

Load Session File

For the purposes of this example, start with a session file that has some preloaded tracks. To load the file, click **Open**. Navigate to *matlabroot\examples\bioinfo*, where *matlabroot* is the folder where you have installed MATLAB. Select *rs4986893.json*.

The screenshot displays the Genomics Viewer (IGV) interface. On the left, a 'TRACKS' panel lists three tracks: 1. Human (GRCh38/hg38) sequence, 2. NA18564 alignment, and 3. Refseq Genes. The main view shows the alignment data for sample NA18564 on chromosome 10, centered around a 38 bp region. The alignment track shows a low coverage area with an orange bar. The Refseq Genes track shows the CYP2C19 gene structure.

The session contains three tracks:

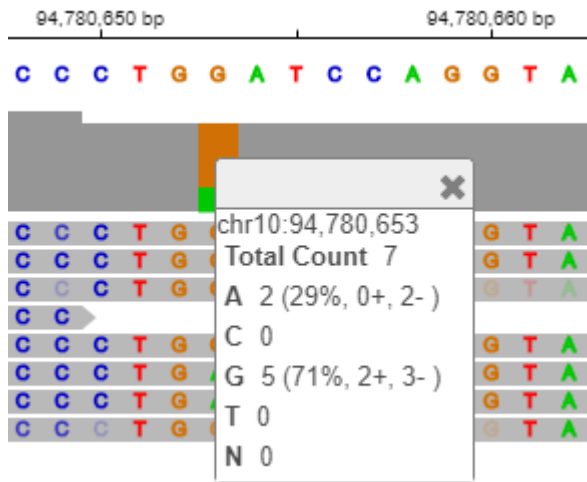
- *Human (GRCh38/hg38)* as a reference
- *NA18564* as low coverage alignment data
- Refseq Genes

The low coverage alignment data comes from a female Han Chinese from Beijing, China. The sample ID is *NA18564* and the sample has been identified with the *CYP2C19**3 mutation [4].

Explore Low Coverage Data

In this session file, the alignment data has been centered around the location of the mutation on the *CYP2C19* gene.

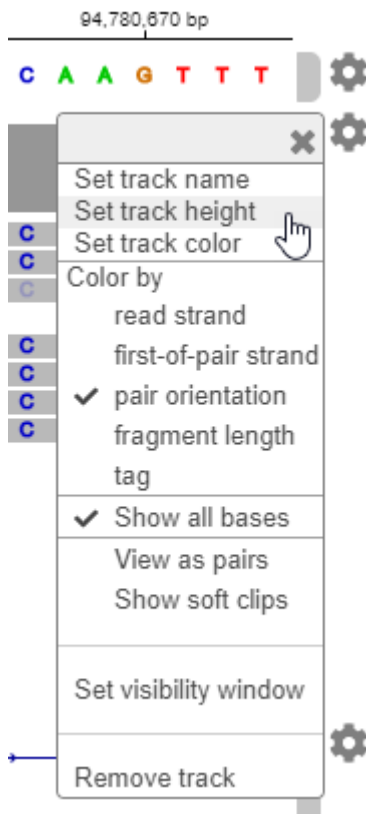
- 1 Click the orange bar in the coverage area to look at the position and allele distribution information.



It shows that 71% of the reads have G while 29% have A at the location *chr10:94,780,653*. This data is a low coverage data and may not show all the occurrences of this mutation. A high coverage data will be explored later in the example.

Close the data tip window.

- 2 You can customize the various aspects of the data display in the app. For example, you can change the track height to make more room for later tracks. Click the second gear icon. Select *Set track height*. Enter 200.

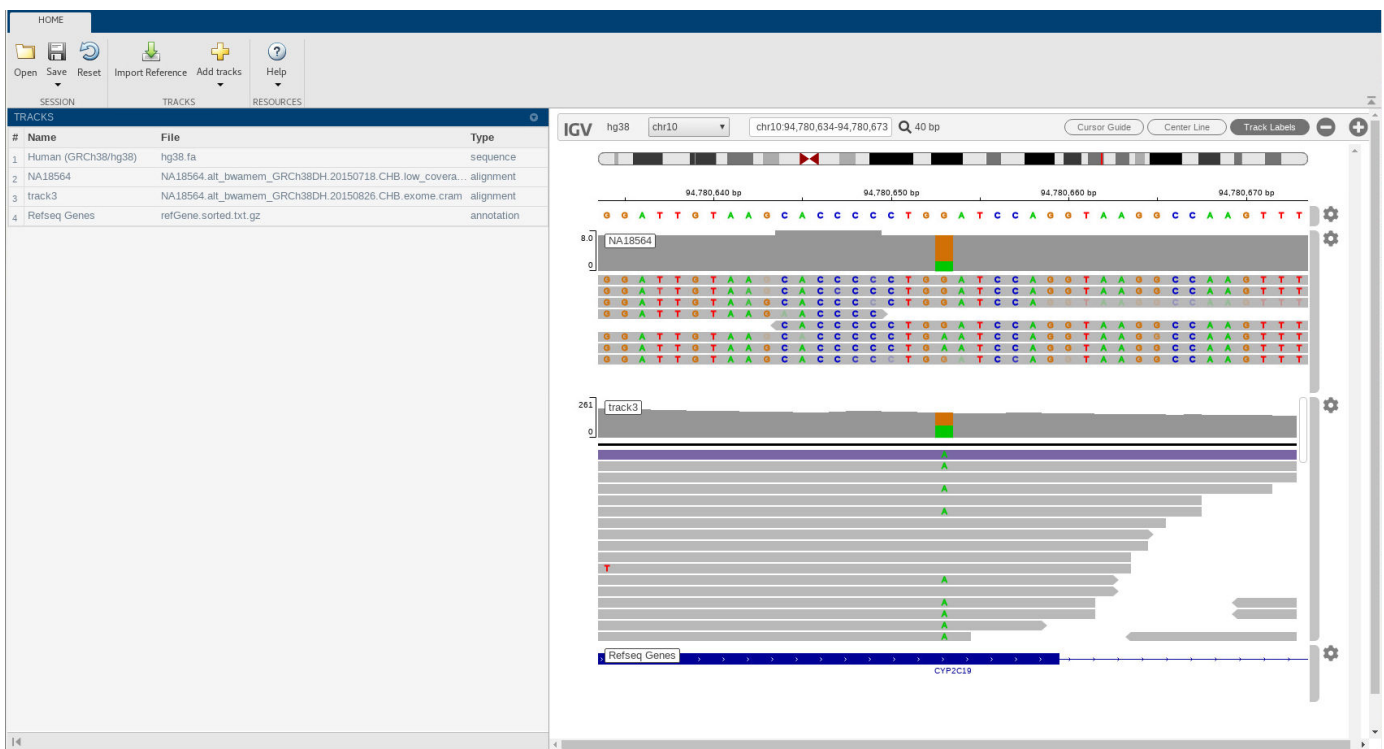


For details on the embedded IGV app and its available options, visit [here](#).

Explore High Coverage Data

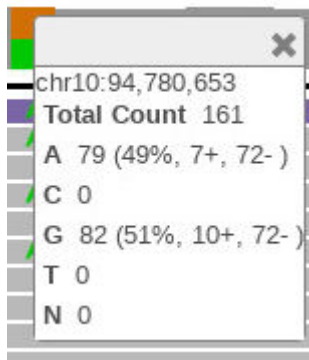
You can look at the high coverage data from the same sample to see the occurrences of this mutation.

- 1 Go to The International Genome Sample Resource website.
- 2 Search for the sample *NA18564*.
- 3 Download the *Exome* alignment file that is in the .CRAM format.
- 4 Also download the corresponding index file that is in the .CRAI format. Save the file in the same location as the source .CRAM file.
- 5 Click the (+) icon on the **Home** tab. Select the downloaded .CRAM file and click **Open**.



The high coverage data appears as track3. You can now see many occurrences of the mutation in several reads.

- 6 Click the orange bar in the coverage area to see the allele distribution. It shows that G is replaced by A in almost 50% of the time.



References

- [1] Robinson, J., H. Thorvaldsdóttir, W. Winckler, M. Guttman, E. Lander, G. Getz, J. Mesirov. 2011. Integrative Genomics Viewer. *Nature Biotechnology*. 29:24-26.
- [2] Thorvaldsdóttir, H., J. Robinson, J. Mesirov. 2013. Integrative Genomics Viewer (IGV): High-performance genomics data visualization and exploration. *Briefings in Bioinformatics*. 14:178-192.
- [3] <https://ghr.nlm.nih.gov/gene/CYP2C19>
- [4] https://www.coriell.org/0/Sections/Search/Sample_Detail.aspx?Ref=NA18564&Product=DNA

See Also

Genomics Viewer | Sequence Alignment | Sequence Viewer

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.

- “Exploring a Nucleotide Sequence Using Command Line” on page 3-2
- “Exploring a Nucleotide Sequence Using the Sequence Viewer App” on page 3-15
- “Explore a Protein Sequence Using the Sequence Viewer App” on page 3-26
- “Compare Sequences Using Sequence Alignment Algorithms” on page 3-30
- “View and Align Multiple Sequences” on page 3-43

Exploring a Nucleotide Sequence Using Command Line

In this section...

“Overview of Example” on page 3-2
 “Searching the Web for Sequence Information” on page 3-2
 “Reading Sequence Information from the Web” on page 3-4
 “Determining Nucleotide Composition” on page 3-5
 “Determining Codon Composition” on page 3-8
 “Open Reading Frames” on page 3-11
 “Amino Acid Conversion and Composition” on page 3-13

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.

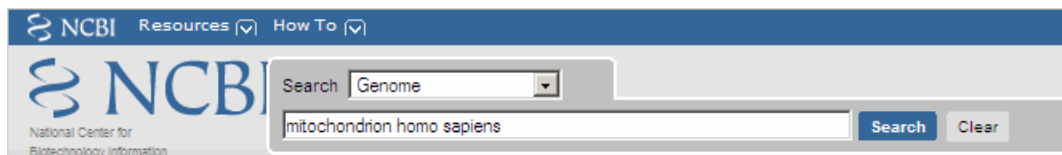
First research information about the human mitochondria and find the nucleotide sequence for the genome. Next, look at the nucleotide content for the entire sequence. And finally, determine open reading frames and extract specific gene sequences.

- 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.



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

The screenshot shows the NCBI Genomes database search results. The search criteria are 'Genome' for 'mitochondrion homo sapiens'. The results list 49 items, with the first item being 'NC_003415: Ancylostoma duodenale mitochondrion, complete genome'. The details for this entry are: DNA; circular; Length: 13,721 nt; Organelle: mitochondrion; Created: 2002/02/21. The interface includes a search bar, navigation tabs (Limits, Preview/Index, History, Clipboard, Details), and a results list with a 'Page 1 of 3 Next' indicator.

NCBI

Search Genome for mitochondrion homo sapiens

Display Summary Show 20 Send to

All: 49

Items 1 - 20 of 49 Page 1 of 3 Next

1: [NC_003415](#) Links

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](#)

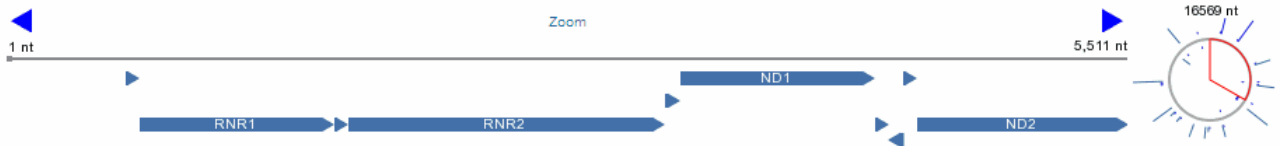
[Links](#)

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:



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

Display [Overview](#) Show [20](#) Send to

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 =
GATCACAGGTCTATCACCTATTAACCACTCACGGGAGCTCTCCATGCAT
TTGGTATTTTCGTCTGGGGGTGTGCACGCGATAGCATTGCGAGACGCTG
GAGCCGGAGCACCTATGTCGAGTATCTGTCTTTGATTCCTGCCTCATT
CTATTATTATCGCACCTACGTTCAATATTACAGGCGAACATACCTACTA
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

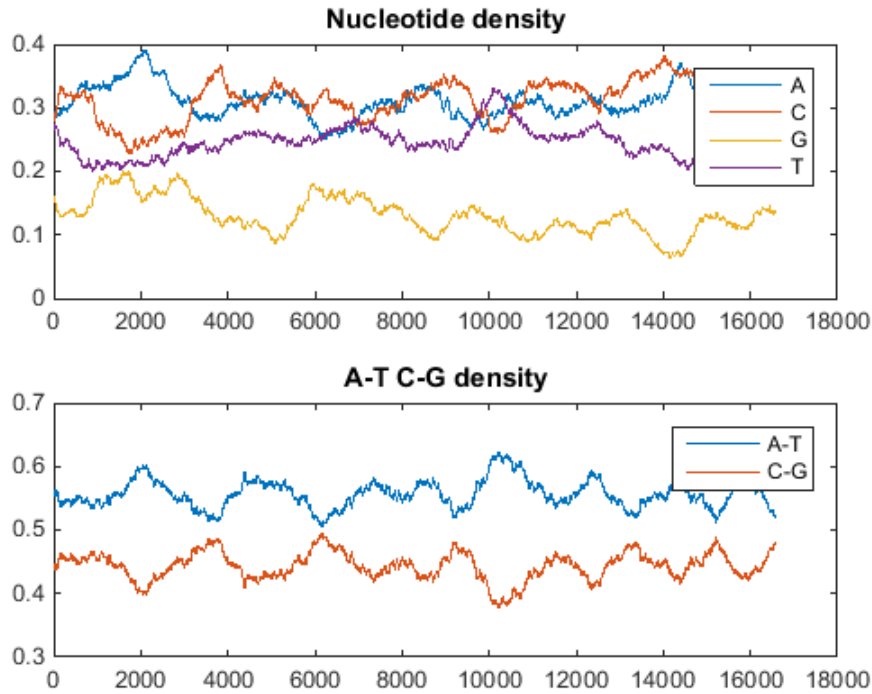
The following procedure illustrates how to determine the monomers and dimers, and then visualize data in graphs and bar plots. Sections of a DNA sequence with a high percent of A+T nucleotides usually indicate intergenic parts of the sequence, while low A+T and higher G+C nucleotide percentages indicate 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 3-4.

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

```
ntdensity(mitochondria)
```

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



- 2 Count the nucleotides using the `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))
```

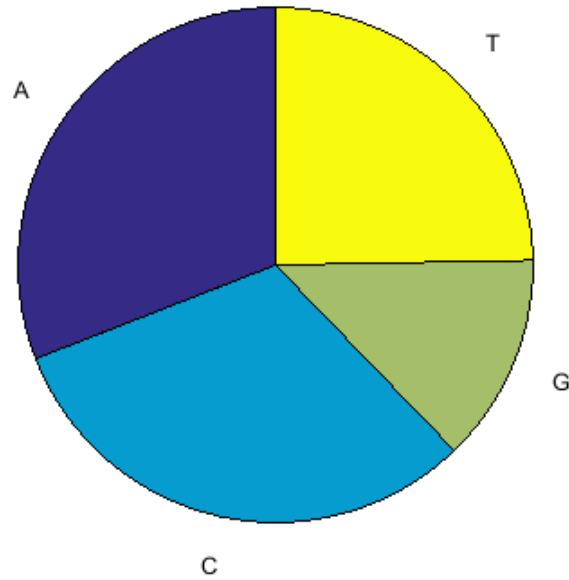
As expected, the nucleotide counts on the reverse complement strand are complementary to the 5'-3' strand.

```
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.

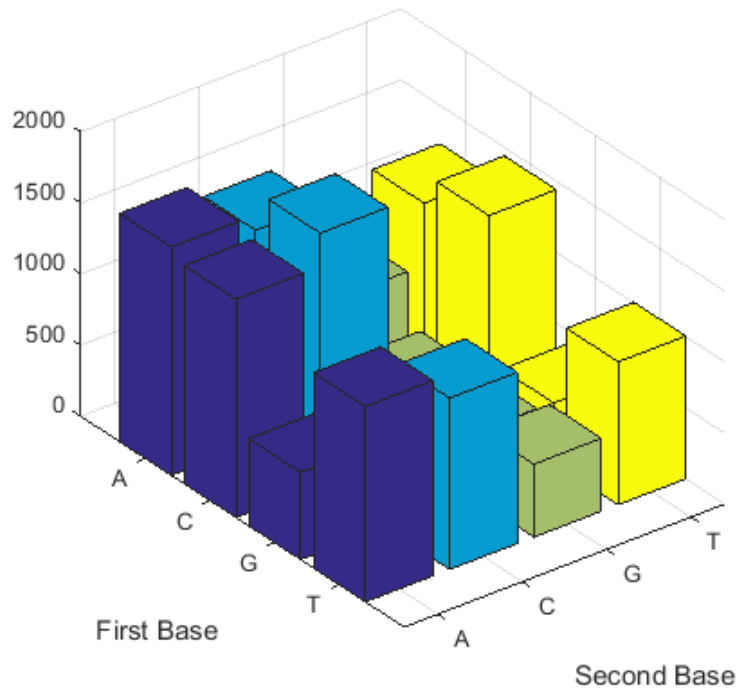


- 5 Count the dimers in a sequence and display the information in a bar chart.

```
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 3-4.

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

```
codoncount(mitochondria)
```

The codon counts for the first reading frame displays.

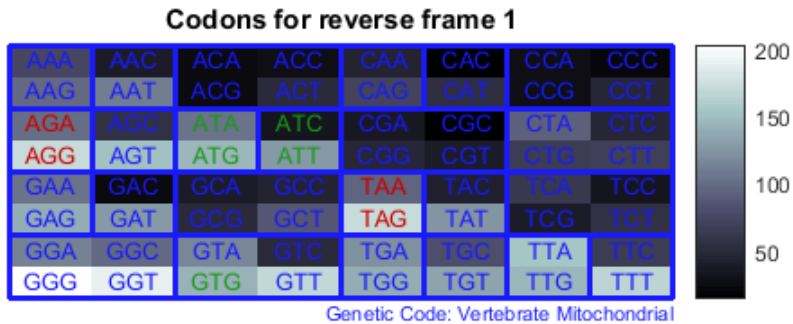
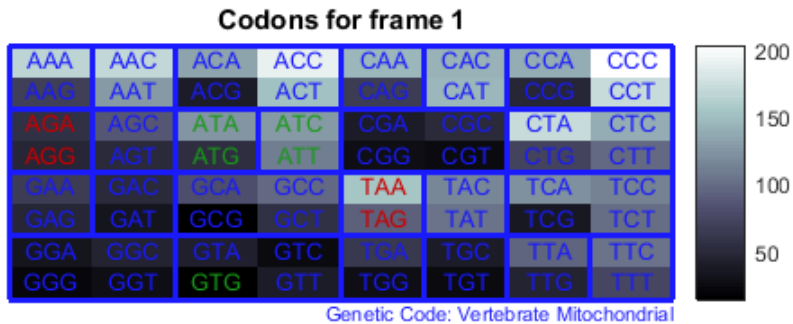
AAA - 167	AAC - 171	AAG - 71	AAT - 130
ACA - 137	ACC - 191	ACG - 42	ACT - 153
AGA - 59	AGC - 87	AGG - 51	AGT - 54
ATA - 126	ATC - 131	ATG - 55	ATT - 113
CAA - 146	CAC - 145	CAG - 68	CAT - 148
CCA - 141	CCC - 205	CCG - 49	CCT - 173
CGA - 40	CGC - 54	CGG - 29	CGT - 27
CTA - 175	CTC - 142	CTG - 74	CTT - 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

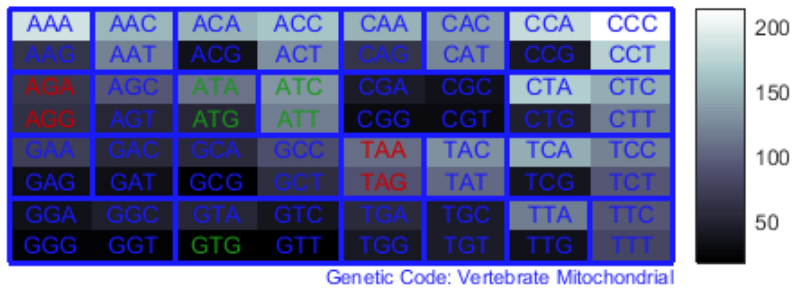
2 Count the codons in all six reading frames and plot the results in heat maps.

```
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
```

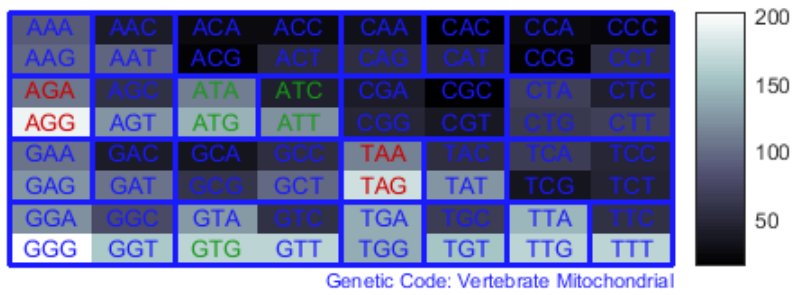
Heat maps display all 64 codons in the 6 reading frames.

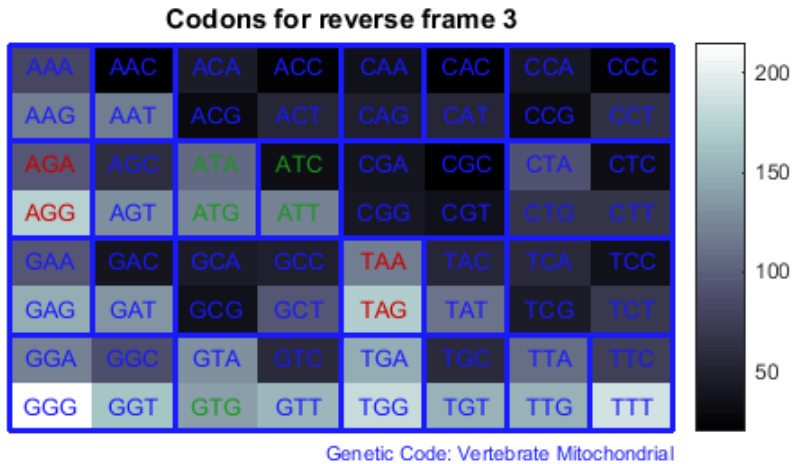
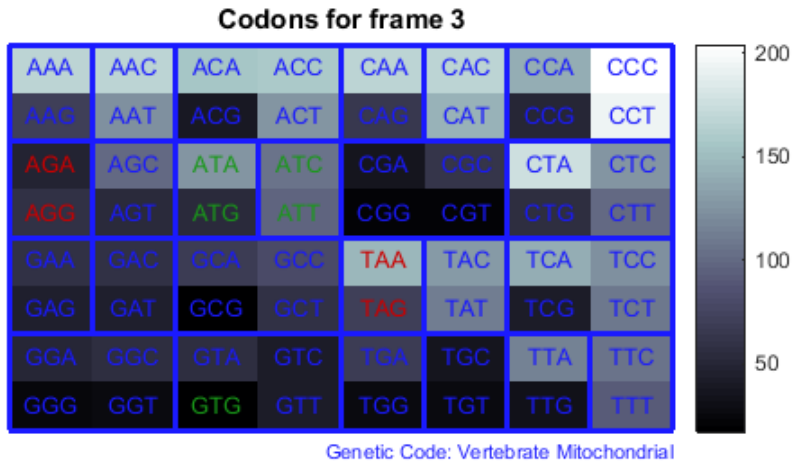


Codons for frame 2



Codons for reverse frame 2





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 3-4.

- 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.

- 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.

```
attaatcccctggcccaaccgctcatctactctaccatctttgcaggcac
actcatcacagcgctaagctcgactgatttttacctgagtaggcctag
aaataaacatgctagcttttattccagttctaaccaaaaaataaacctt
cgttccacagaagctgccatcaagtatttctcagcaagcaaccgcatc
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 3-11.

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

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

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

```
MNPLAQPVIYSTIFAGTLITALSSHWFFTWVGLMNMLAFIPVLTKMNP
RSTEA AIKYFLTQATASMI LLMAILFNNMLSGQWMTNTTNQYSSLMIMM
AMAMKLGMAPFHFVWPEVTQGTPLTSGLLLLTWQKLAPISIMYQISPSLN
VLLLLTSLISIMAGSWGGLNQTQLRKILAYSSITHMGWMMAVLPYNPNM
TILNLTIIYIILTTTAFLLLNLSSTTTLLSRTWNKLTWLTPLIPSTLLS
LGGPLPLTGFLPKWAIIEEFTKNNSLIPTIMATITLLNLYFYLRRIYST
SITLLPMSNNVKMKWQFEHTKPTPFLPTLIALTLLLPISPFMLMIL
```

- 2 Compare your conversion with the published conversion in the GenPept database.

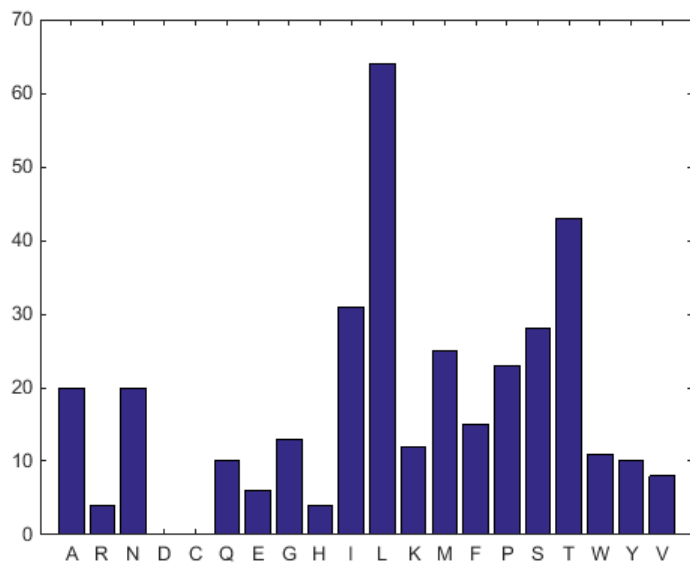
```
ND2protein = getgenpept('YP_003024027','sequenceonly',true)
```

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.

```
aaccount(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.

Exploring a Nucleotide Sequence Using the Sequence Viewer App

In this section...

“Overview of the Sequence Viewer” on page 3-15
“Importing a Sequence into the Sequence Viewer” on page 3-15
“Viewing Nucleotide Sequence Information” on page 3-17
“Searching for Words” on page 3-19
“Exploring Open Reading Frames” on page 3-22
“Closing the Sequence Viewer” on page 3-25

Overview of the Sequence Viewer

The **Sequence Viewer** integrates many of the sequence functions in the Bioinformatics Toolbox toolbox. Instead of entering commands in the MATLAB Command Window, you can select and enter options using the app.

Importing a Sequence into the Sequence Viewer

The first step when analyzing a nucleotide or amino acid sequence is to import sequence information into the MATLAB environment. The **Sequence Viewer** 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.

Note Data in public repositories is frequently curated and updated; therefore, the results of this example might be slightly different when you use up-to-date sequences.

- 1 In the MATLAB Command Window, type

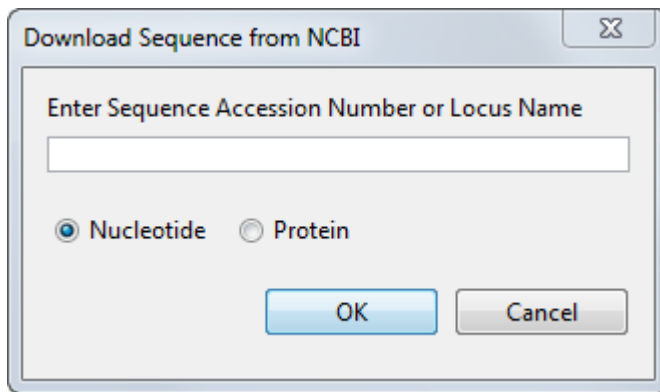
```
seqviewer
```

Alternatively, click **Sequence Viewer** on the **Apps** tab.

The **Sequence Viewer** 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.



- 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.

Biological Sequence Viewer - NM_000520

File Edit Sequence Display Window Help

Line length: 60

Sequence View

NM_000520: Homo sapiens

- Sequence
- ORF
- Full Translation
- Annotated CDS
- CDS with Translation
- Complement Sequence
- Reverse Complement Sequence
- Features
- Comments

Base Count

A:	593	21.6%
C:	750	27.3%
G:	716	26.0%
T:	692	25.2%

NM_000520: Homo sapiens hexosaminidase subunit alpha (HEXA), transcript variant 2, mRNA

Position: 2751 bp

10 20 30 40 50 60

```

1  tcacatcaca acgacttggtg gttttaatcc tccgtttttc tgccttctgaa gttacttcag
61  cctggcaagt cctttaccctc cccgtaggcc tggcagctg catcacaaca ttcaagattc
121 accctagagc catctgggaa actttcttct ccaggtcgcc ctgcgtccct gcctccccac
181 cccgttcttc tcgagtcggg ttagctgtct agttccatca cggccggcac ggcgcagggg
241 gtggccgggt atttactgct ctactgggcc cgtgaacagt ctggcgagcc gaggcagttgc
301 cgacgccggg cacaaatccgc tgcacgtagc aggagcctca ggtccaggcc ggaagtgaaa
361 gggcaggggt tgggtccctcc tggggtcgca ggcgcagagc cgcctctggt cacgtgatcc
421 gcgataaagt cacggggggc ccgctcacc taccagggtc gaccagggtc agccccctcc
481 gagaggggag accagcgggc catgacaagc tccagggttt ggtttctgct gctgctggcg
541 gcagcgttgc caggacgggc gacggccctc tggccctggc ctcagaactt ccaaaccctc
601 gaccagcgtc agtcccttta ccgaaacaac tttcaatttc agtacgatgt cagctcggcc
661 ggcagcccg gctgctcagt cctcgaagag gcttccagc gctatcgtga cctgcttttc
721 ggttccgggt cttggccccc tccctaccct acagggaaac ggcatacact ggagaagaat
781 gtgttggttg tctctgtagt cacacctgga tgaaccagc ttcctacttt ggagtcagtg
841 gagaattata ccctgacct aaatgatgac cagtgtttac tctctctgta gactgtctgg
901 ggagctctcc gaggctctgga gacttttagc cagcttgttt ggaaatctgc tgagggcaca
961 tcttttatca caaagactga gattgaggac tttccccgct ttcctcaccg gggcctgctg
1021 ttggataact ctcgccatta cctgcactc tctagatccc tggaaactct ggtatctatg
1081 ggtacaata aattgaaagt gttccactgg catctggtag atgaccttc cttccatatt
1141 gagagcttca cttttccaga gctcatgaga aagggtcctt caaacctgt caccacatc
1201 tacacagcac aggatgtgaa ggaggtcatt gaatacgcac ggcctccggg tatccgtgtg
1261 cttgcagagt ttgacctcc tggccacct ttgtcctggg gaccagggtat ccctggatta

```

4.7 BP/Pixel

X2 Zoom in X2 Zoom out

Map View

Sequence

CDS

1 1000 2000 2751

Viewing Nucleotide Sequence Information

After you import a sequence into the **Sequence Viewer** app, 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.

Biological Sequence Viewer - NM_000520

File Edit Sequence Display Window Help

Line length: 60

Sequence View

NM_000520: Homo sapiens

- Sequence
 - ORF
 - Full Translation
 - Annotated CDS
 - CDS with Translation
 - Complement Sequence
 - Reverse Complement Sequence
 - Features
 - Comments

Base Count

A:	593	21.6%
C:	750	27.3%
G:	716	26.0%
T:	692	25.2%

NM_000520: Homo sapiens hexosaminidase subunit alpha (HEXA), transcript variant 2, mRNA.

Position: Words found: 33 2751 bp

10 20 30 40 50 60

1 tcacatcaca acgacttggtg gttttaatcc tccgtttttc tgccttctgaa gttacttcag

+1

+2

+3

-1

-2

-3

61 cctggcaagt cctttacctc cccgtaggcc tggcgagctg catcacaaca ttcaagattc

+1

+2

+3

-1

-2

-3

121 accctagagc catctgggaa actttcttct ccaggtcgcc ctgcttcctc gcctccccac

+1

+2

+3

-1

-2

-3

181 cccgtttttc tcgagtcggg tgagctgtct agttccatca cggccggcac ggcccgaggg

+1

+2

+3

-1

-2

-3

241 gtggccggtt atttactgct ctactgggcc cgtgaacagt ctggcgagcc gagcagttgc

+1

+2

+3

-1

-2

-3

301 cracccccra cacaatccac taccatcac aaraacctca atcccaarcc araaactraaa

4.7 BP/Pixel X2 Zoom in X2 Zoom out

Map View

Sequence

ORF

CDS

1 1000 2000 2751

- 4 Click **Annotated CDS** to show the protein coding part of a nucleotide sequence.

The screenshot displays the Biological Sequence Viewer interface for the sequence NM_000520: Homo sapiens hexosaminidase subunit alpha (HEXA), transcript variant 2, mRNA. The main window shows the nucleotide sequence with a line length of 60. The sequence is color-coded by base (A: green, C: blue, G: red, T: orange). A red box highlights the CDS region from position 541 to 841. The sequence is as follows:

```

1  tcacatcaca acgacttggt gttttaatcc tccgtttttc tgccttctgaa gttacttcag
61  cctggcaagt cctttacctc cccgtaggcc tggcgagctg catcacaaca ttcaagattc
121 accctagagc catctgggaa actttcttct ccaggctcgc ctgctgcttc gctctccac
181 cccgttcttc tcgagtcggg tgagctgtct agttccatca cggccggcac ggccgcaggg
241 gtggccgggt atttactgct ctactgggcc ctggaacagt ctggcgagcc gagcagttgc
301 cgacgcccgg cacaaatccg tcgacgtagc aggagcctca ggtccaggcc ggaagtgaaa
361 gggcaggggt tgggtctctc tgggtctgca gggcgagagc cgcctctggt cacgtgatcc
421 gccgataagt cacggggggc ccgctcacc gaccagggtc tcacgtggcc agccccctcc
481 gagaggggag accagcgggc catgacaagc tccaggcttt ggttttcgct gctgctggcg
541 gcagcgttcc caggacgggc gacggccctc tggccctggc ctcagaactt ccaaactccc
601 gaccagcgtc acgtctctta ccgaaacaac tttcaattcc agtacgatgt cagctcggcc
661 ggcgagcccg gctgctcagt cctcgacgag gccttccagc gctatcgtga cctgcttttc
721 ggttccgggt cttggccccg tccttacctc acagggaaac ggcatacact ggagaagaat
781 gtgttgggtg tctctgtagt cacacctgga tgtaaccagc ttcttacttt ggagtcagtg
841 gagaattata ccctgacct aatgatgac cagtgtttac tcctctctga gactgtctgg

```

The Base Count table shows the following statistics:

Base	Count	Percentage
A	593	21.6%
C	750	27.3%
G	716	26.0%
T	692	25.2%

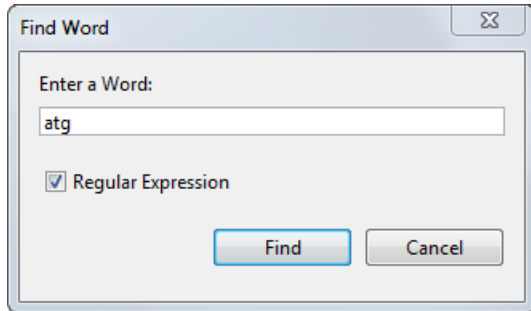
The Map View at the bottom shows the sequence, ORF, and CDS regions. The CDS region is highlighted in purple, and the ORF region is highlighted in red. The sequence is shown in a map view with a scale from 1 to 2751 bp.

Searching for Words

You can also search for characteristic words or sequence patterns using regular expressions. You can enter the IUB/IUPAC nucleotide and amino acid symbols that are automatically converted to corresponding nucleotides and amino acids accordingly. For details about how symbols are interpreted, see the **Nucleotide Conversion** and **Amino Acid Conversion** tables of seq2regex.

For instance, if you search for the word 'TAR' with the **Regular Expression** box checked, the app highlights all the occurrences of 'TAA' and 'TAG' in the sequence since $R = [AG]$.

- 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**.



The **Sequence Viewer** searches and displays the location of the selected word.

Biological Sequence Viewer - NM_000520

File Edit Sequence Display Window Help

Line length: 60

Sequence View

NM_000520: Homo sapiens

- Sequence
 - ORF
 - Full Translation
 - Annotated CDS
 - CDS with Translation
- Complement Sequence
- Reverse Complement Sequence
- Features
- Comments

Base Count

A:	593	21.6%
C:	750	27.3%
G:	716	26.0%
T:	692	25.2%

NM_000520: Homo sapiens hexosaminidase subunit alpha (HEXA), transcript variant 2, mRNA

Position: Words found: 33 2751 bp

10 20 30 40 50 60

1 tcacatcaca acgacttggtg gttttaatcc tccgtttttc tgccttctgaa gttacttcag
 61 cctggcaagt cctttacctc cccgtaggcc tggcgagctg catcacaaca ttcaagattc
 121 accctagagc catctgggaa actttttctt ccaggtcgcc ctgctgcttc gcctccccac
 181 cccgtttctc tcgagtcggg tgagctgtct agttccatca cggccggcac ggcgcagggg
 241 gtggccgggt atttactgct ctactgggcc cgtgaacagt ctggcgagcc gaggcagttgc
 301 cgacgccggg cacaatccgc tgcacgtagc aggagcctca ggtccaggcc ggaagtgaaa
 361 gggcagggtg tgggtccctc tgggttcgca ggcgcagagc cgcctctggt cactgatc
 421 gccgataagt cacggggggc ccgctcacc gaccagggtc tcacgtggcc agccccctcc
 481 gagaggggag accagcgggc catgacaagc tccaggcttt ggttttcgtc gctgctggcg
 541 gcagcgttcc caggacgggc gacggccctc tggccctggc ctccagaact ccaaacctcc
 601 gaccagcgct acgtccctta cccgaacaac tttcaattcc agtacgatgt cagctcggcc
 661 ggcagccccg gctgctcagt cctcgacgag gccttccagc gctatcgtga cctgcttttc
 721 ggttccgggt cttggccccg tccttacctc acagggaac ggcatacact ggagaagaat
 781 gtgttggttg tctctgtagt cacacctgga tghtaaccagc ttcttacttt ggagtcagtg
 841 gagaattata ccctgacct aaatgatgac cagtgtttac tcctctctga gactgtctgg
 901 ggagctctcc gaggctctgga gacttttagc cagcttgttt ggaatctgc tgagggcaca
 961 ttctttatca acaagactga gattgaggac tttccccgct ttctcaccg gggcttgctg

HEX A

4.7 BP/Pixel X2 Zoom in X2 Zoom out

Map View

Sequence

ORF

CDS

3

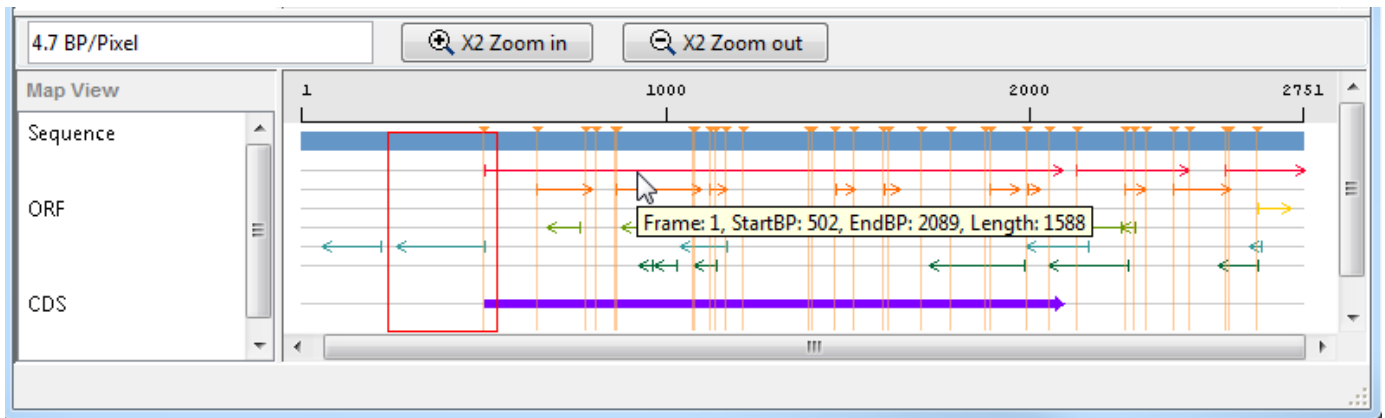
Clear the display by clicking the Clear Word Selection button  on the toolbar.

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.

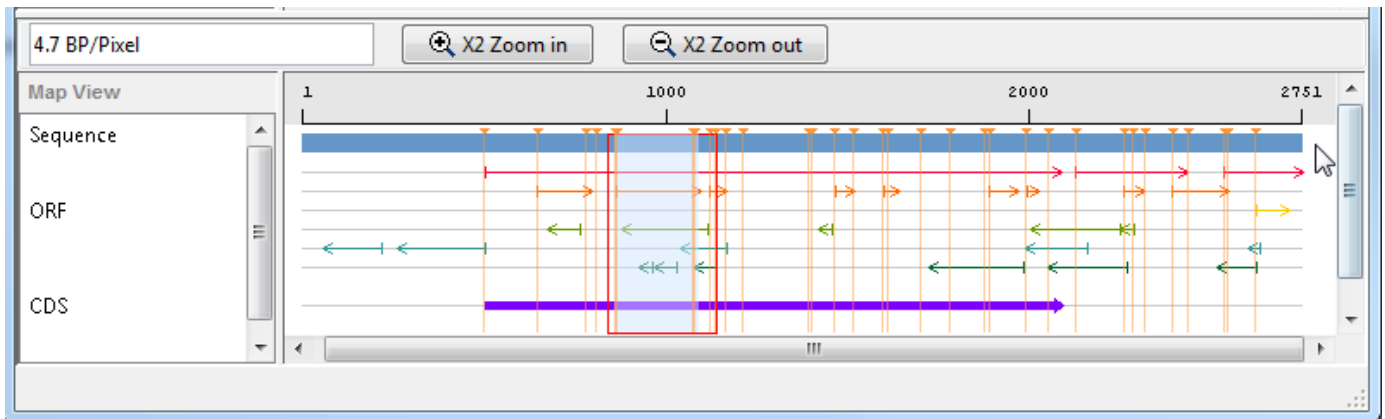
- 1 In the left pane, click **ORF**.

The **Sequence Viewer** displays the ORFs for the six reading frames in the lower-right pane. Hover the cursor over a frame to display information about it.

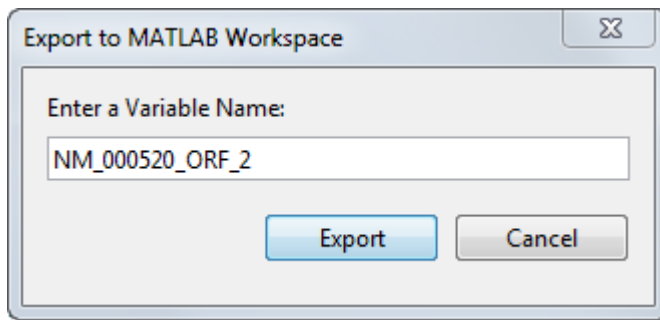


- 2 Click the longest ORF on reading frame 2.

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**.



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 Viewer** adds a tab at the bottom for the new sequence while leaving the original sequence open.

The screenshot shows the Biological Sequence Viewer window for NM_000520_ORF_2. The main window displays the DNA sequence in a color-coded format. The sequence is as follows:

```

1  atgatgacca gtgtttactc ctctctgaga ctgtctgggg agctctccga ggtctggaga
61  ctttagcca gcttgtttgg aaatctgctg agggcacatt ctttatcaac aagactgaga
121 ttgaggactt tccccgttt cctcacggg gcttgctgtt ggatacatct cgccattacc
181 tgccactctc tagcatcctg gacactctgg atgtcatggc gtacaataaa tt

```

Below the sequence, the Base Count is displayed:

Base	Count	Percentage
A:	48	20.7%
C:	60	25.9%
G:	54	23.3%
T:	70	30.2%

The interface also includes a Map View at the bottom, showing a horizontal bar representing the sequence from position 1 to 232. The zoom level is set to 0.4 BP/Pixel, and there are buttons for X2 Zoom in and X2 Zoom out.

- In the left pane, click **Full Translation**. Select **Display > Amino Acid Residue Display > One Letter Code**.

The **Sequence Viewer** displays the amino acid sequence below the nucleotide sequence.

The screenshot displays the Biological Sequence Viewer app interface. The main window shows the sequence NM_000520_ORF_2, which is 232 base pairs long. The sequence is displayed in a grid format with line length set to 60. The sequence is color-coded by base: A (green), C (blue), G (red), and T (black). The translation is shown below the sequence, with amino acids color-coded: M (grey), M (grey), T (grey), S (grey), V (grey), Y (grey), S (grey), S (grey), L (grey), R (grey), L (grey), S (grey), G (grey), E (grey), L (grey), S (grey), E (grey), V (grey), W (grey), R (grey), * (grey), * (grey), P (grey), V (grey), F (grey), T (grey), P (grey), L (grey), * (grey), D (grey), C (grey), L (grey), G (grey), S (grey), S (grey), P (grey), R (grey), S (grey), G (grey), D (grey), D (grey), D (grey), Q (grey), C (grey), L (grey), L (grey), L (grey), S (grey), E (grey), T (grey), V (grey), W (grey), G (grey), A (grey), L (grey), R (grey), G (grey), L (grey), E (grey), L (grey), L (grey), A (grey), S (grey), L (grey), F (grey), G (grey), N (grey), L (grey), L (grey), R (grey), A (grey), H (grey), S (grey), L (grey), S (grey), T (grey), R (grey), L (grey), R (grey), F (grey), * (grey), P (grey), A (grey), C (grey), L (grey), E (grey), I (grey), C (grey), * (grey), G (grey), H (grey), I (grey), L (grey), Y (grey), Q (grey), Q (grey), D (grey), * (grey), D (grey), T (grey), F (grey), S (grey), Q (grey), L (grey), V (grey), W (grey), K (grey), S (grey), A (grey), E (grey), G (grey), T (grey), F (grey), F (grey), I (grey), N (grey), K (grey), T (grey), E (grey), L (grey), R (grey), T (grey), F (grey), P (grey), A (grey), F (grey), L (grey), T (grey), G (grey), A (grey), C (grey), C (grey), W (grey), I (grey), H (grey), L (grey), A (grey), I (grey), T (grey), * (grey), G (grey), L (grey), S (grey), P (grey), L (grey),S (grey),S (grey),P (grey),G (grey),L (grey),A (grey),V (grey),G (grey),Y (grey),I (grey),S (grey),P (grey),L (grey),P (grey),I (grey),E (grey),D (grey),F (grey),P (grey),R (grey),F (grey),P (grey),H (grey),R (grey),G (grey),L (grey),L (grey),L (grey),D (grey),T (grey),S (grey),R (grey),H (grey),Y (grey),T (grey),G (grey),C (grey),C (grey),A (grey),C (grey),T (grey),C (grey),T (grey),C (grey),T (grey),C (grey),T (grey),G (grey),G (grey),A (grey),C (grey),T (grey),G (grey),G (grey),G (grey),T (grey),A (grey),C (grey),A (grey),T (grey),A (grey),A (grey),A (grey),T (grey),T (grey),C (grey),H (grey),S (grey),L (grey),A (grey),S (grey),W (grey),T (grey),L (grey),W (grey),M (grey),S (grey),W (grey),R (grey),T (grey),I (grey),N (grey),A (grey),T (grey),L (grey),* (grey),H (grey),P (grey),G (grey),H (grey),S (grey),G (grey),C (grey),H (grey),G (grey),V (grey),Q (grey),* (grey),I (grey),L (grey),P (grey),L (grey),S (grey),S (grey),I (grey),L (grey),D (grey),T (grey),L (grey),D (grey),V (grey),M (grey),A (grey),Y (grey),N (grey),K (grey).

The Base Count table shows the following data:

Base	Count	Percentage
A	48	20.7%
C	60	25.9%
G	54	23.3%
T	70	30.2%

The Map View shows a horizontal bar representing the sequence from position 1 to 232. The zoom level is set to 0.4 BP/Pixel. The app has three tabs open: Untitled, NM_000520, and NM_000520_ORF_2.

Closing the Sequence Viewer

Close the **Sequence Viewer** from the MATLAB command line using the following syntax:

```
seqviewer('close')
```

Explore a Protein Sequence Using the Sequence Viewer App

In this section...

“Overview of the Sequence Viewer” on page 3-26

“Viewing Amino Acid Sequence Statistics” on page 3-26

“Closing the Sequence Viewer” on page 3-28

“References” on page 3-29

Overview of the Sequence Viewer

The **Sequence Viewer** integrates many of the sequence functions in the Bioinformatics Toolbox toolbox. Instead of entering commands in the MATLAB Command Window, you can select and enter options using the app.

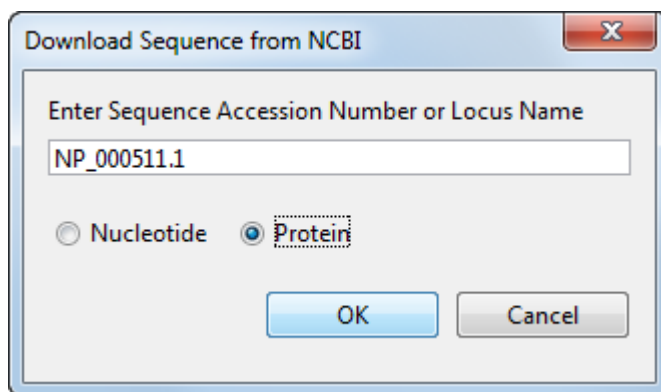
Viewing Amino Acid Sequence Statistics

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.

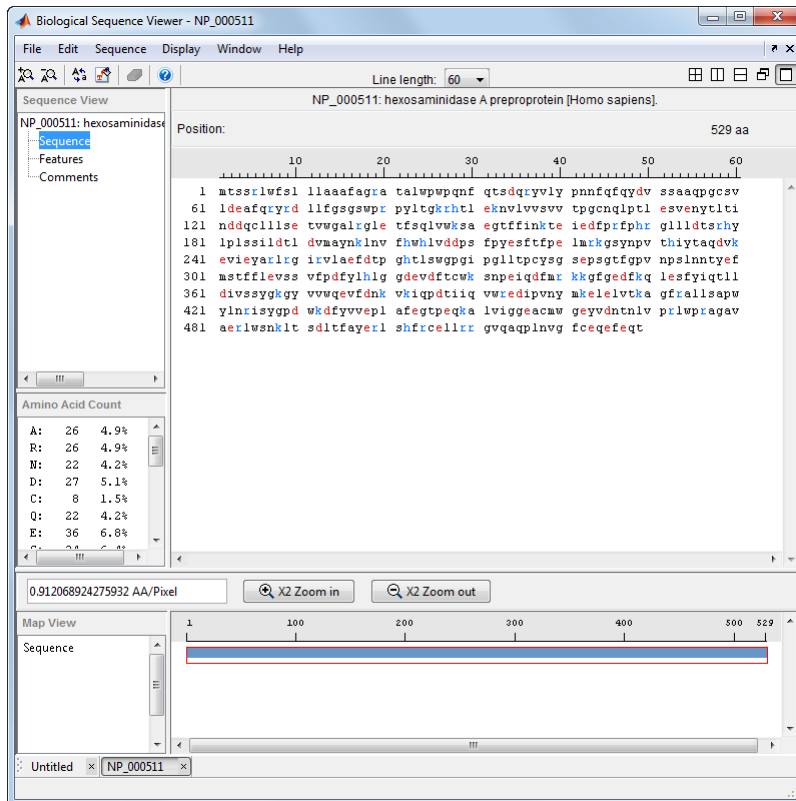
- 1 Select **File > Download Sequence from > NCBI**.

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**.

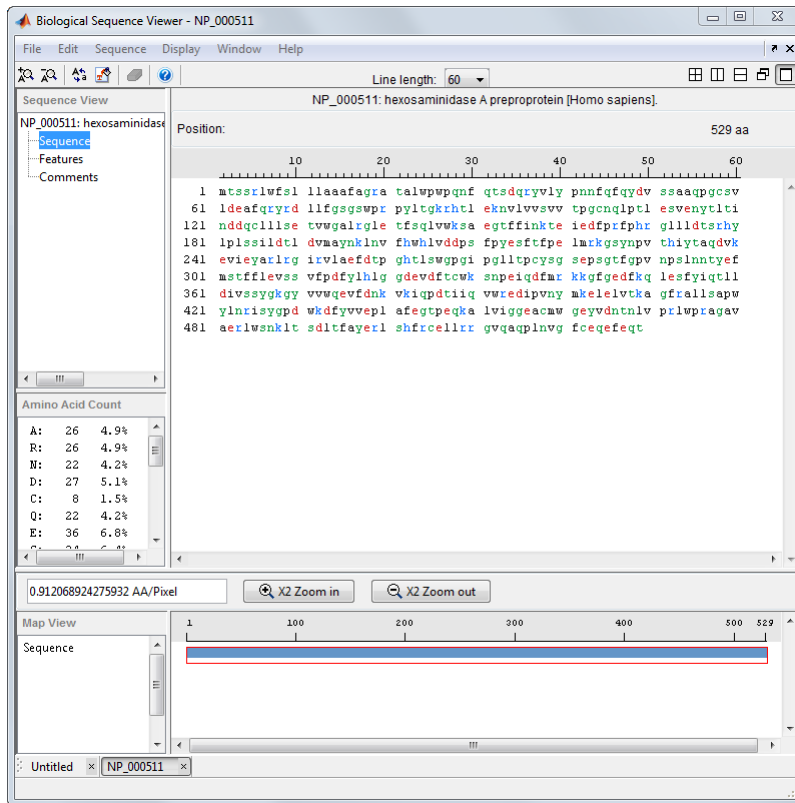


The **Sequence Viewer** accesses the NCBI database on the Web and loads amino acid sequence information for the accession number you entered.



- 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.



Amino Acid Color Scheme	Color Legend
Charge	<ul style="list-style-type: none"> Acidic — Red Basic — Light Blue Neutral — Black
Function	<ul style="list-style-type: none"> Acidic — Red Basic — Light Blue Hydropobic, nonpolar — Black Polar, uncharged — Green
Hydrophobicity	<ul style="list-style-type: none"> Hydrophilic — Light Blue Hydrophobic — Black
Structure	<ul style="list-style-type: none"> 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 on page 3-29.

Closing the Sequence Viewer

Close the **Sequence Viewer** from the MATLAB command line using the following syntax:

```
seqviewer('close')
```

References

- [1] Taylor, W.R. (1997). Residual colours: a proposal for aminochromography. *Protein Engineering* 10, 7, 743-746.

Compare Sequences Using Sequence Alignment Algorithms

In this section...

“Overview of Example” on page 3-30

“Find a Model Organism to Study” on page 3-30

“Retrieve Sequence Information from a Public Database” on page 3-31

“Search a Public Database for Related Genes” on page 3-33

“Locate Protein Coding Sequences” on page 3-34

“Compare Amino Acid Sequences” on page 3-36

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.

Find a Model Organism to Study

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

First, 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/books/NBK22250/')
```

The MATLAB Help browser opens with the Tay-Sachs disease page in the Genes and Diseases section of the NCBI web site. 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.

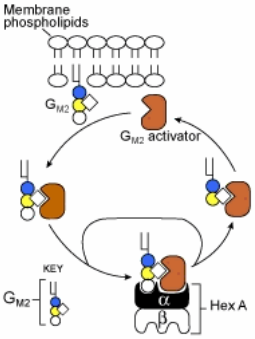
Location: <http://www.ncbi.nlm.nih.gov/books/NBK22250>

NCBI Resources How To Sign in to NCBI

Bookshelf This Book Search Limits Advanced Help

Contents Bookshelf ID: NBK22250 Print View < Prev Next >

Tay-Sachs disease



Model for G_{M2} ganglioside metabolism. Under normal conditions, β -hexosaminidase works in the lysosome of nerve cells to breakdown unwanted ganglioside G_{M2} , a component of the nerve cell membrane. This requires three components: an α -subunit, a β -subunit and an activator subunit. In Tay Sachs disease, the alpha subunit of hexosaminidase malfunctions, leading to a toxic build-up of the G_{M2} ganglioside in the lysosome. [Adapted from: Chavany, C. and Jendoubi, M. (1998) *Mol. Med. Today*, 4: 158-165, with permission.]

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 early death to a chronic adult form that exhibits neuron dysfunction and psychosis.

Genes and Disease [Internet]. National Center for Biotechnology Information (US). [Show details](#)

[Table of Contents Page](#) | [Cite this Page](#)

Download
PDF version of this page (261K)

Gene sequence
Genome view see gene locations
Entrez Gene collection of gene-related information
BLink related sequences in different organisms

The literature
Research articles online full text
Books online books section
OMIM catalog of human genes and disorders
GeneReviews a medical genetics resource

Websites
Fact Sheet from National Institute of Neurological Disorders and Stroke
NTSAD National Tay-Sachs and Allied Diseases Association

2 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.

Retrieve 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 Window, 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.

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.

[Display Settings:](#) Summary, 20 per page, Sorted by Default order

[Send to:](#)

i Found 28006 nucleotide sequences. Nucleotide (60) GSS ([27946](#))

Results: 1 to 20 of 60

<< First < Prev Page of 3 Next > Last >>

[HEXA {HEXA4bpDeltass mutation, exon 11} \[human, **Tay-Sachs** disease patient, mRNA Partial Mutant, 84 nt\]](#)

1. 84 bp linear mRNA

Accession: S76984.1 GI: 912781

[GenBank](#) [FASTA](#) [Graphics](#)

[HEXA {HEXAdeltass mutation, exon 11} \[human, **Tay-Sachs** disease patient, mRNA Partial Mutant, 80 nt\]](#)

2. 80 bp linear mRNA

Accession: S76982.1 GI: 912780

[GenBank](#) [FASTA](#) [Graphics](#)

[HEXA {HEXA4bp mutation, exon 11} \[human, **Tay-Sachs** disease patient, mRNA Partial Mutant, 84 nt\]](#)

3. 84 bp linear mRNA

Accession: S77043.1 GI: 912779

[GenBank](#) [FASTA](#) [Graphics](#)

[HEXA {HEXA4bpDeltaA mutation, exon 11} \[human, **Tay-Sachs** disease patient, mRNA Partial Mutant, 78 nt\]](#)

4. 78 bp linear mRNA

Accession: S76980.1 GI: 912777

[GenBank](#) [FASTA](#) [Graphics](#)

[Human beta-hexosaminidase A alpha-chain \(with the classic form **Tay-Sachs** deletion\) gene, partial cds](#)

5. 351 bp linear DNA

Accession: J02820.1 GI: 184482

[GenBank](#) [FASTA](#) [Graphics](#) [Related Sequences](#)

[Homo sapiens hexosaminidase A \(alpha polypeptide\) \(HEXA\), mRNA](#)

6. 2,437 bp linear mRNA

Accession: NM_000520.4 GI: 189181665

[GenBank](#) [FASTA](#) [Graphics](#) [Related Sequences](#)

- 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]
```

Search 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.

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

117. [Mus musculus 9.5 days embryo parthenogenote cDNA, RIKEN full-length enriched library, clone:B130019N09 product:hexosaminidase A, full insert sequence](#)

1,839 bp linear mRNA

Accession: [AK080777.1](#) GI: 26348756

[GenBank](#) [FASTA](#) [Graphics](#) [Related Sequences](#)

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]
```

Locate 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 agttgocgacgccccggcacaatccgctgcacgtagcaggagcctcagggtccaggccggaagtga
000065 aagggcagggtgtgggtcctcctggggctgcaggcgcagagccgcctctggtcacgtgattcgc
000129 cgataagtcacggggggcgcgcctcacctgaccagggtctcacgtggccagccccctccgagagg
000193 ggagaccagcggggccatgacaagctccaggctttgggttttgcgtgctggtgggcagcgttgc
000257 caggacggggcagcggcctctggcctggcctcagaacttccaaacctccgaccagcgtacgt
000321 cctttaccgaaacaactttcaattccagtaagatgtcagctcggccgocgacgccggctgtcoa
000385 gtocctgacgaggccttcacagcctatogtgaactgcttttcgggtccgggtottggcccgtc
000449 cttacctcacagggaacggcatacaactggagaagaatgtgttggttgtctctgtagtcaacc
000513 tggatgtaaccagcttctactttggagtcagtgagaattataccctgaccataaatgatgac
000577 cagtgtttactcctctctgagactgtctggggagctctccagggtctggagacttttagccagc
000641 ttgtttggaaatctgctgagggcacattctttatcaacaagactgagattgaggactttccocg
000705 ctttcctccacggggcttgcgtgttgatacatctgcattacctgcccactctctagcatcctg
000769 gacactctggatgtcatggcgtacaataaattgaacgtgttccactggcatctggtagatgatc
000833 ctccctcccatatgagagcttccactttccagagctcagagaagggtcctacaacctgt
000897 caccacatctacacagcacaggatgtgaaggaggtcattgaaacgcaacgctccgggtatc
000961 cgtgtgctgcagagtttgacaactcctggccacaactttgtcctggggaccagggtatccctggat
001025 tactgactccttgcactctgggtctgagccctctggcacctttggaccagtgaatccagctot
001089 caataatacctatgagttcatgagcacattctcttagaagtcagctctgtctccagatttt
001153 tatcttcatcttgaggagatgaggttgatttccctgctggaagtcacaccagagatccagg
001217 actttatgaggaagaaaggctcoggtgaggacttcaagcagctggagctctctacaatccagac
001281 gctgctggacatogtctctcttatggcaagggtatgtggtgtggcaggagggttttgataat
001345 aaagtaagattcagccagacacaatcacaacgggtgtggcagaggatattccagtgaaactata
001409 tgaaggagctggaactggtcaccaaggccggctccggccctctctctgccccctggtaact
001473 gaaccgtatacctatggccctgactggaaggatttctacatagtggaaccctggcatttgaa
001537 ggtaccctgagcagaaggctctggtgattggtggagaggcttgatgtggggagaatattgtgg
001601 acaacacaaacctggtccccaggctctggcccagagcaggggctgttgccgaaaggctgtggag
001665 caacaagttgacatctgacctgacattttgcctatgaaogtttgtcacactccgctgtgaattg
001729 ctgaggcggagggtccaggcccaaccctcaatgtaggctctctgtgagcaggagtttgaacaga
001793 cctgagccccaggcaccgaggagggtgctggctgtagggtgaatggtagtgaggccaggcttcca
001857 ctgcatcctggccagggaacggagcccttgcttctgctgccccttgctgctgcccctgtgct
001921 tggagagaaaagggcgggtgctggcctcgcattcaataaagagtaatgtggcattttctata
001985 ataaacatggattacctgtgtttaaaaaaaaaagtgatgaaatggcgttagggtaagggcacagcc
002049 aggtggagtcagtgctgcccctgaggtcttttaagttgagggtcgggaatgaaacctatagc
002113 ctttgtgtgtctotgccttgctgtgagctatgtcactccctcccactcctgacctattcca
002177 gacacctgccctaactcctcagcctgctcactcactctctgattatctccaaggcgttggtat
002241 tatggaaaaagatgtaggggttgagggtgtctggacagtggggagggtccagacccaacct
002305 ggtcacagaagagcctctccccatgcaatactcaccctccctccctagagctattctcct
002369 ttgggtttctgtgcttcaattttatacaaccattatttaaatattattaacacatattgtt
002433 ctota
```

- 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 gctgctggaaggggagctggccgggtgggcccattggccggctgcaggctctgggttccgctgctgc
000065 tggcggcggcgttggcttgcctggccacggcactgtggcctggccccagtaacccaacctc
000129 ccaccggcgtacacctgtaccccaacaactccagttccggtaaccatgtcagttcggcccg
000193 caggcggcgtgcctgcctcgcagggccttcgacgctaccgtaacctgctctcgggtccg
000257 gctcttggccccgacccagcttctcaataaacagcaaacgttggggaagaacattctgggtgt
000321 ctccgtcgtcacagctgaatgtaatgaatttccataatttggagtcggtagaaaattacacctc
000385 accattaatgatgaccagtgcttactcgcctctgagactgtctggggcgcctccgaggtctgg
000449 agacttccagtcagcttgttggaaatcagctgagggcagctctcttcaacaagacaaagat
000513 taaagacttccctcgattccctcaccggggcgtactgctggatacatctcgccattacctgca
000577 ttgtctagcatcctggatcacctggatgcatggcacaataaataaacctgttccactggc
000641 acttgggtggacgactctcctcccatatgagagcttcaacttcccagagctcaccagaaaggg
000705 gtccctcaacctgtcactcacatctacacagcaccaggatgtgaaggaggtcatgaatacgc
000769 aggtctcggggatccctgtgctggcagaatttgacactcctggccacactttgtcctggggc
000833 caggtgccccgggttatcaaccttgcctctcgggtctcctctctcggcaccatttggacc
000897 ggtgaaccccagctcacaacagcactatgactcatgagcactctcctggagatcagctca
000961 gtcttcccggacttttctccacctgggaggggatgaagtgcactcaccctgctggaagtcca
001025 accccaacatccaggcctcctgaagaaaaagggcttactgactcgaagcagctggagctct
001089 ctacatccagagcgtgctggacatgctctgattatgacaagggctatgtggtgtggcaggag
001153 gtatttgataataaagtgaaggttcggccagatacaatcatacaggtgtggcgggaagaaatgc
001217 cagtagagtacatgttggagatgcaagatataccagggtggcttccgggcccctgctgtctgc
001281 tcctggtacctgaacctgtaaagtatggcctgactggaaggacatgtacaagtggagccc
001345 ctggcgttccatggtacgcctgaacagaaggctctggtcattggaggggagggcctgtatgtgg
001409 gagagtatgtggacagcaccacctggctcccagactctggcccagagcgggtgcccgtcctga
001473 gagactgtggagcagtaacctgacaactaatatagacttgcctttaaacttctgtcgcatttc
001537 cgttgtgagctggtgaggagaggaaatccaggcccagcccacagtgtaggctgctgtgagcagg
001601 agtltgagcagacttgagccaccagtgctgaacaccaggaggtgctgtcctttgagtcagct
001665 gcgctgagcaccaggaggggtgctggccttaagagagcaggctcccggggcagggtaatcttc
001729 actgctcctccggccaggggagagcacccttgcctgtgcccctgtgactacagagaaggagg
001793 ctggtgctggcactgggtgttcaataaagatctatgtggcattttctc
```

Compare 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.

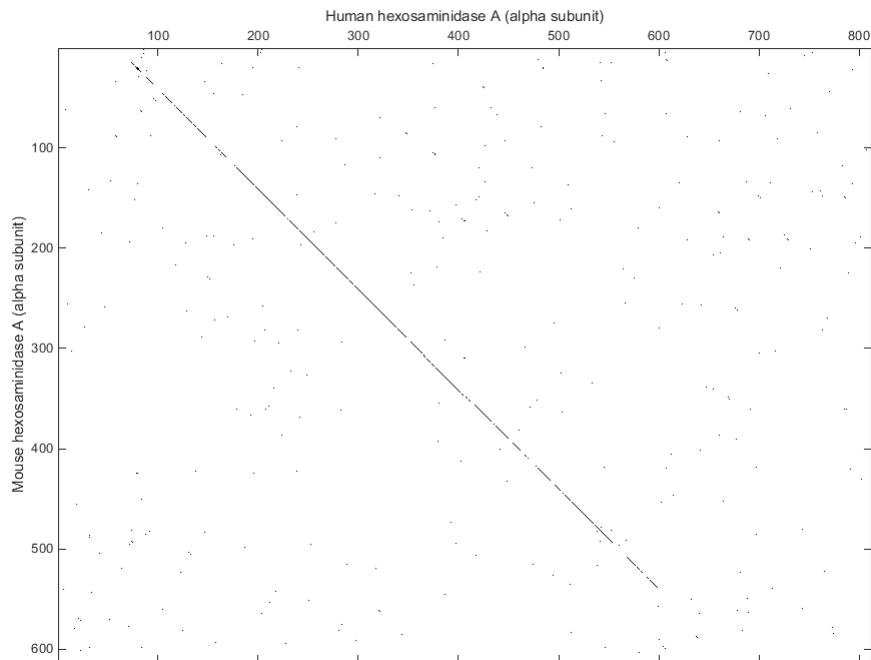
- 1 Using the open reading frames identified previously, convert the human and mouse DNA sequences to the amino acid sequences. Because both the human and mouse HEXA genes were in the first reading frames (default), you do not need to indicate which frame. Type

```
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.



- 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  SCRRPAQSAAARSRLRSRPEVKGGQGVGPPGVAGAEPLVT*FADKSRGRSPDQGLTWPAAPER
      ||          |          |          |          |
001  -----AA-----GR-----G-----A-----G-R-----W-----

065  GDQRAMTSSRLWFSLLLAAAFAGRATALWPPQNFQTSDDQRYVLYPNNFQFOYDVSSAAQPGCS
      ||:| ||| |||||:| |||||:| | | | | | | | | | | | | | | | | | | | | | | |
010  ---AMAGCRLWVSLLLAAALACLATALWPPQYIQTYHRRYTYLPNNFQFRYHVSSAAQAGCV

129  VLDEAFQRYRDLDFGSGSWPRPYLTGKRHTLEKNVLVSVVTPGCNQLPTLESVENYTLTINDD
      |||||:| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
070  VLDEAFRRYRNLDFGSGSWPRPSFSNKQOTLGKNILVSVVTAECNEFPNLESVENYTLTINDD

193  OCLLLSETVWGALRGLETFSQLVWKSAGTFFINKTEIEDFPRFPHRGLLLDTSRHYLPLSSIL
      ||| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
134  OCLLASETVWGALRGLETFSQLVWKSAGTFFINKTKIKDFPRFPHRGVLLDTSRHYLPLSSIL

257  DTLDVMAYNKNLVFHWHLVDDPSFPYESFTFPELMRKGSYNPVTHIYTAQDVKEVIEYARLRGI
      |||||:| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
198  DTLDVMAYNKNLVFHWHLVDDSSFPYESFTFPELTRKGSFNPVTHIYTAQDVKEVIEYARLRGI

321  RVLAEFDTPGHTLSWGPPIGLLLPCYSGSEPSGTFGPVNPSLNNTYEFMSTFFLEVSVVFPDF
      |||||:| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
262  RVLAEFDTPGHTLSWGPAPGLLLPCYSGSHLSGTFGPVNPSLNSTYDFMSTLFLFLEISSVFPDF

385  YLHLGGDEVDFTCWKSNIQDFMRKKGFGEDFKQLESFYIQTLLDIVSSYKGYVWVQEVFDN
      |||||:| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
326  YLHLGGDEVDFTCWKSNIQAFMKKKGF-TDFKQLESFYIQTLLDIVSDYKGYVWVQEVFDN

449  KVKIQPDTIIQVWREDIPVNYMKELELVTKAGFRALLSAPWYLNRIYGPDKWDFYIVEPLAFE
      |||:| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
389  KVKVRPDTIIQVWREEMPVEYMLEMQDITRAGFRALLSAPWYLNRVKYGPDKWDMYKVEPLAFH

513  GTPEOKALVIGGEACMWGEYVDNTNLVPRWLPRAGAVAERLWSNKLTSDLTFAYERLSHFRCEL
      |||||:| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
453  GTPEOKALVIGGEACMWGEYVDSNTNLVPRWLPRAGAVAERLWSNLTNIDFAFKRLSHFRCEL

577  LRRGVOAQLNVGFCEOEFEOT*APGTEEGAGCR*MVVEPGFHCILARGRSPLPSCPLPACPCA
      :| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
517  VRRGIOAQPISVGCCEOEFEOT*A--T--SA--E---HPG-----G-----C---CP--

641  WRERGRWCWRSHSIXSNVAFFYNKHGLPVFKKSVNGVVRVRAQPGWSOCLPLRSFKLRAGNETYS
      | : : | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
552  -----L-SQ-LR--*A-----P--RR-V--LALR-E---O-VP--G-Q---G-*SFT

705  LCAVLPCL*AMSLPSHS*PYSRHL*SSACSLHFCIISPRRWYMEKDVGAWRCSGQWGGLOTQP
      | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
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 =
```

```
MTSSRLWFSLLLAAAFAGRATALWPPQNFQTSQRYVLYPNNFQFQYDV
SSAAQPGCSVLDEAFQRYRDLLFGSGSWPRPYLTGKRHTLEKNVLVSVV
TPGCNQLPTLESVENYTLTINDDQCLLLSETVWGALRGLETFSQLVWKS
EGTFFINKTEIEDFPRFPHRGLLDTSRHYLPLSSILDTLDMAYNKLNV
FHWHLVDDSPFYESFTFPELMRKGSYNPVTHIYTAQDVKEVIEYARLRG
IRVLAEFDTPGHTLSWGPPIPGLLTPCYSGSEPSGTGFPVNPSTNNTYEF
MSTFFLEVSSVFPDFYLHLGGDEVDFTCWKSNIQDFMRKKGFGEFQK
LESFYIQTLLDIVSSYGKGYVWQEVFDNKVKIQPDTIIQVWREDIPVNY
MKELELVTKAGFRALLSAPWYLNRIISYGPDWKDFYIVEPLAFEGTPEQKA
LVIGGEACMWGEYVDNTNLVPRLWPRAGAVAERLWSNKLTSDLTFAYERL
SHFRCELLRRGVQAQPLNVGFCEQEFEQT*
```

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

```
mouseProteinORF =
```

```
MAGCRLWVSLLLAAALACLATALWPPQYIQTYHRRYTLYPNNFQFRYHV
SSAAQAGCVVLDEAFRRYRNLLFGSGSWPRPSFSNKQQTGKNILVSVV
TAECNEFPNLESVENYTLTINDDQCLLASETVWGALRGLETFSQLVWKS
EGTFFINKTKIKDFPRFPHRGVLLDTSRHYLPLSSILDTLDMAYNKFNV
FHWHLVDDSSFPYESFTFPELTRKGSFNPVTHIYTAQDVKEVIEYARLRG
IRVLAEFDTPGHTLSWGPAPGLLTPCYSGSHLSGTGFPVNPSTNSTYDF
MSTLFLEISSVFPDFYLHLGGDEVDFTCWKSNIQAFMKKKGFDFKQL
ESFYIQTLLDIVSDYDKGYVWQEVFDNKVKVRPDTIIQVWREEMPVEYM
LEMQDITRAGFRALLSAPWYLNRVKYGPDWKDMYKVEPLAFHGTPEQKAL
VIGGEACMWGEYVDSTNLVPRLWPRAGAVAERLWSSNLTNIDFAFKRLS
HFRCELVRRGIQAQPIISVGCCEQEFEQT*
```

- 6 Globally align the trimmed amino acid sequences. Type

```
[GlobalScore_trim, GlobalAlignment_trim] = nwalignment(humanProteinORF,...
                                                         mouseProteinORF);
showalignment(GlobalAlignment_trim)
```



```

mouseORFs = seqshoworfs(mouseHEXA.Sequence)

mouseORFs =

1x3 struct array with fields:
    Start
    Stop

humanPORF = nt2aa(humanHEXA.Sequence(mouseORFs(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-AMTSSRLWFSLLLAAAFAGRATALWPWPQNFQTS DQRYV . . .
|| | ||: ||| |||||:| ||||| :| :|: . . .
RGAGRWAMAGCRLWVSLLLAAALACLATALWPWPQYIQTYHRRYT . . .

```

- 9** Show the alignment in color.

```
showalignment(LocalAlignment)
```


View and Align Multiple Sequences

In this section...

“Overview of the Sequence Alignment App” on page 3-43

“Visualize Multiple Sequence Alignment” on page 3-43

“Adjust Sequence Alignments Manually” on page 3-44

“Rearrange Rows” on page 3-52

“Generate Phylogenetic Tree from Aligned Sequences” on page 3-54

Overview of the Sequence Alignment App

The **Sequence Alignment** app integrates many sequence and multiple alignment functions in the toolbox. Instead of entering commands in the MATLAB Command Window, you can use this app to visually inspect a multiple alignment and make manual adjustments.

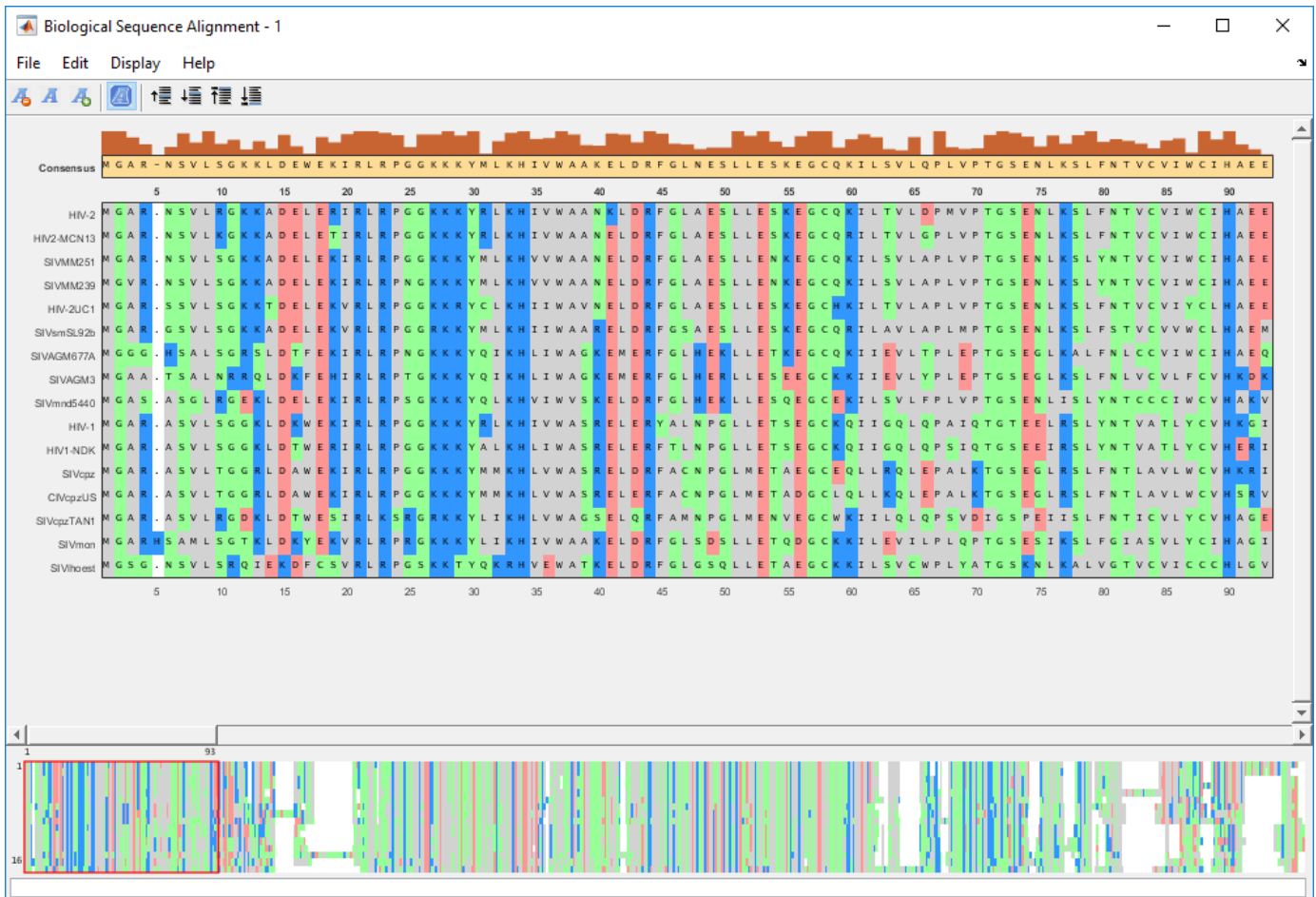
Visualize Multiple Sequence Alignment

- 1 Read a multiple sequence alignment file of the gag polyprotein for several HIV strains.

```
gagaa = multialignread('aagag.aln')
```

- 2 View the aligned sequences in the **Sequence Alignment** app.

```
seqalignviewer(gagaa);
```

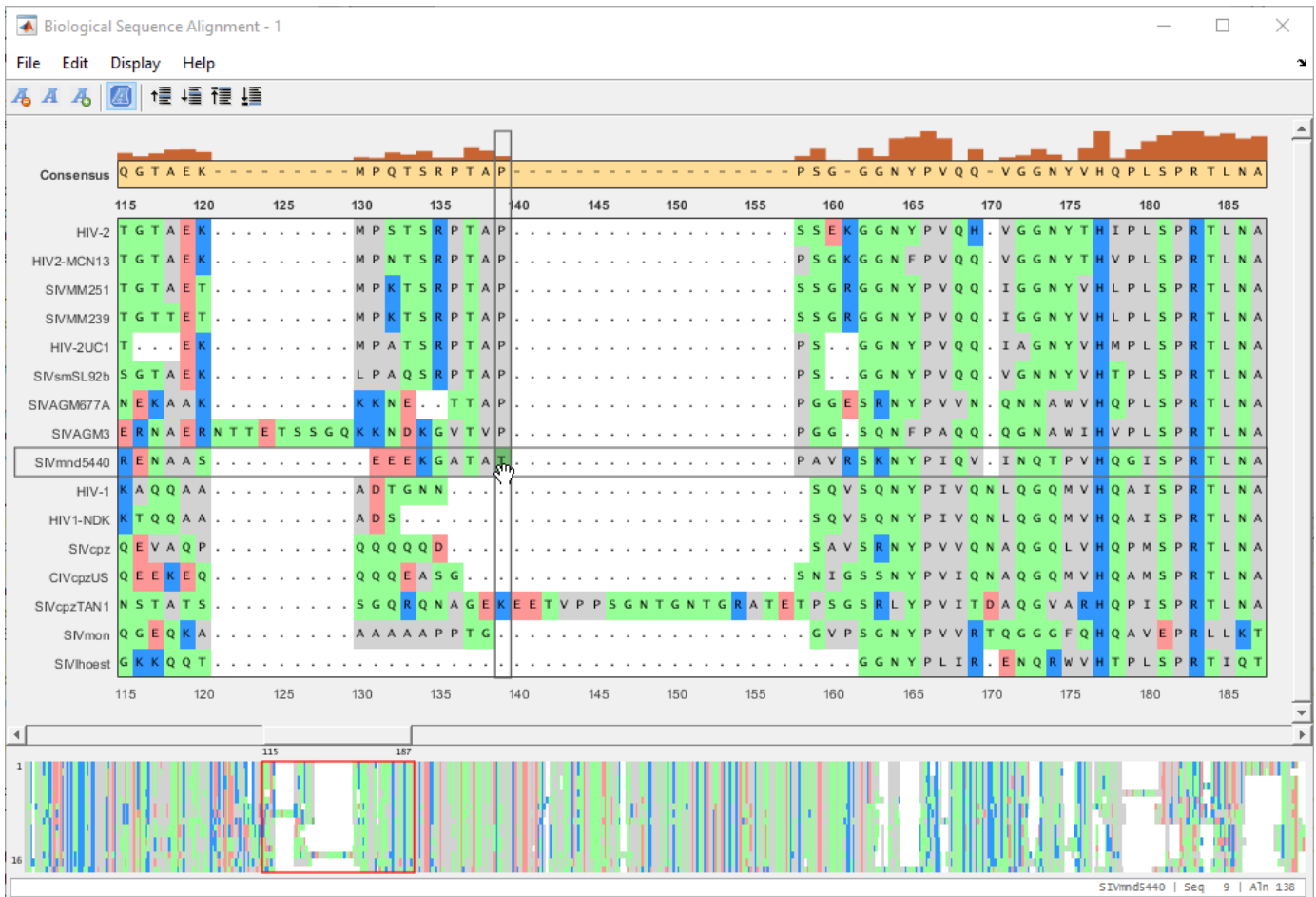


Adjust Sequence Alignments Manually

Algorithms for aligning multiple sequences do not always produce an optimal result. By visually inspecting the alignment, you can identify areas whose alignment can be improved by a manual adjustment.

- 1 To better visualize the sequence alignments, you can zoom in by selecting **Display > Zoom in**. Select this option multiple times until you achieve the zoom level you want.
- 2 Identify an area where you could improve the alignment.

3 Sequence Analysis



- 6 You can also move multiple residues (a subsequence). Suppose you want to move a subsequence to available gaps. First select the gap region that you want to fill in.

Biological Sequence Alignment - 1

File Edit Display Help

Copy Ctrl+C
Delete Sequences
Select All Ctrl+A
Deselect All
Move Rows(s) up
Move Rows(s) down
Move Rows(s) to Top
Move Rows(s) to Bottom
Remove Empty Columns

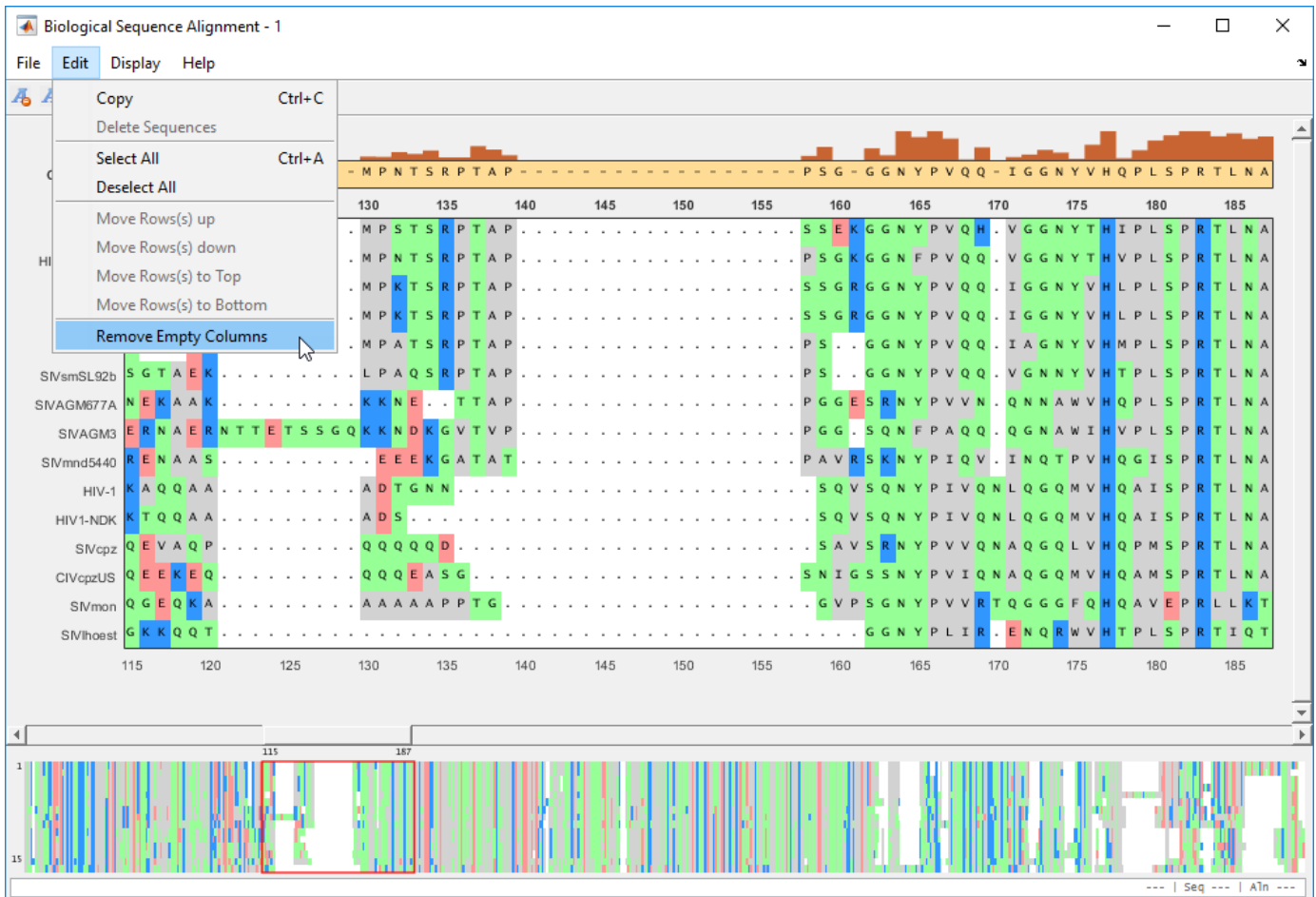
MPQTSRPTAP PSGGGNYPVQQ - VGGNYVHQPLSPRTLNA
130 135 140 145 150 155 160 165 170 175 180 185

HIV-1 MPSTSRPTAP SSEKGGNYPVQH . VGGNYTHIPLSPRTLNA
HIV-1-NDK MPNTSRPTAP PSGKGGNFVPVQQ . VGGNYTHVPLSPRTLNA
SIVsmSL92b MPKTSRPTAP SSGRGGNYPVQQ . IGGNYVHPLSPRTLNA
SIVAGM677A MPKTSRPTAP SSGRGGNYPVQQ . IGGNYVHPLSPRTLNA
SIVAGM3 MPATS RPTAP PS . . . GGNYPVQQ . IAGNYVHMP LSPRTLNA
SIVmnd5440 SGTAEK LPAQSRPTAP PS . . . GGNYPVQQ . VGGNYHTPLSPRTLNA
HIV-1 NEKAAK KKNE . . . TTAP PGGESRNYPVVN . QNNAWVHQPLSPRTLNA
HIV-1-NDK ERNAERNTTETS S GQ KKNDKGVTV P PGG . SQNFPAQQ . QGNAWI HVPLSPRTLNA
SIVcpz RENAAS EEEKGATAT PAVRSKNYPIQV . INQTPVHQGISPRTLNA
SIVcpzUS KAQAQA ADTGNN SQVSQNYPIVQNLQGQMVHQ AISPRTLNA
SIVcpzTAN1 KEVAQP QQQQD SAVSRNYPVVQNAQQQLVHQ PMSRTLNA
SIVmon QEEKEQ QQQEASG SNISSSNYPV IQNAQQQMVHQ AMSRTLNA
SIVhoest NSTATS SGQRQNAGEKEEETVPPSGNTGNTGRATE T P S G S R L Y P V I T D A Q G V A R H Q P I S P R T L N A
115 120 125 130 135 140 145 150 155 160 165 170 175 180 185

1
16

--- | Seq --- | Aln ---

9 Remove empty columns by selecting **Edit > Remove Empty Columns**.



10 After the edit, you can export the aligned sequences or consensus sequence to a FASTA file or MATLAB Workspace from the **File** menu.

Rearrange Rows

You can move the rows (sequences) up or down by one row. You can also move selected rows to the top or bottom of the list.

The screenshot shows the 'Biological Sequence Alignment - 1' window. The 'Edit' menu is open, and the 'Move Rows(s) to Bottom' option is highlighted. The main window displays a sequence alignment grid with columns numbered 115 to 185. The sequences listed on the left are: SIVsmSL92b, SVAGM677A, SIVAGM3, SIVmnd5440, HIV-1, HIV1-NDK, SIVcpz, CIVcpzUS, SIVmon, and SIVhoest. A red box highlights a region in the alignment grid between columns 115 and 185. Below the grid is a color-coded bar representing the alignment, with a red box highlighting a specific region.

The selected sequence moves to the bottom of the list.



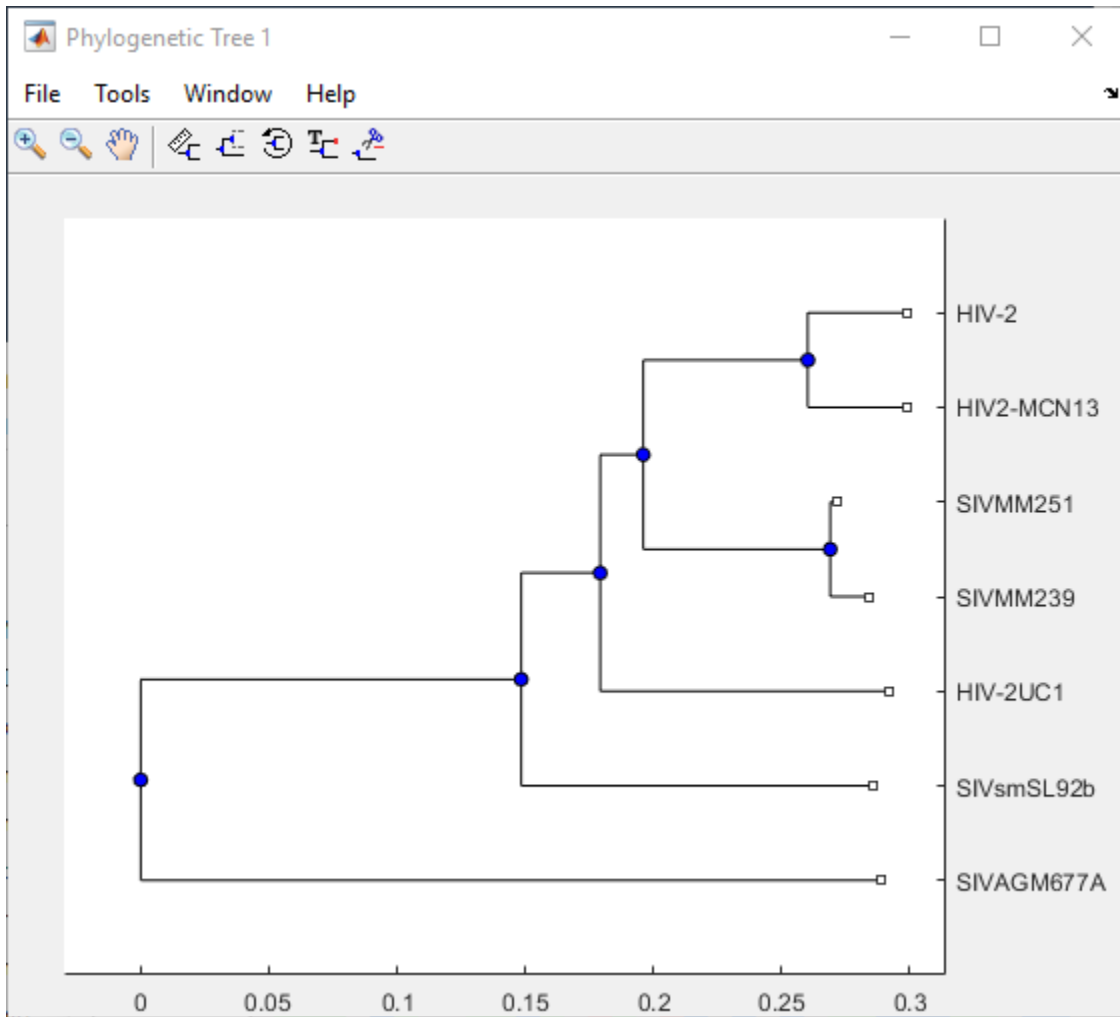
Generate Phylogenetic Tree from Aligned Sequences

You can generate a phylogenetic tree using the aligned sequences from within the app. You can select a subset of sequences or use all the sequences to generate a tree.

Select **Display > View Tree > Selected...** to generate a tree from selected sequences.



A phylogenetic tree for the sequences is displayed in the **Phylogenetic Tree** app. For details on the app, see "Using the Phylogenetic Tree App" on page 5-2.



See Also

NGS Browser | [Sequence Alignment](#) | [Sequence Viewer](#) | [seqalignviewer](#)

More About

- "Sequence Alignments" on page 1-7
- "Aligning Pairs of Sequences"

Microarray Analysis

- “Managing Gene Expression Data in Objects” on page 4-2
- “Representing Expression Data Values in DataMatrix Objects” on page 4-5
- “Representing Expression Data Values in ExptData Objects” on page 4-9
- “Representing Sample and Feature Metadata in MetaData Objects” on page 4-12
- “Representing Experiment Information in a MIAME Object” on page 4-16
- “Representing All Data in an ExpressionSet Object” on page 4-19
- “Visualizing Microarray Images” on page 4-23

Managing Gene Expression Data in Objects

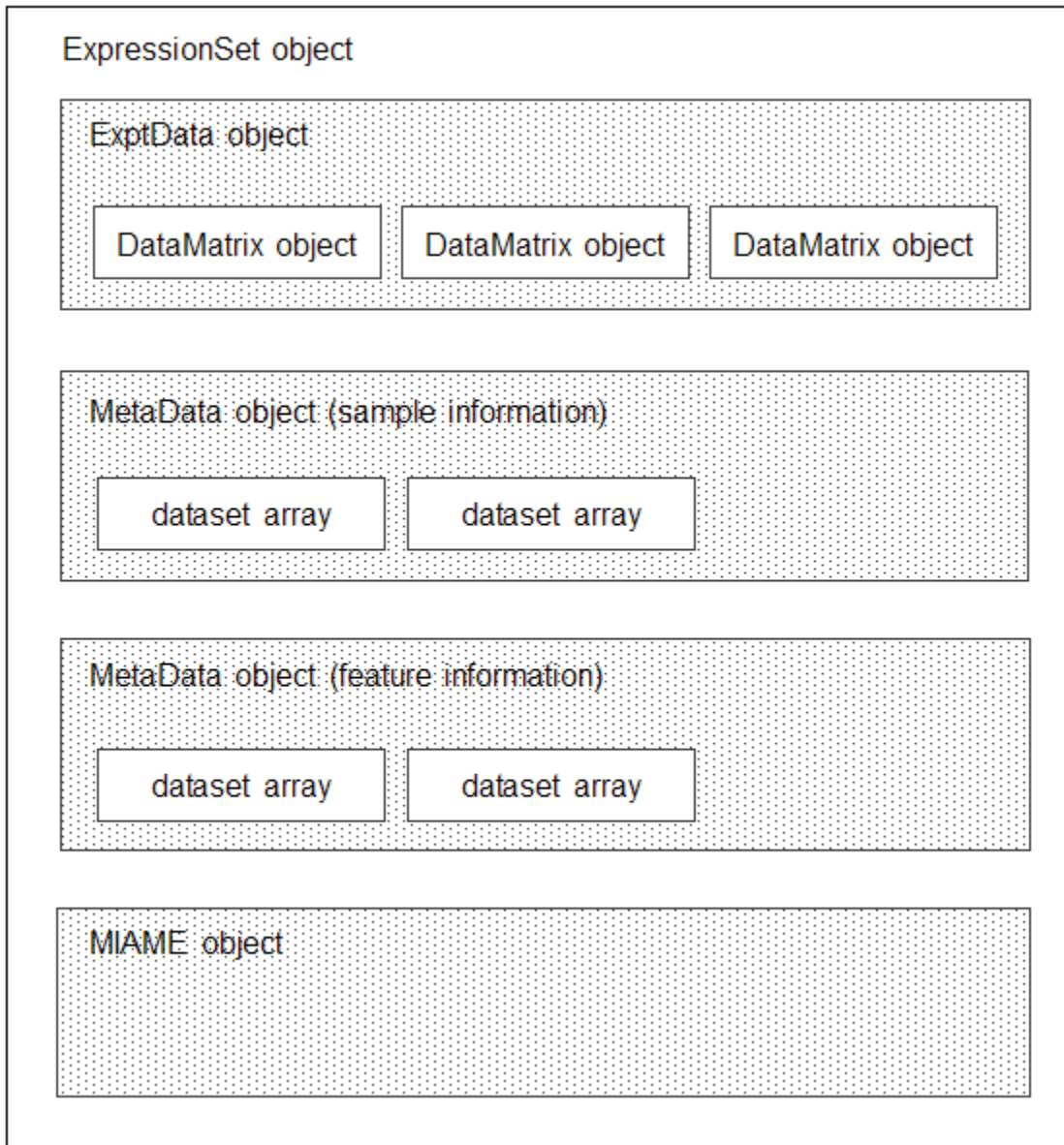
Microarray gene expression experiments are complex, containing data and information from various sources. The data and information from such an experiment is typically subdivided into four categories:

- 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.



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 4-9

- “Representing Sample and Feature Metadata in MetaData Objects” on page 4-12
- “Representing Experiment Information in a MIAME Object” on page 4-16
- “Representing All Data in an ExpressionSet Object” on page 4-19

Representing Expression Data Values in DataMatrix Objects

In this section...

“Overview of DataMatrix Objects” on page 4-5
 “Constructing DataMatrix Objects” on page 4-5
 “Getting and Setting Properties of a DataMatrix Object” on page 4-6
 “Accessing Data in DataMatrix Objects” on page 4-6

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.

```
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” on page 4-9
 “Constructing ExptData Objects” on page 4-9
 “Using Properties of an ExptData Object” on page 4-10
 “Using Methods of an ExptData Object” on page 4-10
 “References” on page 4-11

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” on page 4-12
 “Constructing MetaData Objects” on page 4-13
 “Using Properties of a MetaData Object” on page 4-15
 “Using Methods of a MetaData Object” on page 4-15

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” on page 4-16

“Constructing MIAME Objects” on page 4-16

“Using Properties of a MIAME Object” on page 4-17

“Using Methods of a MIAME Object” on page 4-18

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,,KivelÃ¶
  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 in an ExpressionSet Object

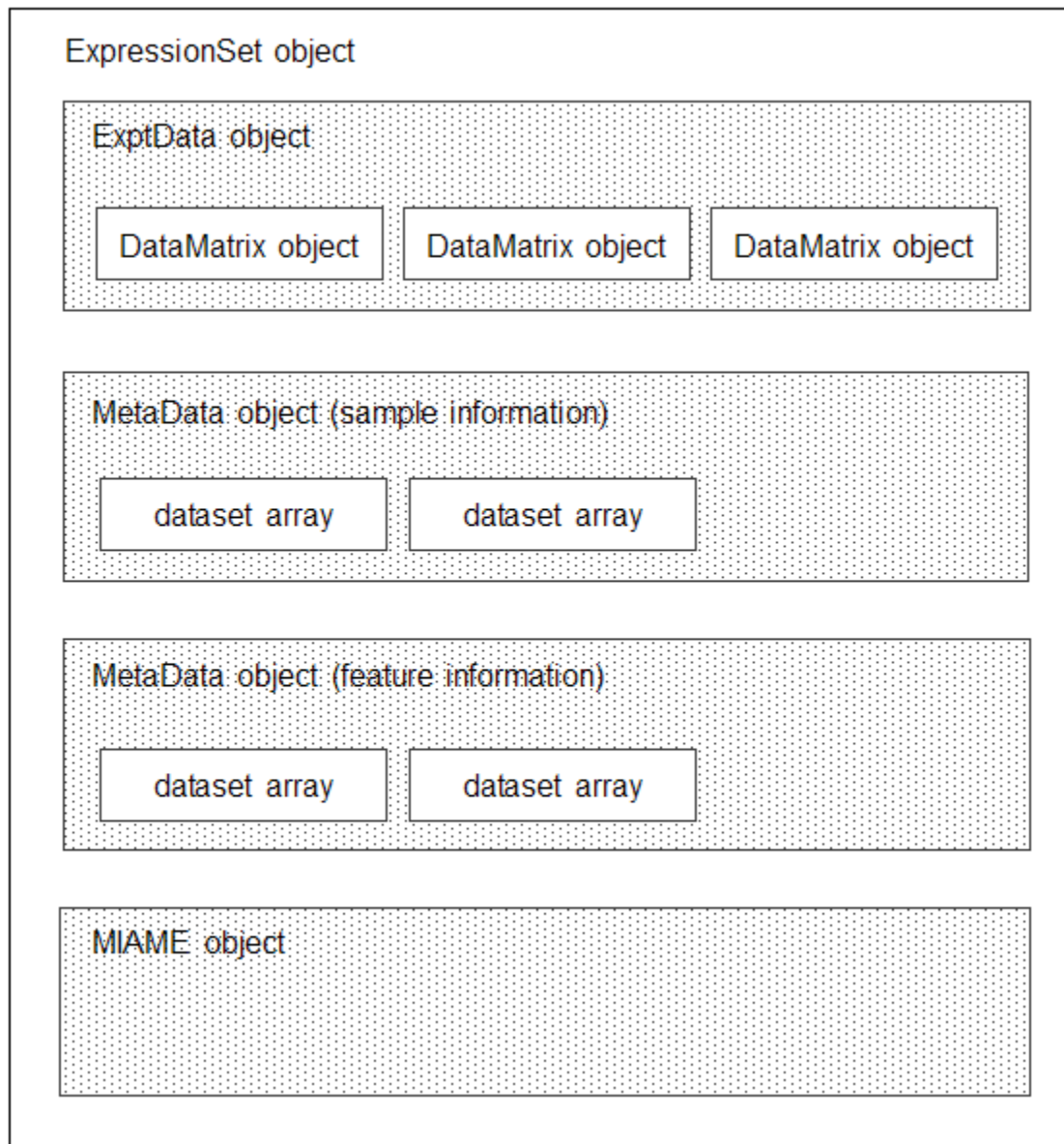
In this section...
“Overview of ExpressionSet Objects” on page 4-19
“Constructing ExpressionSet Objects” on page 4-20
“Using Properties of an ExpressionSet Object” on page 4-21
“Using Methods of an ExpressionSet Object” on page 4-21

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.



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 4-9
- “Representing Sample and Feature Metadata in MetaData Objects” on page 4-12
- “Representing Experiment Information in a MIAME Object” on page 4-16

- 1 Import the `bioma` package so that the `ExpressionSet` 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,,Kivelä
```

```
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.

Visualizing Microarray Images

In this section...

“Overview of the Mouse Example” on page 4-23
 “Exploring the Microarray Data Set” on page 4-23
 “Spatial Images of Microarray Data” on page 4-25
 “Statistics of the Microarrays” on page 4-33
 “Scatter Plots of Microarray Data” on page 4-34

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_alpd.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

<https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE30>

The GenePix GPR-formatted file `mouse_alpd.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 4-23.

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

```
pd = gprread('mouse_alpd.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 R2'
'F Pixels'
'B Pixels'
'Sum of Medians'
'Sum of Means'
'Log Ratio'
'F635 Median - B635'
'F532 Median - B532'
'F635 Mean - B635'
'F532 Mean - B532'
'Flags'
```

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

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

A list of the first 20 gene names is displayed:


```

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

```

Spatial Images of Microarray Data

This procedure illustrates how to visualize microarray data by plotting image maps. The function `mimage` 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, `mimage` 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 4-23.

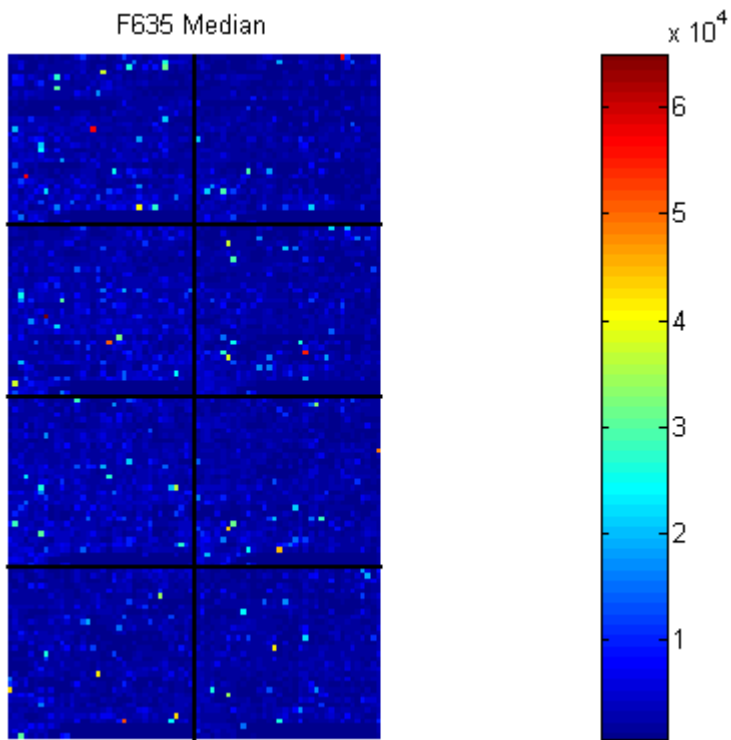
- 1 Plot the median values for the red channel. For example, to plot data from the field `F635` Median, type

```

figure
mimage(pd, 'F635 Median')

```

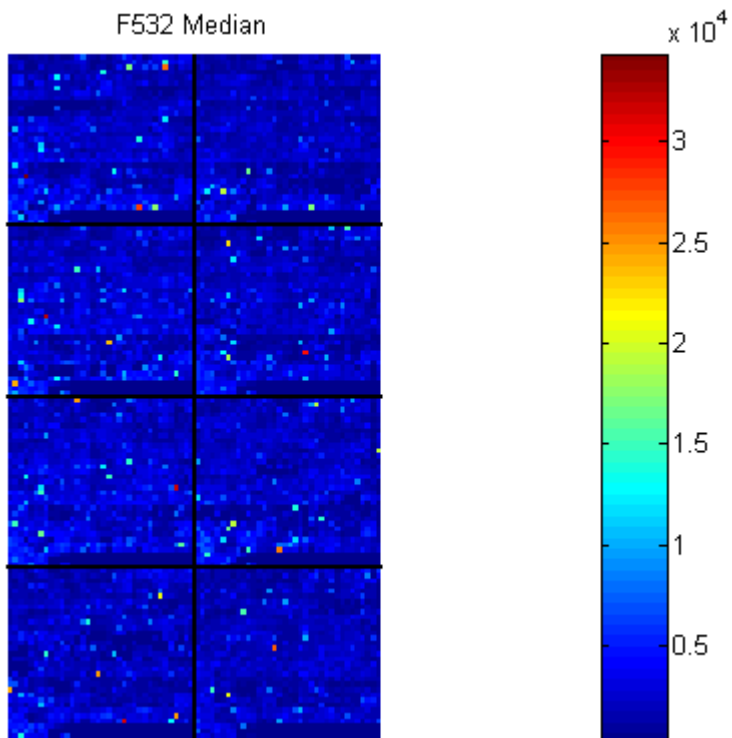
The MATLAB software plots an image showing the median pixel values for the foreground of the red (Cy5) channel.



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

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

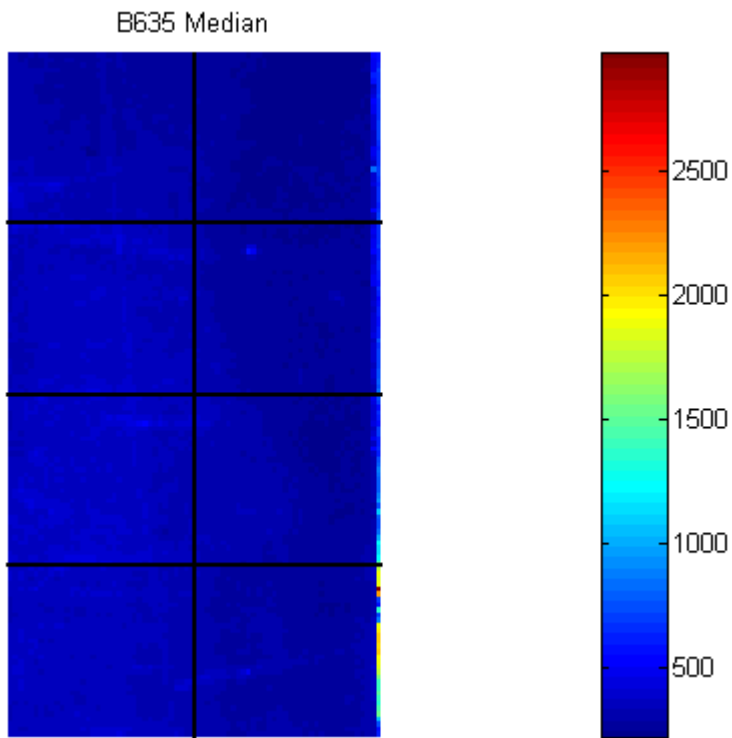
The MATLAB software plots an image showing the median pixel values of the foreground of the green (Cy3) channel.



- 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  
mimage(pd, 'B635 Median')
```

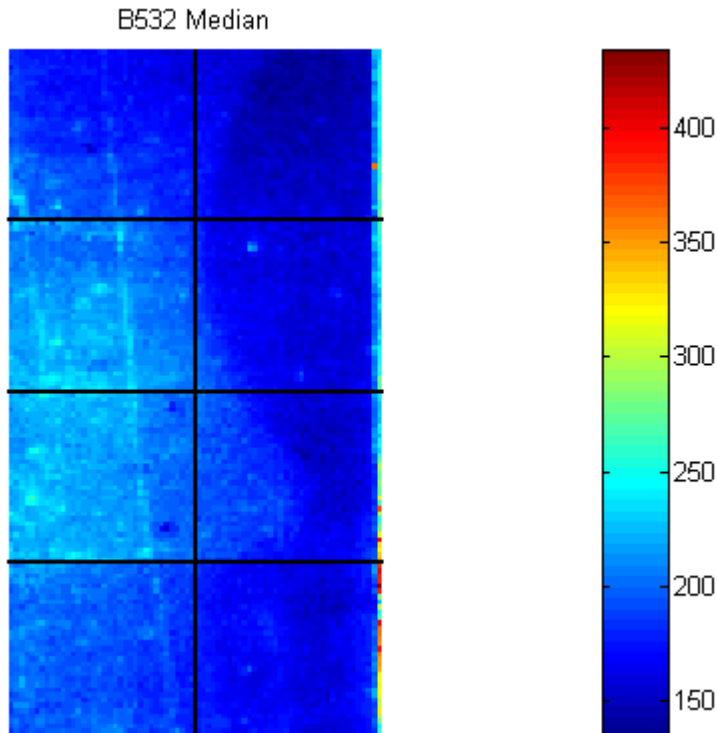
The MATLAB software plots an image for the background of the red channel. Notice the very high background levels down the right side of the array.



- 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  
mimage(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_alwt.gpr')
```

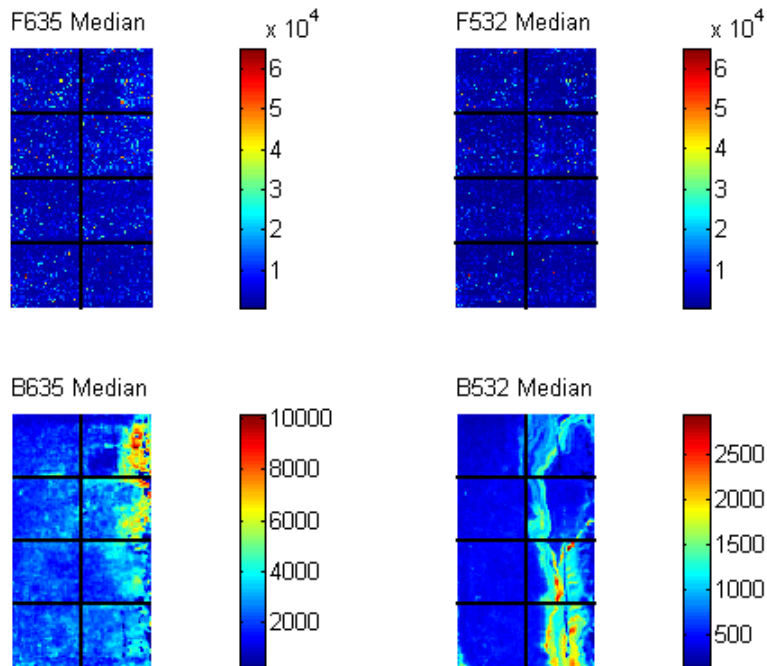
The MATLAB software creates a structure and displays information about the structure.

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

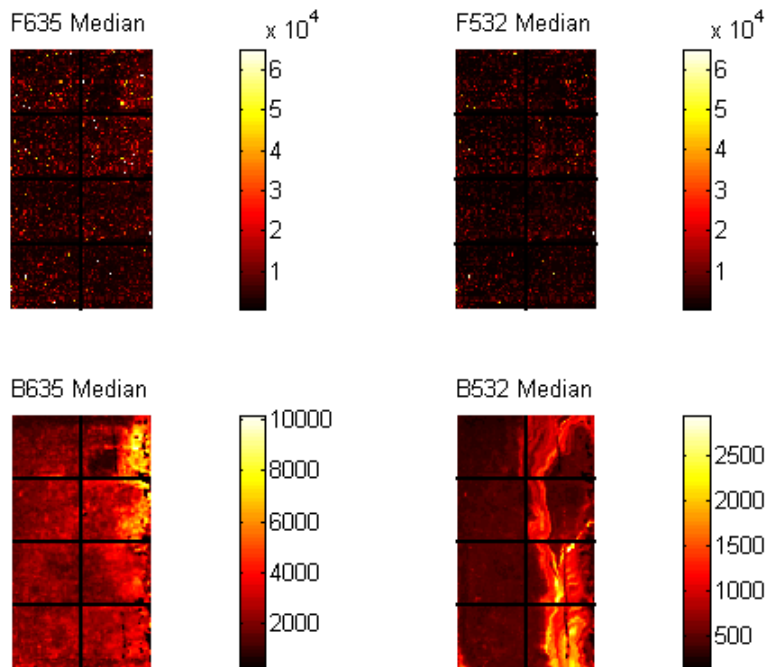
The MATLAB software plots the images.



- 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 `colormap editor` function.

`colormap hot`

The MATLAB software plots the images.



- 8** The function `mimage` 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
```

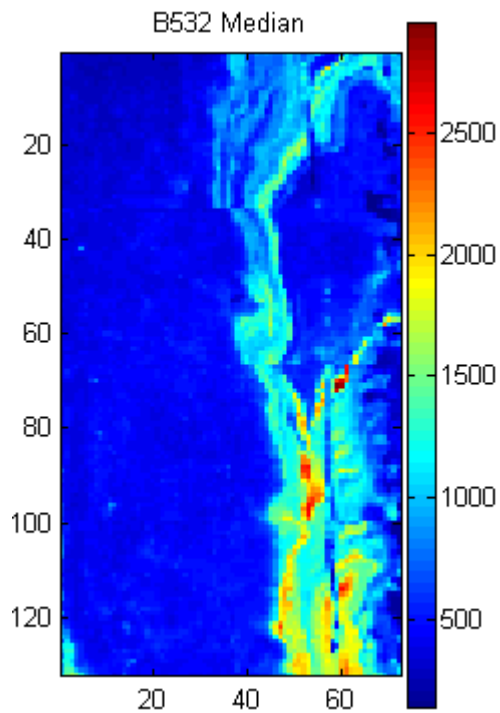
- 9** Extract that column from the field `Data`.

```
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.

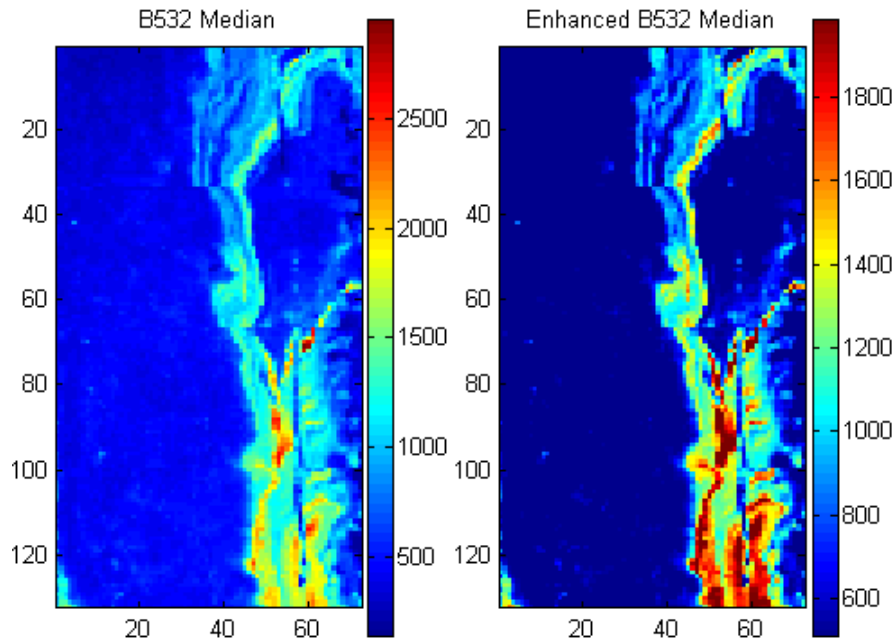


- 11** Bound the intensities of the background plot to give more contrast in the image.

```
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.



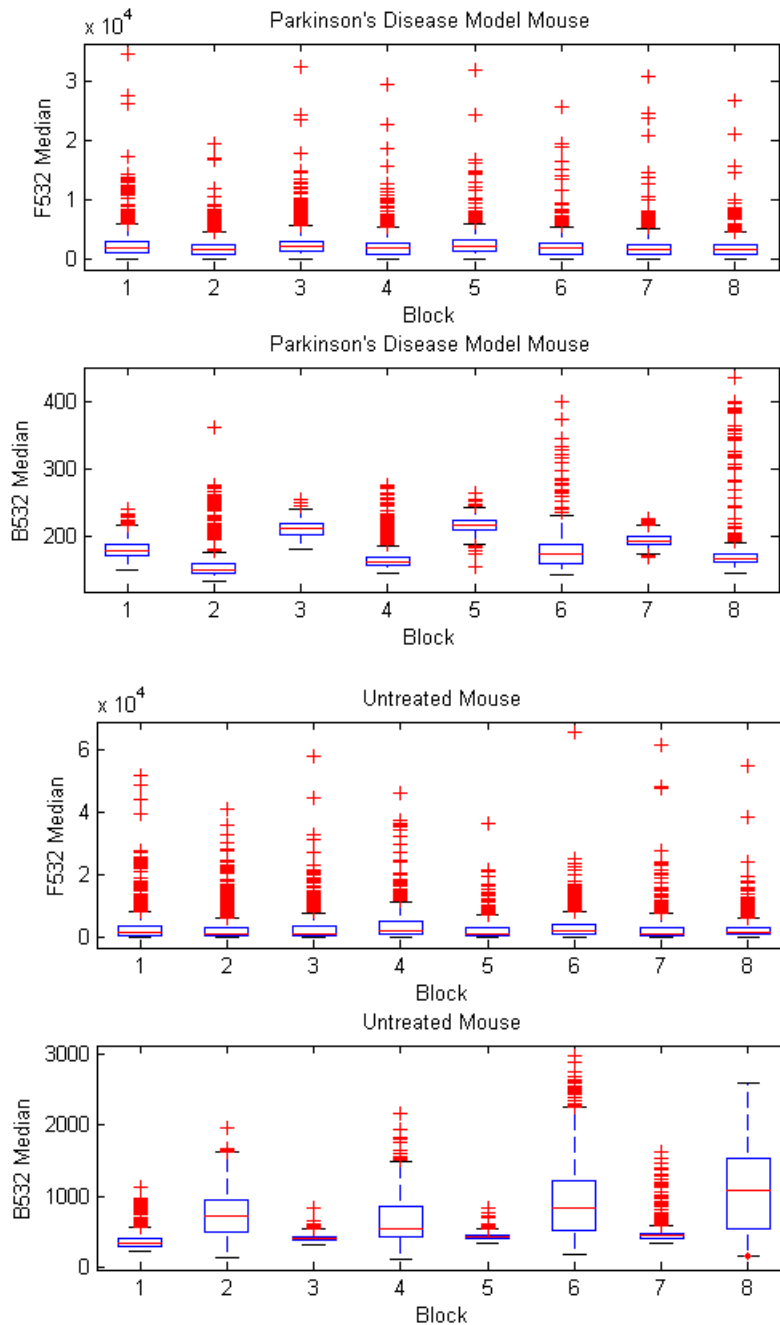
Statistics of the Microarrays

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.



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
```

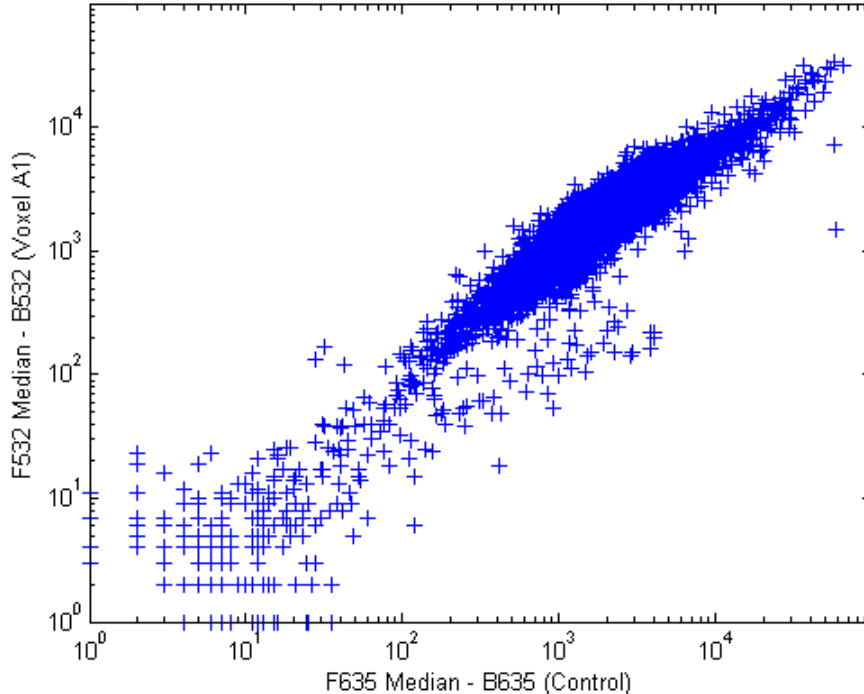
```
cy3DataCol =
    35
```

- 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.)
```

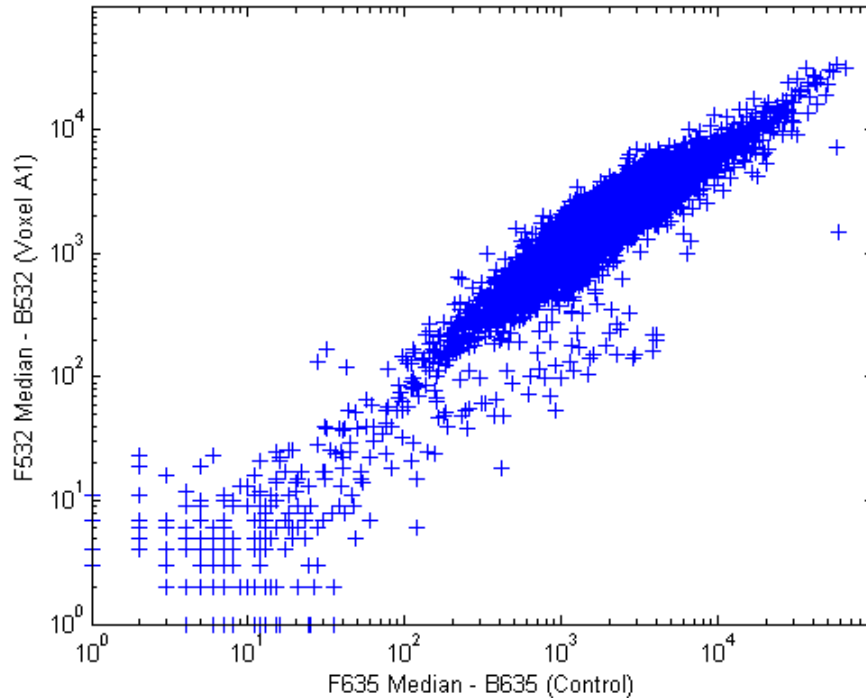


Notice that this function gives some warnings about negative and zero elements. This is because some of the values in the 'F635 Median - B635' and 'F532 Median - B532' columns are zero or even less than zero. Spots where this happened might be bad spots or spots that failed to hybridize. Points with 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)');
```

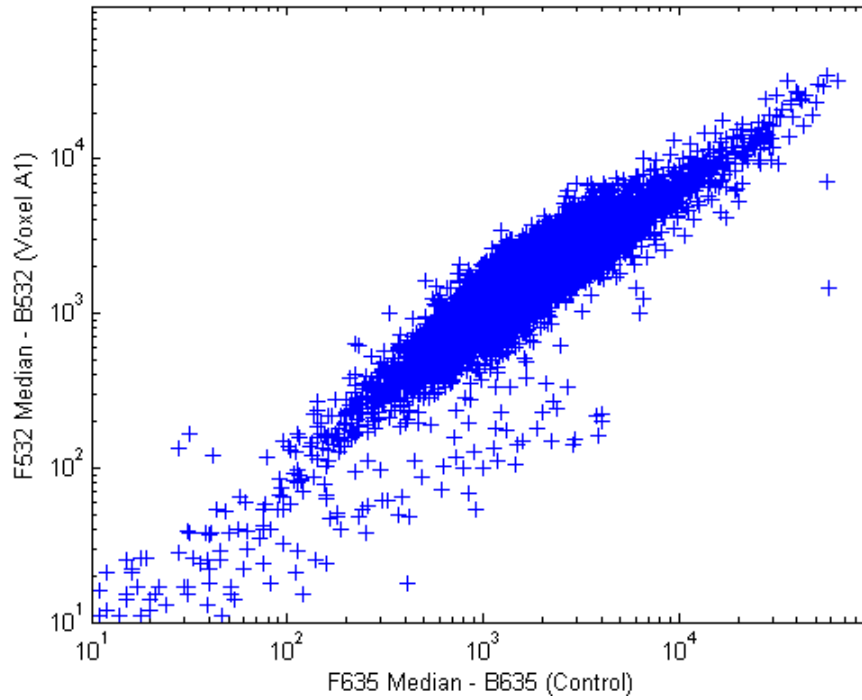
The MATLAB software plots the image.



- 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);
```

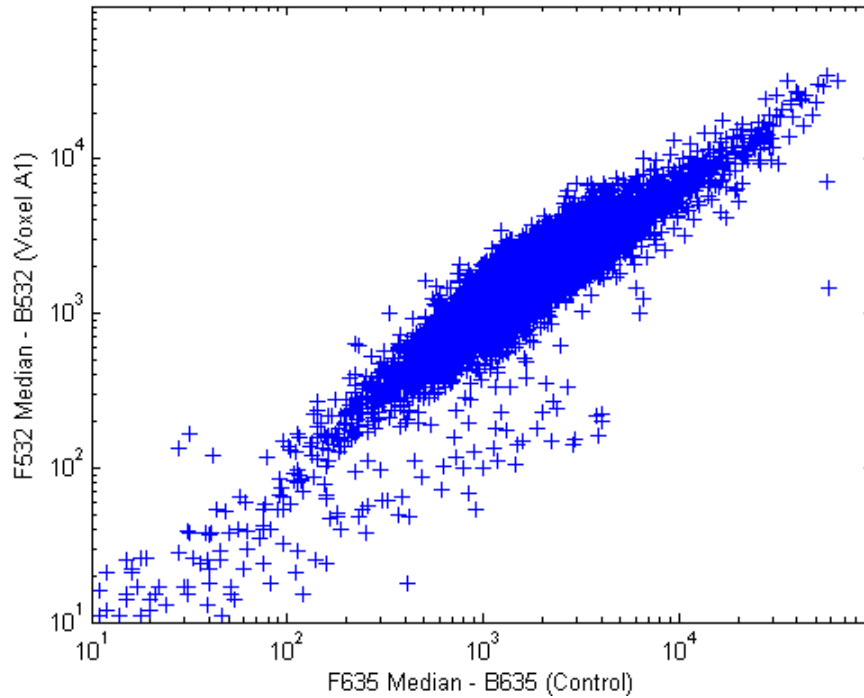
The MATLAB software plots the image.



- 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.

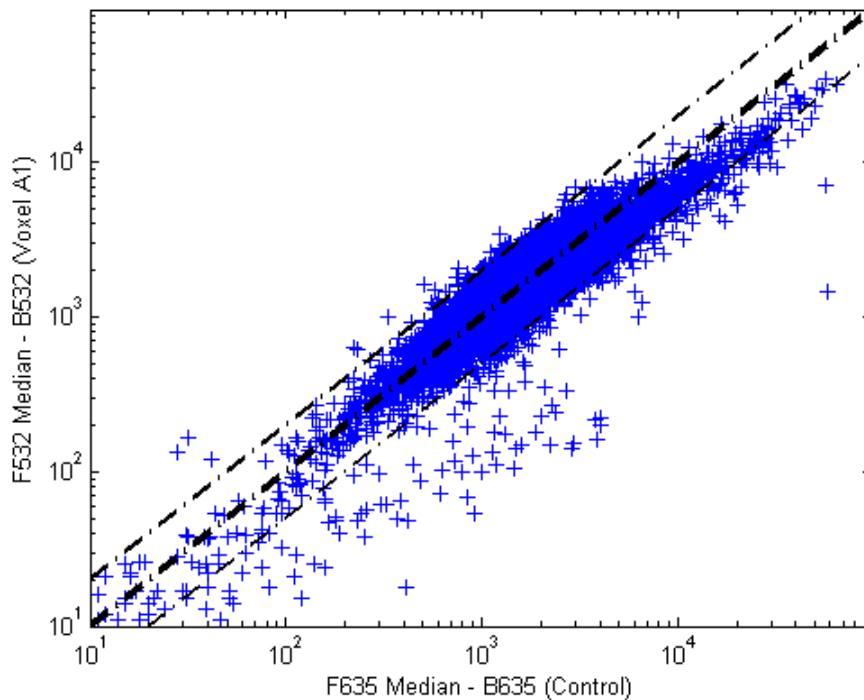


This plot shows the distribution of points but does not give any indication about which genes correspond to which points.

- 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.

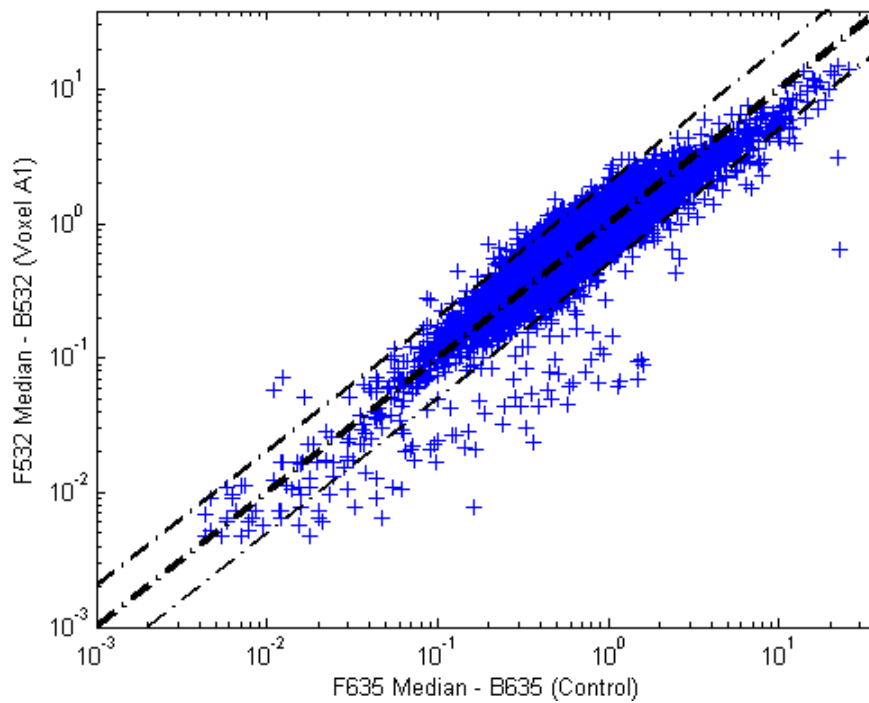
- 8 Normalize the points to evenly distribute them on either side of the line. Use the function `manorm` to perform global mean normalization.

```
normcy5 = manorm(cy5Data);
normcy3 = manorm(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)');
```

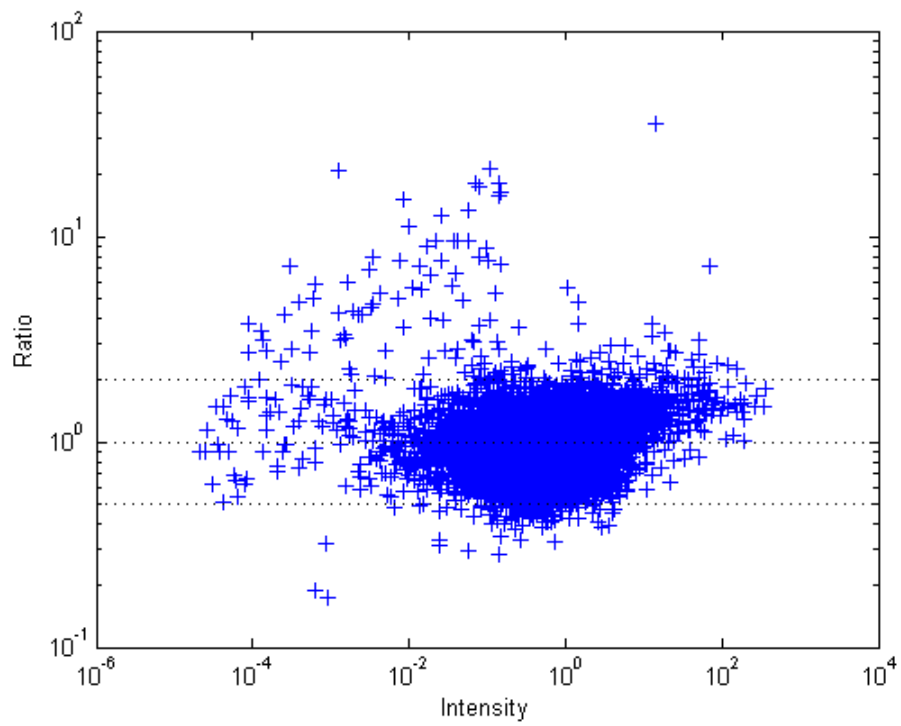
The MATLAB software plots the image.



- 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.

Phylogenetic Analysis

Using the Phylogenetic Tree App

In this section...

“Overview of the Phylogenetic Tree App” on page 5-2

“Opening the Phylogenetic Tree App” on page 5-2

“File Menu” on page 5-3

“Tools Menu” on page 5-11

“Window Menu” on page 5-17

“Help Menu” on page 5-18

Overview of the Phylogenetic Tree App

The Phylogenetic Tree app allows you to view, edit, format, and explore phylogenetic tree data. With this app you can prune, reorder, rename branches, and explore distances. You can also open or save Newick or ClustalW tree formatted files. The following sections give a description of menu commands and features for creating publishable tree figures.

Opening the Phylogenetic Tree App

This section illustrates how to draw a phylogenetic tree from data in a `phytree` object or a previously saved file.

The Phylogenetic Tree app 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')
```

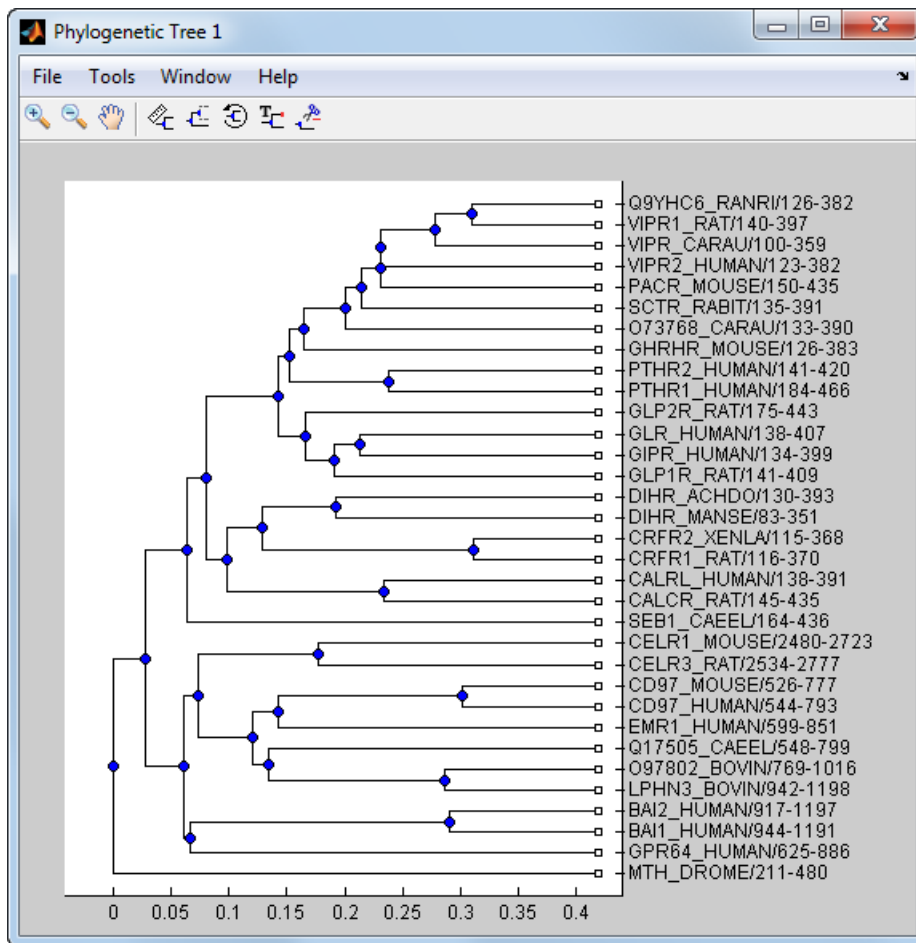
The MATLAB software creates a `phytree` object.

```
Phylogenetic tree object with 33 leaves (32 branches)
```

- 2 View the phylogenetic tree using the app.

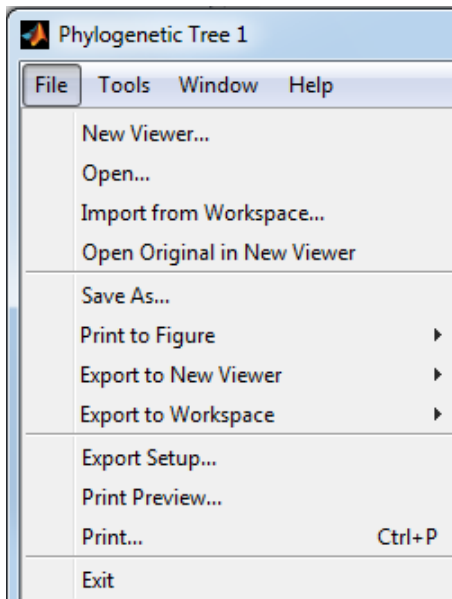
```
phytreeviewer(tr)
```

Alternatively, click **Phylogenetic Tree** on the **Apps** tab.



File Menu

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

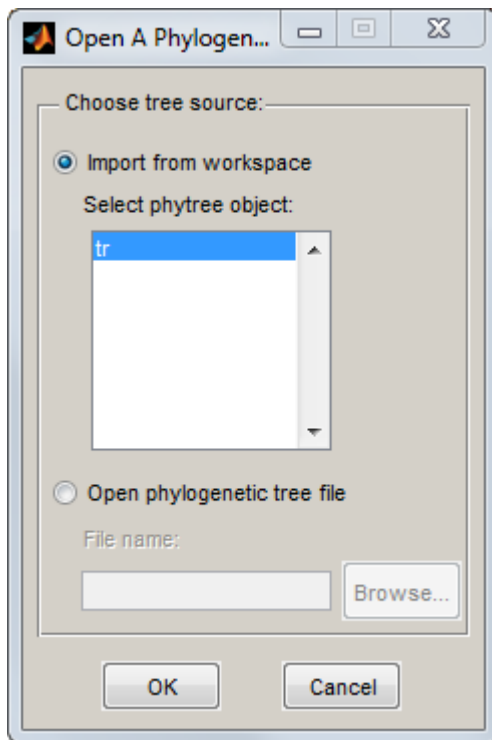


New Viewer Command

Use the **New Viewer** command to open tree data from a file into a second Phylogenetic Tree viewer.

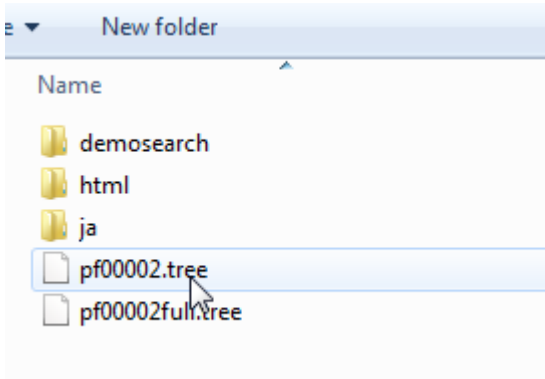
- 1 From the **File** menu, select **New Viewer**.

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.



A second Phylogenetic Tree viewer 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 the app.

- 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 app uses the file extension `.tree` for Newick-formatted files, but you can use any Newick-formatted file with any extension.

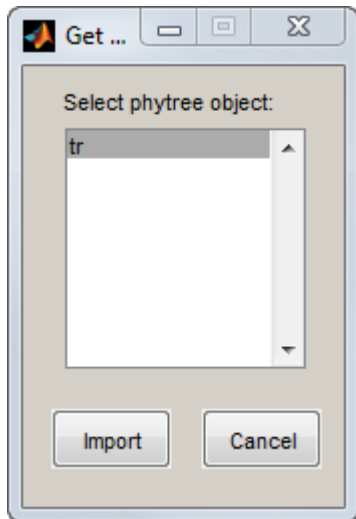
The app replaces the current tree data with data from the selected file.

Import from Workspace Command

Use the **Import from Workspace** command to read tree data from a `phytree` object in the MATLAB Workspace and display the data using the app.

- 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 app replaces the current tree data with data from the selected object.

Open Original in New Viewer

There may be times when you make changes that you would like to undo. The **Phylogenetic Tree** app does **not** have an undo command, but you can get back to the original tree you started viewing with the **Open Original in New Viewer** command.

From the **File** menu, select **Open Original in New Viewer**.

A new Phylogenetic Tree viewer 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**.

The app 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 Viewer Command

Because some of the Phylogenetic Tree viewer 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 Viewer**, and then select either **With Hidden Nodes** or **Only Displayed**.

A new Phylogenetic Tree viewer opens with a copy of the tree.

- 2 Use the new figure to continue your analysis.

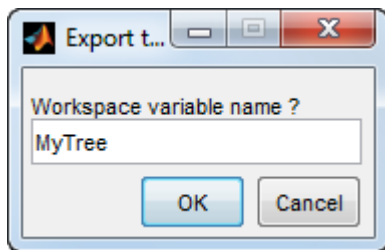
Export to Workspace Command

The **Phylogenetic Tree** app can open Newick-formatted files with tree data. However, it does not create a `phyt tree` 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`.



- 3 Click **OK**.

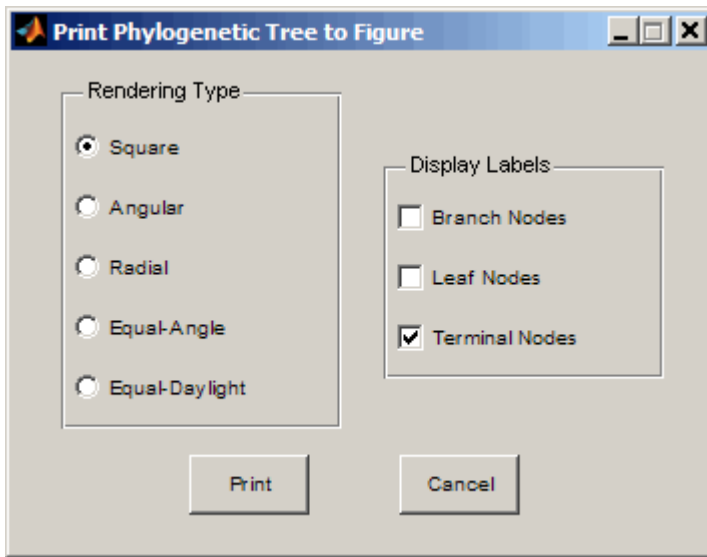
The app creates a `phyt tree` object in the MATLAB Workspace.

Print to Figure Command

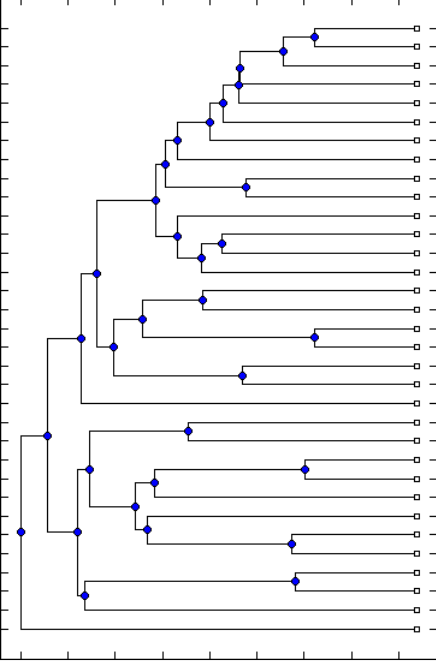
After you have explored the relationships between branches and leaves in your tree, you can copy the tree to a MATLAB Figure window. Using a Figure window lets you use all the features for annotating, changing font characteristics, and getting your figure ready for publication. Also, from the Figure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

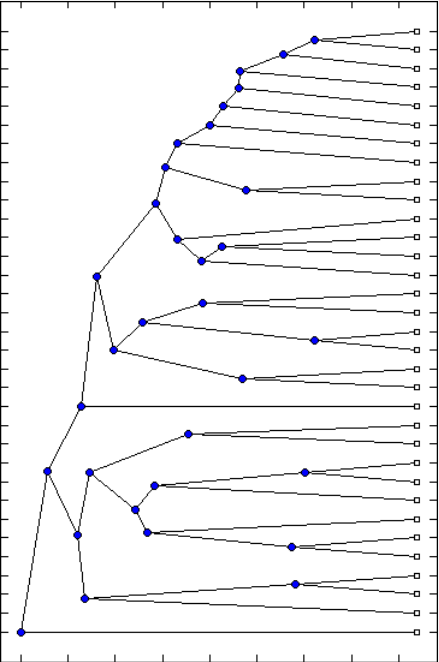
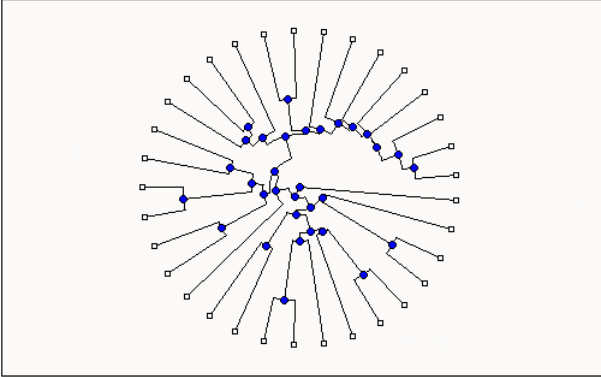
- 1 From the **File** menu, select **Print to Figure**, and then select either **With Hidden Nodes** or **Only Displayed**.

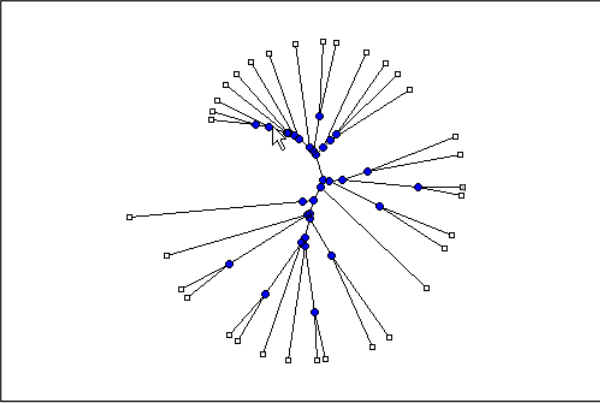
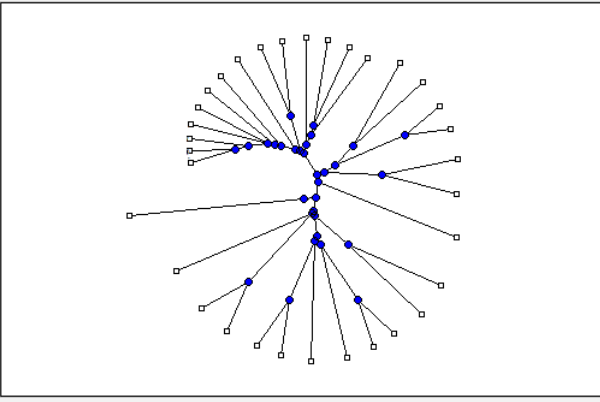
The **Print Phylogenetic Tree to Figure** dialog box opens.



- 2 Select one of the **Rendering Types**.

Rendering Type	Description
'square' (default)	

Rendering Type	Description
'angular'	 An angular phylogenetic tree rendering. The tree is oriented vertically with the root at the bottom left. The branches extend upwards and to the right, forming a fan-like shape. The tips of the branches are marked with small blue circles, and the terminal nodes are marked with small white squares. The tree is set against a background with a vertical axis on the left and a horizontal axis at the bottom, both with tick marks.
'radial'	 A radial phylogenetic tree rendering. The tree is oriented horizontally with the root at the center. The branches extend outwards in all directions, forming a circular or fan-like shape. The tips of the branches are marked with small blue circles, and the terminal nodes are marked with small white squares. The tree is set against a plain white background.

Rendering Type	Description
'equalangle'	 <p>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.</p>
'equaldaylight'	 <p>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.</p>

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.

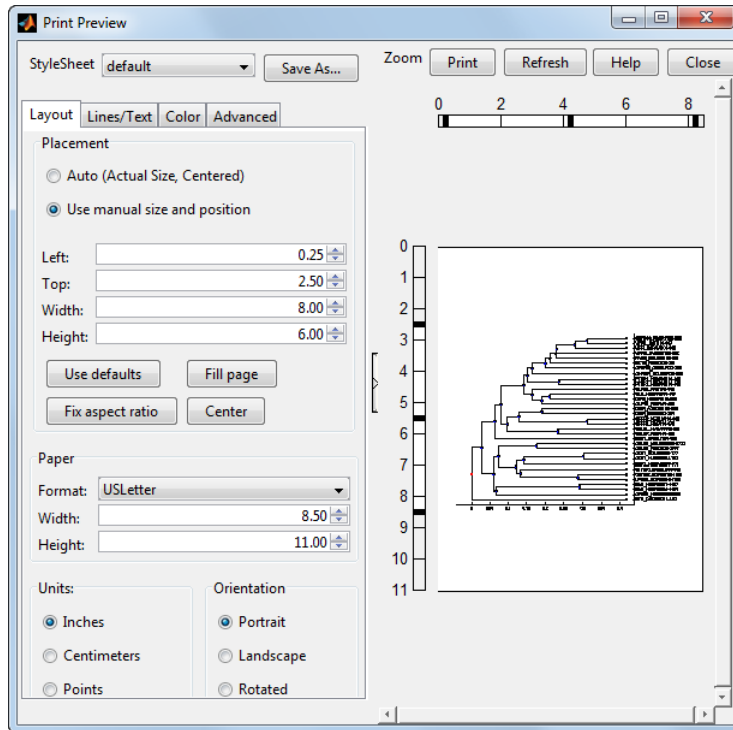
A new Figure window opens with the characteristics you selected.

Print Preview Command

When you print from the **Phylogenetic Tree** app or a MATLAB Figure window (with a tree published from the viewer), you can specify setup options for printing a tree.

- 1 From the **File** menu, select **Print Preview**.

The **Print Preview** window opens, which you can use to select page formatting options.



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

Print Command

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

- 1 From the **File** menu, select **Print**.

The **Print** dialog box opens.

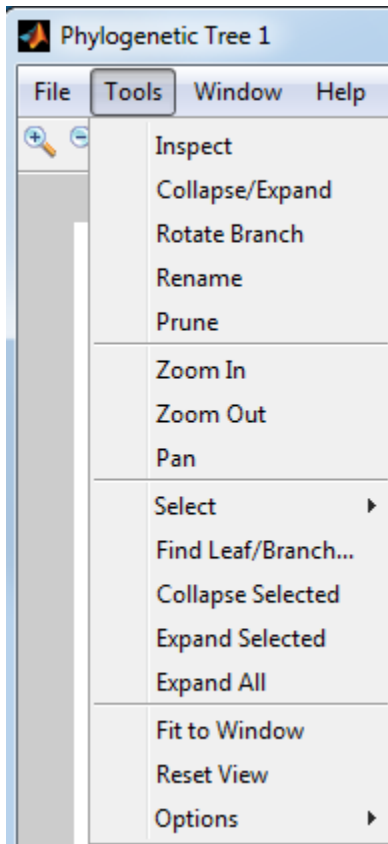
- 2 From the **Name** list, select a printer, and then click **OK**.

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

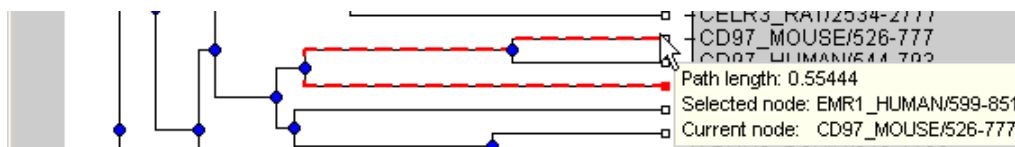
Viewing a phylogenetic tree in the **Phylogenetic Tree** app provides a rough idea of how closely related two sequences are. However, to see exactly how closely related two sequences are, measure the distance of the path between them. Use the **Inspect** command to display and measure the path between two sequences.

- 1 Select **Tools > Inspect**, or from the toolbar, click the **Inspect Tool Mode** icon .

The app 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 app 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.



Collapse and Expand Branch Mode

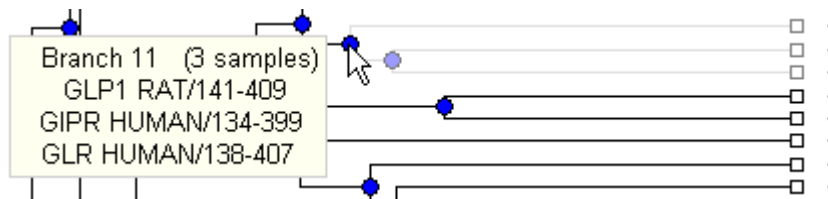
Some trees have thousands of leaf and branch nodes. Displaying all the nodes can create an unreadable tree diagram. By collapsing some branches, you can better see the relationships between the remaining nodes.

- 1 Select **Tools > Collapse/Expand**, or from the toolbar, click the **Collapse/Expand Branch Mode** icon .

The app is set to collapse/expand mode.

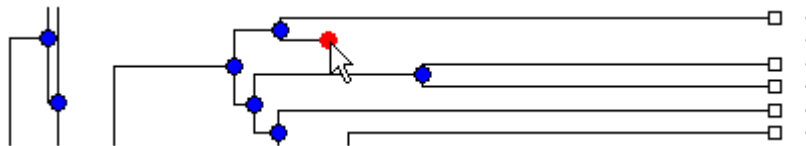
- 2 Point to a branch.

The paths, branch nodes, and leaf nodes below the selected branch appear in gray, indicating you selected them to collapse (hide from view).



- 3 Click the branch node.

The app 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**.

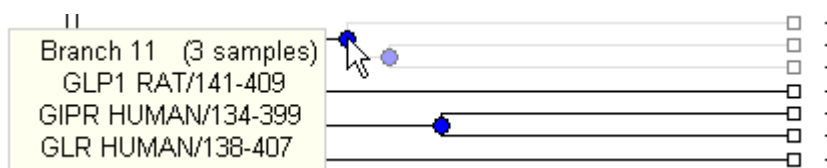
Rotate Branch Mode

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.

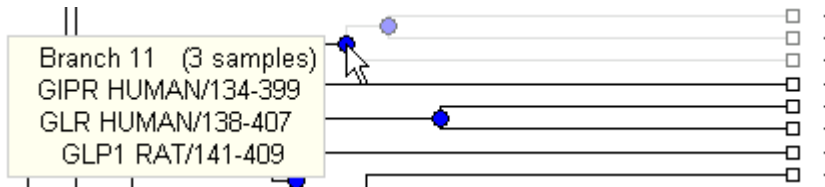
- 1 Select **Tools > Rotate Branch**, or from the toolbar, click the **Rotate Branch Mode** icon .

The app is set to rotate branch mode.

- 2 Point to a branch node.



- 3 Click the branch node.




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.

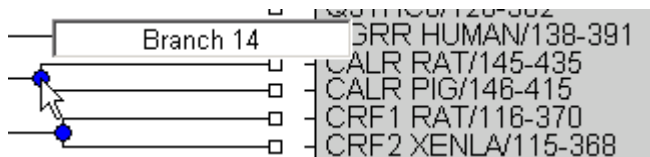
Rename Leaf or Branch Mode

The **Phylogenetic Tree** app takes the node names from a phyt tree 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 .

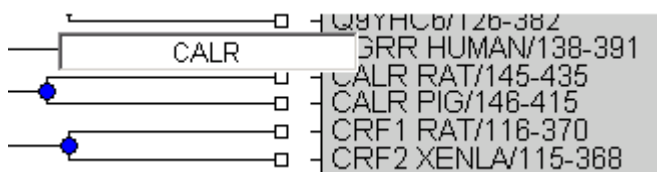
The app is set to rename mode.

- 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**.

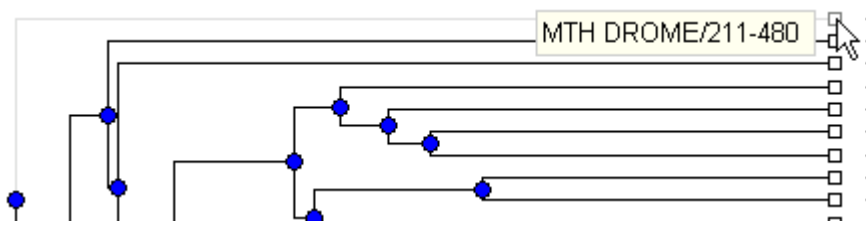
Prune (Delete) Leaf or Branch Mode

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 app 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 app 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.

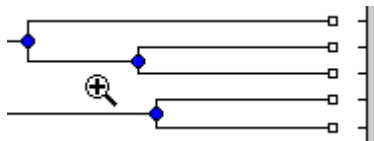
Tip After pruning nodes, you can redraw the tree by selecting **Tools > Fit to Window**.

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.

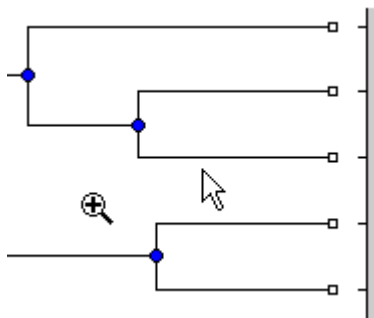
- 1 Select **Tools > Zoom In**, or from the toolbar, click the **Zoom In** icon .

The app 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
- **Propagate 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**

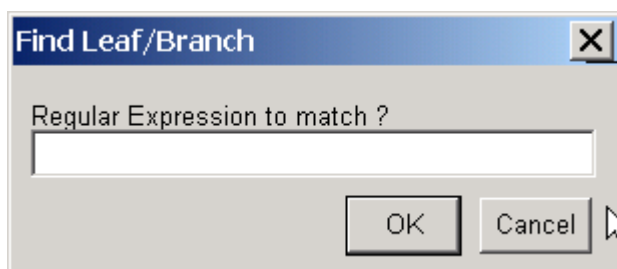
Clear all selected nodes by clicking anywhere else in the Phylogenetic Tree app.

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 **Phylogenetic Tree** app reference.

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