Signals & Systems – Handout #1
MATLAB – A Brief Introduction

H-1.1 Entering (Complex) Numbers, Variables, and Strings:

The two most important data types in `MATLAB` are strings and double precision (real or complex) numbers. Predefined numbers in `MATLAB` are `pi` and the imaginary units `j` and/or `i`. Special numbers are `nan` (not a number, e.g. as the result of a `0/0` operation) and `inf` (infinity, e.g. as the result of a `1/0` operation). Simple examples of number constants are:

```
>> 1.3e-10, 3.0+5.9i, pi*pi, j*3.9, 5*i-7, nan+j*nan, -inf
```

Strings are entered between delimiting `'`-characters:

```
>> 'String Example', 'abcdefghijklmnop', '1234567890', '()[]{}'
```

Variables do not need to be declared and may be of any data type. Variable names consist of a letter, followed by any number of letters, digits, or underscores. `MATLAB` uses only the first 31 characters of a variable name. `MATLAB` is case sensitive; it distinguishes between uppercase and lowercase letters. `A` and `a` are *not* the same variable.

```
>> String_Var = 'String Example', Number = pi + j*5.0
```

To view the value assigned to any variable, simply enter the variable name.

H-1.2 Entering Elementary Matrices and Vectors:

A matrix is entered in `MATLAB` with delimiting `[ ]`-characters. Values are entered row-by-row. Rows are separated by a `;`-character. Elements within rows may (optionally) be separated by a `,`-character:

```
>> A = [ 1 , 2 , 3 , 4 ; 4 3 -2 1 ; 4 3 1 2 ; 2 4 3 1 ]
```

The transpose of matrix `A` is computed with `A.'`. The hermitian transpose (i.e. conjugate transpose) is computed via `A'`. The diagonal elements of the matrix can be extracted with `diag(A)`. A vector can be defined as a row vector `rv` or a column vector `cv`:

```
>> rv = [ 1 2 3 4 ], cv = [ 1 ; 2 ; 3 ; 4 ]
```

Matrix multiplication is accomplished with the `*` operator, e.g. `A*A*cv`. Matrix inversion is accomplished with the command `inv(A)`. The determinant of a matrix can be computed with `det(A)`. Matrices and vectors can be flipped from left to right with `fliplr(A)`. Flipping matrices and vectors from up to down can be accomplished with `flipud(A)`.

Row vectors with incremental changes from element to element can be conveniently defined with the `:`-operator. In order to generate the vector `[ 1 2 3 4 5 6 7 ]` we can simply type `1:7`. Row vectors with increments other than +1 can be generated with the double-`:`-notation. The term `1:-0.2:0` generates the vector `[ 1 0.8 0.6 0.4 0.2 0 ]`.

An alternative way of generating row vectors with incremental changes from element to element is through the command `linspace`. A row vector with `N` uniformly distributed elements between `a` and `b` can be generated with `linspace(a,b,N)`. For example the command `linspace(0,1,5)` generates the vector `[ 0 0.25 0.5 0.75 1 ]`.

Elements of matrices and vectors are accessible via the `()` notation. Considering the matrix `A` from above we can access the element `-2` in the second row and third column with `A(2,3)`. Changing elements in a matrix can be accomplished with an assignment similar to `A(2,3)=+2` for example. Sub-matrices and/or vectors can be generated with calls like `A([ 1 3 ],[ 2 3 ])` which extracts a new 2×2 matrix from the intersection of rows 1 and 3 with columns 2 and 3. We can change multiple elements at a time with assignment like

```
>> A([ 1 3 ],[ 2 3 ]) = [ 1 1 ; 2 2 ]
```

The size (i.e. dimensions) of a matrix can be found with the `size(A)` command.

Access to elements of vectors can be done with a single index within `()`. Consider the column vector `cv` from above. We can get access to its third element simply with `cv(3)`. The length of a vector can be found with `length(cv)`. `MATLAB` also defines a *single index access* for matrices. The term `A([1:16]')`, for example, strings out all 16 elements of the matrix `A` from above into one long column vector. The same is accomplished with the simpler notation `A(:)`. Note that a *single index access* to a matrix counts through the matrix in a column-by-column fashion.

The concatenation of vectors and matrices can be accomplished by extension of the `[,;]` notation with the `,`-character and the `;`-character. A horizontal concatenation of matrix `A` from above, for example, is written as `[ A , A ]`. Similarly, a vertical concatenation is written as `[ A ; A ]`.

There are a few `MATLAB` commands for the convenient generation of some elementary vectors and/or matrices. The command `eye(5)` for example generates a 5×5 identity matrix. The command `ones(2,3)` generates a 2×3 matrix filled with `1`'s. Similarly, the command `zeros(2,3)` generates a 2×3 matrix filled with `0`'s. Commands `rand` and `randn` can be used to likewise generate matrices and/or vectors with random numbers. The resulting random numbers are uniformly distributed between 0 and 1 for `rand` and normally distributed with mean 0 and variance 1 for `randn`.

H-1.3 Getting Help:

A very comprehensive help window can be opened from within `MATLAB` by entering the command:

```
>> helpdesk
```

The `helpdesk` window provides access to a complete list of all `MATLAB` commands as well as a very well written *Getting Started* section. Specific information about any `MATLAB` command or function can be directly obtained via the `help` command. If we want to receive help on the use of the function `reshape` for example, we can simply type:

```
>> help reshape
```

A list of more generic help topis is printed on the screen if we just type `help` without any keyword of function name. Getting help on `MATLAB` operator symbols is, unfortunately, not possible by putting the operator symbol after the help command. Instead, one has to follow the help command with an *operator keyword.* If we want to get some information on the + operator, for example, then we must enter help plus. A list of the keywords for the MATLAB operator symbols can be listed by typing help ops.

H-1.4 Elementary Operators:

The most important arithmetic operators in MATLAB are + (addition), - (subtraction), * (multiplication), / (division), and ^ (power). When applied to scalar numbers then these operators work in the usual way. MATLAB also allows the use of parenthesis ().

In addition, MATLAB lets us use these operators on vectors and matrices. In this case the symbols +, -, *, /, and ^ mean: matrix addition, matrix subtraction, matrix multiplication, matrix "division", and matrix "power" respectively. (With the exception of + and -) these "matrix" style operations are generally quite different from the frequently also needed "element-by-element" style operation. Consider the two vectors a=[1 2 3] and b=[2 3 2]. If we want to form a new vector c=[a(1)*b(1) a(2)*b(2) a(3)*b(3)] then we cannot accomplish this with matrix multiplication *. Instead, MATLAB offers an element-by-element style multiplication operator .* so that we can simply write c=a.*b. The operators ./ and .^ are defined analogously with respect to element-by-element division and element-by-element power.

The basic relational operators that MATLAB provides are: equal ==, not equal ~=, less than <, greater than >, less or equal <=, and greater or equal >=. The basic logical operators are: and &, or |, and not ~.

H-1.5 Mathematical Functions:

MATLAB offers a large number of predefined mathematical functions. Most of these functions are implemented to also work with vectors and/or matrices as input arguments. The output of such functions is always a matrix or vector of the same size as the input matrix/vector. The function is applied to the matrix or vector in an element-by-element fashion. For example, the matrix A=[0 pi/2 ; pi 3*pi/2] can be used as an input to the sine function B=sin(A). The resulting matrix B is given by [0 1 ; 0 -1]. A list of elementary mathematical functions such as sin, cos, log, and exp is obtained from:

```
>> help elfun
```

More specialized mathematical functions can be listed with:

```
>> help specfun
```

H-1.6 Command Window Input and Output:

MATLAB typically displays the result/output of a command line on the MATLAB main window screen. The output can be suppressed if the command line is terminated with a ;-character. Multiple MATLAB commands may be concatenated on a single command line if they are separated with a ,-character or a ;-character. Again, the result display of each command is suppressed with the ;-character and not suppressed with the ,-character. A controlled output of numbers, strings, arrays, and other "objects" can be accomplished with the `disp` command:

```
>> disp('Hello World.'); disp([ 10 12 14 ])
```

The way MATLAB displays numbers on the screen can be changed with the `format` command (see `help format` for details). To input numbers and strings on the MATLAB main window screen during program execution one can use the `input` command:

```
>> InputNumber = input('Please, enter a number: ');
```

To input a string one can use:

```
>> InputString = input('Please, enter a string: ','s');
```

H-1.7 Program Control:

The two most important program control features of (almost) any programming language are *loops* and *conditional statements.* MATLAB provides `for`-loops, `while`-loops, `if`-statements, and `switch`-statements. A simple program to add all numbers from 1 to 100, for example, would be:

```
>> s=0; for n=1:100; s=s+n; end; disp(s);
```

We could also use a `while`-loop to the same effect:

```
>> s=0; n=0; while n<=100; s=s+n; n=n+1; end; disp(s);
```

It should be pointed out that the run time execution of loops in MATLAB is generally very slow. It is, thus, desirable to avoid loops as much as possible. We can, for example, compute the sum of the numbers from 1 to 100 much faster with:

```
>> s=sum(1:100); disp(s);
```

A simple example for a conditional statement is provided by:

```
>> n=input('n='); if n>0; disp('n>0'); else; disp('n<=0'); end;
```

Note that MATLAB also provides an `elseif` branch to simplify nested conditional statements. A convenient way to test for a number of conditions is provided by the `switch`-statement:

```
>> n=input('n='); switch n, case 1; 'n=1', otherwise; 'n~=1', end;
```

Please, refer to `help switch` for more information.

H-1.8 Saving and Loading Data:

The two MATLAB functions `save` and `load` can be used to store and re-load any matrix, vector, string, or object to and from the hard drive. The resulting files are typically `.mat`-files. If we want to save the MATLAB variables `A`, `cv`, and `rv`, for example, into a file named `test.mat` we can simply type:

```
>> save test.mat A cv rv
```

Reloading the data is as easy as:

```
>> load test.mat
```

If no specific variables are listed with the `save` command then the entire variable workspace of `MATLAB` will be saved. This is convenient to temporarily save all results and have the ability to get back to things later.

H-1.9 Scripts, Functions, and M-Files:

`MATLAB` programs are generally saved in `.m`-files. A new `.m`-file with the name `testscript.m` can be created with `edit testscript`. Note that the *filename* of the `.m`-file is equal to the *command name* (without extension `.m`) that `MATLAB` use to execute the program contained in the file. For example, assume that we generate the `.m`-file `testscript.m`, fill it with `MATLAB` commands and save it. We can then execute the program from the `MATLAB` main figure window by simply entering:

```
>> testscript
```

For this procedure to work, however, it is necessary that the directory in which the `.m`-file is stored resides on the `MATLAB` search path. Please, refer to `help path` for more information about how to control the `MATLAB` search path.

Providing comments within your program is very important to increase a programs *readability.* In `MATLAB` any line that is preceded with a `%`-character is considered a comment and, thus, ignored for program execution. Comment lines at the beginning of a script (or function) have a special role: they are displayed on the `MATLAB` main figure window when we issue the command `help scriptname`, where `scriptname` is the name of the `.m`-file. It is, thus, possible to extend `MATLAB`s capability of on-line help to user defined scripts and functions as well.

Other than scripts, `MATLAB` lets us also define functions. Such user defined functions are generated via a script that contain the line

```
function [A,B,C,...]=function_name(a,b,c,...)
```

as its very first line in the `.m`-file. The `function_name` should match the filename of the `.m`-file. Parameters `a`, `b`, `c`, etc. are the input parameters and parameters `A`, `B`, `C`, etc. are the output parameters (variable names other than `a`, `b`, `c`, `A`, `B`, `C`, are of course permissible as well). Please note that `MATLAB` does not support *call-by-reference* parameter handling. All parameters are passed to-and-from the function on a *call-by-value* basis. A simple example is provided by the following function for the computation of $f(x) = x \cdot e^{-x^2/(2\sigma)}$:

```
function f=xgauss(x,sigma)
%XGAUSS Computation of a linearly scaled bell curve.
%   F = XGAUSS(X,SIGMA) computes the function F = X*EXP(-X*X/2*SIGMA).
%   The input argument X may be an arbitrarily dimensional numeric
%   array. SIGMA must be a scalar. The function is evaluated on an
%   element-by-element basis.

f=x.*exp((-0.5/sigma)*x.*x); return
```

If we put the above lines into an `.m`-file with the name `xgauss.m` then we can call the function directly from the `MATLAB` command line:

```
>> x = linspace(-1,1,5); y=xgauss(x,1); disp(y')
```

Please note that we can use the `MATLAB` command `return` inside of a script or function to terminate the script or function. It is the responsibility of the user to make sure that all output parameters are well defined by the time the `return` command is executed.

H-1.10 ELEMENTARY PLOTTING:

One of the strengths of `MATLAB` is its versatility in graphical data representation. The most elementary command that is available for the graphical display of one-dimensional functions (and signals) is the `plot` command. The `plot` command takes typically two input parameters: a vectors with the x-alignment (horizontal) and a vector with the y-alignment (vertical) of the sample points. Let us use the notation (x, y) to define the x and y coordinates of a point in a plane. Let us further assume that we want to connect the three points $(0, 1.5)$, $(2, -1.2)$, and $(3, 0.5)$ with straight lines. We put the x and y coordinates into respective vectors and use the `plot` command to draw the lines:

```
>> x = [ 0 2 3 ]; y = [ 1.5 -1.2 0.5 ]; plot(x,y);
```

The above set of command can be used very conveniently to plot "analog" signals. The following line shows how we can plot a sine signal $s(t) = \sin(t)$ between $t = -2$ and $t = 10$:

```
>> t = linspace(-2,10,300); s = sin(t); h = plot(t,s);
```

The plot command always returns a so called *graphics handle* `h`. The graphics handle can be used to (retroactively) change the properties of the drawn lines. If we want to change the color of the above sine signal to red, the line style to dotted, and the line width to three points then we can use the command:

```
>> set(h,'Color','r','LineStyle',':','LineWidth',3);
```

The line style, color, and line width are not the only properties of the plotted line that can be changed. A complete list of all properties of the graphics object with handle `h` is displayed on the screen when we type:

```
>> get(h)
```

We can also use `get(h,'LineStyle')` to specifically query the current line style of the object. Other object properties can be queried analogously. The on-line help to `plot` provides valuable additional information about how to format/configure the resulting line. Permissible properties for the line style are `'-'` (solid), `':'` (dotted), `'-.'` (dashdotted), and `'--'` (dashed). Permissible codes for the color are `'b'` (blue), `'g'` (green), `'r'` (red), `'c'` (cyan), `'m'` (magenta), `'y'` (yellow), and `'k'` (black). Additionally we can also have `MATLAB` enhance a plot with grid lines:

```
>> grid on;
```

Grid lines can be switched off again with `grid off`.

If we do not like the way MATLAB computes the x and y coordinate limits of the axis system then we can override the default value with the `axis` command. Assume that we want to change the displayed range of the sine function from above to $x = -3 \ldots 11$ and $y = -1.1 \ldots 1.1$. The change is simply accomplished by issuing:

```
>> axis([ -3 11 -1.1 1.1 ]);
```

Note that every new `plot` command clears the axis system of all previous graphical objects. If we want to plot multiple signals into the same axis system then we need to use the `hold` command:

```
>> t = linspace(-2,10,300); s = sin(t); h1 = plot(t,s); hold on;
>> c = cos(t); h2 = plot(t,c); hold off; axis([ -2 10 -1.1 1.1 ]);
>> set(h1,'Color','r'); set(h2,'Color','g'); grid on;
```

It is usually important to label a plot appropriately. The x-axis and the y-axis can be labelled with `xlabel` and `ylabel`. A title can be placed on top of the axis with the `title` command:

```
>> xlabel('Time'); ylabel('Amplitude'); title('Sine and Cosine');
```

Furthermore, if we have multiple lines within a plot then we should clarify which line is which. We can create a "legend" of line styles with the `legend` command. Text-labels for the `legend` command are provided in the order that the lines were displayed on screen:

```
>> legend('Sine Function','Cosine Function');
```

It is occasionally beneficial to use two (or more) separate axis systems to plot the results of an analysis or computation. A convenient way to generate multiple axis systems within the same figure is through the `subplot(m,n,p)` command. It creates a (if necessary new) matrix of m-by-n small axis systems and selects the p^{th} axis for the current plot. Again, let's consider the example of plotting a sine and a cosine signal; this time, however, in two separate axis systems:

```
>> t = linspace(-2,10,300); s = sin(t); c = cos(t);
>> subplot(2,1,1); plot(t,s); axis([ -2 10 -1.1 1.1 ]);
>> subplot(2,1,2); plot(t,c); axis([ -2 10 -1.1 1.1 ]);
```

Oftentimes, we are dealing with sampled data and not with continuous (analog) signals. Interpolation with straight lines between sample points would be (potentially) highly inappropriate. In these situations we can use the `stem` command instead of the `plot` command:

```
>> t = linspace(-2,10,20); s = sin(t); stem(t,s);
```

Lastly, we may occasionally need to visualize discretized two dimensional functions as well. One way to do this effectively is with the *scaled image plot* command `imagesc`. It represents different values on the z-axis via differently shaded colors. As an example, consider a plot of the two-dimensional MATLAB example function `peaks`:

```
>> TwoDimlMatrix = peaks(100); imagesc(TwoDimlMatrix);
```

H-1.11 Brief Function and Command Reference:

This section is a brief summary of some of MATLABs most important functions and commands. The list is by no means complete. Students are encouraged to check with MATLABs `helpdesk` for a complete list. Information about individual functions is readily obtained via the `help` command.

H-1.11.1 Basic Information about Numeric Arrays:

`isempty`	Determine if input is empty matrix.
`isequal`	Test arrays for equality[1].
`isfloat`	Determine if input is floating-point array.
`isinteger`	Determine if input is integer array.
`islogical`	Determine if input is logical array.
`isnumeric`	Determine if input is numeric array.
`isscalar`	Determine if input is scalar.
`isvector`	Determine if input is vector.
`disp`	Display text or array.
`length`	Length of vector.
`ndims`	Number of dimensions.
`numel`	Number of elements.
`size`	Size of matrix.

[1]See also function: `isequalwithequalnans`.

H-1.11.2 Basic Numeric Array Operations and Manipulation:

`cat`	Concatenate arrays along specified dimension.
`vertcat`	Concatenate arrays vertically.
`horzcat`	Concatenate arrays horizontally.
`repmat`	Replicate and tile array.
`fliplr`	Flip matrices left-right.
`flipud`	Flip matrices up-down.
`flipdim`	Flip matrix along specified dimension.
`permute`	Rearrange dimensions of multidimensional array.
`ipermute`	Inverse permute dimensions of multidimensional array.
`reshape`	Reshape array.
`squeeze`	Remove singleton dimensions from array.
`rot90`	Rotate matrix 90 degrees.
`tril`	Lower triangular part of matrix.
`triu`	Upper triangular part of matrix.
`diag`	Diagonal matrices and diagonals of matrix.
`sqrtm`	Matrix square root.
`expm`	Matrix exponential.
`cross`	Vector cross product.
`dot`	Vector dot product.

H-1.11.3 Searching, Sorting, Indexing, and Accumulating:

`find`	Find indices of nonzero elements.
`sort`	Sort array elements in ascending or descending order.
`sortrows`	Sort rows in ascending order.
`ind2sub`	Multiple subscripts from linear index.
`sub2ind`	Linear index from multiple subscripts.
`end`	Indicate last index of array.
`max`	Maximum value of array.
`min`	Minimum value of array.
`prod`	Product of array elements.
`cumprod`	Cumulative product.
`cumsum`	Cumulative sum.
`sum`	Sum of array elements.

H-1.11.4 Elementary Matrices and Arrays:

`eye`	Identity matrix.
`linspace`	Generate linearly spaced vectors.
`logspace`	Generate logarithmically spaced vectors.
`meshgrid`	Generate X and Y matrices for three-dimensional plots.
`ndgrid`	Arrays for multidimensional functions and interpolation.
`ones`	Create array of all ones.
`rand`	Uniformly distributed random numbers and arrays.
`randn`	Normally distributed random numbers and arrays.
`zeros`	Create array of all zeros.

H-1.11.5 Matrix Analysis and Linear Algebra:

`cond`	Condition number with respect to inversion.
`det`	Determinant.
`norm`	Matrix or vector norm.
`null`	Null space.
`orth`	Orthogonalization.
`rank`	Matrix rank.
`rref`	Reduced row echelon form.
`subspace`	Angle between two subspaces.
`trace`	Sum of diagonal elements.
`inv`	Matrix inverse.
`pinv`	Moore-Penrose pseudoinverse of matrix.
`eig`	Find eigenvalues and eigenvectors.
`svd`	Singular value decomposition.
`chol`	Cholesky factorization.
`qr`	Orthogonal-triangular decomposition.
`linsolve`	Solve linear systems of equations.
`funm`	Evaluate general matrix function.
`lu`	LU matrix factorization.

H-1.11.6 Elementary Mathematical Functions:

`acos`	Inverse cosine.
`acosh`	Inverse hyperbolic cosine.
`acot`	Inverse cotangent.
`acoth`	Inverse hyperbolic cotangent.
`asin`	Inverse sine.
`asinh`	Inverse hyperbolic sine.
`atan`	Inverse tangent.
`atanh`	Inverse hyperbolic tangent.
`atan2`	Four-quadrant inverse tangent.
`cos`	Cosine.
`cosh`	Hyperbolic cosine.
`cot`	Cotangent.
`coth`	Hyperbolic cotangent.
`sin`	Sine.
`sinh`	Hyperbolic sine.
`tan`	Tangent.
`tanh`	Hyperbolic tangent.
`exp`	Exponential.
`log`	Natural logarithm.
`log2`	Base 2 logarithm.
`log10`	Common (base 10) logarithm.
`sqrt`	Square root.
`abs`	Absolute value.
`angle`	Phase angle.
`conj`	Complex conjugate.
`cplxpair`	Sort numbers into complex conjugate pairs.
`imag`	Complex imaginary part.
`isreal`	Determine if input is real array.
`real`	Complex real part.
`sign`	Signum.
`unwrap`	Unwrap phase angle.
`fix`	Round towards zero.
`floor`	Round towards minus infinity.
`ceil`	Round towards plus infinity.
`round`	Round towards nearest integer.
`mod`	Modulus after division.
`rem`	Remainder after division.
`factor`	Prime factors.
`factorial`	Factorial function.
`gcd`	Greatest common divisor.
`isprime`	Determine if input is prime number.
`lcm`	Least common multiple.
`nchoosek`	All combinations of N elements taken K at a time.
`perms`	All possible permutations.
`primes`	Generate list of prime numbers.

H-1.11.7 Polynomials:

`conv`	Convolution and polynomial multiplication.
`poly`	Polynomial with specified roots.
`polyfit`	Polynomial curve fitting.
`polyval`	Polynomial evaluation.
`roots`	Polynomial roots.

H-1.11.8 Interpolation:

`interp1`	One-dimensional data interpolation.
`interp2`	Two-dimensional data interpolation.
`interpft`	One-dimensional FFT interpolation.
`ppval`	Piecewise polynomial evaluation.
`spline`	Cubic spline data interpolation.

H-1.11.9 Numerical Integration:

`quad`	Numerically evaluate integral (adaptive Simpson).
`quadl`	Numerically evaluate integral (adaptive Lobatto).
`trapz`	Trapezoidal numerical integration.

H-1.11.10 Specialized Mathematical Functions:

`erf`	Error function.
`erfc`	Complementary error function.
`erfcinv`	Inverse complementary error function.
`erfcx`	Scaled complementary error function.
`erfinv`	Inverse error function.
`gamma`	Gamma function.

H-1.11.11 Data Analysis:

`diff`	Differences and approximate derivatives.
`conv`	Convolution and polynomial multiplication.
`conv2`	Two-dimensional convolution.
`deconv`	Deconvolution and polynomial division.
`detrend`	Remove linear trend or mean from the data.
`filter`	Filter data with IIR or FIR filter.
`filter2`	Two-dimensional digital filter.
`fft`	One-dimensional discrete Fourier transform.
`fft2`	Two-dimensional discrete Fourier transform.
`fftw`	FFTW library run-time algorithm tuning.
`ifft`	Inverse one-dimensional discrete Fourier transform.
`ifft2`	Inverse two-dimensional discrete Fourier transform.
`nextpow2`	Next higher power of two.
`mean`	Average or mean value of arrays.
`median`	Median value of arrays.
`mode`	Most frequent value of array.
`std`	Standard deviation.
`var`	Variance.

H-1.11.12 Creating and Manipulating Strings:

`blanks`	Create string of space characters.
`char`	Convert to character array (string).
`cellstr`	Create cell array of strings from character array.
`datestr`	Convert date and time to string format.
`deblank`	Strip trailing blanks from end of string.
`lower`	Convert string to lowercase.
`sprintf`	Write formatted data to string.
`sscanf`	Read string under format control.
`strcat`	Concatenate strings horizontally.
`strjust`	Justify character array.
`strread`	Read formatted data from string.
`strrep`	Find and replace substring.
`strtrim`	Remove leading and trailing whitespace from string.
`strvcat`	Concatenate strings vertically.
`upper`	Convert string to uppercase.
`findstr`	Find string within another, longer string.
`ischar`	Determine if input is character array.
`isletter`	Detect elements that are alphabetic letters.
`isspace`	Detect elements that are ASCII white spaces.
`strcmp`	Compare strings.
`strcmpi`	Compare strings, ignoring case.
`strfind`	Find one string within another.
`strmatch`	Find possible matches for string.
`strncmp`	Compare first n characters of strings.
`strncmpi`	Compare first n characters of strings, ignoring case.
`strtok`	Return selected parts of string.
`str2double`	Convert string to double-precision number.
`str2num`	Convert string to number.
`num2str`	Convert number to string.

H-1.11.13 Evaluating String Expressions:

`eval`	Execute string containing MATLAB expression.
`evalc`	Evaluate MATLAB expression with capture.
`evalin`	Execute MATLAB expression in specified workspace.

H-1.11.14 Set Operations:

`intersect`	Find set intersection of two vectors.
`ismember`	Detect members of set.
`setdiff`	Find set difference of two vectors.
`issorted`	Determine if set elements are in sorted order.
`setxor`	Find set exclusive OR of two vectors.
`union`	Find set union of two vectors.
`all`	Determine if all array elements are nonzero.
`any`	Determine if any array elements are nonzero.
`unique`	Find unique elements of vector.

H-1.11.15 Control Flow:

`break`	Terminate execution of for or while loop.
`case`	Execute block of code if condition is true.
`catch`	Specify how to respond to error in try statement.
`continue`	Pass control to next iteration of for or while loop.
`else`	Conditionally execute statements.
`elseif`	Conditionally execute statements.
`end`	Terminate conditional block of code.
`error`	Display error message.
`for`	Execute block of code specified number of times.
`if`	Conditionally execute statements.
`otherwise`	Default part of switch statement.
`return`	Return to invoking function.
`switch`	Switch among several cases, based on expression.
`try`	Attempt to execute block of code, and catch errors.
`while`	Repeatedly execute statements while condition is true.

H-1.11.16 Loading and Saving Data:

`load`	Load workspace variables from disk.
`save`	Save workspace variables on disk.
`imread`	Read image from graphics file.
`imwrite`	Write image to graphics file.
`fileparts`	Return parts of file name and path.
`filesep`	Return directory separator for platform in use.
`fullfile`	Build full filename from parts.

H-1.11.17 Sound and Microsoft WAVE Functions:

`sound`	Convert vector into sound.
`soundsc`	Scale data and play as sound.
`wavplay`	Play sound on PC-based audio output device.
`wavread`	Read Microsoft WAVE (.wav) sound file.
`wavrecord`	Record sound using PC-based audio input device.
`wavwrite`	Write Microsoft WAVE (.wav) sound file.

H-1.11.18 Miscellaneous:

`now`	Return current date/time code number.
`pause`	Halt execution temporarily.
`function`	Declare M-file function.
`input`	Request user input.
`nargin`	Return number of function input arguments.
`nargout`	Return number of function output arguments.
`varargin`	Accept variable number of arguments.
`varargout`	Return variable number of argument.
`waitbar`	Display wait bar.
`print`	Print graph or save graph to file.

H-1.11.19 Basic Plots and Graphs:

`errorbar`	Plot graph with error bars.
`loglog`	Plot using log-log scales.
`polar`	Polar coordinate plot.
`plot`	Plot vectors or matrices.
`plot3`	Plot lines and points in 3-D space.
`plotyy`	Plot graphs with Y tick labels on the left and right.
`semilogx`	Semi-log scale plot.
`semilogy`	Semi-log scale plot.
`bar`	Vertical bar chart.
`pie`	Pie plot.
`contour`	Contour (level curves) plot.
`stem`	Plot discrete sequence data.
`stairs`	Stairstep graph.
`hist`	Plot histograms.
`image`	Display image object.
`imagesc`	Scale data and display image object.

H-1.11.20 Annotating Plots:

`annotation`	Create annotation objects.
`clabel`	Add contour labels to contour plot.
`datetick`	Date formatted tick labels.
`gtext`	Place text on 2-D graph using mouse.
`legend`	Graph legend for lines and patches.
`texlabel`	Produce the TeX format from character string.
`title`	Titles for 2-D and 3-D plots.
`xlabel`	X-axis labels for 2-D and 3-D plots.
`ylabel`	Y-axis labels for 2-D and 3-D plots.
`zlabel`	Z-axis labels for 3-D plots.
`textarrow`	Properties for annotation textbox.

H-1.11.21 Axis, Object, and Figure Access:

`figure`	Create figure (graph) windows.
`axes`	Create axes object.
`patch`	Create patch object (polygons).
`text`	Create text object (character strings).
`arrow`	Properties for annotation arrows.
`doublearrow`	Properties for double-headed annotation arrows.
`ellipse`	Properties for annotation ellipses.
`line`	Properties for annotation lines.
`rectangle`	Properties for annotation rectangles.
`delete`	Delete files or graphics objects.
`get`	Get object properties.
`set`	Set object properties.
`ishandle`	True if value is valid object handle.

H-1.11.22 Axis, Object, and Figure Control:

`box`	Axis box for 2-D and 3-D plots.
`hold`	Hold current graph.
`axis`	Plot axis scaling and appearance.
`grid`	Grid lines for 2-D and 3-D plots.
`subplot`	Create axes in tiled positions.
`gcf`	Get current figure handle.
`gca`	Get current axes handle.
`clc`	Clear figure window.
`clf`	Clear figure.
`cla`	Clear axes.
`close`	Close specified window.
`drawnow`	Complete any pending drawing.

H-1.11.23 Predefined Dialog Boxes:

`dialog`	Create and display dialog box.
`errordlg`	Create and display error dialog box.
`helpdlg`	Create and display help dialog box.
`inputdlg`	Create and display input dialog box.
`listdlg`	Create and display list selection dialog box.
`msgbox`	Create and display message dialog box.
`pagesetupdlg`	Display page setup dialog box.
`printdlg`	Display print dialog box.
`questdlg`	Display question dialog box.
`uigetdir`	Display standard dialog box for retrieving a directory.
`uigetfile`	Display standard dialog box for retrieving files.
`uigetpref`	Display dialog box for retrieving preferences.
`uiputfile`	Display standard dialog box for saving files.
`uisave`	Display standard dialog box for saving workspace variables.
`uisetcolor`	Display dialog box for setting an object's ColorSpec.
`uisetfont`	Display dialog box for setting an object's font.
`warndlg`	Display warning dialog box.

H-1.11.24 Microsoft Excel Functions:

`xlsfinfo`	Determine if file contains Microsoft Excel (.xls) spreadsheet.
`xlsread`	Read Microsoft Excel spreadsheet file (.xls).
`xlswrite`	Write Microsoft Excel spreadsheet file (.xls).

H-1.11.25 Low-Level File I/O:

`fopen`	Open file or obtain information about open files.
`fclose`	Close one or more open files.
`feof`	Test for end-of-file.
`fgetl`	Return next line of file as string without line terminator(s).
`fgets`	Return next line of file as string with line terminator(s).
`fread`	Read binary data from file.
`fwrite`	Write binary data to file.