An Introduction to MATLAB

Collection Editor:
Anders Gjendemsjø

An Introduction to MATLAB

Collection Editor:
Anders Gjendemsjø

Authors:
Anders Gjendemsjø
Jason Laska

Online:
< http://cnx.org/content/col10323/1.3/ >

C O N N E X I O N S

Rice University, Houston, Texas

This selection and arrangement of content as a collection is copyrighted by Anders Gjendemsjø. It is licensed under the Creative Commons Attribution 2.0 license (http://creativecommons.org/licenses/by/2.0/).
Collection structure revised: January 20, 2006
PDF generated: October 26, 2012
For copyright and attribution information for the modules contained in this collection, see p. 27.

Table of Contents

1 An Introduction to MATLAB 1
2 Using MATLAB 3
3 Graphical representation of data in MATLAB 7
4 Scripts and Functions in MATLAB 13
5 Vectorizing loops in MATLAB 17
6 Writing C Functions in MATLAB (MEX-Files) 19
7 Introductory Computer Assignment for MATLAB 25
Index 26
Attributions 27

Chapter 1

An Introduction to MATLAB[1]

MATLAB, short for Matrix Laboratory, is a simple and flexible programming environment for a wide range of problems such as signal processing, optimization, linear programming and so on. The basic MATLAB software package can be extended by using add-on toolboxes. Examples of such toolboxes are: Signal Processing, Filter Design, Statistics and Symbolic Math.

Comprehensive documentation for MATLAB is available at Mathworks.com[2] . In particular, an excellent (extensive) getting started guide is available at Getting started with MATLAB[3] . There is also a very active newsgroup for MATLAB related questions, **comp.soft-sys.matlab**

MATLAB is an interpreted language. This implies that the source code is not compiled but interpreted on the fly. This is both an advantage and a disadvantage. MATLAB allows for easy numerical calculation and visualization of the results without the need for advanced and time consuming programming. The disadvantage is that it can be slow, especially when bad programming practices are applied.

[1] This content is available online at <http://cnx.org/content/m13255/1.1/>.
[2] http://www.mathworks.com
[3] http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf

Chapter 2

Using MATLAB[1]

2.1 Matlab Help

MATLAB has a great on-line help system accessible using the help command. Typing

`help <function>`

will return text information about the chosen function. For example to get information about the built-in function sum type:

`help sum`

To list the contents of a toolbox type help <toolbox>, e.g. to show all the functions of the signal processing toolbox enter

`help signal processing`

If you don't know the name of the function but a suitable keyword use the `lookfor` followed by a keyword string, e.g.

`lookfor 'discrete fourier'`

To explore the extensive help system use the "Help menu" or try the commands `helpdesk` or `demo`.

2.2 Matrices, vectors and scalars

MATLAB uses matrices as the basic variable type. Scalars and vectors are special cases of matrices having size 1x1, 1xN or Nx1. In MATLAB, there are a few conventions for entering data:

- Elements of a row are separated with blanks or commas.
- Each row is ended by a semicolon, ;.
- A list of elements must be surrounded by square brackets, [ ]

Example 2.1

It is easy to create basic variables.

`x = 1` (scalar)
`y = [2 4 6 8 10]` (row vector)
`z = [2; 4; 6; 8; 10]` (column vector)
`A = [4 3 2 1 0; 1 3 5 7 9]` (2 x 5 matrix)

Regularly spaced values of a vector can be entered using the following compact notation

`start:skip:end`

Example 2.2

A more compact way of entering variables than in Example 1 (Example 2.1) is shown here:

[1] This content is available online at <http://cnx.org/content/m13254/1.5/>.

```
y= 2 :  2 :  10
A=[4:-1:0;1:2:9]
```

If the skip is omitted it will be set to 1, i.e., the following are equivalent
`start:1:end` and `start:end`
To create a string use the single quotation mark " ' ", e.g. by entering `x = 'This is a string'`.

2.3 Indexing matrices and vectors

Indexing variables is straightforward. Given a matrix M the element in the i'th row, j'th column is given by `M(i,j)`. For a vector v the i'th element is given by `v(i)`. Note that the lowest allowed index in MATLAB is 1. This is in contrast with many other programming languages (e.g. JAVA and C), as well as the common notation used in signal processing, where indexing starts at 0. The colon operator is also of great help when accessing specific parts of matrices and vectors, as shown below.

Example 2.3
This example shows the use of the colon operator for indexing matrices and vectors.
`A(1,:)` returns the first row of the matrix A.
`A(:,3)` returns the third column of the matrix A.
`A(2,1:5)` returns the first five elements of the second row.
`x(1:2:10)` returns the first five odd-indexed elements of the vector x.

2.4 Basic operations

MATLAB has built-in functions for a number of arithmetic operations and functions. Most of them are straightforward to use. The Table (Table 2.1: Common mathematical operations in MATLAB) below lists the some commonly used functions. Let x and y be scalars, M and N matrices.

Common mathematical operations in MATLAB

	MATLAB
xy	`x*y`
x^y	`x^y`
e^x	`exp(x)`
$\log(x)$	`log10(x)`
$\ln(x)$	`log(x)`
$\log2(x)$	`log2(x)`
MN	`M*N`
M^{-1}	`inv(M)`
	continued on next page

M^T	`M'`
$\det(M)$	`det(M)`

Table 2.1

- Dimensions - MATLAB functions length and size are used to find the dimensions of vectors and matrices, respectively.
- Elementwise operations - If an arithmetic operation should be done on each component in a vector (or matrix), rather than on the vector (matrix) itself, then the operator should be preceded by ".", e.g .*, .^ and ./.

Example 2.4
Elementwise operations, part I

Let $A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$. Then `A^2` will return AA $= \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$, while `A.^2` will return $\begin{pmatrix} 1^2 & 1^2 \\ 1^2 & 1^2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.

Example 2.5
Elementwise operations, part II

Given a vector x, and a vector y having elements $y(n) = \frac{1}{\sin(x(n))}$. This can be easily be done in MATLAB by typing `y=1./sin(x)` Note that using `/` in place of `./` would result in the (common) error `Matrix dimensions must agree`.

2.5 Complex numbers

MATLAB has excellent support for complex numbers with several built-in functions available. The imaginary unit is denoted by `i` or (as preferred in electrical engineering) `j`. To create complex variables $z_1 = 7 + i$ and $z_2 = 2e^{i\pi}$ simply enter `z1 = 7 + j` and `z2 = 2*exp(j*pi)`

The Table below gives an overview of the basic functions for manipulating complex numbers, where z is a complex number.

Manipulating complex numbers in MATLAB

	MATLAB
$\mathrm{Re}(z)$	`real(z)`
$\mathrm{Im}(z)$	`imag(z)`
$\lvert z \rvert$	`abs(z)`
$\mathrm{Angle}(z)$	`angle(z)`

continued on next page

z^*	`conj(z)`

Table 2.2

2.6 Other Useful Details

- A **semicolon** added at the end of a line tells MATLAB to suppress the command output to the display.
- MATLAB and **case sensitivity**. For variables MATLAB is case sensitive, i.e., b and B are different. For functions it is case insensitive, i.e., sum and SUM refer to the same function.
- Often it is useful to **split a statement** over multiple lines. To split a statement across multiple lines, enter three periods `"..."` at the end of the line to indicate that it continues on the next line.

Example 2.6
Splitting $y = a + b + c$ over multiple lines.

```
y = a...
+ b...
+ c;
```

Chapter 3

Graphical representation of data in MATLAB[1]

3.1 Graphical representation of data in MATLAB

MATLAB provides a great variety of functions and techniques for graphical display of data. The flexibility and ease of use of MATLAB's plotting tools is one of its key strengths. In MATLAB graphs are shown in a figure window. Several figure windows can be displayed simultaneously, but only one is active. All graphing commands are applied to the active figure. The command `figure(n)`will activate figure number `n` or create a new figure indexed by `n`.

3.2 Tools for plotting

In this section we present some of the most commonly used functions for plotting in MATLAB.

- `plot`- The plot and stem functions can take a large number of arguments, see help plot and help stem. For example the line type and color can easily be changed. `plot(y)` plots the values in vector `y`versus their index. `plot(x,y)` plots the values in vector `y`versus `x`. The `plot` function produces a piecewise linear graph between its data values. With enough data points it looks continuous.
- `stem`- Using `stem(y)`the data sequence `y`is plotted as stems from the x-axis terminated with circles for the data values. `stem` is the natural way of plotting sequences. `stem(x,y)` plots the data sequence `y` at the values specified in `x`.
- `xlabel('string')`- Labels the x-axis with `string`.
- `ylabel('string')`- Labels the y-axis with `string`.
- `title('string')`- Gives the plot the title `string`.

To illustrate this consider the following example.

Example 3.1
In this example we plot the function y = x2 for x 2 [-2; 2].

```
x = -2:0.2:2;

y = x.^2;

figure(1);
```

[1]This content is available online at <http://cnx.org/content/m13252/1.1/>.

```
plot(x,y);

xlabel('x');

ylabel('y=x^2');

title('Simple plot');

figure(2);

stem(x,y);

xlabel('x');

ylabel('y=x^2');

title('Simple stem plot');
```

This code produces the following two figures.

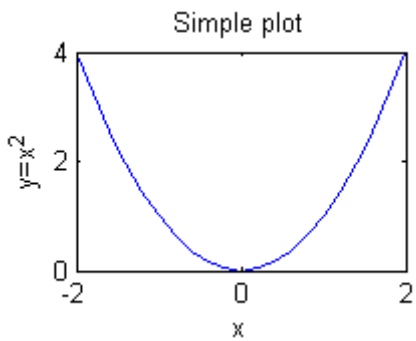


Figure 3.1

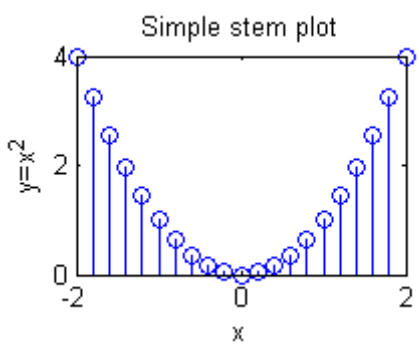


Figure 3.2

Some more commands that can be helpful when working with plots:

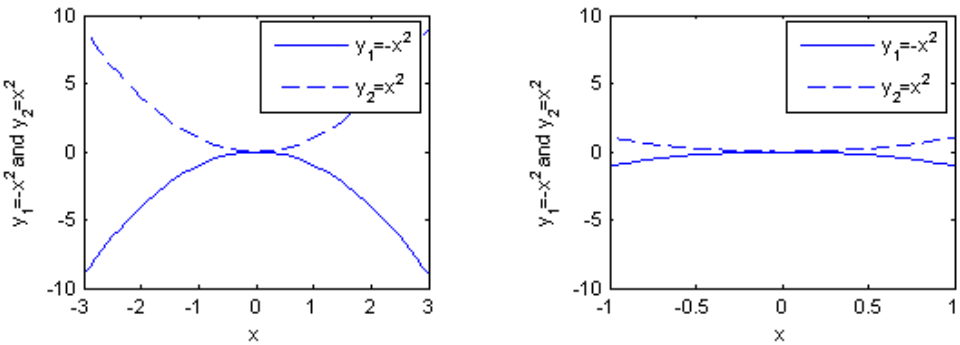
- hold on / off - Normally hold is off. This means that the plot command replaces the current plot with the new one. To add a new plot to an existing graph use `hold on`. If you want to overwrite the current plot again, use `hold off`.
- `legend('plot1','plot2',...,'plot N')`- The `legend` command provides an easy way to identify individual plots when there are more than one per figure. A legend box will be added with strings matched to the plots.
- `axis([xmin xmax ymin ymax])`- Use the `axis` command to set the axis as you wish. Use `axis on/off` to toggle the axis on and off respectively.
- `subplot(m,n,p)` -Divides the figure window into `m` rows, `n` columns and selects the `pp`'th subplot as the current plot, e.g `subplot(2,1,1)` divides the figure in two and selects the upper part. `subplot(2,1,2)` selects the lower part.
- `grid on/off` - This command adds or removes a rectangular grid to your plot.

Example 3.2

This example illustrates `hold`, `legend` and `axis`.

```
x = -3:0.1:3; y1 = -x.^2; y2 = x.^2;

figure(1);

plot(x,y1);

hold on;

plot(x,y2,'-');

hold off;

xlabel('x');

ylabel('y_1=-x^2 and y_2=x^2');

legend('y_1=-x^2','y_2=x^2');

figure(2);

plot(x,y1);

hold on;

plot(x,y2,'-');

hold off;

xlabel('x');

ylabel('y_1=-x^2 and y_2=x^2');

legend('y_1=-x^2','y_2=x^2');

axis([-1 1 -10 10]);
```

The result is shown below.



(a) (b)

Figure 3.3

Example 3.3

In this example we illustrate subplot and grid.

```
x = -3:0.2:3; y1 = -x.^2; y2 = x.^2;


subplot(2,1,1);

plot(x,y1);

xlabel('x'); ylabel('y_1=-x^2');

grid on;

subplot(2,1,2);

plot(x,y2);

xlabel('x');

ylabel('y_2=x^2');
```

Now, the result is shown below.

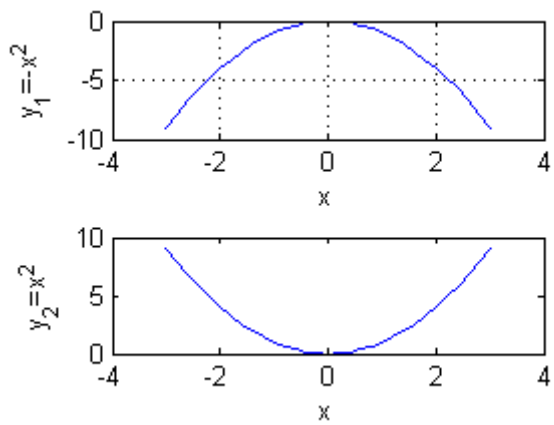


Figure 3.4

3.3 Printing and exporting graphics

After you have created your figures you may want to print them or export them to graphic files. In the "File" menu use "Print" to print the figure or "Save As" to save your figure to one of the many available graphics formats. Using these options should be sufficient in most cases, but there are also a large number of adjustments available by using "Export setup", "Page Setup" and "Print Setup".

To streamline the graphics exportation, take a look at exportfig package at Mathworks.com, URL: http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=727[2] .

3.4 3D Graphics

We end this module on graphics with a sneak peek into 3D plots. The new functions here are `meshgrid` and `mesh`. In the example below we see that `meshgrid`produces `x`and `y`vectors suitable for 3D plotting and that `mesh(x,y,z)` plots `z` as a function of both `x` and `y`.

Example 3.4

Example: Creating our first 3D plot.

```
[x,y] = meshgrid(-3:.1:3);

z = x.^2+y.^2;

mesh(x,y,z);

xlabel('x');

ylabel('y');

zlabel('z=x^2+y^2');
```

[2] http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=727

This code gives us the following 3D plot.

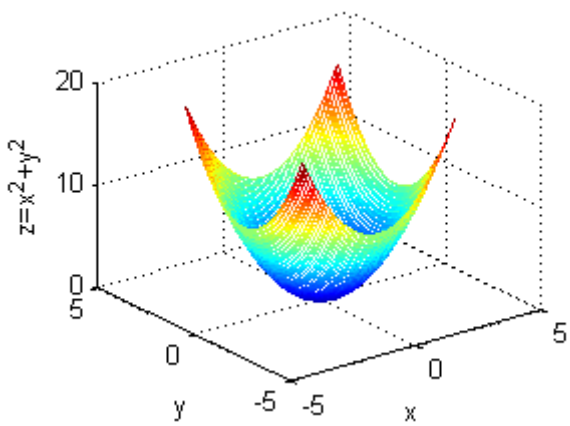


Figure 3.5

Chapter 4

Scripts and Functions in MATLAB[1]

4.1 Script files

Script files, also called M- files as they have extension `.m`, make MATLAB programming much more efficient than entering individual commands at the command prompt. A script file consists of MATLAB commands that together perform a specific task. The M-file is a text file which can be created and edited by any plain text editor like Notepad, emacs or the built-in MATLAB editor. To create a script in MATLAB use: `File - New - M -File` from the menu. An example script is shown below.

Example 4.1
Our first script.

```
n = 0:pi/100:2*pi; %create an index vector
y = cos(2*pi*n);   %create a vector y
plot(n,y);         %plot y versus n
```

As shown above the `%`-sign allows for comments. Saving the script as `foo.m` it can be executed as `foo` from the command prompt or by clicking the run button in the MATLAB editor. Script files are very practical and should be the preferred alternative compared to the command prompt in most cases.

4.2 Program flow

As in most programming languages program flow can be controlled by using statements such as `for`, `while`, `if`, `else`, `elseif`, and `switch`. These statements can be used both in M-files and at the command prompt, the latter being highly inconvenient. Below we show some examples. Use help to get more details.

- `for`- To print "Hello World" 10 times write

  ```
  for n=1:10
     disp('Hello World')
  end
  ```

  `for` loops can in many cases be avoided by vectorizing your code, more about that later.
- `if`, `elseand` `elseif` - Classics that never go out of style.

[1]This content is available online at <http://cnx.org/content/m13253/1.1/>.

```
if a == b
   a = b + 1
elseif a > b
   a = b - 1
else
   a = b
end
```

4.3 User Defined Functions

Sometimes it is convenient to create your own functions for use in MATLAB. Functions are program routines, usually implemented in M-files. Functions can take input arguments and return output arguments. They operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

Example 4.2
Create a function for calculating the sum of the $N+1$ first terms of geometric series. Assume $N < \infty$.

Solution: The sum of the N + 1 terms of a geometric series is given by ssum $= \sum_{n=0}^{N} a^n$. An implementation of this sum as a function accepting the input arguments a and N is shown below.

```
function ssum = geom(a,N)
  n=0:N;
  ssum = sum(a.^n);
end
```

The function `geom`can then be called, e.g from the command prompt. The function call `geom(0.9,10)`returns 6.8619.

To illustrate some more MATLAB programming we take on the task of creating a MATLAB function that will compute the sum of an arbitrary geometric series, ssum $= \sum_{n=0}^{N} a^n$.

Example 4.3
Create a function to calculate the sum of an arbitrary geometric series.

Solution: For $N < \infty$ we know that the sum converges regardless of a. As N goes to ∞ the sum converges only for $a < 1$, and the sum is given by the formula $\sum_{n=0}^{\infty} a^n = \frac{1}{1-a}$. A possible implementation is given as:

```
function ssum = geomInf(a,N)
   if(N==inf)
      if(abs(a)>=1)
         error('This geometric series will diverge.');
      else
         ssum=1/(1-a);
      end
   else
      n=0:N;
      ssum = sum(a.^n);
   end
end
```

Note that in the two examples above we could have used the formula for the sum of a finite geometric series. However we chose to create a vector and use the function `sum` to illustrate MATLAB concepts.

4.4 Learn From Existing Code

Wouldn't it be great to learn from the best? Using the command `type` followed by a function name the source code of the function is displayed. As the built in functions are written by people with excellent knowledge of MATLAB, this is a great feature for anyone interested in learning more about MATLAB.

Chapter 5

Vectorizing loops in MATLAB[1]

5.1

In MATLAB one should try to avoid loops. This can be done by vectorizing your code. The idea is that MATLAB is very fast on vector and matrix operations and correspondingly slow with loops. We illustrate this by an example.

Example 5.1
Given $a_n = n$, and $b_n = 1000 - n$ for $n = 1, ..., 1000$. Calculate $\sum_{n=1}^{1000} a_n b_n$, and store in the variable `ssum`.

Solution: It might be tempting to implement the above calculation as

```
a = 1:1000;
b = 1000 - a;
ssum=0;
for n=1:1000 %poor style...
   ssum = ssum +a(n)*b(n);
end
```

Recognizing that the sum is the inner product of the vectors a and b, ab^T, we can do better:

```
ssum = a*b' %Vectorized, better!
```

For more detailed information on vectorization, please take a look at MathWorks' Code Vectorization Guide[2] .

[1] This content is available online at <http://cnx.org/content/m13251/1.4/>.
[2] http://www.mathworks.com/support/tech-notes/1100/1109.html

Chapter 6

Writing C Functions in MATLAB (MEX-Files)[1]

6.1 Introduction

The MATLAB M-File is very good for putting together functions or scripts that run many of MATLAB's fast Built-In functions. One nice thing about these files is that they are never compiled and will run on any system that is already running MATLAB. MATLAB achieves this by interpreting each line of the M-File every time it is run. This method of running the code can make processing time very slow for large and complicated functions, especially those with many loops because every line within the loop will be interpreted as a new line, each time through the loop. Good MATLAB code avoids these things by using as many Built-In features and array operations as possible (because these are fast and efficient). Sometimes this is not enough...

MATLAB has the capability of running functions written in C. The files which hold the source for these functions are called MEX-Files. The mexFunctions are not intended to be a substitue for MATLAB's Built-In operations however if you need to code many loops and other things that MATLAB is not very good at, this is a good option. This feature also allows system-specific APIs to be called to extend MATLAB's abilities (see the Serial Port Tutorial for an example of this).

This document is arranged in the following manner:

- The MEX-Function: Interface to MATLAB
- Getting and Creating Data
- Calling Built-In Functions from a MEX-File
- Compiling
- Useful Functions not Mentioned Here

These are some of the basic topics that will allow you to create a MEX-file in a short time. There are many other features and abilities that MATLAB has which can be explored in the MATLAB documentation.

6.2 The MEX-Function: Interface to MATLAB

When writing programs in C, it is always assumed that the program will start execution from the main(). MEX -Files are similar in that they always start execution from a special function called the mexFunction. This function has return type void and is the "gateway" between the MATLAB function call, and your C code.

[1] This content is available online at <http://cnx.org/content/m12348/1.2/>.

Example 6.1

```
//You can include any C libraries that you normally use
#include "math.h"
#include "mex.h"   //--This one is required

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    //All code and internal function calls go in here!

    return;
}
```

In order to make a mex-function, you must include the "mex.h" library. This library contains all of the APIs that MATLAB provides. There are four input parameters to the mexFunction which correspond to the way a function is called in MATLAB - (ex: `[z0,z1] = jasonsFunction(x,y,z);`)

- `nlhs` (Type = int): This paramter represents the number of "left hand side" arguments. So in my example function call, nlhs = 2 (the outputs are z0 and z1).
- `plhs` (Type = array of pointers to mxArrays): This parameter is the actual output arguments. As we will see later, an mxArray is MATLAB's structure for holding data and each element in plhs holds an mxArray of data.
- `nrhs` (Type = int): Similar to nlhs, this paramter holds the number of "right hand side" arguments.
- `prhs` (Type = const array of pointers to mxArrays): This array hold all of the pointers to the mxArrays of input data for instance, prhs[0] holds the mxArray containing x, prhs[1] holds the mxArray containing y, etc).

6.3 Getting and Creating Data

The main MATLAB structure used for holding data in MEX-Files is the mxArray. This structure can hold real data, complex data, arrays, matrices, sparse-arrays, strings, and a whole host of other MATLAB data-structures. Using data from some of the basic structures is shown here, but refer to the MATLAB help for using other data structures.

6.3.1 Get that Data

Lets use my example from above (if you forgot: `[z0,z1] = jasonsFunction(x,y,z);`). Assume that x is a 2-D matrix, y is a string, and z is an integer. Here we wills ee how to extract and use these different types of data.

We have access to the input paramter x by a pointer held in the array prhs. In C, when referencing an array by index, the variable is automatically dereferenced (ie: you dont need to use a star). For clarity, I will copy the variable x over to an mxArray pointer named xData (This does not need to be done for the code to work).

Example 6.2

```
//---Inside mexFunction---

//Declarations
mxArray *xData;
```

```
double *xValues;
int i,j;
int rowLen, colLen;
double avg;

//Copy input pointer x
xData = prhs[0];

//Get matrix x
xValues = mxGetPr(xData);
rowLen = mxGetN(xData);
colLen = mxGetM(xData);

//Print the integer avg of each col to matlab console
for(i=0;i<rowLen;i++)
{
    avg=0;
    for(j=0;j<colLen;j++)
    {
        avg += xValues[(i*colLen)+j];
        //Another Method:
        //
        //avg += *xValues++;
    }
    avg = avg/colLen;
    printf("The average of row %d, is %d",i,(int)avg);
}
```

The function `mxGetPr` is used to get a pointer to the real data xData. This function takes a pointer to an mxArray as the intput paramter, and returns a pointer array of doubles. A similar function `mxGetPi` can be used for complex data. `mxGetN` and `mxGetM` return integers of the lengths of the row and column in the matrix. If this were an array of data, one of these return values would be zero. MATLAB gives the matrix as rows first, then columns (if you were to traverse the matrix linearly) so to jump by position, (x,y) maps to x*colLen+y. MATLAB organizes its arrays this way to reduce cache misses when the row traversal is on the outside loop. It is good to code it this way if you are working for efficiency. `printf()` will print out to the MATLAB command prompt.

Getting a string is very similar, but has its own method. The example below shows the procedure for getting a string. Again, I will copy the input to a pointer called yData.

Example 6.3

```
//---Inside mexFunction---

//Declarations
mxArray *yData;
int yLength;
char *TheString;

//Copy input pointer y
yData = prhs[1];
```

```
//Make "TheString" point to the string
yLength = mxGetN(yData)+1;
TheString = mxCalloc(yLength, sizeof(char)); //mxCalloc is similar to malloc in C
mxGetString(yData,TheString,yLength);
```

This last example shows how to get a simple integer. This is the method that has always worked for me, but it seems kind of strange so I imagine there is another way to do this.

Example 6.4

```
//---Inside mexFunction---

//Declarations
mxArray *zData;
int Num;

//Copy input pointer z
zData = prhs[2];

//Get the Integer
Num = (int)(mxGetScalar(zData));

//print it out on the screen
printf("Your favorite integer is: %d",Num);
```

Three data types have been shown here. There are several others and the MATLAB help as well as the MATLAB example code shows how to use them. Now to export the data....

6.3.2 Returning Data to MATLAB

Assigning return values and data to the left hand side parameters is very similar to getting the data from the last section. The difference here is that memory must be allocated for the data strucure being used on the output. Here is an example of how to return a 2-D matrix. This code will take the input x and return a copy of the matrix to z0 with every point in x multiplied by 2. Note that I am not copying the name of the output mxArray pointer into another variable.

Example 6.5

```
//---Inside mexFunction---

//Declarations
mxArray *xData;
double *xValues, *outArray;
int i,j;
int rowLen, colLen;

//Copy input pointer x
xData = prhs[0];

//Get matrix x
```

```
xValues = mxGetPr(xData);
rowLen = mxGetN(xData);
colLen = mxGetM(xData);

//Allocate memory and assign output pointer
plhs[0] = mxCreateDoubleMatrix(colLen, rowLen, mxREAL); //mxReal is our data-type

//Get a pointer to the data space in our newly allocated memory
outArray = mxGetPr(plhs[0]);

//Copy matrix while multiplying each point by 2
for(i=0;i<rowLen;i++)
{
    for(j=0;j<colLen;j++)
    {
        outArray[(i*colLen)+j] = 2*xValues[(i*colLen)+j];
    }
}
```

6.4 Calling Built-In Functions from a MEX-File

While it may be nice to write functions in C, there are so many useful and fast pre-written functions in MATLAB that it would be a crime if we could not use them. Luckily, The Mathworks (creators of MATLAB) has provided this capability. Built-In functions have a parameter list similar to the mexFunction itself. This example uses the built-in function z = conv(x,y);

Example 6.6

```
//---Inside mexFunction---

//Declarations
mxArray *result;
mxArray *arguments[2];

//Fill in the input parameters with some trash
arguments[0] = mxCreateDoubleMatrix(1, 20, mxREAL);
arguments[1] = mxCreateDoubleMatrix(1, 10, mxREAL);

//In the real world I imagine you would want to actually put
//some useful data into the arrays above, but for this example
//it doesnt seem neccesary.

//Call the Function
mexCallMATLAB(1,&result,2,arguments,"conv");

//Now result points to an mxArray and you can extract the data as you please!
```

6.5 Compiling

Compiling the MEX-Files is similar to compiling with gcc or any other command line compiler. In the MATLAB command prompt, change your current directory to the location of the MEX source file. Type: `mex filename.c` into the MATLAB command window. MATLAB may ask you to choose a compiler. Choose the compiler with MATLAB in its directory path. Your function will be called with the same name as your file. (ex: `mex jasonsFunction.c` produces a function that can be called from MATLAB as `[z0,z1] = jasonsFunction(x,y,z);`)

After compiling MATLAB produces the actual MEX binary that can be called as a normal MATLAB function. To call this function, you must be in the same directory with the binary. The binary goes by different names depending what system you compiled the source on (ex: Windows=.dll MacOSX=.mexmac Solaris=.mexsol Linux=.mexlx). Your MEX-function will have to be compiled on each type of system that you want to run it on because the binaries are operating system specific.

6.6 Other Useful Functions

Here is a nice list of useful functions in the mex library that make life a lot easier. Most of these work in similar fashion to those functions described above. The full list can be found in the MATLAB help documentation with many examples. There are also some example files in the MATLAB extern directory (MATLAB/extern/examples/mx or mex).

- mxDuplicateArray
- mexErrMsgTxt
- mxMalloc
- mxRealloc
- mxCreateString
- mxDestroyArray
- mxFree
- mxGetCell
- mxGetData
- and many more...

Chapter 7

Introductory Computer Assignment for MATLAB[1]

The Introductory assignment is currently only available in pdf format[2] . Solution in pdf format[3] .

[1] This content is available online at <http://cnx.org/content/m13256/1.5/>.
[2] http://cnx.org/content/m13256/latest/CAMatlabIntro.pdf
[3] http://cnx.org/content/m13256/latest/SolutionCAIntro.pdf

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

A Assignment, § 7(25)

C c functions, § 6(19)
c language, § 6(19)
Computer, § 7(25)

F fast functions in matlab, § 6(19)
Function, § 4(13)

H Help, § 2(3)

I Introduction, § 1(1)
Introductory, § 7(25)

L Loops, § 5(17)

M Matlab, § 1(1), § 4(13), § 5(17), § 6(19), § 7(25)
Matrix, § 2(3)
mex, § 6(19)

P Performance, § 5(17)

S Script, § 4(13)

V Vector, § 2(3)
Vectorizing, § 5(17)

W Workspace, § 2(3)

Attributions

Collection: *An Introduction to MATLAB*
Edited by: Anders Gjendemsjø
URL: http://cnx.org/content/col10323/1.3/
License: http://creativecommons.org/licenses/by/2.0/

Module: "An Introduction to MATLAB"
By: Anders Gjendemsjø
URL: http://cnx.org/content/m13255/1.1/
Page: 1
Copyright: Anders Gjendemsjø
License: http://creativecommons.org/licenses/by/2.0/

Module: "Using MATLAB"
By: Anders Gjendemsjø
URL: http://cnx.org/content/m13254/1.5/
Pages: 3-6
Copyright: Anders Gjendemsjø
License: http://creativecommons.org/licenses/by/2.0/

Module: "Graphical representation of data in MATLAB"
By: Anders Gjendemsjø
URL: http://cnx.org/content/m13252/1.1/
Pages: 7-12
Copyright: Anders Gjendemsjø
License: http://creativecommons.org/licenses/by/2.0/

Module: "Scripts and Functions in MATLAB"
By: Anders Gjendemsjø
URL: http://cnx.org/content/m13253/1.1/
Pages: 13-15
Copyright: Anders Gjendemsjø
License: http://creativecommons.org/licenses/by/2.0/

Module: "Vectorizing loops in MATLAB"
By: Anders Gjendemsjø
URL: http://cnx.org/content/m13251/1.4/
Page: 17
Copyright: Anders Gjendemsjø
License: http://creativecommons.org/licenses/by/2.0/

Module: "Writing C Functions in MATLAB (MEX-Files)"
By: Jason Laska
URL: http://cnx.org/content/m12348/1.2/
Pages: 19-24
Copyright: Jason Laska
License: http://creativecommons.org/licenses/by/1.0

Module: "Introductory Computer Assignment for MATLAB"
By: Anders Gjendemsjø
URL: http://cnx.org/content/m13256/1.5/
Page: 25
Copyright: Anders Gjendemsjø
License: http://creativecommons.org/licenses/by/2.0/

An Introduction to MATLAB
This course gives a basic introduction to MATLAB. Concepts covered include basic use, graphical representation and tips for improving your MATLAB code. Also included is an introductory computer assignment to test yourself after finishing the course.

About Connexions
Since 1999, Connexions has been pioneering a global system where anyone can create course materials and make them fully accessible and easily reusable free of charge. We are a Web-based authoring, teaching and learning environment open to anyone interested in education, including students, teachers, professors and lifelong learners. We connect ideas and facilitate educational communities.

Connexions's modular, interactive courses are in use worldwide by universities, community colleges, K-12 schools, distance learners, and lifelong learners. Connexions materials are in many languages, including English, Spanish, Chinese, Japanese, Italian, Vietnamese, French, Portuguese, and Thai. Connexions is part of an exciting new information distribution system that allows for **Print on Demand Books**. Connexions has partnered with innovative on-demand publisher QOOP to accelerate the delivery of printed course materials and textbooks into classrooms worldwide at lower prices than traditional academic publishers.