

# Math 216 Differential Equations

## Matlab - A Short Introduction

In Math 216, the principal computational tool is a software package called Matlab. Matlab (Matrix laboratory) is an interactive system designed for matrix computations: solving systems of linear equations, multiplying matrices, computing eigenvalues and eigenvectors, and much more. In this brief introduction, we will present some of the main features of Matlab, emphasizing some of the commands and functions that will be useful in Math 216. In a sense this introduction is a short version of the excellent manual by Mark S. Gockenbach, *A Practical Introduction to Matlab*. This manual is available online at

<http://www.math.mtu.edu/~msgocken/intro/intro.html>

and a postscript version (better for printing) is available from

<http://www.math.mtu.edu/~msgocken/intro/intro.ps>

We encourage the reader to experiment with the commands and to try out the examples while reading the material.

Matlab is available on most of the public computers on campus, and you may also purchase a copy of the software for your own computer if you like. On Macintosh computers, Matlab is launched either by clicking on the corresponding icon in the dock (if it is present), or by using the Finder to open the Applications folder, and then clicking on the Matlab icon in that folder. On Windows computers, Matlab can be accessed via the Start menu. Matlab is also available in a unix environment on Solaris-based machines that can be found throughout the university. If you are using a public computer and you want your Matlab work to be saved for future use, you need to set your working directory to one in your IFS space. Then whatever you save will be there the next time you log in. To exit Matlab, choose **Quit** from the **MATLAB** menu at the top of the screen. Other Matlab commands can be typed directly into the Matlab **Command Window**, after the Matlab prompt, `>>`. The command `clear` wipes out Matlab work done in the session and brings you back to the beginning of the session.

## Vectors and Matrices; Variables and Functions; Getting Help

When using Matlab, one of the most useful things to remember is that every variable is a matrix. The following commands show how to assign numbers, vectors, and matrices to variables. The Matlab prompt is `>>`; the commands below are typed in after the prompt, and concluded with a carriage return. The response of Matlab appears on the next line.

```
>> a=5  
a =
```

5

```
>> v=[1;3;0]
```

```
v =
```

```
1  
3  
0
```

```
>> A=[1,2,3;4,5,6;7,8,9]
```

```
A =
```

```
1    2    3  
4    5    6  
7    8    9
```

Notice that the rows of a matrix are separated by semicolons, while the entries on any given row are separated by commas (spaces may also be used). As mentioned above, each of these three variables is regarded as a matrix; the scalar  $\mathbf{a}$  is a  $1 \times 1$  matrix, the vector  $\mathbf{v}$  is a  $3 \times 1$  matrix, and  $\mathbf{A}$  is a  $3 \times 3$  matrix.

Indices of variables must be positive integers. Thus,  $\mathbf{x}(0)$  and  $\mathbf{x}(1.2)$  are not allowed, nor is  $\mathbf{A}(0,1)$  permitted. The size of a matrix can be found using the function `size(A)`, which returns the pair of numbers  $m, n$  when  $\mathbf{A}$  is an  $m \times n$  matrix. The length of the vector  $\mathbf{v}$  is found by using the function `length(v)`.

When entering a matrix, if you want to prevent Matlab from displaying it immediately afterward (for example, if it is a very large matrix), you need to end the line with a semicolon, as the following example shows.

```
>> x=[1,2,6];
```

Matlab says nothing because you ended the line with a semicolon. However, it has nonetheless remembered your definition of the vector  $\mathbf{x}$ .

Another useful thing to remember is that the complex unit  $\sqrt{-1}$  is represented by either one of the built-in variables `i` or `j`. The following examples demonstrate how complex numbers are displayed in Matlab. They also show that the square root function is a built-in feature:

```
>> sqrt(-1)
```

```
ans =
```

```
0 + 1.0000i
```

The variable `ans` contains the result of the most recent computation which can then be used as an ordinary variable in subsequent computations:

```
>> 10000-4*2*3
```

```
ans =
```

```
9976
```

```
>> sqrt(ans)
```

```
ans =
```

```
99.8799
```

```
>> (-100+ans)/4
```

```
ans =
```

```
-0.0300
```

Another built-in variable that is often useful is `pi`:

```
>> pi
```

```
ans =
```

```
3.1416
```

Only a few digits of the transcendental number  $\pi$  are displayed. More significantly only a finite number of digits of  $\pi$  are known to Matlab. This is because Matlab only deals in approximate arithmetic of real numbers. In particular, numbers smaller than a certain size cannot be represented by Matlab. This minimum size is stored in a built-in Matlab variable called `eps`:

```
>> eps
```

```
ans =
```

```
2.2204e-16
```

Besides the square root function, other common functions are predefined. They include:

<code>abs</code>	absolute value
<code>angle</code>	phase angle of a complex number
<code>sqrt</code>	square root
<code>real,imag</code>	real part, imaginary part of complex numbers
<code>conj</code>	complex conjugation
<code>round</code>	rounds to the nearest integer
<code>fix</code>	rounds towards zero
<code>floor</code>	rounds towards $-\infty$
<code>ceil</code>	rounds towards $\infty$
<code>sign</code>	signum function
<code>rem</code>	remainder (needs two input variables)
<code>sin,cos,tan</code>	usual trigonometric functions
<code>asin,atan</code>	usual inverse trigonometric functions

Here are some examples:

```
>> cos(1)^2+sin(1)^2
```

```
ans =
```

```
1
```

```
>> exp(1)
```

```
ans =
```

```
2.7183
```

```
>> log(ans)
```

```
ans =
```

```
1
```

Matlab has a comprehensive online help system which includes a list of built-in special functions and routines, as well as a list of other commands on which help is available. To obtain the list just type `help`. To get help on a particular command, type `help` followed by the command. An equally useful way to get help is to use the Matlab **Help** window, which is accessed by going to the **Help** menu at the top of the screen. The **Help** window allows you to search the Matlab manuals for information on the topic of your choice.

For example:

```
>> help pi
```

```
PI      3.1415926535897....
```

```
PI = 4*atan(1) = imag(log(-1)) = 3.1415916535897....
```

Another command is `help help` which describes how to find help. To view a few demonstrations, try typing `demo` in the **Command Window**. The demonstrations can also be accessed from the **Help** window.

## Operations on Matrices

Using Matlab we can perform standard arithmetic operations on matrices: addition, subtraction, and multiplication, as well as more advanced computations: finding row echelon form, finding eigenvalues and eigenvectors of a matrix and much more. Some of these latter operations are useful in studying systems of differential equations.

### Matrix arithmetic

If **A** and **B** are matrices, then Matlab can compute the sum, difference and the product of these two matrices (when these operations are well-defined). To do this, it is enough to type `A+B`, `A-B`, and `A*B`, respectively. Recall that order is important in matrix multiplication:

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

```
A =
```

```
    1    2
    3    4
```

```
>> B=[0,1;1,0]
```

```
B =
```

```
    0    1
    1    0
```

```
>> A*B
```

```
ans =
```

```
    2    1
    4    3
```

```
>> B*A
```

```
ans =
```

```
    3    4
    1    2
```

If  $A$  is a square ( $n \times n$ ) matrix, then typing  $A^2$  yields the matrix product  $A \cdot A$ . In general typing  $A^m$  gives the  $m$ -fold product  $A \cdot A \cdot \dots \cdot A$  ( $m$  factors).

Generally, applying to a matrix  $A$  any of the built-in functions returns a matrix of the same dimensions containing the values of the function as if it had been applied “componentwise”:

```
>> A=[0,pi/2;pi,3*pi/2]
```

```
A =
```

```
      0      1.5708
  3.1416      4.7124
```

```
>> sin(A)
```

```
ans =
```

```
      0      1.0000
  0.0000     -1.0000
```

We say that built-in Matlab functions are *vectorized*, indicating that they can be effortlessly applied to large vectors or matrices without the need to write complicated loops to cycle through the indices, as in some lower-level programming languages. If you want Matlab to multiply two matrices  $A$  and  $B$  that have the same dimension in a componentwise fashion (rather than the usual matrix multiplication) you should use the operator `.*` rather than `*`. Similarly, componentwise division of two matrices of the same dimensions can be accomplished by writing `A./B` which creates a matrix whose entries are  $A(i,j)/B(i,j)$ .

Matlab also contains several commands to make it easy to input certain standard types of matrices. For example, the built-in function `zeros(m,n)` returns an  $m \times n$  matrix of zeros. Similarly, `ones(m,n)` returns an  $m \times n$  matrix of ones. If  $v$  is a vector of  $n$  components, then the function `diag(v)` returns a square  $n \times n$  diagonal matrix whose entries are all zero with the exception of those running down the diagonal, which are set equal to the corresponding entries of the vector  $v$ . The case when  $v = \text{ones}(1,n)$  is especially important because this gives a diagonal matrix with all ones down the diagonal; this is the  $n \times n$  identity matrix. Matlab provides a special built-in function in this case: `eye(n)` (“eye” sounds like the letter  $I$ , which is usually used to denote the identity matrix).

In the special case of a matrix with only one row, we have a row vector instead of a matrix, and Matlab provides some specialized ways to input certain types of row vectors that occur frequently in practice. In Matlab, the notation `lo:step:hi` denotes a row vector whose first entry is `lo` and whose consecutive entries differ by `step`, and whose last entry does not exceed `hi`. For example,

```
>> t=1:2:9
```

```
t =
```

```
      1      3      5      7      9
```

The parameters `lo`, `step`, and `hi` do not have to be integers. For example,

```
>> y=0:.1:.4
```

```
y =
```

```
    0    0.1000    0.2000    0.3000    0.4000
```

Also, if the parameter `step` is omitted, Matlab assumes it is equal to one:

```
>> t=2:4
```

```
t =
```

```
    2    3    4
```

### More advanced operations

As mentioned above, Matlab can be used to do more advanced operations on matrices. For example, if **A** is a square matrix, then by typing

```
>> det(A)
```

Matlab computes its determinant.

In solving systems of linear equations, computing both the row echelon form of a matrix and finding the eigenvalues and eigenvectors of the matrix are very important tools. Let's see how to do this using Matlab. If **A** is a matrix then typing

```
>> rref(A)
```

Matlab finds a row echelon form of **A**. The particular row echelon form Matlab finds is called the row-reduced echelon form (row canonical form). This is a matrix for which all the pivots are 1 and all entries above the pivots are zero. Usually, a row echelon form is not unique. For example, the matrix

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & 2 \end{bmatrix}$$

is in row echelon form, but not in row-reduced echelon form. However, this matrix is equivalent (by replacing the first row by the sum of the two rows) to

$$\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & -2 \end{bmatrix}$$

which is in row-reduced form. Matlab would return the latter for any matrix row-equivalent to it.

If **A** is a square  $n \times n$  matrix, then typing

```
>> [r,s] = eig(A)
```

(the letters  $\mathbf{r}$  and  $\mathbf{s}$  are arbitrary choices) causes Matlab to compute a square  $n \times n$  matrix  $\mathbf{r}$  whose columns are eigenvectors of  $\mathbf{A}$ . The other object returned by `eig` is a diagonal  $n \times n$  matrix  $\mathbf{s}$  whose diagonal entries are the eigenvalues of  $\mathbf{A}$  corresponding to the eigenvectors returned in the columns of  $\mathbf{r}$ . Technically speaking, all the columns of  $\mathbf{r}$  are eigenvectors of  $\mathbf{A}$  only if  $\mathbf{A}$  is a *diagonalizable* matrix. If the eigenvalues are all distinct, then  $\mathbf{A}$  is diagonalizable, and sometimes  $\mathbf{A}$  is diagonalizable even if it has some repeated eigenvalues.

To be more precise, let us consider an example. Define a matrix

$$A = \begin{bmatrix} 1 & 0 \\ 2 & 2 \end{bmatrix}$$

by typing into Matlab

```
>> A=[1,0;2,2]
```

```
A =
```

```
    1    0
    2    2
```

Then ask Matlab to compute the eigenvalues and eigenvectors by typing

```
>> [r,s]=eig(A);
```

Here Matlab doesn't produce any output because we ended the line with a semicolon. To see what Matlab has stored in the matrices  $\mathbf{r}$  and  $\mathbf{s}$ , type

```
>> r
```

```
r =
```

```
    0    0.4472
  1.0000   -0.8944
```

```
>> s
```

```
s =
```

```
    2    0
    0    1
```

Thus, we see by looking at the diagonal matrix  $\mathbf{s}$  that the matrix  $A$  has two distinct eigenvalues, namely  $\lambda = 2$  and  $\lambda = 1$ . From looking at the matrix  $\mathbf{r}$  we then see that an eigenvector of  $A$  corresponding to the eigenvalue  $\lambda = 2$  is

$$v = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

and an eigenvector of  $A$  corresponding to the eigenvalue  $\lambda = 1$  is

$$v = \begin{bmatrix} 0.4472 \\ -0.8944 \end{bmatrix}.$$

Of course, eigenvectors are only defined up to multiplication of all entries by the same nonzero constant, so, for example,

$$v = \begin{bmatrix} 0 \\ -5 \end{bmatrix}$$

is also an eigenvector of  $A$  corresponding to the eigenvalue  $\lambda = 2$ .

## Graphs

Among the many features of Matlab, the ability to create graphs is one of the most useful. Here, we will describe how to deal with the usual situations, including plots of points in the Cartesian plane and graphs of the built-in functions.

To plot a number of ordered pairs  $(x, y)$  connected by straight lines, we just build row vectors  $\mathbf{x}$  and  $\mathbf{y}$  containing the  $x$  and  $y$  values and ask Matlab to make a plot:

```
>> x=1:5;
>> y=0:.1:.4;
>> plot(x,y)
```

Of course, since the vectors  $\mathbf{x}$  and  $\mathbf{y}$  contain two different coordinates of the same set of points in the plane, these vectors have to have the same length. If one wants to exhibit only the points (without connecting them with straight lines) the last command can be replaced by `plot(x,y,'o')`.

Because the Matlab functions are vectorized, constructing the needed vectors to graph a built-in function is easy. We first construct a vector of the desired  $x$  values, and then the corresponding  $y$  values come from applying the built-in function to the vector  $\mathbf{x}$  containing the  $x$  values:

```
>> x=0:.2:2*pi;
>> y=cos(x);
>> plot(x,y)
```

This produces a plot of the cosine function over the interval  $0 \leq x < 2\pi$ . To create a plot of a function such as  $y = 2x/(x+3)$ , which involves multiplication and division, we need to use the vectorized versions of these operations, writing `.*` for multiplication and `./` for division:

```
>> x=-1:.1:1;
>> y=2.*x./(3+x);
>> plot(x,y)
```

Some useful commands in making more sophisticated plots are the following:

- `xlabel('x axis label')`, `ylabel('y axis label')` labels the horizontal and vertical axes, respectively, in the current plot.

- `title('plot title')` adds a title to the current plot.
- `axis([a b c d])` changes the viewing window on the current graph to  $a \leq x \leq b$ ,  $c \leq y \leq d$ .
- `grid` adds a rectangular grid to the current plot.
- `hold on` freezes the current plot so that the subsequent plots you tell Matlab to make will be displayed on the same axes with the current one.
- `hold off` releases the current plot; the next plot will erase the current before displaying.
- `subplot` puts multiple plots in one graphics window.
- `legend` creates a small box inside the current plotting window that distinguishes and identifies multiple plots in the same window.
- `num2str(N)` returns the value of `N` as a string, which is helpful in producing informative titles of graphs.

Here is an example of using some of these commands. First, let's make a list of  $x$ -values:

```
>> x=0:.01:500;
```

Now we make a simple plot, illustrating a waveform exhibiting “beats”:

```
>> plot(x,sin(x)+sin(.9*x))
```

Matlab opens a new window containing the plot. Suppose now we want to view the same plot zooming in on the portion with  $28 \leq x \leq 31$  and  $-0.5 \leq y \leq 0.5$ . This modification of the current plot is easily accomplished using the `axis` command:

```
>> axis([28 31 -0.5 0.5])
```

You can issue repeated `axis` commands, in order to try to find the part of the plot that is the most interesting. To place a background grid on the current plot, which can be helpful in locating the coordinates of points on the plot, just use `grid`:

```
>> grid
```

You can plot several different graphs on the same axes. One way to do this is to plot the graphs separately and tell Matlab not to erase the current plot in between with the `hold on` command:

```
>> plot(x,sin(x/10)+sin(0.09*x),'r')
>> hold on;
>> plot(x,cos(x/5),'g')
>> hold off;
```

Here the 'r' and 'g' tell Matlab to make the plots in red and green color respectively. Another way to get the same result is to use a single plot command:

```
>> plot(x,sin(x/10)+sin(.09*x),'r',x,cos(x/5),'g')
```

Regardless of how the two graphs were put onto the same set of axes, you can add a legend to the plot to indicate which curve is which as follows:

```
>> legend('sin(x/10)+sin(.09*x)', 'cos(x/5))
```

Adding a title to the plot is also easy:

```
>> title('Two curves')
```

Finally, while colors are a good way to distinguish different curves on the screen, it is often easier to distinguish them on a black-and-white printout if they correspond to different kinds of lines. For example,

```
>> plot(x,sin(x/10)+sin(.09*x),'-',x,cos(x/5),'--')
```

plots the beats with a solid curve, and the cosine graph with a dashed curve. These plot directives can be combined. For example, 'or' means plot points only, and make them red, while '--g' means connect the points plotted with a green dashed curve.