Matlab for Math 203

Ethan Akin, Peter Brinkmann,
Edward Grossman and Raymond Hoobler

Department of Mathematics
City College of New York

November, 2007

Table of Contents

Introduction		1
Lab 01	Introduction to Matlab	3
Lab 02	m-Files	8
Lab 03	Symbolic Manipulation and Anonymous Functions	15
Lab 04	Graphing Surfaces and Plotting Lines in Space	21
Lab 05	Curves in Space	26
Lab 06	Graphing Surfaces and Their Tangent Planes	28
Lab 07	Gradients and Max/Min Problems	31
Lab 08	Graphing Parametrized Surfaces	35
Lab 09	Multiple Integrals	40
Lab 10	Iteration and Taylor Series Approximation	45
Appendix A:	Symbolic Expressions in Matlab	49
Appendix B:	Publishing m-files to HTML	55
Matlab Review Sheet		59
Sample Lab Exam Questions		61
Index		64

Introduction

MATLAB is a widely used software package with a lot of numerical and symbolic power. The program uses arrays of numbers to do many things.

This booklet is intended to be an introduction to Matlab. Our intention is to go through approximately one Lab per week with occasional pauses to get caught up. For each lab there is work to be done in class, some follow-up reading and some homework to be handed in the following week. The work requires a computer equipped with Matlab and so you should allot some time during the week (Mon-Thurs) when the Artino Lab is open. The Lab is being used for these Math 203 classes on Friday and so is not available then. Any computer equipped with Matlab (version 7) will do, provided that it includes the SYMBOLIC TOOLBOX which we will be using.

This set of notes is not as complete as a regular Matlab text would be. It needs to be supplemented with notes that you take during in-class work and from your use of the HELP command.

The labs are designed to introduce some of the graphic and calculus capabilities of MATLAB.Since the lab period is only one hour, you should prepare for it by reading ahead of time the material and problems to be covered. You will be working in pairs at the computers, but you will be tested individually at the end of the semester. So be sure to spend some of your time working at the keyboard. In addition you will probably have to spend some time outside of class working on the homework exercises. But by the end of the semester you should have developed a reasonable facility with Matlab that you can use in other courses.

Learning to use a software package like Matlab takes up a lot of time, especially in the beginning. As with any new computer package, the learning goes best if you can relax, play around and enjoy the pretty output on the screen.

There are two manuals which you should be aware of, although we won't be using them. They do provide the details omitted from these notes.

A Guide to Matlab for Beginners and Experienced Users, 2nd Ed. by Brian R. Hunt, Ronald L. Lipsman and Jonathan M. Rosenberg, (Cambridge University Press, 2006) is a well-written text which, as the title suggests, starts you out at the beginning and then proceeds on to material beyond what we will be using. It is a very nice book, but it is not cheap (about \$ 50).

Mastering Matlab 7 by Duane C. Hanselman and Bruce L. Littlefield (Prentice Hall, 2004) is comprehensive, but still readable, also expensive (about \$ 70, but cheaper at Amazon).

In addition, there is a free site which is up-to-date and very helpful: www.utexas.edu/its/rc/tutorials/matlab/.

Finally, in the Matlab Help window, accessed under Help on the toolbar, there is a tab for Demos. Clicking on the folder Matlab you will see a list of many helpful illustrated documents and videos.

Warning: Sometimes you will want to interrupt a calculation. You may have made a mistake and are hung up in a loop, or perhaps a calculation is just taking too long. It is important to know that CTRL+C (that is, hold down the button labeled Control or Ctrl, and press c)is the equivalent of (and is more effective than) screaming "Stop" at the screen.

Lab 1: Introduction to Matlab

Matlab Objects and Operations: number arrays: vectors and matrices, arithmetic of arrays and operations on arrays, anonymous and built-in functions, `pi` (but not `e`), Edit menus, colors and line styles

Matlab Commands: `linspace`, colon operator (`:`), semicolon (`;`), `size`, `length`, transpose(`'`), `diff`, `sum`, `prod`, `max`, `min`, `plot`, `hold on`, `hold off`, `hold`, `figure`, `help`, (optional: `polar`)

The Mat in Matlab stands for matrix. A matrix is a rectangular array of numbers. Manipulating such arrays is a central theme in Matlab.

Work in class: Try out some manipulations with arrays:

```
a = [1 3 0 -1 pi]
b = [3:7 a]
b = [3:7; a]
b = b'
b = b'
c = 3*a + 100
d = a.^3
e = linspace(0,1,5)
f = [0:.2:1]
g = a.*e./(1+e)
h = [a  d  c]
a(4)
b(1,3)
b(5)
size(a)
size(b)
length(e)
length(f)
k = h(1:2:length(h)-1)
m = max(k)
n = sum(a)
p = diff(h)
```

Beginning with two arrays `a` and `b` of the same size the sum is `a + b`. Multiplication by a scalar 17 or π is written `17*a` or `pi*a`. You cannot omit the `*` writing `3a` as we often do in algebra because Matlab interprets this as the name of some new object rather than three times the old object `a`. For array multiplication, division and exponentiation, e.g. `a.*b`, `a./b` and `a.^3`, you must include the period '`.`'. This is called the *vectorized* expression.

We will later make use of the fact that the position of an entry of a matrix b can be specified in either of two ways: $b(i, j)$ is the entry in row i and column j. For $b(N)$ you begin to count, down the first column, the down the second, and so forth until you reach position N and so N can be any integer from 1 to m times n.

Drawing pictures with Matlab, using its graphics, will be the most important application for us. For example, the response to the command `plot(x,y)`, applied to a pair of vectors of the same length, consists essentially of three steps:

1. Pair off the two number arrays to get a list of pairs of numbers.
2. Plot the points whose coordinates are on the list.
3. Connect the successive dots with line segments.

A useful test of your understanding is to plot the line segment which connects the points with coordinates $(2, 3)$ and $(5, 0)$.

Observe how the ARROW UP key lets you repeat a command which you earlier executed without typing the whole thing out again. As you keep hitting the ARROW UP key the command line scrolls back through the previously typed commands. If you type a letter or two first, then it scrolls back only through the earlier commands which began with the same few letters.

Work in class: Over the interval $[0, \pi]$ we graph the functions $f(x) = \pi \sin(\frac{x}{2})$ and $g(x) = \frac{x^2}{\pi}$ on the same figure by using the following commands:

```
x = linspace(0, pi);
y1 = pi * sin(x/2);
y2 = x.^2 / pi;
```

```
plot(x, y1, x, y2)
figure
plot(x, y1)
hold on
plot(x, y2)
```

Observe that the results on the two figures are (almost)the same. When two graphs are plotted at once on the same figure, as in `plot(x,y1,x,y2)`, two different colors are automatically used. You can choose the colors and line styles, by writing, for example, `plot(x,y1,x,y2,'r')` or `plot(x,y1,':g',x,y2,'-.')`. The commands for colors, line styles and point markers (which are used to label individual points) are listed in the following table (see also `help plot`).

Line style	Symbol	Point marker	Symbol	Color	Symbol
solid(default)	-	plus	+	red	r
dashed	- -	circle	o	green	g
dotted	:	asterisk	*	blue	b
dash-dot	-.	point	.	cyan	c
		cross	x	magenta	m
		square	s	yellow	y
		diamond	d	black	k
		up triangle	$\wedge$	white	w
		down triangle	v		
		right triangle	>		
		left triangle	<		
		pentagram	p		
		hexagram	h		

Notice that you use single quote marks to call for these. Both the left and right QUOTE marks are typed using ' next to the ENTER key.

In the above program we used one of the BUILT-IN FUNCTIONS, `sin(x)`. Also built in are `cos(x)` and `tan(x)`. Others are the square root function `sqrt(x)` and the exponential function e^x which is written `exp(x)`. The natural logarithm function is `log(x)`. The common, base 10, log is `log10(x)`. These functions are *vectorized*, meaning that you can substitute either a single number or an array.

You can build your own functions as *anonymous functions*:

Work in class: Try the following code:

```
f = @(x,y) x^2 + 3*x*y
a = [0:3]
b = [3:6]
c = f(a,b)
```

Oops. What was the error and how do you fix it?

Assignments: Read from the screen the Matlab responses to `help length`, `help size`, `help linspace` and `help plot`. Similarly use `help` to understand the commands `max`, `min`, `sum`, `prod` and `diff`.

In the Lab, during the week: Do the following exercises.

1. For the function $f(x) = (1+x)/(1+x^2)$, define an anonymous function. Then do two plots. For the coarse plot use `x = [-3:.5:3]` and `y = f(x)`. Then let `a = [x ; y]'` to get a table of values. For the fine plot use `x = linspace(-3,3)`. Don't forget the semicolon ';' after the definitions of `x` and `y` this time.
2. With `t = linspace(0,2*pi);` let `x = cos(t)./3;` and `y = sin(t);` Now look at `plot(x, y)`. This is the plot of the parametric representation of a curve. What curve is it?

   Label your figures using the Figure Window Toolbar. Use Edit / Axes Properties / Labels. To show the grid use Edit / Axes Properties / Scale and check the On box. To change the color or line style of a graph, first move the cursor to the graph, click on it and then use Edit / Current Object Properties / Style.

3. (Optional) Matlab draws nice polar graphs. Look at `help polar`. Define: `t = linspace(0,2*pi);` then look at `polar(t,sin(k*t))` for different values of k. For example, try $k = 2, 4, 5$. You can put them all on the same figure by using `hold on`. It is then best to use different colors, like `polar(t,sin(4 * t),'r')` for red. Notice that if you use `hold on` before you ask for the first polar graph you get a different picture from what you get if you apply `hold on` after your first polar graph (Try it). Can you explain what is going on? Hint: Try using `hold on` alone and see what you get.

Written homework to be handed in: You should do these without using the computer.

1. Assuming that

```
x = [0:.1:1]
y = linspace(0,1,10)
A = [0:5;-5:0;1,3,7,9,-10]
```

compute

(a) `x(1), y(1), x(2), y(2)`

(b) `A(2, 3), A(5)`

(c) `length(x)`, `length(y)`, `size(x)`, `size(A)`

(d) `3*A.^2-1`, `A'`

2. Is it true that `pi*linspace(0,1,10)` equals `linspace(0,pi,10)`? Explain your answer.

Lab 2: m-Files

Matlab Objects and Operations: m-files (function m-files and scripts), anonymous functions and function handles, logic operators, character arrays (strings)

Matlab Commands: `<, <=, >, >=, ==, ~=`,
`max`, `min`, `fzero`, `sort`, `unique`,
`pause`, `disp`, `num2str`,
`title`, `text`, `xlabel`, `ylabel`, `grid on`,
(optional: `echo on`, `echo off`,`comet`)

For any computation or plot which requires more than two or three commands it is helpful to use an *m-file*, also called a *script*. Then, during the inevitable debugging, you don't have to retype the whole list of commands.

Work in class: We graph the function given by:

$$f(x) = \begin{cases} x^2 + x & \text{if } \ x < 1 \\ \sin(x) & \text{if } \ x \geq 1. \end{cases}$$

First, we define an anonymous function:

```
f = @(x)(x.^2+x).*(x<1) + sin(x).*(x>=1)
```

Recall from Lab 1 the pattern of an anonymous function. It begins with a declaration of variables - in parentheses following the @ symbol. Then the formula is written out. It is important to use the vectorized formula - put in the dots.

This particular function has a branched definition with two cases. These are handled by the relation operators. The sentence $2 \leq 10$ is written in Matlab `2 <= 10`. This is a true sentence and so is interpreted by Matlab as the number 1. The false sentence $2 = 10$, written `2 == 10` is interpreted as 0. Try out

```
a = [0 : 5]
a < 2
a == a.^2
```

Write the following as an m-file and save it as `Lab2Class`. You can cut-and-paste (use Ctrl c and Ctrl v) rather than rewrite the formula for `f` which we are now calling `Jump` . Each separate command will occur on a numbered line. This is helpful because when (not if) Matlab detects an error, it tells you the number of the line where the problem is located. The gaps below (after `echo on` and `Jump`, for example) are for your convenience to group the commands in readable chunks. In typing, just ignore them and don't bother skipping a line. The computer doesn't need that kind of help.

```
% This is a comment.  It is ignored as an instruction.
clear all; close all
echo on

%First we define the anonymous function again.
Jump = @(x)(x.^2+x).*(x<1) + sin(x).*(x>=1)
x1 = linspace(-1, 1);
x2 = linspace(1, 2 * pi);
x  = [x1, x2];
y1 = x1.^2-2;
y2 = sin(x2);
y  = Jump(x);

plot(x1, y1)
hold on
plot(x2, y2)
grid on
pause

figure
plot(x, y)
hold on
grid on
pause

[M,i] = max(y);
disp(['maximum point is  (',num2str(x(i)),',', num2str(M),')'])
pause
```

```
plot(x(i), M, 'rd')


[m,j] = min(y);
disp(['minimum point is  (',num2str(x(j)),',', num2str(M),')'])
pause


plot(x(j), m, 'gd')
echo off
```

IMPORTANT: When you change an m-file, for example when you are debugging it (correcting mistakes), always remember to hit SAVE before you run it again or else Matlab will run the old version.

Run the file `Lab2Class`, compare the two figures, and explain the difference. Why didn't we use `plot(x1,y1,x2,y2)` in the first figure? Is it true that `y == [y1, y2]` ? If so, could we have done without the anonymous function in this case?

The third portion of the m-file computes the maximum and minimum points on the graph of the function and puts red and green diamonds at the points in the second figure. The instruction `[M,i] = max(y)` yields two values. The first, labeled M, is the maximum value and the second, labeled i, is the place where the maximum value is located (or actually its first occurrence). Look at

```
a = pi * [-2, 7, 0, 1, 7, 2]
[M,i] = max(a)
```

For the x coordinate we use $x(i)$, that is, the entry of the x vector with the same index.

Notice that we used the COLOR and POINT MARKER commands red ('r') and diamond ('d') to mark the maximum point.

When you do an exercise like this there is some question how to do the labeling for a Figure. The command line labels, `title`, `xlabel`, `ylabel`, `grid on` are used when you do the labeling as part of an m-file. It is quicker, and easier, to use the Figure Window Toolbar as described in Lab 1. Your script would then simply plot the functions. Then, after running the script, add the labeling and tick marks. Be sure to use different colors for the graphs in your display and different line styles for your printed output. Experiment using the Toolbar so that you can see how there commands work.

We used the commands `echo on` and `echo off` at the beginning and end of the m-file. I recommend this, but it is not required. This has the effect of showing the commands on the screen, in addition to their outputs, when you run the m-file.

It is a good habit to begin every m-file with the commands `clear all` and `close all`. This clears any previously assigned values of variables and closes the current figures. If you don't put in the initial `close all` command and you run the program several times, to debug it for example, then you may keep opening new figures that you have to close by hand. Furthermore, it may be hard to figure out which figure is which.

In an anonymous function the whole formula beginning with @ is called a FUNCTION HANDLE so `Jump` <u>is</u> a function handle. Often you want to refer to the function in an instruction and you use its handle. The command `fzero` uses a function handle and a numerical guess to find a root near the guess. Here are two different versions of the same instruction.

```
a = fzero(@(x)(x.^2+x).*(x<1) + sin(x).*(x>=1),3)
a = fzero(Jump,3)
```

There is an alternative way to define a function using a special sort of m-file called a function m-file. We won't be using them and so you needn't worry about them. What you do need to know is that the built-in functions, like cosine, written `cos`, <u>are</u> really function m-files and the name, like `cos`, is not a function handle. You can convert it to a function handle by attaching `@`. Compare

```
a = fzero(cos,1)
b = fzero(@cos,1)
```

While we won't be using function m-files, I will mention the two advantages of function m-files. They are useful when the formulae become very long and complicated because like other m-files they are preserved in memory until deleted, and so can be debugged. The maintenance in memory is itself the other advantage. The anonymous functions are only temporary. Try asking for `fzero`(f,3). Remember that `f` was the original label we used for function handle now called `Jump`. The problem is that the `clear all` instruction at the beginning of `Lab2Class` wiped `f` out of memory.

Assignments: Read `help max`, `help min` and `help fzero` to see how these commands work.

Below, you will find an m-file. It illustrates the effect of round-off error when you try to compute $Lim_{x\to\infty}(1+\frac{1}{x})^x$ by substituting larger and larger values. Type it out and run it to observe what happens. You first rapidly approach the correct value: $e = 2.718....$ Then, suddenly, the values jump around and finally collapse to 1. You should be able to figure out why the computer gives the value 1 when you substitute very large values of x. In the script the command `comet` is used instead of `plot`. When used with `hold on` it generates the plot by moving from point to point instead of showing the entire graph all at once. Notice also the different labeling commands. Although you won't be held responsible for it, using `comet` is fun. However, you have to fix the axis values in advance (here x goes from -1 to 70 and y goes from -1 to 8) otherwise the axis will keep changing to fit the curve and this will look confusing. Try it by running the m-file without the axis command. To do this just put a `%` in front of the line. Notice, too, how the tick marks are set. The `gca` command says "get current axes".

```
clear all; close all
x = [0:.01:45, 45:.001:55, 55:.1:70];
y = (1 + 2.^(-x)).^(2.^x);
```

```
figure
axis([-1 70 -1 8]);
title('y = (1+(1/x))^x');
xlabel('log_2(x)');
ylabel('y');
set(gca, 'Xtick', [0:10:70], 'Ytick', [-1:.5:8]);
hold on
comet(x,y)
plot(x,y)
```

Lab Homework: This is to be submitted via Digital Dropbox or printed out and handed in. Follow the instructions given in class.

1. Consider the function f with branched definition

$$f(x) \quad = \quad \begin{cases} x \ - \ 2x^{\frac{3}{2}} & 0 \le x \le 1 \\ x \ - \ 2 \ - \ \cos(\frac{\pi x}{2}) & 1 < x \le 3. \end{cases}$$

Write an anonymous function for f and Plot the graph of the function over the interval to allow you to estimate its zeros. Use these estimates in the `fzero` command to find the positive zeros (that is, the positive values of x at which the function equals zero). On the plot of the function set the Xtick marks to include the zeros and the integer values in the interval.

Remember that you should use the function handle of an anonymous function when you apply the command `fzero`. In setting Xtick and Ytick the list has to consist of numbers in increasing order. You can first list the values in any order, for example, a = [0:3, zero1, zero2] and then use the `sort` command. That is, replace a by `sort`(a). The only difficulty left is that `sort` does not eliminate repeats and no repeats are allowed are allowed in the list. Use instead the command `unique`. Replacing a by `unique`(a) sorts the elements of a and eliminates the repeats.

2. Plot the function given by $f(x) = 2\sin(2x) - 3\cos(\frac{x}{2})$ over the interval $[0, 2\pi]$ using steps of length .001 (How?). Use the commands `max` and `min` to estimate the maximum and minimum points. Include the maximum and minimum points as tick marks on the x-axis and the maximum and minimum values as tick marks on the y-axis.

Lab 3: Symbolic Manipulation and Anonymous Functions

Matlab Objects and Operations: symbolic arrays,
symbolic differentiation and integration,
obtaining an anonymous function from a symbolic formula,
numerical integration

Matlab Commands: `syms`, `sym`, `double`,
`subs`, `diff`, `int`, `quad`,
`figure(gcf)`, `input`, `whos`,
(optional: `char`, `eval`, `while ... end`)

Work in class: One of the advantages of Matlab is the ease of plotting several functions on the same graph. We illustrate this by plotting a function, its derivative and a tangent line (at a point whose x-coordinate you choose) in the same figure by using the following script. In the process we introduced the Symbolic Toolbox which is used to do symbolic differentiation and integration.

```
clear all; close all
f = @(x) x.*cos(x.^2);
syms x
fxsym = diff(f(x))
fx = @(x) subs(fxsym)
pause

u = linspace(-.5, 3);
v = f(u);
w = fx(u);
whos
pause

plot(u, v)
grid on; hold on
pause
```

```
plot(u, w, 'g')
pause

p = input('x-value for tangent line point?  ')
q = f(p) + fx(p) * (u-p);
figure(gcf); plot(u, q, 'k'); plot(p, f(p), 'rd')
pause

Fsym = int(f(x))
F = @(x) subs(Fsym)
disp(['The integral from 0 to 2 of f is F(2)-F(0) = ',
        num2str(F(2)-F(0))])
disp(['The numerical approximation given by quad is  ',
        num2str(quad(f, 0, 2))])
```

The important command `p = input('question?   ')` is used for interactive work. There is a pause which allows you to enter the answer you choose to the question in the parentheses. When you type and hit ENTER the program continues and uses for p the value you entered.

This script also illustrates several aspects of the relationship between symbolic expressions and anonymous functions. We began with `f`, which is an anonymous function (`f` itself is the function handle). Since `syms x` turns `x` into a symbolic object, we obtain a symbolic object `f(x)` when we substitute into `f`. We use the notation `fxsym` for the symbolic object that we get by applying symbolic differentiation to the symbolic object `f(x)` (not to the anonymous function `f`). Now we use `subs` in order to turn the symbolic object `fxsym` into an anonymous function that we can evaluate numerically.

We used the important substitution command `subs` to convert a symbolic expression to an anonymous function. In general, it is used to replace a variable in a symbolic expression. `subs(g,x,h)` means substitute for the symbolic variable `x` into the symbolic expression `g` the numerical or symbolic expression `h`. Try these out:

```
syms x
g = x^2
a = [1 : 5]
g(a)
b = subs(g,x,a)
c = subs(g,a)
z = subs(g,sin(x))
G = @(x) subs(g)
G(a)
G(x)

syms y
g = g*y
z = subs(g,x,sin(x))
w = subs(g,y,sin(x))
G = @(x,y) subs(g)
G(x,y)
G(a,y)
G(x,a)
```

Notice that the command `subs` automatically vectorizes the expression when the substitution is performed. Also, when there is only a single variable it can be omitted in the command. Finally, observe that when used to define an anonymous function the specification of variables is left to the declaration beginning with `@`.

Finally, recall that in Lab 1 we applied the command `diff` to numerical arrays to get the vector of successive differences. Here the same command applied to a symbolic object computes the symbolic derivative.

The Symbolic Toolbox also allows us to manipulate with fractions and exact expressions in radical form.

```
a = pi/2
f = @(x) x.^2.*sin(x)
syms x
g = cos(x/2) + sin(x/10)
```

```
b = f(a)
c = f(sym(a))
d = double(c)
b = subs(g,a)
c = subs(g,sym(a))
d = double(c)
```

With the introduction of symbolic objects, we have now met four of the five types of Matlab expressions which will occur in this course. They are:

- NUMERICAL ARRAYS: These are the numerical vectors and matrices. Examples: `a = 2`, `b = linspace(1,pi)`, `c = [1:5; 2 6 9 0 1]`.
- FUNCTION HANDLES: Associated with anonymous functions. Using `@` they can be attached to function m-files or built-in functions. Examples: `@(x) x.^2`, `@exp`.
- SYMBOLIC OBJECTS: After certain variables have been reserved as symbolic, for example, by using `syms` x, symbolic objects can be defined directly or obtained by symbolic differentiation or integration. Examples: `g = sin(x^3)`, `gx = diff(g)`, `G = int(g)`. Notice that `g` looks like a function handle, but it isn't one. It is a symbolic object. `g(a)` yields an error message.
- CHARACTER ARRAYS: Also called STRINGS, these are arrays, like vectors or matrices, but not of numbers, merely of typed expressions which are not interpreted. Expressions in single quotes are strings. In our work we have already used `disp()` which just displays the string in parentheses. Example:
  `disp(['The number pi is ' num2str(pi)])`.
  The command `num2str(pi)` converts the number pi to a string. Compare: `a = 3 + 2`, `b = '3 + 2'`, `c = eval(b)`. The command `eval( )` evaluates the string. The command `char( )` converts a symbolic object to a string while the command `sym( )` goes the other way, converting a string to a symbolic object. For example, `syms x` is equivalent to writing `sym('x')`.

- CELL ARRAYS: Also called LISTS, we won't use these much, but we include them for completeness. A list is written with braces `{ }` instead of parentheses `( )` or brackets `[ ]`, e.g., `L = {5, pi/2, 2}`. If we reserve `syms x y z` and then define the symbolic object `M = x * sin(y) * log(z)`, then we can substitute the values 5, $\pi/2$ and 2 for x, y and z by using `subs(M, {x,y,z}, L)`.

It is often hard to keep track of which expressions are which among strings (= character arrays), symbolic objects and functions. You can check by typing `whos`.

Assignments: Read `help quad`. Look over APPENDIX A: SYMBOLIC EXPRESSIONS IN MATLAB which is included at the end of this booklet.

Lab Homework: This is to be submitted via Digital Dropbox or printed out and handed in. Follow the instructions given in class.

Consider the function given by $f(x) = sin(x^2)cos(x)$. Write out an m-file to do the following:

1. Define the function and its derivative as symbolic expressions and as anonymous functions.

2. Graph both the function and its derivative over the interval $[0, \pi]$, using different linestyles for the two graphs. End this portion with a `pause`.

3. Looking at the graphs, observe roughly where the first positive root of the derivative occurs. Enter this as `guess =` and then input your rough numerical guess. Then apply `a = fzero( , guess)` to get a more precise estimate for this root. Be careful: Which function goes in `fzero()`? That is, you are looking for the place where something equals zero. What is the something?

4. Evaluate the function to obtain the corresponding critical point (a, b). Get the current figure back and plot a diamond at the critical point.

5. (Optional) You can list and plot all of the critical points by including in your m-file program a loop of the following sort.

```
k = 1; a = [ ];
while k == 1
  guess = input('guess for next root?   ')
  p = fzero(fx, guess)
  figure(gcf);  plot(p, f(p), 'd')
  a = [a, p]
  k = input('Enter 1 if  more roots. Otherwise enter 0')
end
figure(gcf)
pause
disp('Table of Critical Points Between 0 and pi')
disp([a; f(a)]')
```

Lab 4: Graphing Surfaces and Plotting Lines in Space

Matlab Objects and Operations: rotating and
zooming in/out for surface plots,
anonymous functions of several variables, the constant EPS

Matlab Commands: `cross, meshgrid, surf`, `plot3`,
`shading interp, axis equal`,
`contour, clabel, contour3`
(optional: `mesh`)

We introduce three dimensional plotting by graphing planes and parameterized lines. The command `meshgrid` is used for surface plotting the way `linspace` or the colon operator is used for lines and curve plotting in the plane. It is used to generate input locations. In response to `[x,y] = meshgrid(-1 : 2, 0 : .5 : 2)` you get two matrices, each has size [5 4] (that is, 5 rows and 4 columns) corresponding to the `x` values $-1, 0, 1, 2$ and the `y` values $0, .5, 1, 1.5, 2$. In the `x` matrix the 4 x values are repeated in 5 identical rows. In the `y` matrix the 5 y values are repeated in 4 identical columns. Thus, in the $(2, 3)$ position (second row, third column) is the second y value and the third x value. The y is looked at first because the conventions are: rows then columns; x horizontal and y vertical.

However, you have to be careful about the number of divisions you use. If you use 100 divisions for x and y (the default value for `linspace`) then the response to the `[x,y] = meshgrid(...)` command defines each of `x` and `y` as a 100×100 matrix, which has 10,000 entries. Of course, the computer can easily handle data sets of that size but graphing becomes very slow and the grid is too fine. On the other hand, you don't want the grid to be too coarse either. In general, it is best to use between 10 and 20 subdivisions for `x` and `y`. Don't forget the semicolon `;` .

Work in class: Consider the two planes with equations

$$\begin{aligned} x + y + z &= 2 \\ 3x - 4y + 5z &= -8 \end{aligned}$$

We graph them and plot the line of intersection between them. We want to make sure that the rectangle over which we graph the two planes contains the projection of some points that lie in both planes. So we begin by computing a point of intersection and then use a square centered on the xy coordinates of the point we find. We set $z = 0$ and then solve the two equations for x and y. You should try this by hand so that you can understand the Matlab procedure for solving the system. It is given in the first three lines of the script.

```
clear all; close all
A = [1, 1; 3, -4]
b = [2; -8]
inter = A\b
disp('An intersection point is      ')
p = [inter', 0]
pause

[x,y] = meshgrid(p(1) + [-1:.2:+1], p(2) + [-1:.2:+1]);
z = 2 - x - y;
surf(x, y, z)
pause
shading interp; hold on;
pause

z = (-8 - 3 * x + 4 * y)/5;
surf(x, y, z)
pause

disp('parallel vector for intersection line is     ')
v = cross([1, 1, 1],[3, -4, 5])
pause

M = (abs(v(1)) + abs(v(2)))/2;
t = [-1 : 1]/M
x = p(1) + t * v(1); y = p(2) + t * v(2); z = p(3) + t * v(3);
plot3(x, y, z, 'k'); plot3(p(1), p(2), p(3), 'rd')
figure(gcf)
```

The cross product of two three dimensional vectors is given by the command `cross`. For the dot product of two vectors a and b with the same length we use `sum(a.*b)`. The command `surf` is used to graph surfaces. We used it above for the planes. The command `plot3` is used to graph curves in space given by parametric equations. We used it to graph the line of intersection.

We graph surfaces of the form $z = f(x, y)$ by starting with the `[x,y] = meshgrid(...)` and then by expressing z directly as a formula, as in `z = x.^2  +  y.^2.` or by first defining an anonymous function and then substituting as in `z = F(x,y).`

Work in class: We use the function given by $f(x, y) = x^2 - y^2$ to illustrate the different surface plots and `contour` commands.

```
clear all; close all
f = @(x,y) (x.^2 - y.^2)
[x, y] = meshgrid(-1 : .02 : 1);
z = f(x, y);
mesh(x, y, z)
pause

figure; surf(x, y, z)
pause

shading interp; hold on
pause

contour3(x, y, z, 'k')
pause

clabel(contour3(x, y, z, 5, 'k'))
pause

figure; contour3(x, y, z)
pause
```

```
figure; contour(x, y, z)
hold on;
pause

clabel(contour(x, y, z, 5, 'k'))
```

Here we see different surface plots given by Matlab. Alternative pictures of the surface itself are given by `mesh` (which we won't need, but you should be aware of it), `surf` and `surf` followed by `shading interp`. The command `contour` draws contour curves in the xy plane while `contour3` draws them up on the surface. As illustrated here, `contour3` is best used in a figure after `surf` `shading interp` `hold on` . You can adjust the number of contour lines with a numerical instruction: `contour3(x,y,z,20,'k')` draws 20 black contour lines on the surface. The command `clabel` is used to label the contour curves with the z value.

With the surface plots it is instructive (and fun) to use the zoom and rotation commands which are on the bar at the top of the figure.

Now we use the same sequence of steps to look at the function given by $f(x,y) = \frac{x^2-y^2}{x^2+y^2}$. We use the graph near zero to consider whether $\lim_{(x,y)\to(0,0)} f(x,y)$ exists or not. Notice that we include `eps` in the denominator to avoid a division by zero and so avoid a hole that would include the origin (exactly the location of interest).

Just go back the the m-file you just used, replace the function f in your m-file by `f = @(x,y) (x.^2 - y.^2)./ (x.^2 + y.^2 + eps )` and run it again.

Lab Homework: This is to be submitted via Digital Dropbox or printed out and handed in. Follow the instructions given in class.

1. Write an m-file that first graphs the plane with equation $x + y - z = 2$. Graph it over a square in the xy plane centered at the point $(3, 1)$. Then, on the same figure, plot the line through $(3, 1, 2)$ that is normal to the plane. Use the `axis equal` command so that your normal actually looks perpendicular to the plane.

2. Plot the graph $f(x, y) = (x^2 - y^2)^2 + y^3$. Make a plot with 20 contour traces on the surface.

3. Plot the graph of the cylinder in space given by $y = x^3$. Here you can't use `[x,y] = meshgrid` (why not?). Try `[x,z] = meshgrid` instead, but use `surf(x,y,z)` as before.

Lab 5: Curves in Space

Matlab Commands: `diff`, `input`

Work in class: We graph a parametrized curve, compute its length and draw a tangent line.

```
clear all; close all;

len = @(x,y,z) sqrt(x.^2 + y.^2 + z.^2)

f = @(t) t.*cos(5*t)
g = @(t) t.*sin(5*t)
h = @(t) t.^2/10

t = linspace(0, 2*pi, 10);
x = f(t); y = g(t); z = h(t);
dx = diff(x); dy = diff(y); dz = diff(z);

coarse = sum(len(dx, dy, dz));
plot3(x,y,z); grid on
disp('Coarse polygonal length estimate equals ');
disp(coarse)
pause

t = linspace(0,2*pi);
x = f(t); y = g(t); z = h(t);
dx =diff(x); dy = diff(y); dz = diff(z);

fine = sum(len(dx, dy, dz));
plot3(x,y,z); grid on ; hold on
disp('Fine polygonal length estimate equals ');
disp(fine)
pause

syms t;
x = f(t); y = g(t); z = h(t);
dx = diff(x)
```

```
dy = diff(y)
dz = diff(z)

speed = len(dx, dy, dz)
arc = quad(@(t) subs(speed), 0, 2*pi);
disp('Arc length integral equals ');
disp(arc);
pause

t0 = input('t value for tangent line (t in (0,2pi))')
p0 = [f(t0), g(t0), h(t0)]
V0 = subs([dx,dy,dz],t,t0)

s = [-1:1];
plot3(p0(1) + s*V0(1), p0(2) + s*V0(2), p0(3) + s*V0(3), 'g')
plot3(p0(1), p0(2), p0(3), 'rd')
```

Here we see contrasted the two uses of the command `diff`. For the coarse and fine polygonal estimates, we used the numerical command. Later, we obtained dx, dy, dz, symbolic expressions in the variable t, by using symbolic differentiation. Applying the function *len* we obtain *speed* as a symbolic expression in t. In `quad` we used the conversion of *speed* to the handle of an anonymous function.

Lab Homework: This is to be submitted via Digital Dropbox or printed out and handed in. Follow the instructions given in class.

For the curve given parametrically by $x = \cos(t); y = \cos(2t); z = t^2$ obtain fine and coarse estimates of the length between $t = 0$ and $t = 2\pi$ and then use `quad` to integrate numerically.

In detail, plot the curve in red with a spacing of .5, `hold` the graph, and then repeat with a spacing of .1 in blue. For the finer spacing, identify the points by plotting again using $\circ$ as a linestyle. Look back in Lab 1 for the table. Calculate the polygonal length for each of these two spacings by using `sum` and then calculate the length by using `quad` applied to the integral formula.

Lab 6: Graphing Surfaces and Their Tangent Planes

Matlab Objects and Operations: symbolic expressions in several variables, symbolic partial derivatives

Matlab Commands: `diff(  ,x)`, `diff(  ,y)`, `surfc`, `input`

Work in class: When you define an anonymous function in several variables, you declare all of the variables in parentheses after the @ symbol.

```
clear all; close all

syms x y

fsym  = (x^2-x)*y
fxsym = diff(fsym, x)
fysym = diff(fsym, y)

f  = @(x,y) subs(fsym);
fx = @(x,y) subs(fxsym);
fy = @(x,y) subs(fysym);

x0 = input('x-coordinate of the point of tangency    ');
y0 = input('y-coordinate of the point of tangency    ');
z0 = f(x0, y0)

[a b] = meshgrid(x0 + [-2:.2:2], y0 + [-2:.2:2]);
c = f(a, b);
surfc(a, b, c)
shading interp; hold on;
pause

L = z0 + fx(x0, y0)*(a-x0) + fy(x0, y0)*(b-y0);
surf(a, b, L)
plot3(x0, y0, z0, 'rd')
```

Here we have illustrated an alternative way to proceed when we want to do symbolic differentiation. If we had used the approach demonstrated in Lab 3 we would have begun

```
f = @(x,y) (x.^2 - x).*y
syms x y
fx = diff(f(x,y), x)
fy = diff(f(x,y), y)
```

That is, there we began by defining the anonymous function f and then differentiated the symbolic expression `f(x,y)`, which is the same as what we are here calling `fsym`. This time we are starting with the symbolic expression `fsym` and then define the anonymous function f by using the `subs` command just as we do for `fxsym`. The two procedures are completely equivalent. You may use either one (or both). Notice that if we begin with the anonymous function we have to be sure and vectorize. If we begin with the symbolic expression we don't have to because `subs` does it automatically.

When we have symbolic expressions of several variables you have to declare which variable is being differentiated. That is why we didn't just write `diff(fxsym)` or `diff(f(x,y))`.

We also illustrated the `surfc` command which exhibits the surface graph together with the contour curves on the xy plane below.

Matlab Oddity: Matlab sometimes has trouble switching variables. Try this:

```
f = @(x,y) x.^2.* cos(y)
syms x y
gsym = x^2 * y
g = @(x,y) subs(gsym);
a = f(x,y)
b = f(y,x)
c = g(x,y)
d = g(y,x)
```

The problem here is that when using `subs` Matlab substitutes one variable at a time. Think about why this gives you the results you got.

Lab Homework: This is to be submitted via Digital Dropbox or printed out and handed in. Follow the instructions given in class.

1. Exhibit the cone $z^2 = (x^2 + y^2)/4$ and the cylinder $y^2 + z^2 = 1$ together in a single graph over a rectangle in the xy plane. Remember the `axis equal` command. Don't forget the top and the bottom of the cone and the cylinder.

2. Plot the graph of $z = \sin(x^2 + y^2)$ together with its tangent plane at the point $(1, 1, \sin(2))$.

Lab 7: Gradients and Max/Min Problems

Matlab Commands: `max, min`

```
ones, zeros, size
surfc
quiver, quiver3
 for . . . end, pause(n)
```

In this section we look at the gradient vectors for surfaces of the form $z = f(x, y)$. In the xy plane the two-dimensional vector field $grad(f)$ is perpendicular to the contour lines. In three space we define $F(x, y, z) = z - f(x, y)$ and look at the portion of the vector field $grad(F)$ which is attached to the contour surface $F = 0$, i.e. the original surface. We also find numerical estimates for the absolute maximum and minimum of the surface for the portion which we have graphed. This generalizes to surfaces in space the method used in Lab 2 for curves in the plane.

Work in class: After obtaining the partial derivatives, we solve, by hand, the system of equations $Fx = 0, Fy = 0$ to get the critical points.

```
clear all; close all

syms x y
fsym  = f = (10*x.^2 - 5*y.^2 + 3*x.*y ).* exp(.1*(-x.^2 - y.^2))
fxsym = diff(fsym, x)
fysym = diff(fsym, y)

f  = @(x,y) subs(fsym);
fx = @(x,y) subs(fxsym);
fy = @(x,y) subs(fysym);
pause

[u, v] = meshgrid(-5 : .2 : 7);
w = f(u, v);
surfc(u, v, w)
pause
```

```
contour(u, v, w, 25)
pause; hold on

clabel(contour(u,v,w,10))
pause; hold off

contour(u,v,w,25); hold on

[a, b] = meshgrid(-5 : .5 : 7);
quiver(a, b, fx(a, b), fy(a, b), 2)
pause

figure; contour3(u, v, w, 20, 'k'); hold on
pause

clabel(contour3(u, v, w, 5, 'k'))
pause

figure; surf(u, v, w); shading interp
pause

quiver3(a, b, f(a, b), -fx(a, b), -fy(a, b), ones(size(a)), 1/2)
pause

z = w(1 : prod(size(u)));
[M, iM] = max(z);
disp('maximum point is ');
disp([u(iM), v(iM), M]);
[m, im] = min(z);
disp('minimum point is ');
disp([u(im), v(im), m]);
pause

figure; surf(u, v, w); hold on;
plot3(u(iM), v(iM), M, 'rd')
plot3(u(im), v(im), m, 'gd')
```

The new command here is `quiver` which draws arrows in the plane, and its three dimensional version `quiver3` which draws arrows in space. For each you specify a point at which the arrow is based and a vector which is based at the point. You can add values to specify a scale factor and the color. Run the following as an m-file to see how they work. Notice that we specify the axis values so that the arrow does not take up the whole picture. Also, in the little program we don't want the axis to keep changing as we change the length of the vector.

```
xyz = [-2, 7]
quiver(2,1,1,3); axis([xyz xyz]); grid on;
pause
hold on
a = [2, 3]
b = [1, 4]
z = zeros(1, 2)
pause
quiver(z, z, a, b, 'g'); hold off
pause
a = [0, 0, 1; 1, 2, 3]
b = [2, 1, 5; 1, 3, 0]
c = [1, 2, 4; 7, 3, 3]
quiver3(a, b, c, -a, -b, -c); axis([xyz xyz xyz]); grid on;
pause
figure; axis([xyz xyz xyz]); grid on; hold on
for t = [1:25]
    quiver3(a, b, c, -a, -b, -c, .1 * t)
    pause(.5)
end
```

Often we want to specify a vector of zeros or ones. The command `zeros(m, n)` gives an $m \times n$ matrix of zeros. If we start with a matrix a then `zeros(size(a))` gives a matrix of zeros with the same size as a. The command `ones` works the same way.

In the little program we stretched the arrows through 25 steps. The command `pause(n)` pauses for n seconds and then continues. Compare this with `pause` which waits for you to hit any key as a prompt to continue. In the `for ... end` loop there was a half second pause after each step.

The use of the `max` and `min` commands work as they did in Lab 2, but for one complication (of course). Remember that `[M i] = max(z)` gives the maximum value and the location on the z vector of the maximum. So z needs to be a vector. Because of the meshgrid construction w is a matrix. If w is an $m \times n$ matrix, that is, `size(w) = [m n]` then we convert w to a row vector of length mn which equals `prod(size(w))` by using the command `z = w([1 : prod(size(w))])`. Then from `[M i] = max(z)`, `x(i)` and `y(i)` give the x and y coordinates of the maximum point. At this point you should go back to Lab 1 to look up the two different ways of specifying the location of an entry in a matrix.

Lab Homework: This is to be submitted via Digital Dropbox or printed out and handed in. Follow the instructions given in class.

Graph and make a contour plot of $f(x, y) = y^4 + x^3 - 10y^2 - 3x + 20$ over the square $[-3, 3] \times [-3, 3]$. Solve for the critical points by hand to make sure that they are all included. Then plot a gradient field on the surface. Finally, identify the maximum and minimum using `max` and `min`.

Lab 8: Graphing Parametrized Surfaces

Matlab Commands: `ones, zeros`

Work in class: We graph a parametrized surface and illustrate the normal vector obtained as the cross product of the partial velocity vectors.

```
clear all; close all;

syms u v

gsym  = [u^3+v, v^2-u, u*v]
gusym = diff(gsym, u)
gvsym = diff(gsym, v)
Nsym  = cross(gusym, gvsym)

g1  = @(u, v) subs(gsym(1));
g2  = @(u, v) subs(gsym(2));
g3  = @(u, v) subs(gsym(3));
gu1 = @(u, v) subs(gusym(1));
gu2 = @(u, v) subs(gusym(2));
gu3 = @(u, v) subs(gusym(3));
gv1 = @(u, v) subs(gvsym(1));
gv2 = @(u, v) subs(gvsym(2));
gv3 = @(u, v) subs(gvsym(3));
N1  = @(u, v) subs(Nsym(1));
N2  = @(u, v) subs(Nsym(2));
N3  = @(u, v) subs(Nsym(3));

[U, V] = meshgrid(-3:.2:3);
X = g1(U, V);
Y = g2(U, V);
Z = g3(U, V);

mesh(X, Y, Z)
axis equal
hold on
pause
```

```
quiver3(X, Y, Z, gu1(U, V), gu2(U, V), gu3(U, V), 2, 'r')
pause
quiver3(X, Y, Z, gv1(U, V), gv2(U, V), gv3(U, V), 3, 'g')
pause
quiver3(X, Y, Z, N1(U, V), N2(U, V), N3(U, V), 2, 'k')
pause
surf(X, Y, Z); shading interp
```

Matlab Oddity: Notice that the partial derivatives $gu2$ and $gv1$ are constants. This can lead to problems although it didn't here.

```
f = @(x) x.^2
g = @(x) 3
h = @(x) 3 + x - x
a = [-2 : 3]
c = f(a)
d1 = g(a)
d2 = h(a)
d3 = 3 * ones(size(a))
```

You see that $f(a)$ is a vector, obtained by substituting a into the vectorized anonymous function f. On the other hand, $d2$ and $d3$ give what you expect from $g(a)$ a vector of the same length as a with all entries equal to 3. However, $g(a)$ is just the scalar 3.

For most purposes this doesn't matter because arithmetic with scalars is automatically vectorized.

```
c = f(a).* sqrt(f(a))
d = g(a) + g(a).* f(a)
```

In the m-file we started with the command `quiver3` worked ok despite the scalar entries $gu2(U, V)$ and $gv1(U, V)$. However, `quiver` is not so forgiving

```
syms t
xsym = t
```

```
ysym = cos(t).*(2*sin(t) + 1)
xtsym = diff(xsym)
ytsym = diff(ysym)
x = @(t) subs(xsym);
y = @(t) subs(ysym);
xt = @(t) subs(xtsym);
yt = @(t) subs(ytsym);
a = [0:3*pi/100:3*pi];
b = [0: 3*pi/10:3*pi];
plot(x(a),y(a))
grid on; hold on;
pause
quiver(x(b),y(b),xt(b),yt(b))
pause
figure; plot(x(a),y(a))
grid on; hold on;
pause
quiver(x(b),y(b),ones(size(b)),yt(b))
```

Work in class: We use spherical coordinates as a parametrization for the top half of the ellipsoid given by $x^2 + 4y^2 + 4z^2 = 16$. Then on the same graph we plot the parabolic cylinder given by $y = x^2$. By hand you can parametrize the curve of intersection and compute the points where the curve begins and ends. These endpoints occur when $z = 0$ and so are obtained by solving the quadratic equation: $y + 4y^2 = 16$. Finally we graph the curve of intersection, and estimate its arclength

```
clear all; close all

[theta, phi]= meshgrid(0:pi/10:2*pi, 0:pi/10:pi/2);
x = 4*sin(phi).*cos(theta);
y = 2*sin(phi).*sin(theta);
z = 2*cos(phi);
surf (x, y, z)
grid on
hold on
pause
```

```
axis equal
shading interp
pause

[x, z] = meshgrid(-2:.2:2, 0:.1:2.5);
y = x.^2;
surf(x, y, z)
pause


syms t
t0=-1.37
t1=1.37

xsym  = t
ysym  = t^2
zsym  = 2*sqrt(1-(xsym/4)^2-(ysym/2)^2)
speed = @(t) subs(sqrt(diff(xsym)^2+diff(ysym)^2+diff(zsym)^2))

L1 = quad(speed, t0, t1);
disp('arc length integral equals ');
disp(L1);

T = linspace(t0, t1);
x = subs(xsym, T);
y = subs(ysym, T);
z = subs(zsym, T);
plot3(x, y, z, 'rd')

L2 = sum(sqrt(diff(x).^2 + diff(y).^2 + diff(z).^2));
disp('polygonal length estimate equals ');
disp(L2);
```

We have seen in previous work that a surface which is described by the equation $x = f(y, z)$ is graphed using $[y, z] =$ `meshgrid`.

The parametrization of the ellipsoid uses spherical coordinates.

Lab Homework: This is to be submitted via Digital Dropbox or printed out and handed in. Follow the instructions given in class.

1. Plot the Möbius Strip as the parametric surface given by:

$$\begin{aligned} x &= 2\cos(\theta) \;+\; r\cos(\frac{\theta}{2}), \\ y &= 2\sin(\theta) \;+\; r\cos(\frac{\theta}{2}), \\ z &= r\sin(\frac{\theta}{2}) \end{aligned}$$

   with $-\frac{1}{2} \leq r \leq \frac{1}{2}$ and $0 \leq \theta \leq 2\pi$. So the parameters are `r` and $\theta =$ `theta`.

2. Use spherical coordinates to plot that portion of the sphere of radius 3, centered at the origin with latitude angles between $\pm 60°$. Remember that the angle ϕ opens down from the north pole so that the equator is at $\phi = \pi/2$. We want the part of the sphere with latitude angle varying from $60°$ above to $60°$ below the equator.

3. Exhibit the cone $z^2 = (x^2 + y^2)/4$ and the cylinder $x^2 + y^2 = 1$. together in a single graph. Use polar coordinates `r` and $\theta =$ `theta` to parametrize the x and y variables,

Lab 9: Multiple Integrals

Matlab Commands: `dblquad`, `quadv`,
`warning off`, `warning on`

Work in class: We compute the double integral

$$\int_0^2 \int_0^1 x\sqrt{x^2+y}\; dx\; dy$$

first using the symbolic package and then by using the numerical integration command `dblquad`.

```
clear all; close all

syms x y ;
f = @(x,y) x.*sqrt(x.^2 + y)
pause

I1 = int(f(x,y), x)
disp('int_0^1  x sqrt(x^2 + y)  dx equals   ');
D1 = subs(I1,x,1) - subs(I1,x,0)
pause

I2 = int(I1, y)
disp('int_0^2  \int_0^1  x sqrt(x^2 + y)  dx dy  equals ');
D2 = subs(I2,y,2) - subs(I2,y,1)
pause


disp('The numerical estimate using dblquad is ');
disp(dblquad(f, 0, 1, 0, 2));
```

We compute the double integral

$$\int_0^1 \int_0^{x^2} \sqrt{x^3+1}\; dy\; dx$$

first using the symbolic package just as before. When we use `dblquad`, we need a trick because `dblquad` only works over rectangular regions.

```
clear all; close all

syms x y ;
f = @(x,y) sqrt(x.^3+1);
pause

I1 = int(f, y)
disp('int_0^{x^2} sqrt(x^3+1) dy   equals ');
D1 = subs(I1,y,x^2)
pause

I2 = int(D1, x);
disp('int_0^1 int_0^{x^2} sqrt(x^3+1) dy dx   equals ');
D2 = subs(I2,x,1) - subs(I2,x,0)
pause

F = @(x, y) f(x,y).*(v<=u.^2)
pause

[u, v] = meshgrid(linspace(0, 1, 50));
surf(u, v, F(u, v))
pause

disp('The numerical estimate using dblquad is ');
disp(dblquad(F, 0, 1, 0, 1));
```

The requirement that the integration be over a rectangle limits the usefulness of the command `dblquad`. At times the trick we used above leads to inaccurate estimates. What we would like to do is imitate with the numerical integration command `quad` what we did with symbolic command `int`. That is, we would like to compute the double integral as an interated integral. We have to be careful because `quad` is strictly numerical. Compare

```
a = quad(@(x) x.^2,0,1)
quad(@(x) x^2,0,1)
quad(@(x) x.^2 + y,0,1)
syms y; quad(@(x) x.^2 + y,0,1)
```

```
quad(@(x,y) x.^2 + y,0,1)
```

The first one is the standard use of `quad` with the handle for a function of a single variable. The rest give error messages. First we are reminded that the function must be vectorized. The rest indicate that you cannot mix symbolic or undefined variables in with the numerical routine. What does work is

```
I1 = @(y)quad(@(x) x.^2 + y,0,1)
```

What `I1(3)` does is substitute 3 for y and then applies `quad` to `@(x) x.^2 + 3`, to get 3.3333. The remaining problem is that this function is not vectorized. So `I1([0:3])` gives an error message. This means you can't integrate it. You get an error message with `quad(I1,0,1)` as well.

Happily the command `quadv` does what `quad` does and vectorizes the result. Thus, we can replace `dblquad` in our original problem.

```
I1 = @(y)quadv(@(x) x.*sqrt(x.^2 + y),0,1)
I2 = quadv(I1,0,2)
pause
I1 = @(x)quadv(@(y) sqrt(x.^3 + 1),0,x.^2)
I2 = quadv(I1,0,1)
```

In the second example the variable upper limit of integration results in some warning messages. Nonetheless, the program completes the run and gets the correct answer.

Matlab Oddity: In Lab 6 we mentioned that one could either define `f = @(x,y) x.^2 + y` and then, after reserving `syms x y`, use it to get the symbolic expression `fsym = f(x,y)`, or we could go the other way, defining `fsym = x^2 + y` and then using `subs` to define `f = @(x,y) subs(f)`. We remarked that the latter expression was automatically vectorized by `subs`. True enough, except sometimes not. Try

```
syms x y
fsym = x.^2 + y
f = @(x,y) subs(fsym)
dblquad(f,0,1,0,3)
```

Despite our vectorizing the symbolic expression, `dblquad` is complaining that the function f is not vectorized. Thus, it is probably better to begin with the anonymous function except when the symbolic expression turns up after applying `diff`, which is where we first used the `@(  ) subs(...)` construction.

Assignment: In the Lab read the Matlab responses to `help dblquad` and `help int`.

Lab Homework: This is to be submitted via Digital Dropbox or printed out and handed in. Follow the instructions given in class.

1. Compute the double integral

$$\int_0^{\pi/3}\int_0^{\pi/6} x\sin(x+y)\ dxdy$$

using all three methods: symbolic, via `dblquad` and as an iterated numerical integral using `quadv`.

2. Consider the portion of the surface given by $z = f(x,y) = y - x^2$ which lies above the region in the xy plane where $0 \le x \le 1, x^2 \le y \le 1$. Parametrize the surface by using `[s,t] = meshgrid(0:.05:1)` and then $x = s; y = s^2 + t(1-s^2); z = f(x,y)$ and plot the graph.

   Use a double integral to compute the volume of the solid bounded by the surface and the planes $z = 0, x = 0$ and $y = 1$. First use `int` for symbolic integration and then use `quadv`.

3. One can use the same procedure for triple (and higher) integrals. For example,

$$\int_0^1\int_0^z\int_0^y ze^{-y^2}\ dx\ dy\ dz$$

is computed using `quadv`:

```
warning off;
quadv(@(z) quadv(@(y) quadv(@(x) ...
   z.*exp(-y.^2), 0, y), 0, z), 0,1)
warning on;
```

The nesting of the `quadv(@(..)..)` commands works just like the integral notation. The `warning off` command is used (first!) because without it you get a long list of grumpy warnings before the (correct) computation is completed. It is best to turn the warnings back on at the end. Notice also that if a line of instructions is very long you can break it up by typing three periods `...` and then ENTER to go to a new line. Matlab interprets the entire instruction as though it is written on a single line.

Use this method to compute the triple integral of $x + 2y$ over the solid S which is bounded by the parabolic cylinder $y = x^2$ and the planes $z = x$, $x = y$ and $z = 0$. First, plot the solid (remember `hold on`) and then do the triple integral.

Lab 10: Iteration and Taylor Series Approximation

Matlab Commands: `subs`, `polyval`, `for ... end`
`cumprod`, `cumsum`

Work in class: We illustrate the motion of terms of a sequence by iterating the quadratic function given by $f(x) = mx(1-x)$ for different values of the slope m at $x = 0$.

```
clear all; close all

syms x
m = input('Slope at x = 0? ');
f = m*x*(1 - x)
a = input('Initial value ');
for k = 1:20
  a = [a, subs(f,x,a(k))];
end

figure(1);hold on;axis([0,20,min(a)-1,max(a)+1])
for k= 1:20
   plot([k-1,k],[a(k),a(k+1)]);
  pause(.5);
end
plot([0:20],a,'r.')
pause

u= linspace(0,1);
figure(2); plot(u, subs(f,x,u),'k'); hold on;
pause
plot(u,u,'g')
pause
for k= 1:20
   plot([a(k),a(k),a(k+1)],[a(k),a(k+1),a(k+1)]);
  pause(.5);
end
plot(a, subs(f,x,a),'r.')
```

We can do this for a general function which you can input as a symbolic expression. Try using the functions $\cos(x)$ and $\sqrt{x+3}$ with initial value 0 in each case.

```
clear all; close all
syms x
f = input('Type function as symbolic string in the variable  x ');
a = input('Initial value ');
for    k=1 : 20
   a = [a, subs(f,x,a(k))];
end

figure(1);hold on;axis([0,20,min(a)-1,max(a)+1])
for k= 1:20
    plot([k-1,k],[a(k),a(k+1)]);
   pause(.5);
end
set(gca,'Ytick',sort([a(1),a(20)]));
plot([0:20],a,'r.')
b=unique(a);
set(gca,'Ytick',sort(a([1,4,8,21])));
plot([0:20],a,'r.')
[c;b]'
pause

u=linspace(0,1);
figure(2);plot(u,subs(f,x,u),'k'); hold on;
pause
plot(u,u,'g')
pause
for k= 1:20
    plot([a(k),a(k),a(k+1)],[a(k),a(k+1),a(k+1)]);
    pause(.5);
end
plot(a,subs(f,x,a),'r.')
```

Work in class: We compare the graph of $y = e^x$ with the graph of each of the first five partial sums of the MacLaurin series for e^x.

Each of these partial sums is a polynomial. A polynomial in x can be described by listing its coefficients in decreasing order. Zero coefficients act as place holders just like in ordinary numbers and so cannot be omitted. Observe that 5098102 = `polyval([5 0 9 8 1 0 2],10)`.

```
clear all; close all
x = linspace(-1, 3);
coeff = [ 1 ];
plot(x, exp(x), x, polyval(coeff, x))
title('compare with constant term')
pause

for k = 1:10;
  c= coeff(1)/k;
  coeff= [c, coeff];
  plot(x, exp(x), x, polyval(coeff, x))
  title(['terms through the power' num2str(k)])
  pause
end
```

We can do this one as well for a general function which you can input as a symbolic expression.

```
clear all; close all
syms x;
f = input('Type in the function as a symbolic expression in x ')
g = f;
coeff = [subs(g, x, 0)];
a = linspace(-1, 3);
b = subs(f, x, a);
plot(a, b, a, polyval(coeff, a)); axis([-1,3,min(b)-1,max(b)+1]);
title('compare with constant term')
factor = 1;
pause

for k = 1 : 25;
  g= diff(g)
```

```
    factor = factor/k
    c= subs(g, x, 0)*factor;
    coeff= [c, coeff];
    plot(a, b, a, polyval(coeff, a)); axis([-1,3,min(b)-1,max(b)+1]);
    title(['terms through the power' num2str(k)])
    pause
  end
```

A good example to use is $f = cos(5x + x^2)$.

In computing factorials, we can obtain $k!$ for a positive integer k by using the command `prod([1:k])`. Often it is convenient to use $k! = ((k-1)!) \times k$. This is what we did in the `for ... end` loop. The expression $factor$ had a $(k-1)!$ in the denominator. We divided by k to get $k!$. For factorials which are not too large the command `cumprod` is helpful. It returns the cumulative products. That is, in position i `cumprod(a)` gives the product of the first k entries of a. Thus, `cumprod([1:k])` lists the factorials from 1 up to $k!$. Similarly, `cumsum` which returns the cumulative sums. That is, `cumsum(a)` lists the partial sums of the finite sequence a.

For a vector a of length n, the command `polyval(a,b)` computes the value at b of the polynomial function whose coefficients, in decreasing order, are given by a. Thus, if $a = [3, 2, 0, 1]$ then `polyval(a,b)` computes $p(b)$ where p is the function given by $p(x) = 3x^3 + 2x^2 + 1$. Notice that the zero coefficients have to be included as place holders.

The expression is vectorized. That is, if b is a vector then `polyval(a,b)` is the vector with length that of b obtained by substituting each entry into the polynomial.

We can obtain the polynomial as an anonymous function by using `p = @(x) polyval(a,x).`

Assignment: In the Lab read the Matlab response to `help polyval`.

Appendix A: Symbolic Expressions in MatLab

To create symbolic variables in Matlab use the `syms` command:

```
syms x y real
```

Note that commas do not separate the variables you want declared symbolic. The *real* designation is optional. If you are sure that the variables will not take on complex values, then the real option may produce simpler formulas when Matlab manipulates expressions that involve radicals or logarithms.

You can now define a symbolic expression using the usual operators. Note that vectorization is not applicable in symbolic expressions (at least not directly) and does not have to be used in the definition. For example,

```
w=x^2-3*x
```

defines a symbolic expression w. The dependent variable w does not have to be declared symbolic prior to using it.

Evaluating Symbolic Expressions

To evaluate a symbolic expression use the subs command. Consider the following examples:

```
subs(w,2)

ans =
    -2
```

The command `subs(w,2)` substitutes for the independent variable in w the value 2.

```
subs(w,1/3)!

ans =
   -0.8889
```

Here the substitution in w has been done numerically. Since the input is an exact fraction one might also expect that it should be possible to obtain an exact fractional value. Indeed this can be done.

```
subs(w,sym(1/3))

ans =
    -8/9
```

The command `sym` tells Matlab to enter the symbolic fraction 1/3 rather than the numerical fraction. We can equally well convert the symbolic answer to numerical form using the double command.

```
double(ans)

ans =
   -0.8889
```

We mentioned above that symbolic expressions need not be entered in vectorized form. Nonetheless, the `subs` command can evaluate these expressions for an array of inputs.

```
subs(w, [1 -1 1.5])

ans =
   -2.0000    4.0000   -2.2500
```

If we have a symbolic expression in two or more variables the substitution must be carried out more explicitly so that Matlab knows which values are to be substituted for which variables. For example,

```
z=x^2+y^2

z =
x^2+y^2
X=[1 2 3]; Y=[1, 3, 5];  % Lists of values substituted for x and y.

subs(z, {x,y}, {X,Y})

ans =
     2    13    34
```

Observe the set brackets {} that are used to specify the variables in z and the corresponding replacements. These are used to specify LISTS. In this example, Matlab replaces the variables x and y by the pairs obtained from the two arrays X and Y. Thus, it first evaluates z for $x = 1, y = 1$, then for $x = 2, y = 3$, etc. Observe that we have used the fact that Matlab distinguishes between the variables x and X. We use the capital form to hold the numeric values we wish substituted for x. When you use a symbolic variable such as x, you should never assign a numerical value to x through a statement such as $x = 2$. Doing so will destroy the symbolic quality of the variable.

Some Calculus

Differentiation of symbolic expressions is straightforward. Use the `diff` command. For the expression w defined earlier we get

```
diff(w)

ans =
    2*x-3
```

This is the first derivative. Higher order derivatives are readily computed by including the desired order in the diff command. For example, to find a second derivative of w enter

```
diff(w,2)

ans =
    2
```

For a multivariate expression we can use `diff` to find partial derivatives. Referring to z we have

```
diff(z,x)

ans =
    2*x
```

and

```
diff(z,y)

ans =
    2*y
```

The `diff` command is vectorized, that is, it will differentiate all symbolic components in a vector input. This is particularly useful in dealing with curves, for example. Suppose we wish to study the curve defined by the parametric equations $x = t^2, y = t^3$. Let us first add t to our list of symbolic variables, using `syms t real`

Now we define the symbolic parametric equations defining the curve:

```
r=[t^2 t^3]

r =
    [ t^2, t^3]
```

The components of the tangent vector to the curve are given by differentiation of the component functions.

```
dr=diff(r)

dr =
    [ 2*t, 3*t^2]
```

As you know, integration of symbolic expressions may be considerably more difficult than differentiation. The `int` command can be used to find an antiderviative or to obtain an exact expression for a definite integral. For example,

```
int(w)

ans =
    1/3*x^3-3/2*x^2
```

produces $\int(x^2 - 3x)dx$ (without the constant of integration). You can get Matlab to write the expression in a somewhat easier to read form using the `pretty` command.

```
pretty(ans)


                                          3          2
                                   1/3 x~  - 3/2 x~
```

The ~ indicate that the symbolic variable has certain assumptions in its definition. In this case we assumed the variables represented real quantities. If no assumptions are made, the letters appear without the ~ symbol.

Definite integrals can be evaluated by including the limits of integration.

```
int(w, 0, 2)

ans =
-10/3
```

computes $\int_0^2 (x^2 - 3x)dx$. In some cases exact symbolic answers cannot be obtained for the integration problem you wish to consider. In such situations you can evaluate a definite integral numerically using the `quad` command. ("quad" is an abbreviation for quadrature, which is an old-fashioned name for integration). As a numerical routine, the `quad` command requires its input to be vectorized, that is the expression it works with has to be able to accept array inputs. In order to use `quad` on the symbolic expression w, the latter must first be converted to vectorized form using the `vectorize` command (which also converts the symbolic expression to a string).

```
W=vectorize(w)

W =
    x.^2-3.*x
```

Now we can apply quad to the vectorized expression W.

```
quad(W,0,2)

ans =
   -3.3333
```

Of course we obtain the numerical approximation to the exact symbolic answer -10/3 found above. Alternatively, we can convert w to an anonymous function `W = @(x) subs(w)` and then use `quad(W,0,2)` to get the same answer.

An Exercise

1. For the curve C given by the parametric equations $\mathbf{r} = [t^2, 2t^3 - t]$ find symbolic expressions for the tangent vector $\mathbf{r}' = \frac{dr}{dt}$.

2. Using the symbolic expression found in 1. find a unit tangent vector to C at the point corresponding to $t = 1.5$.

3. Set up a symbolic definite integral (using `int`) for the arclength of C between [0, 0] and [1, 1]. Recall that the arclength is given by the integral of the speed, that is, an integral of the form $\int_a^b |\mathbf{r}'(t)| \, dt$. Find the numerical (decimal) value using the `double` command.

4. Using `quad` evaluate the arclength specified in 3. Compare with the answer you found in 3. Which answer is correct? What lesson is to be learned from this?

5. Plot the curve C for $-2 \leq t \leq 2$.

Appendix B: Publishing m-files to HTML

When we ran the m-files described in the text above we produced nice graphs, which you could print using the PRINT button in the graphics window. However, the m-file code is sitting in the Editor window. You could print that as well, as a separate document. We would prefer to produce a report showing the code and the output, including the graphs. Matlab, however, provides a way to actually print your m-file code and its output in a single document. It's called publishing. It allows you to publish your result as a webpage (HTML), a PowerPoint presentation, a Word document, or in several other formats. We will describe how to take your completed m-file and publish it to a webpage, which Matlab calls "Publish to HTML."

In this case, where the m-file produces only a single output at the end, all you do is go to the File menu in the editor and select "Publish to HTML." There is also a button at the left side of the icon bar that does the same thing. In very short order, a window will appear with your code and output (the graph). You can then print that window for a complete record of your work and what it produces. Very neat

For m-files that produce more than one output you divide the m-file into regions called "cells." Any output produced by a command in a cell appears right after the cell, rather than at the end of the document. This makes your document into something like a report, which is the basic idea of this feature. You can add additional "markup" to cells to give them a more distinctive appearance, as well as provide a hyperlinked structure for the webpage that is produced.

If you would like to learn more about the features of publishing, there is an excellent video demo that you can access from the Matlab Help menu. Start from the HELP menu on the toolbar above the main Matlab screen. Click on MATLAB HELP. On the window which comes up there are several tabs on the left, below the phrase HELP NAVIGATOR. Click on the DEMOS tab and then on the + next to MATLAB. Open the folder named DESKTOP TOOLS AND DEVELOPMENT. Look for the video called PUBLISHING M CODE FROM THE EDITOR. The helpful Irishman will show you the many features of the `publish` option. However, we will pick out the special aspects which you are most likely to use.

To see the result of publishing, look at the first of the MATHEMATICS Demos. This is an m-file called BASIC MATRIX OPERATIONS. When you click on it you see the published output of the sort we will get. At the top you see OPEN INTRO.M IN THE EDITOR. When you click on this you see the m-file itself. The important thing to observe is the comment labels,%, and the double comments, %%. The sample we will use is a version of the classwork done in Lab 03.

When you want to publish your m-file, first open it and look at the toolbar in the Editor window. In the menu under CELL click on ENABLE CELL MODE. A second toolbar will open up. If you see DISABLE CELL MODE then the cell mode is already on. Leave it on.

In your m-file, at the top, after a single comment % and a space, type the name of the file. The name of the m-file should include your name. Each separate cell begins with a double comment %% and a space followed by a typed label for the cell. The labels for the separate cells will be listed right at the beginning as "contents". Within each cell only the last figure will be printed into your report, so make sure that each figure that you want to appear occurs in a separate cell.

Make sure that you save all the changes you have made and then click on PUBLISH. That is, under FILE click on PUBLISH TO HTML on just use the icon all the way to the left on the cell toolbar.

```
   % HarryPotterLab03
clear all; close all
%% function and derivative
f = @(x)x.*cos(x.^2)
syms x
fxsym = diff(f(x))
fx = @(x)subs(fxsym)
pause
u = linspace(-.5,3);
v = f(u);
w = fx(u);
whos
pause
%% function plot
plot(u,v); grid on; hold on
pause
%% derivative plot
plot(u,w,'g')
%% tangent line
```

```
p = 1.4
%p = input('x value for tangent point?    ')
q = f(p) + fx(p)*(u - p);
figure(gcf); plot(u,q,'k'); plot(p,f(p),'rd')
pause
%% integral
Fsym = int(f(x))
F = @(x) subs(Fsym)
disp(['The integral from 0 to 2 of f is F(2) - F(0) =  ',
        num2str(F(2)-F(0))])
disp(['The numerical approximation given by quad is  ',
        num2str(quad(f,0,2))])
```

Notice that I disabled the line `p = input('x value ...` with a % and just put in a value (in this case `p = 1.4`) before I published. The PUBLISH scheme can't deal with interactive commands like `input` and error messages result. When using the m-file interactively, rather than publishing it, I uncomment the `input` line and comment the `p = 1.4` instead.

The command `pause` is the one interactive command that you can use and you should do so. When you publish, Matlab will run the m-file and pause, as usual, at each `pause` command. Make sure your cursor is somewhere in the command window and not in your m-file (so you don't type new and unwanted stuff into the m-file) and then hit anykey, as usual. Of course, you can avoid having to do this by commenting all the `pause` commands, but get used to not doing that. In graphing, you can manipulate the figure while it is paused. For example, you can rotate a surface in 3D by using the Figure window. Remember that the final version of the figure is what gets printed in your report. So you can change the figure and then respond to the pause. For this reason it is a good idea to make sure that there is a `pause` after each figure that you want to appear in the report. If you don't like the look, then just `Publish` again. The previous version will be overwritten.

Once you have the HTML file, you can print it. However, the HTML file itself is hidden away in a folder in the Matlab program. You can keep the HTML file, for your own use or to submit your homework electronically. For this purpose it is best to have your own jumpdrive (= flashdrive, memory stick, etc.). Plug it into the USB port in the computer. It will appear in My Computer as REMOVABLE DISK. Now under FILE in the Editor toolbar, click on PREFERENCES. In the window which comes up, under

Editor/Debugger click on Publishing. Under Location the default, subdirectory named html in source file directory has a dot in it. Instead click on Single directory and hit the box to the right which lets you Browse. Find Removable Disk and open it. Hit New Folder and when the folder appears, type the name of the m-file, including your name, to label the folder. Then hit Apply and then OK and the preferences window goes away. Then when you publish the files which result appear in that folder. Do this business with preferences each time you publish a new m-file (but not when you republish one you are working on) so that you can open a new folder with a new label for each new m-file. It is important to store each m-file in a separate folder because the result of publishing an m-file usually consists of several files. In addition to the HTML file itself, the figures are stored as separate PNG files. If you submit the results electronically, send the whole folder or else all of the files in it. Each of these files will be labeled with the name of the m-file. This is why IT IS IMPORTANT THAT THE NAME OF YOUR M-FILE SHOULD INCLUDE YOUR NAME.

As a simple test run for this use of Preferences, I recommend that you publish to the flashdrive an m-file titled simply `Owner ID` of the following type and publish it directly to the drive, no new folder required.

```
% OwnerID
% Harry Potter
% If this drive is found please email me at
hpotter@hogwarts.edu
```

The error message that you see results because the form of the instruction is not the correct one for an anonymous function, but, as you see, this little HTML file will appear on your drive and may help if the stick gets lost.

Finally, once the HTML file is finished and you see it before you, it sometimes happens that the Matlab program remains at Busy, seen on the bottom left. Usually, just hitting Enter will finish things up. Otherwise, try Ctl C.

MatLab Review Sheet

By the end of the course you should be able to use Matlab in the following ways:

1. Define arrays (vectors) using `:` and `linspace`, as well as define more general rectangular arrays (matrices). Suppress output using `;`. Use the `length` and `size` commands.
2. Describe and use vectorized arithmetic operations for arrays, for example, `.*` and `.^` and the transpose operation $'$ for matrices.
3. Use the built in functions for sqrt, sine, log, exponential etc. and also `cross` for cross product of 3D vectors.
4. Use the `diff`, `max`, `min`, `sum` and `prod` commands, understanding how they work on general rectangular arrays. For `max`, for example, use the `[Max,location] = max(...)` command.
5. Call for an entry for a matrix A in two ways, that is, A(row,col) or A(number). This requires understanding that the matrix is implicitly numbered as though it is the vector: $a = A(1$:`prod`(size!(A))), an explicit renaming which is sometimes useful.
6. Exploit the ARROW UP feature to reduce the amount of retyping needed.
7. Use `plot` and `plot3` to plot curves in 2D and 3D, respectively.
8. Set the axis range for each variable.
9. Define a STRING (also called a CHARACTER ARRAY) using single quotes.
10. Use the Edit menu on a figure to label graphs.
11. Use `hold on,` `hold off` and `clear figure`. Use the figure editor menus to adjust colors and line-styles.
12. Use the `help` command.
13. Define and use m-files (also called SCRIPTS). Use the `pause` and `disp` commands.

14. Define and use function m-files.
15. Define anonymous functions. Be able to substitute arrays (so make sure you *vectorize* your formulas).
16. Use `fzero` to find roots.
17. Use `syms` to declare symbolic variables, and define symbolic functions.
18. Use the `subs` command with symbolic expressions. In particular, use it to convert a symbolic expression to an anonymous function.
19. Use `diff` and `int` to perform symbolic calculus. Obtain the derivative and integral as anonymous functions.
20. Use `meshgrid` to define arrays pairs for graphing surfaces.
21. Use `surf` command to graph surfaces. Use `shading interp` to eliminate lines. Deal with variations such as cylindrical figures and such as portions of spheres and ellipsoids which are parameterized using spherical coordinates. Adjust the figure using `axis equal`.
22. Use contour and labeling commands, both `contour` and `contour3` and `clabel` with each one. Adjust the number of contour lines which are shown in each case. Also use `surfc`.
23. Use `quiver` and `quiver3`.
24. Use numerical integration commands `quad` for single variable integration and `quadv` for iterated integrals.
25. Use logical operators like `<=, <, >=, >, ==` .

Sample Lab Exam Questions

Part I These are typical short answer questions

1. How would you use the `linspace` command to generate the sequence `[0: .01 : 1]`?

2. How would you define a to be a vector whose entries are the numbers of the interval $[0, 2\pi]$ which subdivide the interval into 20 intervals of equal length? What is the `size` of a?

3. How would you define a to be a vector whose entries are the numbers of the interval $[0, 25]$ which subdivide the interval into 200 intervals of equal length, making sure that the resulting vector is *not* printed out on the screen?

4. How would you define a to be a vector whose entries are the 20 points of the interval $[0, 2\pi]$ which are equally spaced and which include the endpoints? What is the *size* of a?

5. How would you define A to be the matrix $\begin{pmatrix} 3 & 4 & 7 & 9 \\ 2 & 3 & 2 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$?

   What is the *size* of A ?

6. What command allows the user to superimpose several graphs in the same window?

7. What command will write the text "(2,1)" at the point with coordinates (2,1) in a 2D graphics window?

8. What command is used to reserve x and y as symbolic variables?

9. Write the command or commands which you use to define the vector y whose entries are the values of the function $f(x) = x^3\sqrt{\sin^2(x) + 1}$ at the first ten positive integers?

10. Write the commands which we use to plot the graph of the function $f(x) = x\cos(x^2)$ on the interval $[0, 5]$.

11. Determine to at least two place decimal accuracy at what value of x the function $f(x) = x\sin(\sqrt{x})$ has its maximum value on the interval $[0, 2\pi]$?

12. Write the command used to find the value of x close to $x = .5$ where the function $f(x) = 3\sin(x) - x - 1$ is zero. What is the computed value?

13. Write the commands which we use to plot the graph of the function $f(x, y) = x^2y$ over the square $[-1, 1] \times [-1, 1]$.

14. Write the commands which we use to plot the graph of the curve parameterized by t: $x = t, y = t^2, z = \sin(t)$ for the interval of time $0 \leq t \leq \pi$.

15. Write the commands which we use to compute the indefinite integral $\int x \sin(\sqrt{x})dx$ symbolically. What is the computed formula?

16. Find Matlab's numerical estimate for the value of $\int_0^1 \frac{\sin\ x}{\sqrt{1+x^2}}\ dx$.

17. Find Matlab's numerical estimate for the value of $\int_0^1 \int_0^x \sqrt{y + x^4}\, dy\, dx$.

Part II. Typical longer questions

1. Suppose f is defined as an anonymous function by the command

```
f = @(x,y)  2*x.*y - 3*x.^2 - 4*y.^2 +  14*x + 10*y + 12
```

   (a) State a sequence of commands that will produce 25 labeled contour curves of the function over the square $[0, 5] \times [0, 5]$.

   (b) Based on the contour plot determine whether the function has any critical points in the square $[0, 5] \times [0, 5]$ and for each one determine whether it is a relative maximum, relative minimum or a saddle point. Provide a justification for your answer.

2. A student wishes to create a plot for the space curve given by $x = \cos(t), y = \sin(t), z = t$ for t in the interval $[0, 2\pi]$. The curve should be drawn in black. The student produces the following code:

```
 t= [0 : 2pi/10 : 2pi];
 x = cos(t); y = sin(t);  z = t;
plot(x, y, z, 'b')
```

   (a) What, if anything, is wrong with either the commands or the output?

   (b) State the corrected code for accomplishing the desired result.

3. Give a command or commands that will draw the three sides of the triangle with vertices $A = (1, 1,)$, $B = (3, 2)$ and $C = (2, 2)$.

Index

anonymous functions	Labs 01, 02
anonymous function from a symbolic formula	Lab 03
anonymous functions of several variables	Lab 04
arithmetic of arrays	Lab 01
arrays, numerical	Lab 01
arrays, symbolic	Lab 03
`axis equal`	Lab 04
built-in functions	Lab 01
`char`	Lab 03
character arrays (strings)	Lab 02
`clabel`	Lab 04
colon operator (`:`)	Lab 01
`comet`	Lab 02
`contour`	Lab 04
`contour3`	Lab 04
`cross`	Lab 04
`cumprod`	Lab 10
`cumsum`	Lab 10
`diff`	Labs 01, 03
`diff(  ,x), diff(  ,y)`	Lab 06
differentiation	Lab 03
`disp`	Lab 02
`double`	Lab 03
`dblquad`	Lab 09
`echo on`, `echo off`	Lab 02
Edit menus, colors and line styles	Lab 01
`eps`	Lab 04
`eval`	Lab 03

`figure` Lab 01
`figure(gcf)` Lab 03
`for . . . end` Labs 07, 10
function handles Lab 02
`fzero` Lab 02

`grid on` Lab 02

`help` Lab 01
`hold on`, `hold off`, `hold` Lab 01

`input` Labs 03, 05, 06
`int` Lab 03
integration Lab 03

`linspace` Lab 01
`length` Lab 01
logic operators `<, <=, >, >=, ==, ~=` Lab 02

m-files (function m-files and scripts) Lab 02
`max, min` Labs 01, 02, 07
`mesh` Lab 04
`meshgrid` Lab 04

`num2str` Lab 02
number arrays Lab 01
numerical integration Lab 03

Oddities Labs 06, 08, 09
`ones` Labs 07, 08
operations on arrays Lab 01

partial derivatives Lab 06
`pause` Lab 02
`pause(n)` Lab 07
`pi` Lab 01
`plot` Lab 01
`plot3` Lab 04
`polar` Lab 01
`polyval` Lab 10
`prod` Lab 01

`quad` Lab 03
`quadv` Lab 09
`quiver, quiver3` Lab 07

rotating for surface plots Lab 04

semicolon (`;`) Lab 01
`shading interp` Lab 04
`size` Labs 01, 07
`sort` Lab 02
`subs` Labs 03, 09, 10
`sum` Lab 01
`surf` Lab 04
`surfc` Labs 06, 07
symbolic arrays Lab 03
symbolic differentiation Lab 03
symbolic expressions in several variables Lab 06
symbolic integration Lab 03
symbolic partial derivatives Lab 06
`syms` Lab 03
`sym` Lab 03

`text` Lab 02
`title` Lab 02
transpose(`'`) Lab 01

`unique`	Lab 02
vectors and matrices	Lab 01
`warning off`, `warning on`	Lab 09
`while ... end`	Lab 03
`whos`	Lab 03
`xlabel`	Lab 02
`ylabel`	Lab 02
`zeros`	Labs 07, 08
zooming in/out for surface plots	Lab 04