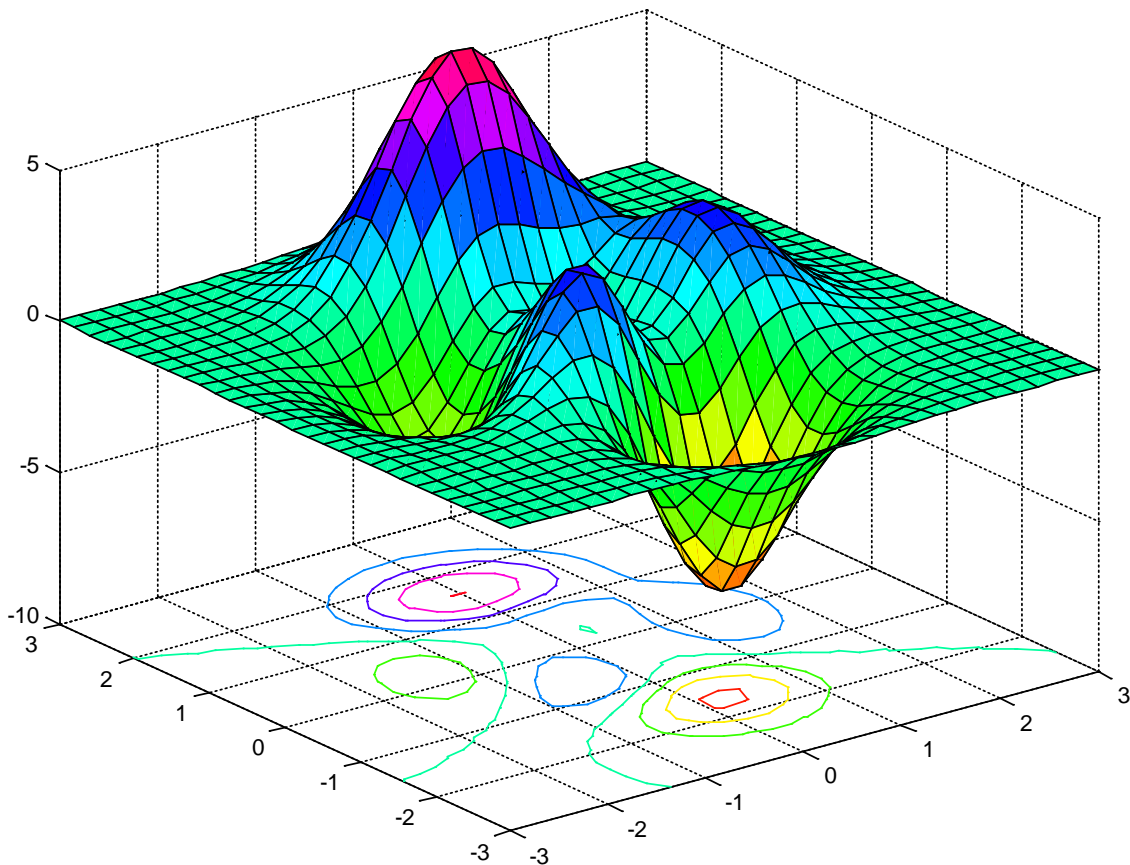




UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING
Department of Electrical and Computer Engineering

ECE221H1S ELECTRIC AND MAGNETIC FIELDS

Lab #1: Introduction to MATLAB Part 2 *Working With Arrays, Two- and Three- Dimensional Plotting, and Scripts and Functions*



LAB #1

The purpose of this lab is to allow you to gain further experience in using MATLAB, by building on what you learned in your first lab. In this lab, you will review some of these basic ideas and gain experience creating three-dimensional plots and writing scripts and functions.

PREPARATION - Individual

Required to do before you come into the lab

- 1) Read through these Lab #1 notes carefully.
- 2) Watch the video demonstration, *Writing a MATLAB Program*. This is available at <http://www.mathworks.com/videos/writing-a-matlab-program-69023.html>.
- 3) Give special consideration to the algorithms you will use for the two functions described in the last section of Section 2.5. **When you arrive to your lab session you will have to submit a handwritten copy of your pseudocode for these two functions to your TA. Make sure you have made a copy (or at least taken a picture) of your preparation before you hand it in, as you will not have this to use during the lab.**

Review – Lab #0: Introduction to MATLAB Part 1

Before you come to your second lab session, you should be comfortable with using MATLAB as a “super” graphing scientific calculator. You should be able to evaluate real or complex number expressions dealing with the standard set of built-in functions (`sin`, `cos`, `tan`, `asin`, `cosh`, `exp`, `sqrt`, `real`, `imag`, `log`, `log10`, etc.) As well, you should understand the basics of working with matrices and vectors, and how to create simple two-dimensional plots. In summary, you should be able to:

- 1) *Define vectors and matrices.*
For example, `A = [1 -5 6; 9 0 1; 2 9 0]` is a 3 x 3 matrix.
- 2) *Carry out matrix calculations.*
For example, using the matrix operators (+, -, *, /, inv, etc.)
- 3) *Carry out array calculations (element-by-element, or dot operators).*
For example, to calculate and plot the expression $(e^x + 1)^2$, for the values of x ranging from -5 to 15, we could use the commands:

```
>> x=linspace(-5,15,101) % x has 101 elements equally spaced from -5 to 15
>> y=(exp(x)+1).^2 % the .^ operator is used to square each element
>> plot(x,y,'b--') % This plots this function with a blue dashed line
```
- 4) *Create and modify two-dimensional figures.*
For example, through the use of the commands: `plot`, `figure`, `clf`, `hold`, `subplot`, `ezplot`, `xlabel`, `ylabel`, `title`, `legend`, `grid`, and `axis`.

IN-LAB WORK - Group

1. Three-Dimensional Plotting

As shown in the *Three Dimensional Graph* command table of *MATLAB Primer* Reference section, there are a number of ways to generate 3D plots in MATLAB. In this lab we will focus on using the `mesh` command.

To start, **plot the function given by** $(x, y) = \frac{1}{1+r} = \frac{1}{1+\sqrt{x^2+y^2}}$, over a 2 m x 4 m region that is centered at the origin. You can do this by following these steps:

1. Define the variables x and y , using the commands:

```
>> x = [-1:0.05:1]; % or we could use x = linspace(-1,1,41);
>> y = [-2:0.05:2]; % or we could use y = linspace(-2,2,81);
```

These variables specify the range over which we want to plot this function.

2. Verify that the sizes of the vectors x and y , are 1 x 41 and 1 x 81, respectively by using the `size` or `length` commands:

```
>> [nr,nc]=size(x) % This gives both the number of rows and columns
>> length(x) % This gives just the number of columns
```

3. To plot $f(x, y)$ over the entire region we must create the *matrix* fx_y that has a value at each point (x, y) . *What must be the size of this matrix given vectors x and y from above?*

This can be done in two ways. **Read through both and then create the matrix fx_y using whichever method you prefer.**

- a. The first method uses `for` loops:

```
for a = 1:length(y)
    for b = 1:length(x)
        fxy(a,b)=1/(1+sqrt(x(b)^2+y(a)^2));
        X(a,b)=x(b); % these matrices are needed
        Y(a,b)=y(a); % for plotting fxy(x,y)
    end
end
```

These nested `for` loops, run through each point in the region and calculate each element of fx_y individually. They also create the *matrices* X and Y which together contain each point where fx_y is defined, i.e., $[X(1,1), Y(1,1)]$, $[X(1,1), Y(1,2)]$, ..., $[X(1,1), Y(1, \text{length}(y))]$, etc.

- b. The second method is much simpler, and makes use of the `meshgrid` command. This command does the same thing as the `for` loop structure, meaning that it converts the *vectors* x and y , to *matrices* with the necessary sizes (X and Y). These matrices can then be used to calculate the matrix fx_y through element-by-element operations, i.e., the dotted operations `./` and `.^`

```
>> [X,Y]=meshgrid(x,y);
```


Then we can create the matrix fx_y by typing:

```
>> fxy = 1./(1+(X.^2+Y.^2).^0.5);
```

4. Check the sizes and contents of the matrices X and Y . What do you notice about how their elements are organized? _____
- _____
- _____

5. Generate the 3D plot of this charge distribution by typing:

```
>> figure;
>> mesh(X,Y,fxy);
>> xlabel('x axis');
>> ylabel('y axis');
>> zlabel('f_{xy}');
```

Experiment with using the zoom and rotate tools in the figure toolbar, .

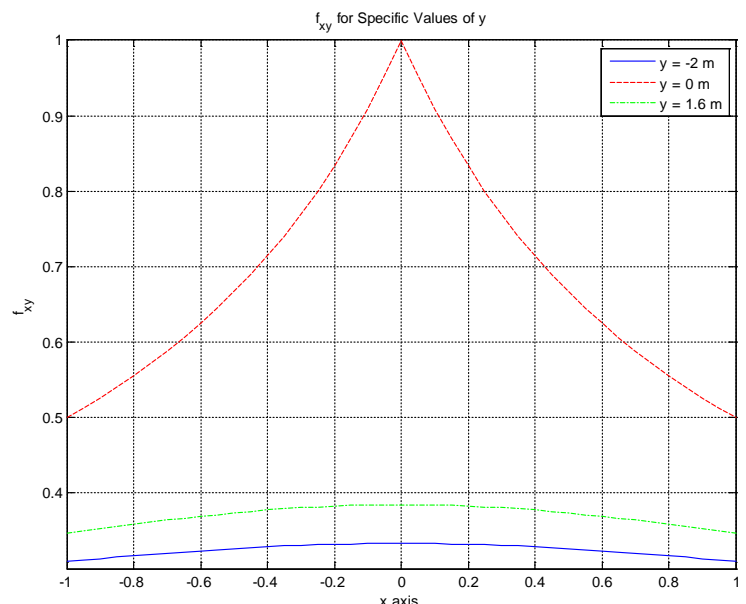
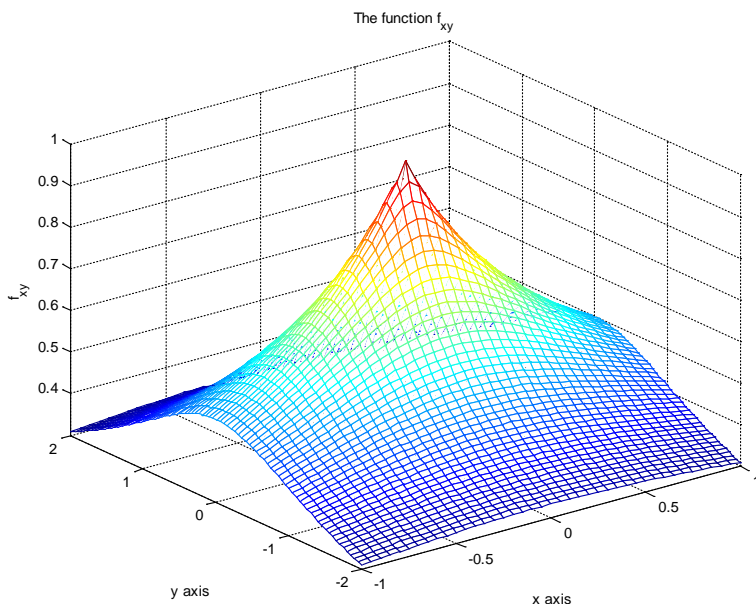
6. Create a 2D plot, which shows the variation of $f(x,y)$ with respect to x , when $y = -2$ m, $y = 0$ m, and $y = 1.6$ m. Your plot should have a single set of properly labeled axes with the three plots clearly identified. Use the `legend` command to help you do this.

The built-in function `find` may be of use to you in determining the indices of these three values of y . For example,

```
>> [r,c]=find(y==1.6)
```

Returns the values $r = 1$ and $c = 73$. This means that the 73rd element of the vector y is equal to 1.6. Since the matrix f_{xy} , which contains all the values of the function $f(x,y)$, has a size of 81×41 , the 73rd row of f_{xy} contains the values of this function over the range $-1 \leq x \leq 1$ for the fixed value of $y = 1.6$ m.

Your two plots should look something like:



2. Scripts and Functions

2.1. Working Directory


Find out the name of your working directory by typing `pwd` ('print working directory') following the MATLAB command prompt. Next, find out what files, if any, exist in your working directory by typing `dir`. Find out how you can change your working directory by using the command `cd` ('change directory').

Note: To run a MATLAB script or function, the file must either be in the working directory, or in a folder which is contained in the MATLAB path variable (see `help path`). The best idea is to make sure your scripts and functions are in the current working directory.

2.2. Scripts

Now create your first MATLAB script by typing the command `edit`, or opening the editor window (see `help edit`). In editor window copy in the following 3 lines of MATLAB code:

```
figure;
x = [0:20];
y = 6*x.^3+8*x.^2-3*x+12;
plot(x,y,'ro-');
grid on;
```

After you finished typing, click on File and choose Save as to save your MATLAB program under the name `myfirstscript.m`. If you successfully saved your program, you should be able to display it by typing `type myfirstscript` at the MATLAB prompt. To run your program, simply type its name as `myfirstscript` at the MATLAB prompt, or press the green arrow in the editor, .

Edit this script so that it plots the values of this polynomial for the range $-10 \leq x \leq 20$ with a blue dashed curve.

2.3. Functions

Next, create a *function* by copying these lines into the editor:

```
function [y]=polynomialvalue(x,plotflag)
% These commented lines directly below the function line will be displayed if
% you type help polynomialvalue at the command prompt. Often it is useful to
% give a copy of the function call so you can clearly see what the input and
% output variables are, meaning:
%     [y]=polynomialvalue(x,plotflag)
% If plotflag is set to 1, the polynomial will be plotted.
% Otherwise the plot will not be created.


y = 6*x.^3+8*x.^2-3*x+12;

if plotflag == 1
    figure;
    plot(x,y,'b--');
end
```

This function accepts the vector x as one of its input variables and returns the vector y to the MATLAB workspace. Note, there is not an “end function” command required for MATLAB functions. Make sure to save this as `polynomialvalue.m` (same name as the function) in your working directory.

Run this function by typing:

```
>> help polynomialvalue
>> [y]=polynomialvalue(linspace(-10,20,100),1);
```

Type `whos` to see your current list of variables stored in memory. Note that only the vector y is returned to the main MATLAB workspace, and that internal variables x and `plotflag` are no longer accessible once the function has finished. **Use the zoom tools** of the figure toolbar, , to find the one real root of this polynomial.

2.4. Debugging Functions in MATLAB

Within the MATLAB m-file editor there are many features that can be used to debug a MATLAB function, such as the setting of breakpoints (see the menu *Debug* and the toolbar buttons). However, one simple debugging tool is to include the keyboard command at the point of interest in the code. This command temporarily halts the execution of the code and gives you keyboard access so that you can check to see if your variables are the correct size or value. To continue the execution you can type `return`, or to stop the code from running you can type `dbquit`.

Try using the keyboard command by inserting it into your `polynomial.m` function after the end statement. You can see that this gives you access to the internal variables of the function and you can then return to the main command prompt, and thus exit the function, by typing `return` or `dbquit`.

2.5. Loops and Conditional Statements in MATLAB

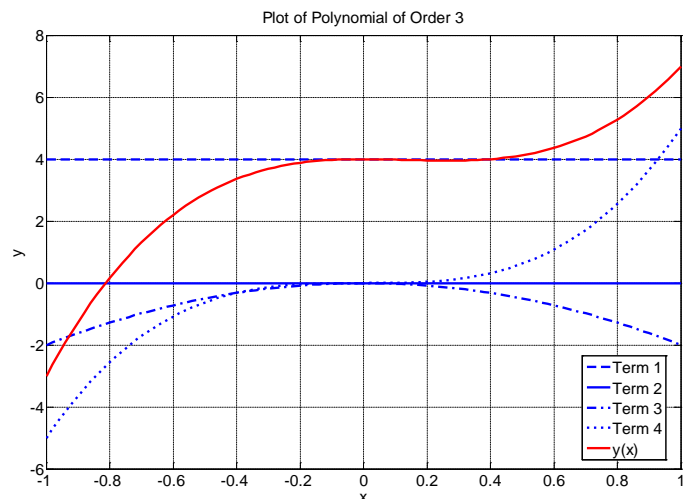
Part 1: Plots of Polynomial Functions

Write your own function that will calculate and plot any arbitrary polynomial function $y(x)$. Your function should have the following characteristics:

- 1) It should take as inputs: (i) a row or column vector that contain the coefficients of the different terms of the polynomial, (ii) the range of x over which you would like the polynomial to be calculated,
- 2) It should return as an output the value of the polynomial, $y(x)$, at each value of x .
- 3) Within the function it should create a single plot which contains a plot of each term as well as the overall polynomial function.

For example, for the polynomial given by $y(x) = 5x^3 - 2x^2 + 4$, the input row vector would be `[4;0;-2;5]`, and the final plot would look similar to that figure shown to the right.

Hint: You may find that the use of a `for` loop and an `if else` conditional statement might be helpful to implement this function.



2.5. Loops and Conditional Statements in MATLAB (cont'd)*Part 2: Simple Integration*

Now write a function that will approximate the value of the integral:

$$\int_{x_1}^{x_2} f(x) dx$$

You **cannot use** any of the built-in integration functions in MATLAB, such as `quad`, `trapz`, or `integral`, and instead must rely on the use of some combination of loop and/or conditional statements.

Your function should have at least two input variables representing the integration limits, x_1 and x_2 , but you do not have to pass in the integrand, $f(x)$, as this can be hardcoded into the function. Your function must be able to calculate the value of the integral to at least a 0.25% accuracy.

Your TA will give you the specific integrand, $f(x)$, as well as the integration limits, x_1 and x_2 , during the lab.

Our integrand is: $f(x) =$

Our limits of integration are: $x_1 =$ $x_2 =$

Therefore, the exact result is: $\int_{x_1}^{x_2} f(x) dx =$

The result from our MATLAB function is: $\int_{x_1}^{x_2} f(x) dx \approx$

Which is accurate to _____ percent (show your calculations below).