# Introduction to CircuitLab and Matlab

Circuit design and analysis is usually a three step process:

- First, you design your circuit on paper.
- Next, you simulate your design to make sure it works.
- Then you build your circuit in lab

The tools we'll be using in this class to do this are:

- Matlab: a tool that makes circuit design and analysis much easier. Think of Matlab as a calculator which can solve 50 equations for 50 unknowns and graph the results.
- CircuitLab: a very friendly circuit simulator which allows you to build a circuit using drag and drop. Once built, you can find the voltages or run a time simulation to see how the voltages change over time (useful for AC analysis).
- Breadboards, oscilloscopes, and multimeters: used in lab to allow you to connect components together to build a circuit.

## CircuitLab

www.CircuitLab.com

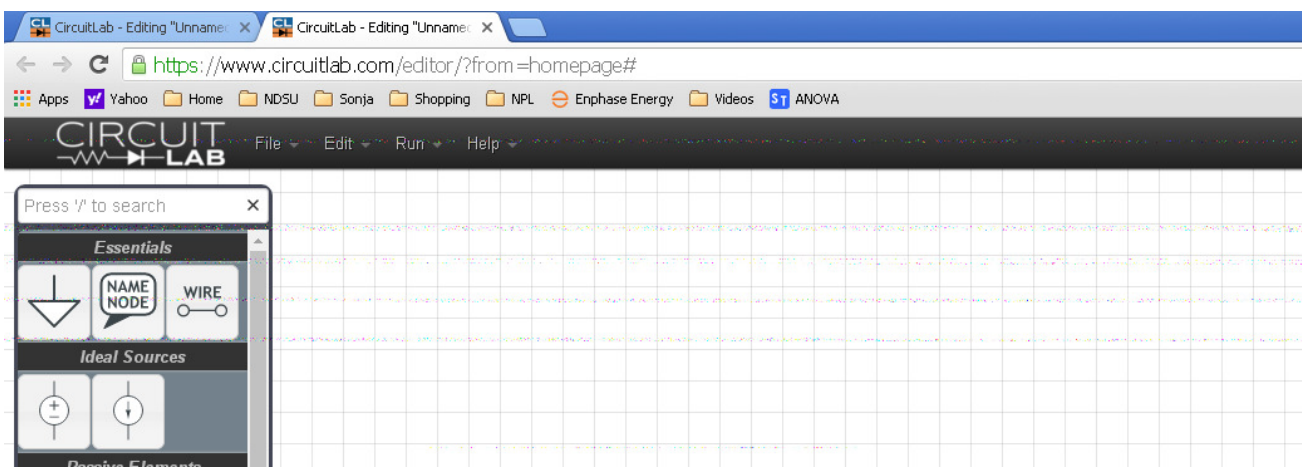


Figure 1: Welcome screen in CircuitLab.

CircuitLab is a circuit simulator which is very friendly to use. It's fairly intuitive so you can just jump right in and start simulating circuits. There are also multiple videos to walk you through building circuits using CircuitLab.

Before you start using CircuitLab, you will need to create an account.

**Note: Register using Use your NDSU email address and it will be free for you to use.**

- **name@ndsu.edu** should work
- **name@ndus.edu** will *not* work.
- Same with Matlab

NDUS is your email address for the entire North Dakota University system. This includes NDSU, UND, Minot State, Dickinson State, etc. NDSU only pays the site license for NDSU - not for other schools in the NDUS system.

CircuitLab is very affordable and costs $34/year for individual users (2025 price). NDSU has an institutional license, however, so it's free for NDSU students to use.

Once you have an account, you can save your circuits for later use.

To build a circuit, drag and drop components from the left (we'll talk about what each one is later in this course)

- R rotates the component
- Double click to change the value

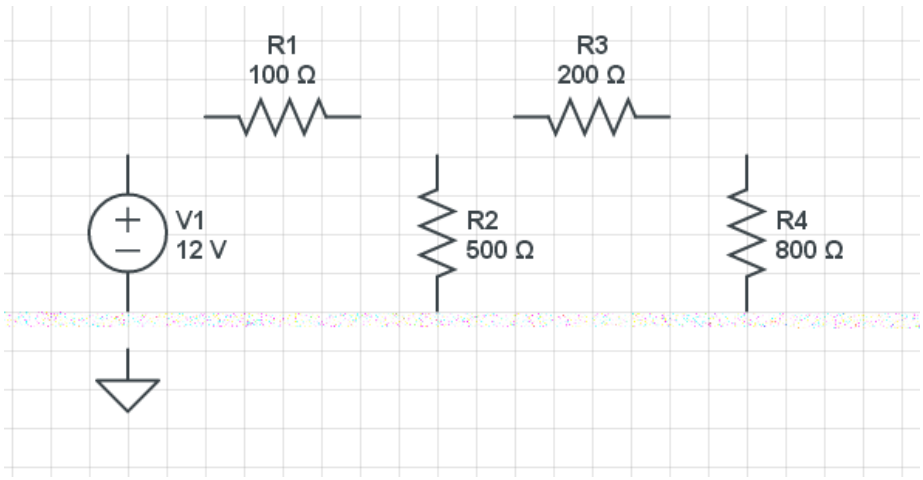


Figure 2: Drag and drop circuit elements to start your circuit

Click and drag to connect the wires together. Adding labels allow you to look at different voltages

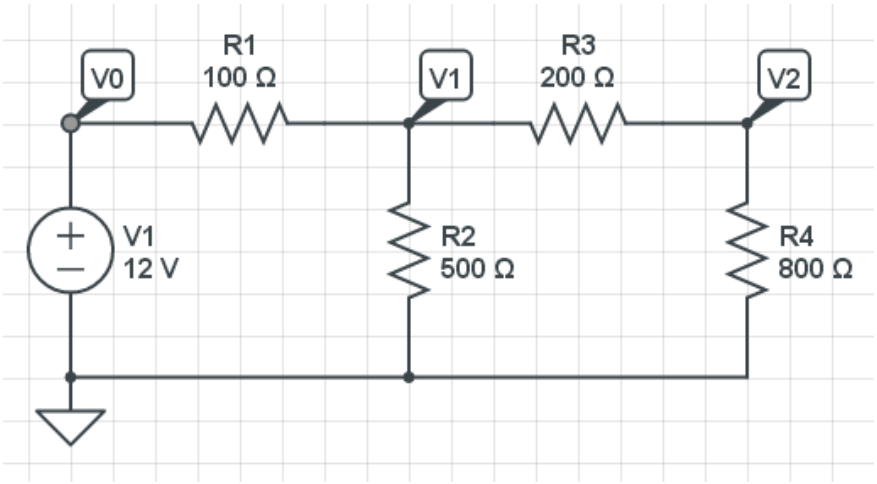


Figure 3: Left click to drag and drop wires, connecting components

To find the voltages, click on

- Simulate
- + Add Expression ( Select V0, V1, and V2 )
- Run DC Solver

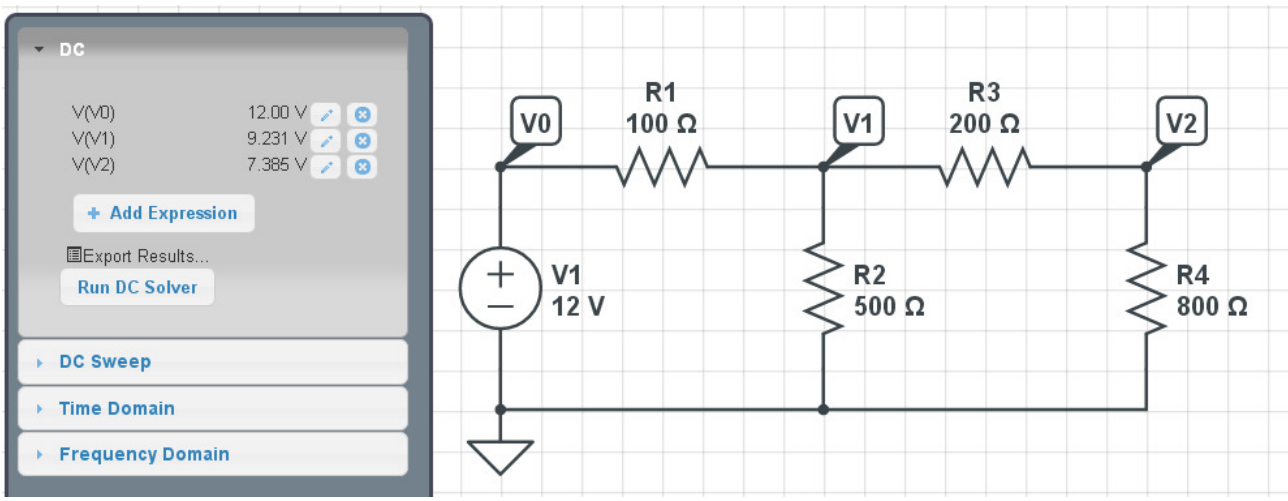


Figure 4: To find the DC voltages, click on +Add Expression to select what you want to find then Run DC Solver

If you build the circuit in lab, you should measure the same voltages with a multimeter (give or take. Resistors have a 1% to 5% tolerance, which will slightly affect the voltages you measure.)

You can also look at transient simulation. Change the source to an AC signal and change R2 and R4 to capacitors

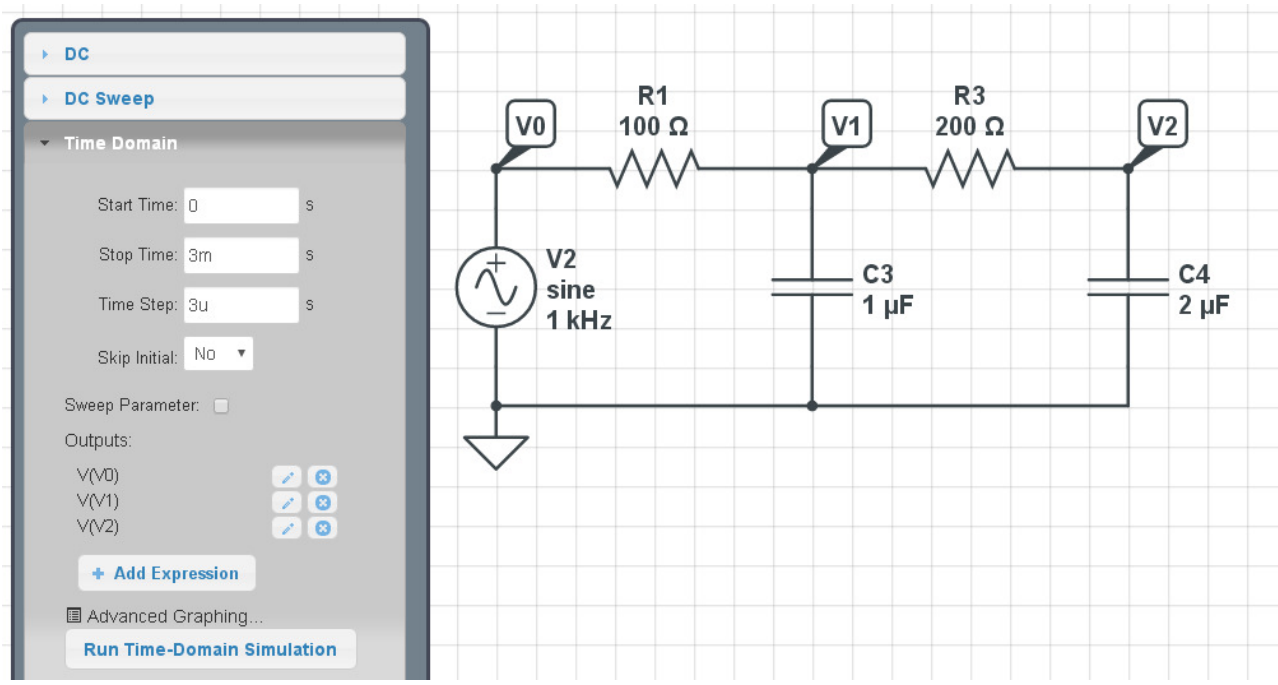


Figure 4: For time-varying signals, run a Time-Domain Simulation.
Make the Time Step 1000x smaller than the Stop Time to generate 1000 points on the plot

Now run a Timer Domain simulation

- The input is a 1kHz sine wave. Run the simulation for a few cycles (3 cycles or 3ms)
- Make the time step 100 to 1000x smaller ( 3us gives 1000 points on the following graph). If you make the time step too small, it will take some time to finish the simulation.
- + Add expression to see V0, V1, and V2
- Click on Run Time Domain Simulation

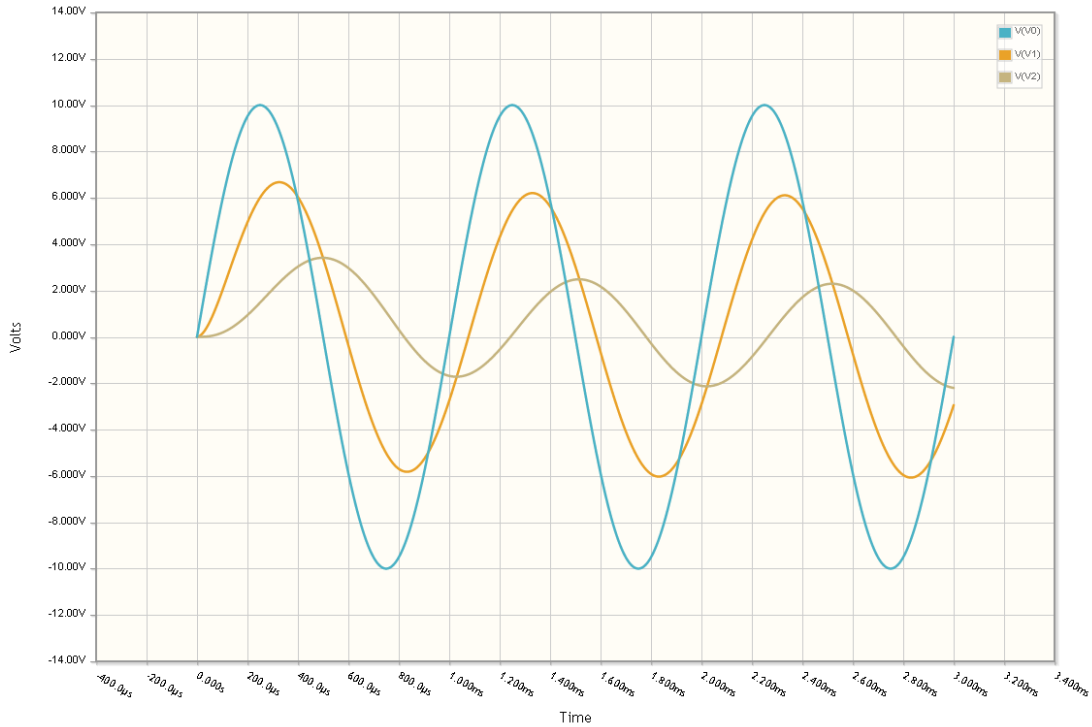


Figure 5: After running a Time-Domain simulation, a graph showing the voltages and/or currents vs. time is displayed

In lab, you would use an oscilloscope to look at the voltages V0, V1, and V2. The oscilloscope would then output an image like that shown above.

# Introduction to Matlab

## Becoming familiar with MATLAB

- The console
- The editor
- The graphics windows
- The help menu
- Saving your data (diary)
- Solving N equations with N unknowns
- Least Squares Curve Fitting

## General environment and the console

When you start up Matlab, the screen should look something like this:

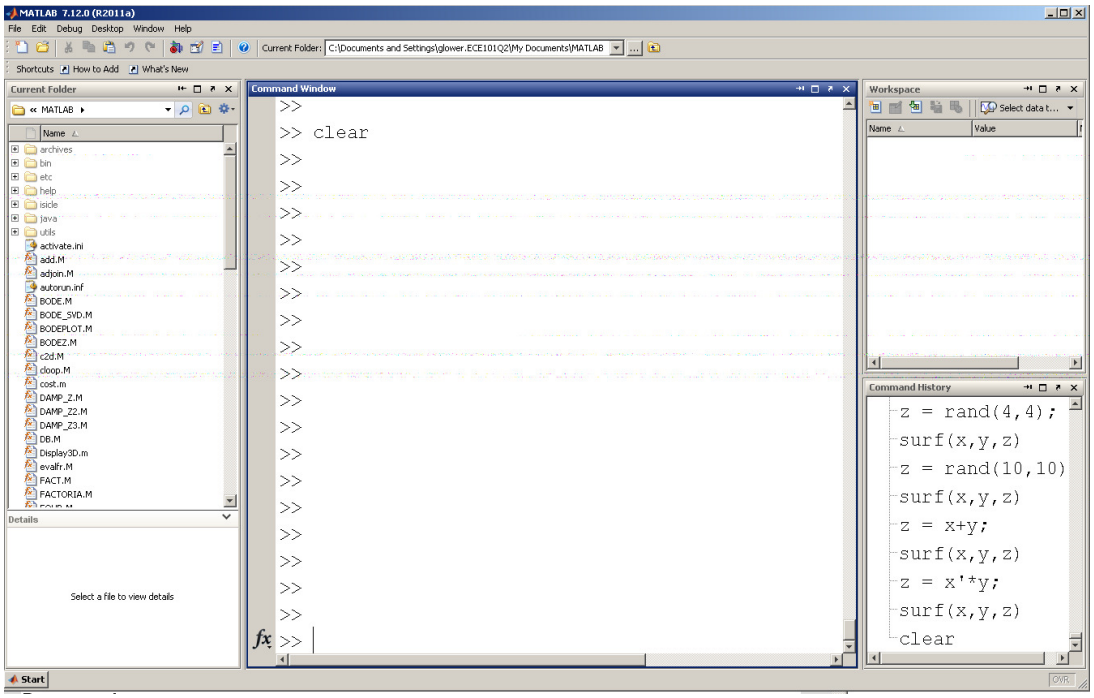


Figure 6: The default display in Matlab includes a directory, command window, workspace, and command history.

Usually, I close all of the windows except the Command Window. Matlab can be used like a calculator - with some of the command options listed in the appendix. For example, if you want to compute

```
( 2 + 3 ) * 5
```

You type it the way it looks:

```
>> (2 + 3)*5

ans =   25
```

In Matlab, you can save the results and use them later - like the memory functions of your calculator. Valid names for your variables have to start with a letter. Note that Matlab is case sensitive. For example

```
>> a = (2+3)*5

a =  25

>> b = 1.3 * a^3

b =  2.0313e+004
```

## Matrices in Matlab

Matlab is also a matrix language. An nxm matrix is an array of numbers arranged in n rows and m columns

$$A_{2x3} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

The syntax to input a matrix are:

```
[           start of a matrix
,           next column (a space also works)
;           next row
]           end of matrix.
```

For example, to input a 2x3 matrix for A:

```
A = [1,2,3 ; 4,5,6]

     1     2     3
     4     5     6
```

Matrix Addition: When adding matrices, each element gets added. You can only add matrices of like dimensions.

```
A = [1,2,3 ; 4,5,6]

     1     2     3
     4     5     6

B = [2,2,2 ; 3,3,3]

     2     2     2
     3     3     3

C = A + B

     3     4     5
     7     8     9
```

Matrix Multiplication: When multiplying matrices, the inner dimension must match

$$C_{mn} = A_{mx} B_{xn}$$

Element (i,k) of matrix C is computed as:

$$c_{ik} = \sum a_{ik} b_{kj}$$

## Sample Matlab Code

If you terminate a line with a semicolon, the result is computed but isn't displayed. If you leave off the semicolon, the results is displayed. For example:

```
x = 2*pi;           % result is computed and stored in x but isn't displayed
x = 2*pi            % ditto but the result is displayed.

   6.2832
```

If you want a different format for the display, you can use the 'sort', 'long', and 'shorteng' commands:

```
format short
pi

   3.1416

format long
pi

  3.141592653589793


format shorteng
pi^30

  821.2893e+012
```

Matrices

- [ start of matrix
- ] end of matrix
- , next element
- ; next row

```
A = [1,2,3]

   1     2     3



B = [1,2,3;4,5,6]

    1     2     3
    4     5     6

C = A'

    1
    2
    3

D = zeros(1,3)
```

```
     0     0     0

E = rand(3,2)

   0.5860    0.0835
   0.2467    0.6260
   0.6664    0.6609
```

for loops:

```
x = zeros(1,5);
for i=1:5
  x(i) = i*i;
  end
x

x =

     1     4     9    16    25
```

while

if

if else end

# Scripts and Functions in Matlab

Matlab is also a programming language. This allows you to run the same code over and over without having to constantly retyping in the code over and over again.

- Scripts are like code you type in the command window. When you run a script, it is like you just typed in the code in the script in the command window
- Functions are subroutines. They allow you to create new commands in Matlab which can be called by other new functions and so on.

This is why Matlab is used by industry so much: you can create a library of functions and scripts which allow you to design your product.

## Scripts:

Suppose you want to see the distribution of rolling two six-sided dice (2d6) 100 times. This is where scripts shine.

- Start with a New Script.
- Get the code to produce 2 random numbers
- Click on the green arrow to run the script (F5)

note: Save the script in the Matlab directory. If you save it somewhere else on your computer, Matlab won't be able to find it.

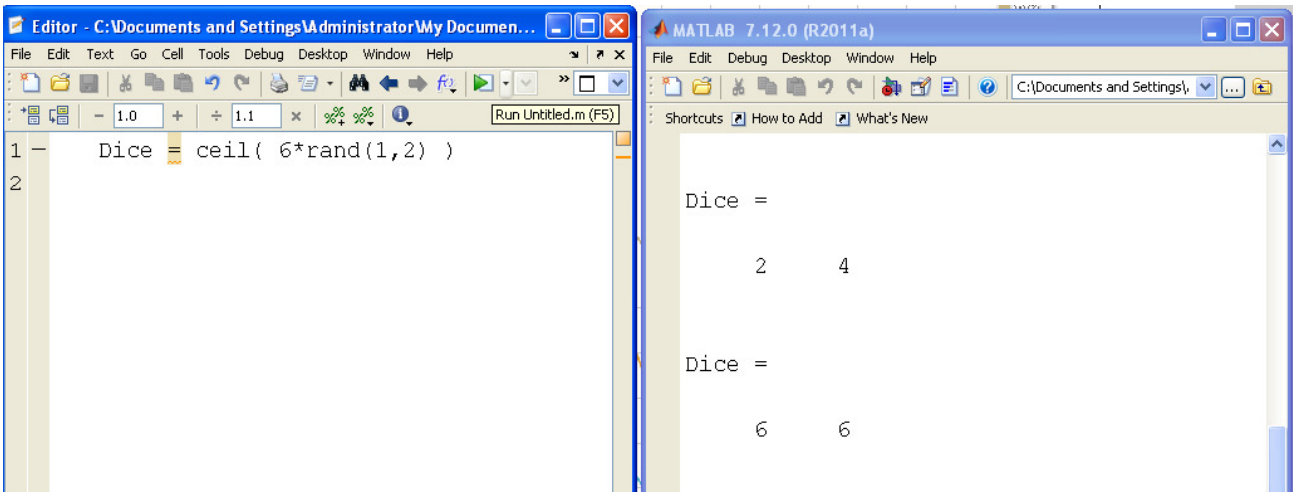


Figure 7: Script Window (left) and Command Window (right)
Click the green arrow to run the script. The result shows up in the command window.

Once you are convinced the script works,

- Get it to sum the two dice
- Then repeat 100 times

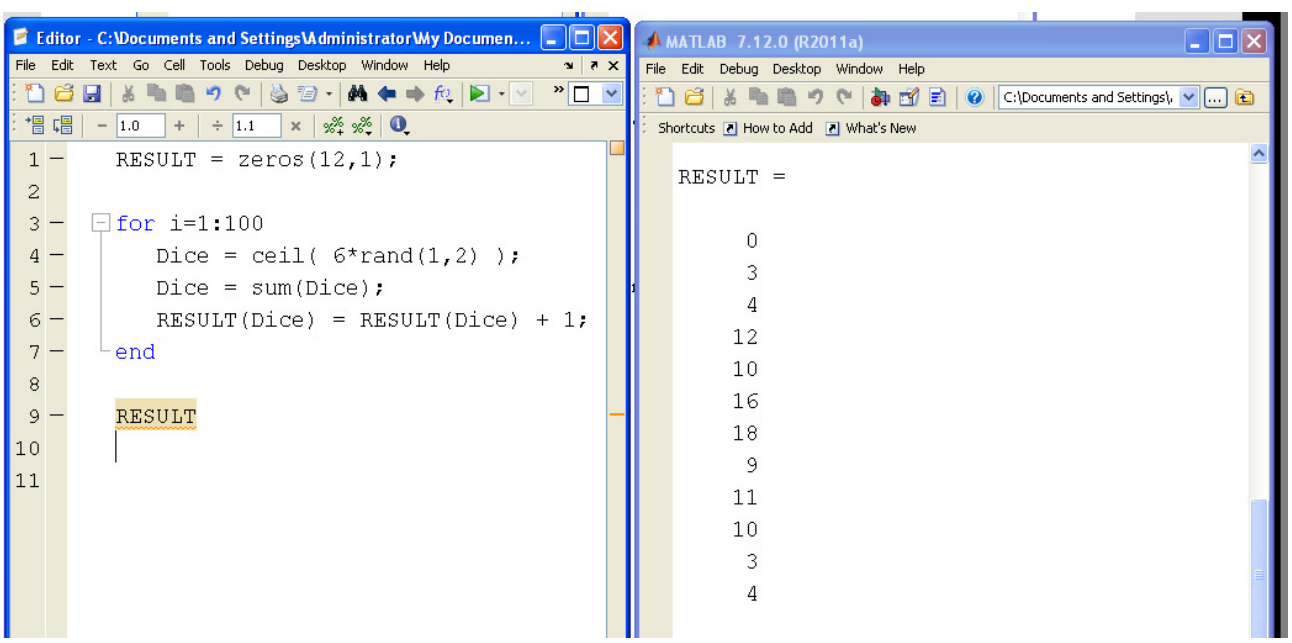


Figure 8: Script for rolling two 6-sided dice (2d6) 100 times (left window) and the result shown in the Command Window (right)

The beauty of scripts is that

- If you want to see what happens if you run it a second time, you can just by clicking the green arrow again.
- If you want to roll the dice 1000 times instead, change one line of code ( i = 1:100 becomes 1:1000 )

## Functions

A function is a subroutine. It allows you to create new functions in Matlab that can be used over and over again.

For example, let's create a function called NDice. It will be passed two parameters

- N: The number of dice
- Sides: The range of numbers (1..6, 1..12, etc.)

In Matlab,

- click on New Function
- Create the subroutine
- Save with the same name as the function

With that you just created a new function in Matlab called NDice. From the command window, you can

- Find the sum of rolling 5d6 (total is 18)
- Find the sum of rolling 12d4 (total is 26)

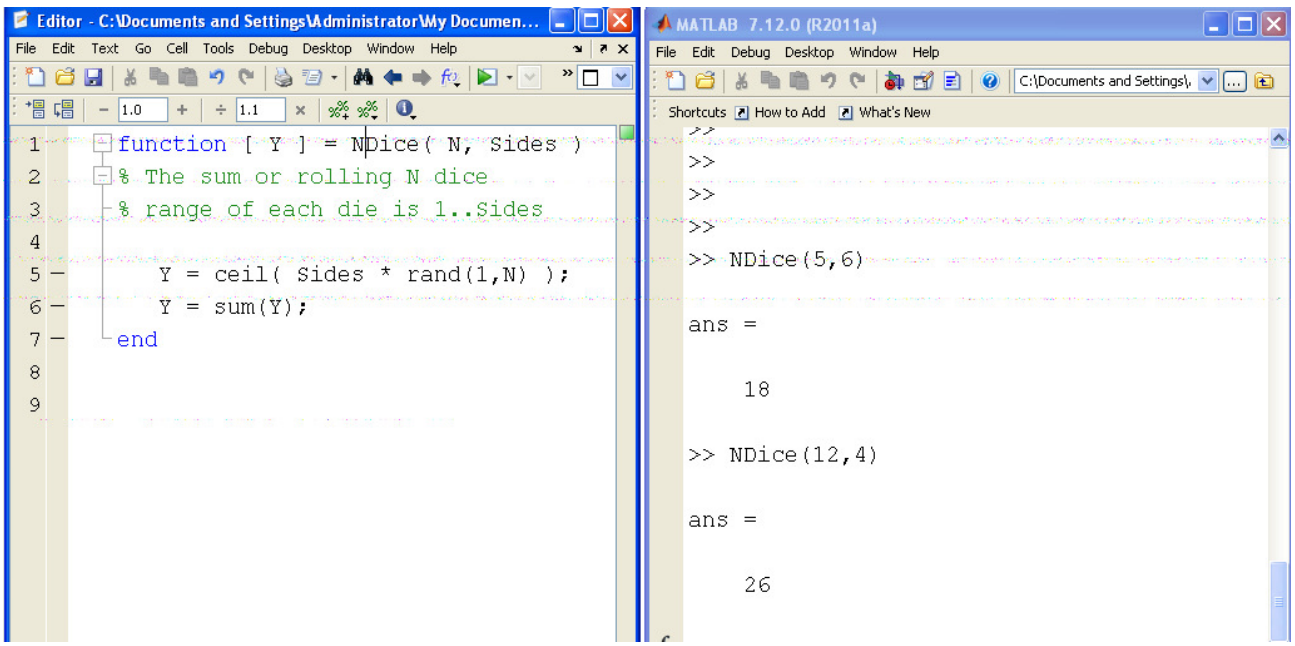


Figure 9: Function for rolling N 6-sided dice (Nd6) and examples of calling it in the Command Window

Now that you have this function, you can clean up the script

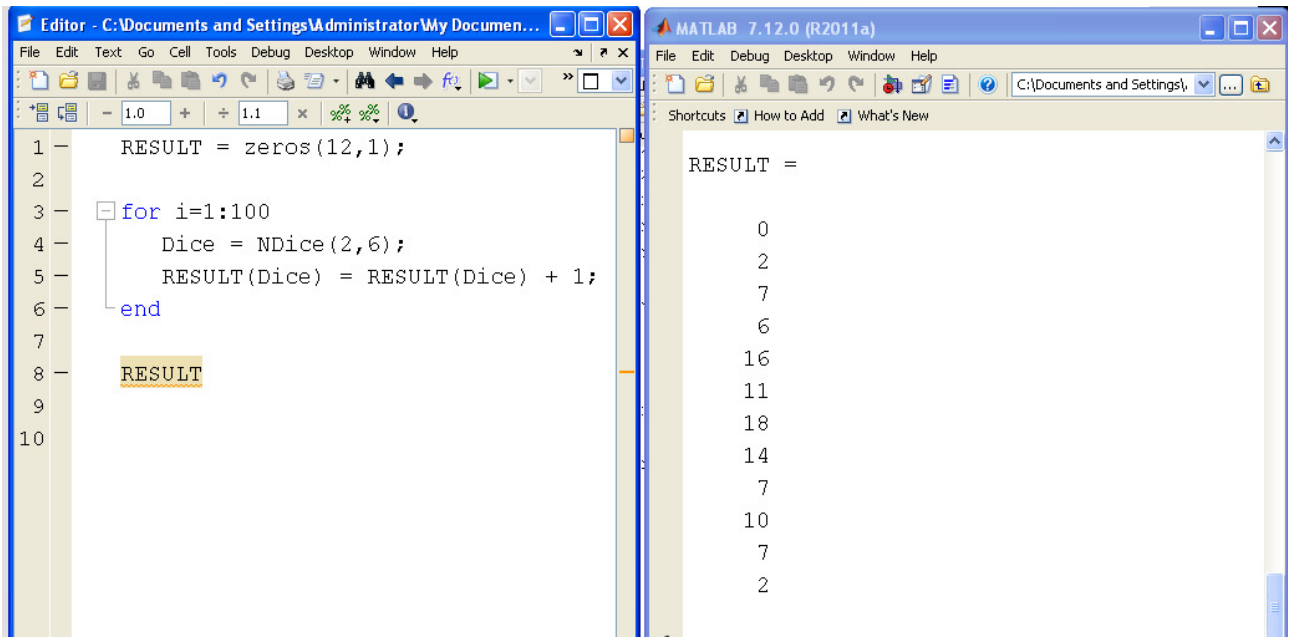


Figure 10: Example of a script that calls a function we just created: NDice().

Note that since this script uses a function we just create, other people won't be able to run this script unless they too create the function NDice.

## Plotting Functions in Matlab:

Matlab has some pretty good graphics capabilities.

| Matlab Plot Command | x axis | y axis | type of function |
|---|---|---|---|
| `plot(x,y)` | linear | linear | $y = ax + b$ |
| `semilogx(x,y)` | log() | linear | $y = a \cdot e^{bx}$ |
| `semilogy(x,y)` | linear | log() | $y = a + b\ln(x)$ |
| `loglog(x,y)` | log() | log() | $y = a \cdot b^{x}$ |
| `subplot(abc)` | Create 'a' rows, 'b' columns of graphs. Starting at #c | | |

Table 1: Various plot functions available in Matlab

For example, plot the function

$$y = 3x + 4$$

```
x = [0:1:10]';
y = 3*x + 4;

subplot(131)
plot(x,y); % connect the points with a line

subplot(132)
plot(x,y,'.');       % plot a dot at each point

subplot(133);
plot(x,y,'.-');      % connect the points and add a dot
```

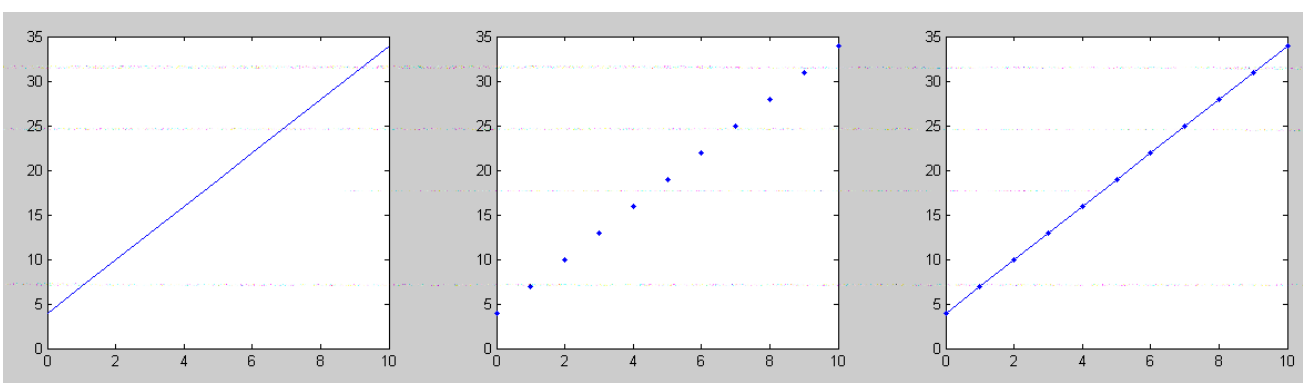


Figure 11: Example of subplot's and plot types in Matlab

Plot how much money you'd have if you invested it for 10 years at

- 1% interest
- 3% interest
- 6% interest
- 9% interest

Three ways to do this:

```
t = [0:0.01:10]';
y1 = 1000 * exp(0.01*t);
y3 = 1000 * exp(0.03*t);
y6 = 1000 * exp(0.06*t);
y9 = 1000 * exp(0.09*t);

% Method #1
plot(t,y1,t,y3,t,y6,t,y9)

% Method #2
plot(t,[y1,y3,y6,y9])

% Method #3
plot(t,y1)
hold on
plot(t,y3)
plot(t,y6)
plot(t,y9)
hold off
```

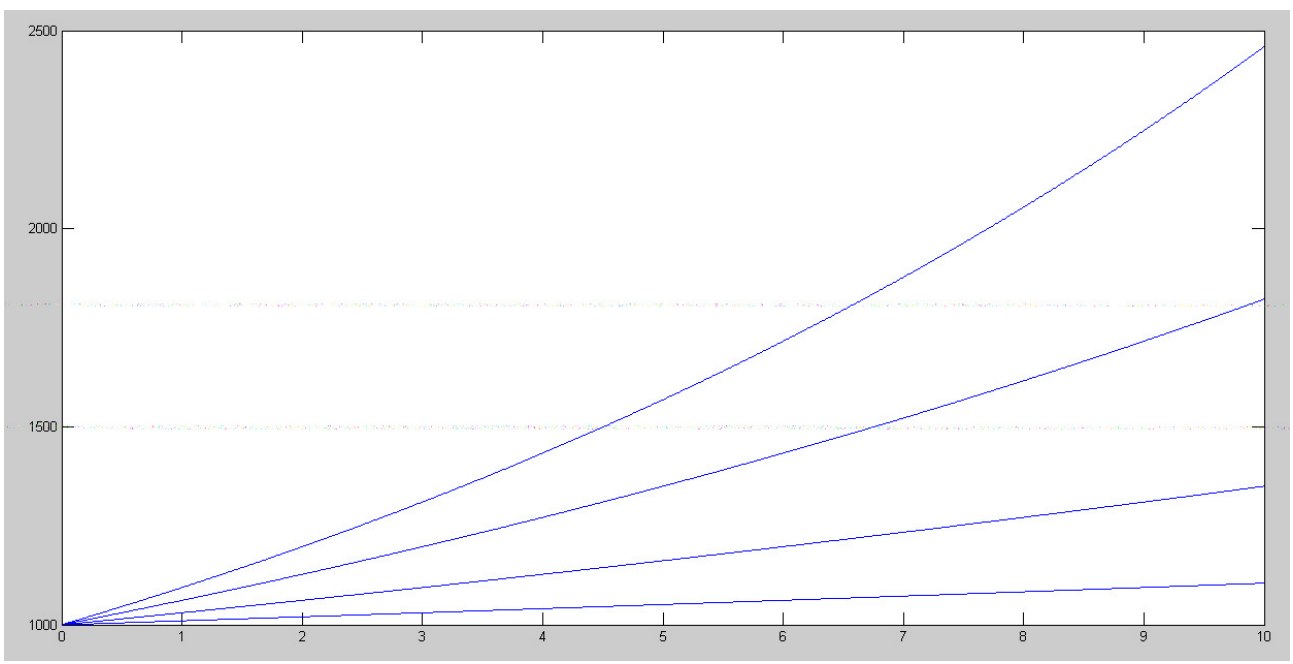


Figure 12: You can plot several functions on the same graph.
This graph shows the time-value-of-money at 1%, 3%, 6%, and 9% interest.

If you invest at 9% interest, you have almost 2.5x more money after 10 years than you'd have at 1% interest.

Polynomials

- poly([a,b,c]) Give a polynomial with roots at (a, b, c)
- roots([a,b,c,d]) Find the roots of the polynomial

$$ax^3 + bx^2 + cx + d = 0$$

Example:

- Determine a polynomial with roots at (1, 2, 3)
- Plot that function,
- Verify that the roots are in fact at (1, 2, 3)

In Matlab:

```
poly([1,2,3])

     1    -6    11    -6
```

meaning

$$y = x^3 - 6x^2 + 11x - 6 = (x-1)(x-2)(x-3)$$

```
roots([1,-6,11,-6])

   3.0000
   2.0000
   1.0000


x = [0:0.01:4]';
y = x.^3 - 6*(x.^2) + 11*x - 6;
plot(x,y);
grid on
```

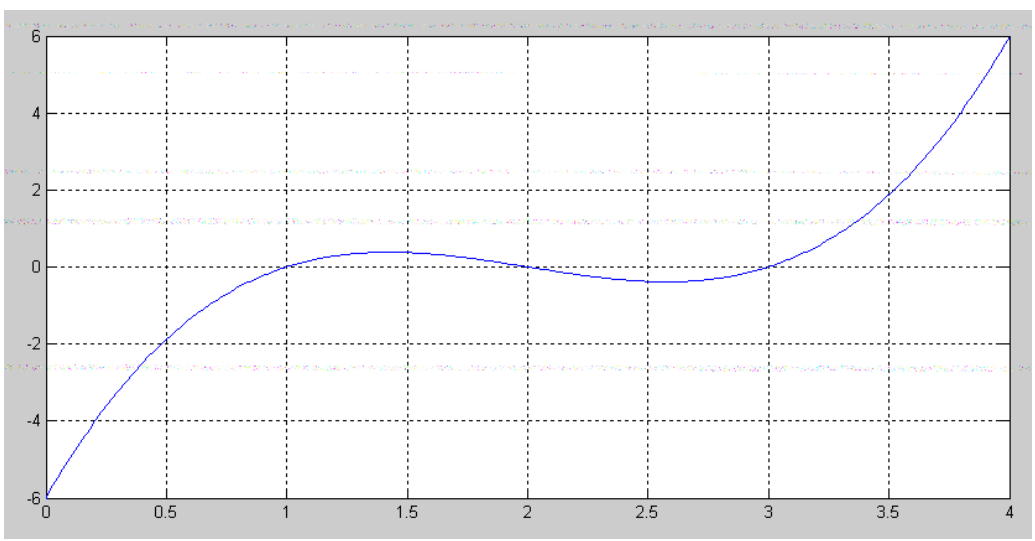


Figure 13: The Matlab function roots() finds the roots of a function - which are also the zero crossing.

## 3-D Plots

```
mesh(z)
```

Draw a 3-d mesh for the array 'z' with the height being the value in the array.

Example

$$z = x^2 + y^2$$

```
x = [-3:0.1:3]';
y = [-3:0.1:3]';
size(x)

ans =

    61     1

z = zeros(61,61);
for i=1:61
  for j=1:61
     z(i,j) = x(i)^2 + y(j)^2;
     end
  end

mesh(z)
```

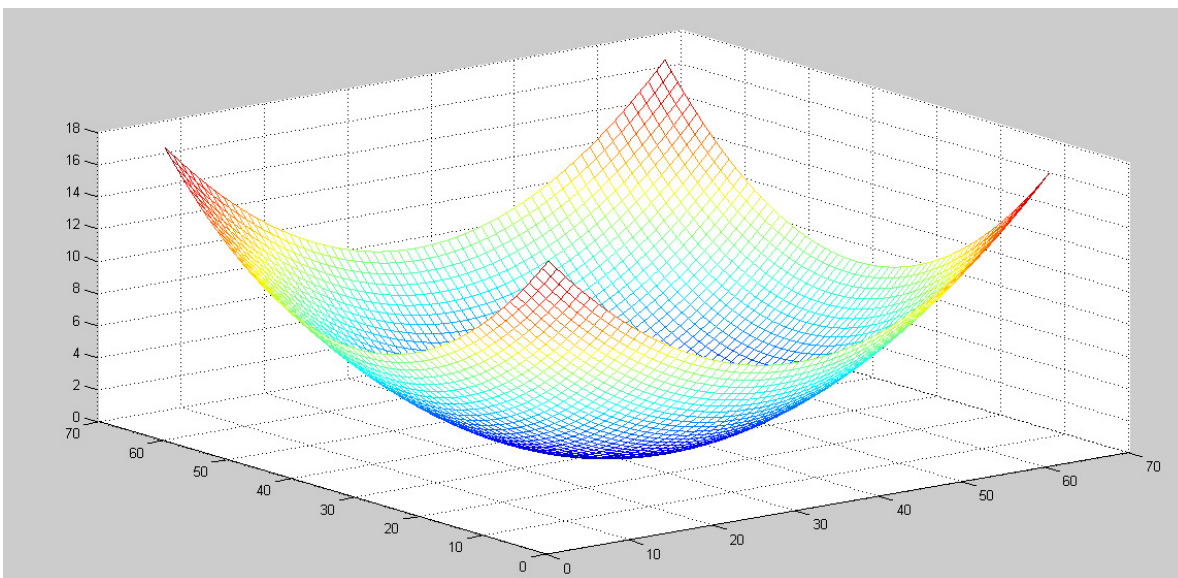


Figure 14: The command mesh() draws a pretty 3-d image representing the height vs. (x,y)

Another pretty plot:

$$r = \sqrt{x^2 + y^2}$$

$$z(x, y) = \left(\frac{\sin(r)}{r}\right)$$

```
x = [-10:0.1:10]';
y = [-10:0.1:10]';
size(x)

  201     1

z = zeros(201,201);
for i=1:201
  for j=1:201
     r = sqrt(x(i)^2 + y(j)^2);
     z(i,j) = sin(r) / (r + 0.000001);
     end
  end
mesh(z)
```

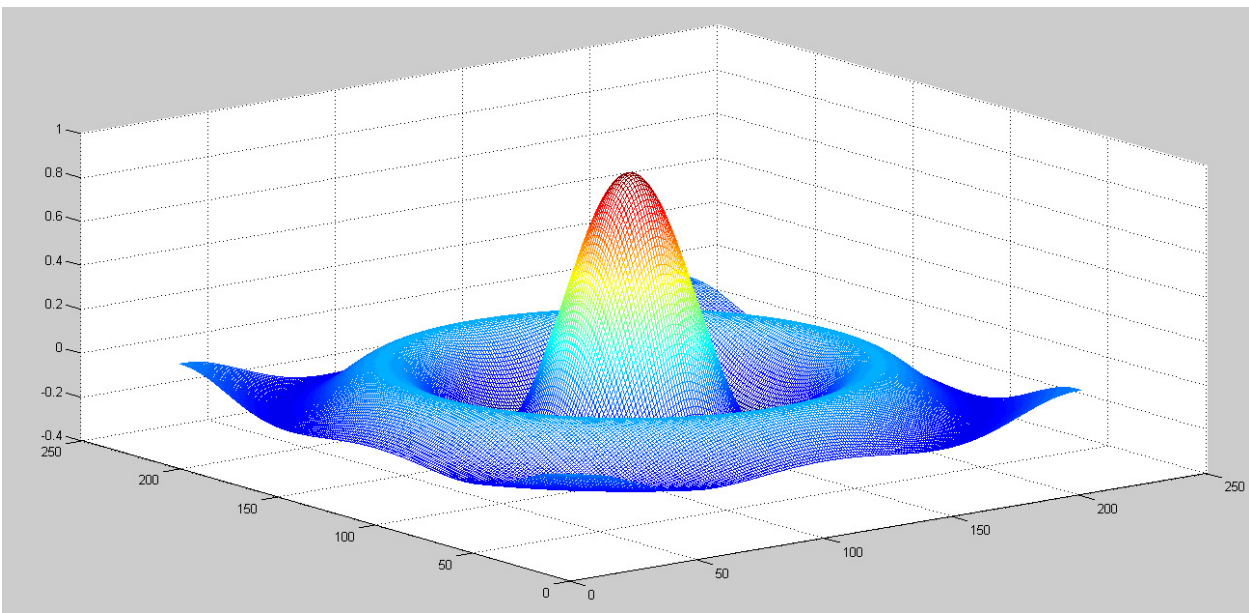


Figure 15: Another pretty graph using the mesh() function in Matlab

## Summary

Matlab is a very power tool that's become the de-facto standard in engineering. At its heart, it's just a calculator similar to your hand calculator. It also adds

- Matrix operations
- Graphics
- Programming capability
- The ability to write and execute routines (Matlab scripts),
- The ability to create and use custom routines (Matlab functions)

to name just a few features.

Matlab is used extensively in several ECE courses you'll take at NDSU. That's part of the reason this course exists. The goal of ECE 111 is to get you familiar with using Matlab along with some of its capabilities. That way, when you get to Circuits I or other courses that use Matlab, you can focus on that course rather than trying to figure out how to use this tool.

In addition to Matlab, CircuitLab is also used extensively in many ECE courses. It too is a very easy to use and intuitive tool that lets you try out your ideas and check your designs. Again, the goal of this course is to get you familiar with CircuitLab so that it will be easier to use once you get to Circuits I and other courses in ECE.

# Matlab Commands

## Display

- format short — display results to 4 decimal places
- format long — display results to 13 decimal places
- format short e — display using scientific notation
- format long e — display using scientific notation

## Analysis

- sqrt(x) — square root of x
- log(x) — log base e
- log10(x) — log base 10
- exp(x) — e^x
- exp10(x) — 10^x
- abs(x) — |x|
- round(x) — round to the nearest integer
- floor(x) — round down (integer value of x)
- ceil(x) — round up to the next integer
- real(x) — real part of a complex number
- imag(x) — imaginary part of a complex number
- abs(x) — absolute value of x, magnitude of a complex number
- angle(x) — angle of a complex number (answer in radians)
- unwrap(x) — remove the discontinuity at pi (180 degrees) for a vector of angles

## Polynomials

- poly(x)
- roots(x)
- conv(x,y)

## Trig Functions

- sin(x) — sin(x) where x is in radians
- cos(x) — cos()
- tan(x) — tan()
- asin(x) — arcsin(x)
- acos(x) — arccos(x)
- atan(x) — arctan(x)
- atan2(y,x) — angle to a point (x,y)

## Probability and Statistics

- factorial(x) — (x-1)!
- gamma(x) — x!

- rand(n,m) create an nxm matrix of random numbers between 0 and 1
- randn(n,m) create an nxm matrix of random numbers with a normal distribution
- sum(x) sum the columns of x
- prod(x) multiply the columns of x
- sort(x) sort the columns of x from smallest to largest
- length(x) return the dimensions of x
- mean(x) mean (average) of the columns of x
- std() standard deviation of the columns of x

## Display Functions

- plot(x) plot x vs sample number
- plot(x,y) plot x vs. y
- semilogx(x,y) log(x) vs y
- semilogy(x,y) x vs log(y)
- loglog(x,y) log(x) vs log(y)
- mesh(x) 3d plot where the height is the value at x(a,b)
- contour(x) contour plot
- bar(x,y) draw a bar graph
- xlabel('time') label the x axis with the word 'time'
- ylabel() label the y axis
- title() put a title on the plot
- grid() draw the grid lines

## Useful Commands

- hold on don't erase the current graph
- hold off do erase the current graph
- diary create a text file to save whatever goes to the screen
- linepace(a, b, n) create a 1xn array starting at a, increment by b
- logspace(a,b,n) create a 1xn array starting at 10^a going to 10^b, spaced logarithmically
- subplot() create several plots on the same screen
- disp('hello') display the message *hello*

## Utilities

- format set the display format
- zeros(n,m) create an nxm matrix of zeros
- eye(n,m) create an nxm matrix with ones on the diagonal
- ones(n,m) create an nxm matrix of ones
- help help using different functions
- pause(x) pause x seconds (can be a fraction). Show the graph as well

- clock the present time
- etime the difference between to times
- tic start a stopwatch
- toc the number of seconds since tic