Octave Tutorial Documentation

Rocco Vulpis

April 25, 2024

Basic Arithmetic Operators

Addition: “+”

Subtraction: “- “

Multiplication: “\*”

Exponentiation: “^” or “\*\*”

Division: “/”

Storing Variables

name of variable = expression

Ex.) >> a = 2

a = 2

The number 2 will be stored to a variable named a and printed on the following line.

Ex.) b = 2;

The number 2 will be stored internally to a variable named b, but when ending the line with a semicolon, the next line will not print the assignment of the variable.

If you try to print a variable that has not been initialized, an error will print on the next line.

Ex.) >> c

Error: ‘c’ undefined near line 1 column 1

There are a few different ways to check what variable names have been assigned.

Commands: “who” or “whos”

When naming a variable, special characters (!, ^, \*, $) or blank spaces may not be used. A variable name may also not start with a number (1, 4, 7, 345). Variable names are also case sensitive.

Variables storing text must be stored single quotes.

Ex.) >> a\_text = ‘Hello’

a\_text = Hello

Vectors

name of variable = [element1 element2 element3 …]

Ex.) >> a = [1 2 3 4]

a =

1 2 3 4

Elements use index numbers.

Ex.) >> a (2)

ans = 3

Replace an element at a given index.

Ex.) >> a (2) = 5

a =

1 2 5 4

Replace multiple elements at once.

Ex.) >> a ([2 3]) = [5 6]

a =

1 2 5 6

Create a 1 by 5 row vector increased by one without manually adding each element.

Ex.) >> a = 1:5

a =

1 2 3 4 5

Accessing the 1st to 3rd element.

Ex.) >> a (1:3)

ans =

1 2 3

Create a column vector.

Variable name = [element1; element2; element3]

Notice that semi colons separate the values in between the square brackets.

linspace Function

linspace *(base, limit, n)*

This function will return a row vector with *n* linearly spaced elements between *base* and *limit*.

Sliding Windows of Data

*y =* movmean *(x, wlen)*

This function will calculate the moving average over a sliding window of length *wlen* on data *x*.

Scatter and Line Plotting

Plotting a point to a graph is done by using the “plot” function.

plot(x, y)

When there’s the need to plot multiple points, same sized vectors can be used as parameters.

Plots can be customized using plot properties.

plot(x, y, property)

Plotting Points Example

>> x = [1 3 6]

>> y = [2 4 1]

>> plot(x, y, ‘or’)

The plot function plots the points (1, 2), (3, 4), (6, 1) with the points being represented as circles that are red (because the third parameter is ‘or’ where o = a circle and r = red).

To set axis, use the “axis” function.

axis([x-min x-max y-min y-max])

How to Plot, Salt, and Smooth

**Plotting**

The first thing to do is pick out a function to graph. For this example, the function being implemented is y = x2.

From here, choose a coefficient of x2, coefficient of x, a constant, range of x values, and the number of points to graph.

>>a = 1;

>>b = 0;

>>c = 0;

>>xBeginning = -80;

>>xEnd = 80;

>>numberOfPoints = 200;

The next step is to create an array of x values using the “linspace” function to evenly space out the range of x values with the desired number of points to graph.

>>linspace(xBeginning, xEnd, numberOfPoints);

Now y can be set equal to ax2 + bx + c to complete the equation.

>> y = a \* x.^2 + b \* x + c;

After setting up the equation, it can now be plotted to a figure. The windows will be manipulated so when the program is executed, they will align from left to right in the middle of the screen. Starting with the first figure, it is going to be 550x500 pixels and be manually positioned at the x position 5 and y position 400 (Essentially in the middle of the screen almost all the way to the left).

The first thing to do is create the figure. There will be three parameters. The first parameter is the number of the figure. The second is a property of the figure and in this case it’s the “position” property so that the window can be positioned. The last parameter of the function is the coordinates and size of the figure because the position argument was used.

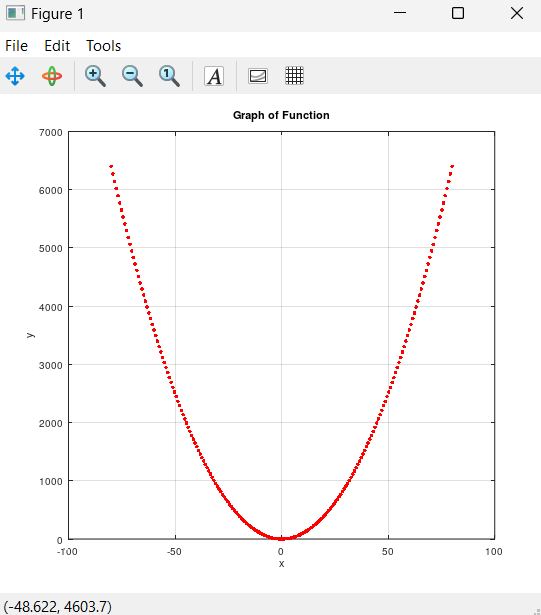
>>figure(1, “position”, [5, 400, 550, 500]);

Creates a 550 by 500 figure positioned on the screen at the coordinates (5, 400).

The function can now be plotted to the figure that was just created.

>>plot(x, y, “. r”);

To make the graph look nicer, labels and grid lines can be added as well.

>>xlabel(“x”);

>>ylabel(“y”);

>>grid on;

Figure 1

**Salting**

Now that the function can be plotted to the graph, it can also be salted and plotted to another graph. The process entails looping through the array of x values and adjusting each y value by adding or subtracting a randomly generated value between 0 and 3500. How salted the graph appears depends on the value added or subtracted to the y value. If the range is larger (0-2000), the graph will look less like the original function. A smaller range will do the opposite and the plotted points will better resemble the original function.

First set up a for loop that starts at 1 and loops through the length of the x values.

>>for i = 1:length(x)

end

Next, inside of the loop body set the random adjust value for each point.

>>for i = 1:length(x)

adjust = randi(3500);

end

Then, also inside of the loop body set a conditional to either have the y value at that specific x value either add or subtract by the randomly generated adjust number.

>>for i = 1:length(x)

adjust = randi(3500);

if randi([0, 1]) == 0

y(i) += adjust;

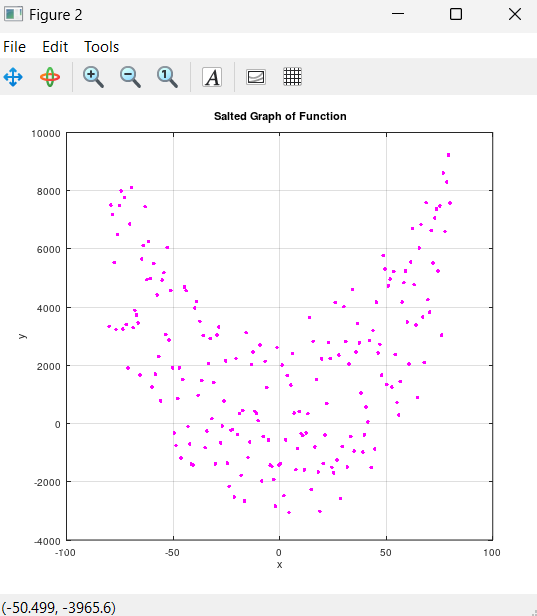
else

y(i) -= adjust;

end

end

The y values have now been salted and can be plotted to a new figure. The steps to graph this are very similar to the original plotting window created previously. The only difference besides the title of the graph is the number of the figure and where the window will be positioned when opened.

>>figure(2,"position", [550,400,550,500]);

>>plot(x, y, ".g");

>>title("Salted Graph of Function");

>>xlabel("x");

>>ylabel("y");

>>grid on;

Figure 2

**Smoothing**

Since the points have been salted, they can be smoothed by taking the average of the current y value and the other y values around it by using a moving window. The process utilizes the “movmean” function by passing in the y values and the set sliding window value.

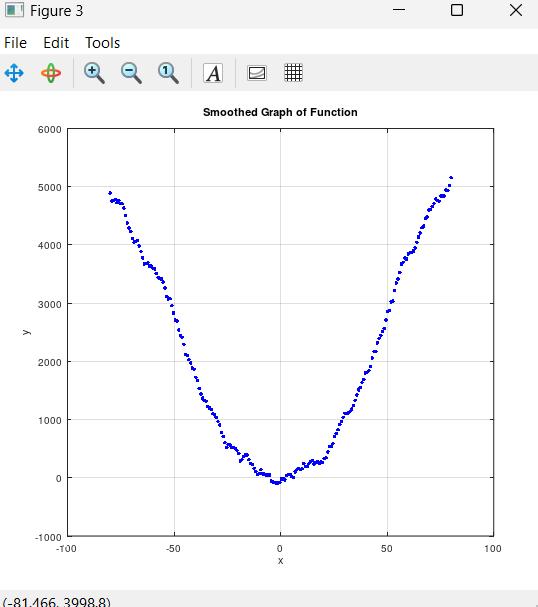
To begin smoothing, first set the window value.

>>windowValue = 60;

Next calculate the moving average of the y points with the windowValue variable and store it in an array called average.

>>average = movmean(y, windowValue);

The average can now be plotted, and the process is very similar to the previous two figures that were created. Again, the only thing that changes is the number of the figure and the coordinates of where the window opens when the program runs.

>>figure(3, “position”, [1100, 400, 550, 500]);

>>plot(x, average, “.b”);

>>title(“Smoothed Graph of Function”);

>>xlabel(“x”);

>>ylabel(“y”);

>>grid on;

Figure 3

References

Basic Octave Tutorial

https://www.youtube.com/watch?v=TqwSlEsbObg

Creating multiple plot windows.

<https://docs.octave.org/v4.4.0/Multiple-Plot-Windows.html>

Setting the size and positions of plot windows.

https://lists.gnu.org/archive/html/help-octave/2009-08/msg00184.html

Generate random numbers.

https://docs.octave.org/v6.4.0/Random-Number-Generation.html

Conditional statements in Octave.

https://docs.octave.org/latest/The-if-Statement.html

Calculating the moving average over a sliding window.

https://docs.octave.org/v6.3.0/Statistics-on-Sliding-Windows-of-Data.html

Linear Space

https://octave.sourceforge.io/octave/function/linspace.html

<https://www.quora.com/How-does-Octaves-linspace-function-work-internally>

Elements used from previous semester's project

https://github.com/roccovulpis/Probability-and-Applied-Stats-Project-2/tree/f54ca2eddf404608e995c81af324c2688d1f2072/PlottingSaltingSmoothing/Octave/Documentation