

Ahmed Malik

When first starting Octave I used these two videos to help me first get a basic understanding of Octave's user interface as well as getting acquainted with the syntax

1. https://www.youtube.com/watch?v=ZOs4egoXPPA&list=PLuR45MKVZJHiQDOdSzGcl7_S0jSOHn8o_&index=2&ab_channel=Mr.STEMEDUTV
2. https://www.youtube.com/watch?v=GjvxquBIPYA&ab_channel=Mr.STEMEDUTV

```
>> 2+3
ans = 5
>> date
ans = 14-Dec-2023
>> help cos
'cos' is a built-in function from the file libinterp/corefcn/mappers.cc

-- Y = cos (X)
    Compute the cosine for each element of X in radians.

    See also: acos, cosd, cosh.

Additional help for built-in functions and operators is
available in the online version of the manual. Use the command
'doc <topic>' to search the manual index.

Help and information about Octave is also available on the WWW
at https://www.octave.org and via the help@octave.org
mailing list.
```

This is an example I followed in one of the videos to get a basic understanding of how Octave works.

It is important to note that this is the command window and not the editor where the code will take place. At first, this did confuse me but after exploring the user interface I quickly learned about the editor as well as the documentation tab at the bottom which contains a table of contents as well as a search bar for all of the functions. This made it much easier to search for exactly what I was looking for,

Exploring the table of contents, this is what I found to help me plot

[15.1 Introduction to Plotting](#)

[15.2 High-Level Plotting](#)

[15.2.1 Two-Dimensional Plots](#)

[15.2.1.1 Axis Configuration](#)

[15.2.1.2 Two-dimensional Function Plotting](#)

[15.2.1.3 Two-dimensional Geometric Shapes](#)

[15.2.2 Three-Dimensional Plots](#)

[15.2.2.1 Aspect Ratio](#)

[15.2.2.2 Three-dimensional Function Plotting](#)

[15.2.2.3 Three-dimensional Geometric Shapes](#)

[15.2.3 Plot Annotations](#)

[15.2.4 Multiple Plots on One Page](#)

[15.2.5 Multiple Plot Windows](#)

Right away, in section 15.2.1, the two-dimensional plots page gave me an understanding of how to plot in Octave, this is what that page states:

15.2.1 Two-Dimensional Plots

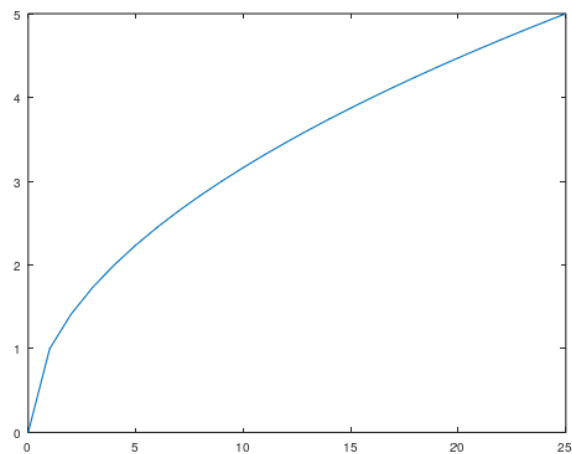
The `plot` function allows you to create simple x-y plots with linear axes. For example,

```
x = -10:0.1:10;  
plot (x, sin (x));  
xlabel ("x");  
ylabel ("sin (x)");  
title ("Simple 2-D Plot");
```

Using the information here and what I know learned from the videos about variables, I came up with the following:

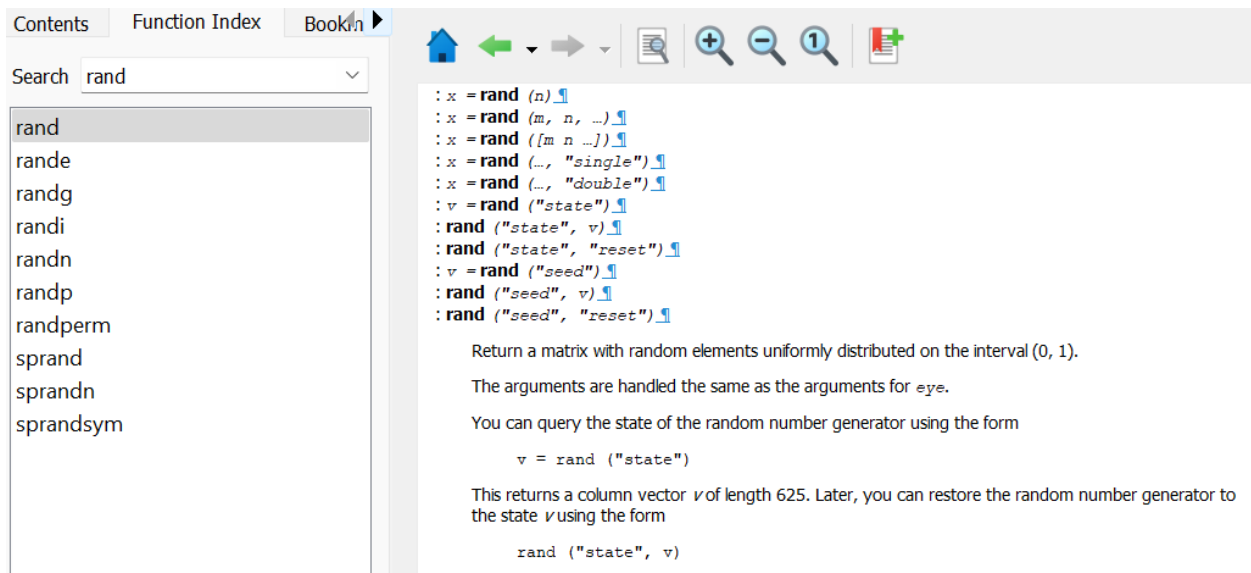
```
x = 0:1:25;  
y = sqrt(x)  
plot (x, y);
```

which gave this graph ->



Now that I had at least one graph, I used section 15.2.5, “Multiple Plot Windows”, to figure out how to generate more than 1 graph separately. This section helped me learn about the “figure()” function which allows us to set up multiple different graphs.

Now knowing this I was ready to start salting, and from the java salter program, I knew that I only needed to know about the “rand” function. I just searched in the function index to learn how to use it



The screenshot shows the MATLAB Function Index window. The search bar contains the text "rand". The left sidebar lists various functions, with "rand" selected. The main pane displays the documentation for the "rand" function, including its syntax and usage examples.

Search: rand

rand
rande
randg
randi
randn
randp
randperm
sprand
sprandn
sprandsym

Syntax:

```
x = rand (n)  
x = rand (m, n, ...)  
x = rand ([m n ...])  
x = rand (... , "single")  
x = rand (... , "double")  
v = rand ("state")  
rand ("state", v)  
rand ("state", "reset")  
v = rand ("seed")  
rand ("seed", v)  
rand ("seed", "reset")
```

Return a matrix with random elements uniformly distributed on the interval (0, 1).

The arguments are handled the same as the arguments for *eye*.

You can query the state of the random number generator using the form

```
v = rand ("state")
```

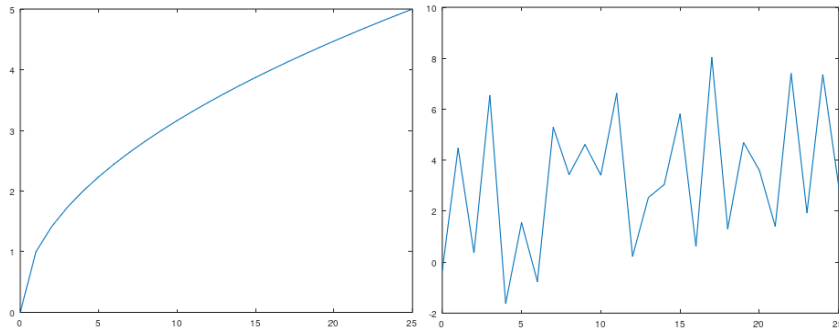
This returns a column vector *v* of length 625. Later, you can restore the random number generator to the state *v* using the form

```
rand ("state", v)
```

After learning about it here, I implemented the figure function to separate the graphs and this is what I came up with:

```
1 figure(1);  
2 x = 0:1:25;  
3 y = sqrt(x)  
4 plot (x, y);  
5  
6 figure(2);  
7 salting_amount = 5.0  
8 salted_y = y + (rand(size(y)) * 2 - 1) * salting_amount;  
9 plot(x, salted_y);
```

This produced these two graphs



Now, the only step left was to smooth it. To figure out how to solve the moving average, I did find a stackoverflow thread that helped me. Shown here:

<https://stackoverflow.com/questions/3114450/octave-time-series-moving-average>

You can use the [FILTER](#) function. An example:

```
t = (0:.001:1)';           %#'
vector = sin(2*pi*t) + 0.2*randn(size(t));   %# time series

wndw = 10;                  %# sliding window size
output1 = filter(ones(wndw,1)/wndw, 1, vector); %# moving average
```

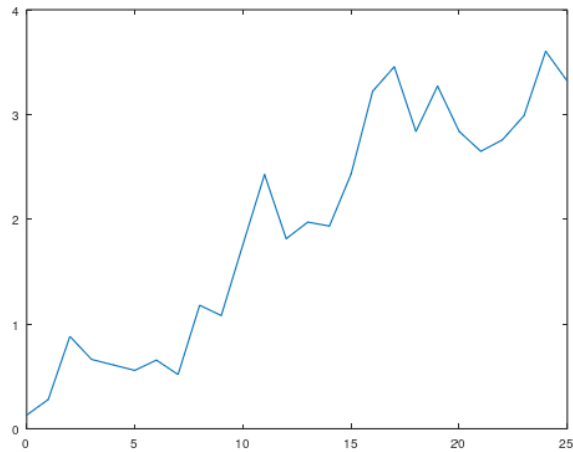
I used the function index again, to help me understand the “filter” function, and the “ones” function.

After a few attempts of playing around with the code, this is what I got

```
11 figure(3)
12 window_value = 10;
13 % Step 1: Create the moving average filter
14 moving_average_filter = ones(1, window_value) / window_value;
15 % Step 2: Apply the filter to the salted data
16 smoothed_y = filter(moving_average_filter, 1, salted_y);
17
18 plot(x, smoothed_y);
```

The `(ones(1, window_value))` represents a simple moving average filter of size `window_value`. Dividing this array by `window_value` normalizes it, making the sum of all elements equal to 1. The `filter` function applies the moving average filter to the `salted_y` data. The 1 in the arguments is the filter's denominator coefficient, which is just 1 in the case of a moving average.

I split the function into two parts so it was easier for me to understand and read, unlike the stackoverflow response that had it in 1 statement. The code gave me this smoothed graph



Concluding, I learned a great deal about Octave including its syntax and its user interface. The plotting, salting, and smoothing was much simpler than the Java implementation because of the optimization for math in octave.

Sources

https://www.youtube.com/watch?v=ZOs4eqoXPPA&list=PLuR45MKVZJHiQDOdSzGcl7_S0jSOHn8o_&index=2&ab_channel=Mr.STEMEDUTV

https://www.youtube.com/watch?v=GjvxquBIPYA&ab_channel=Mr.STEMEDUTV

<https://docs.octave.org/latest/>

<https://stackoverflow.com/questions/3114450/octave-time-series-moving-average>