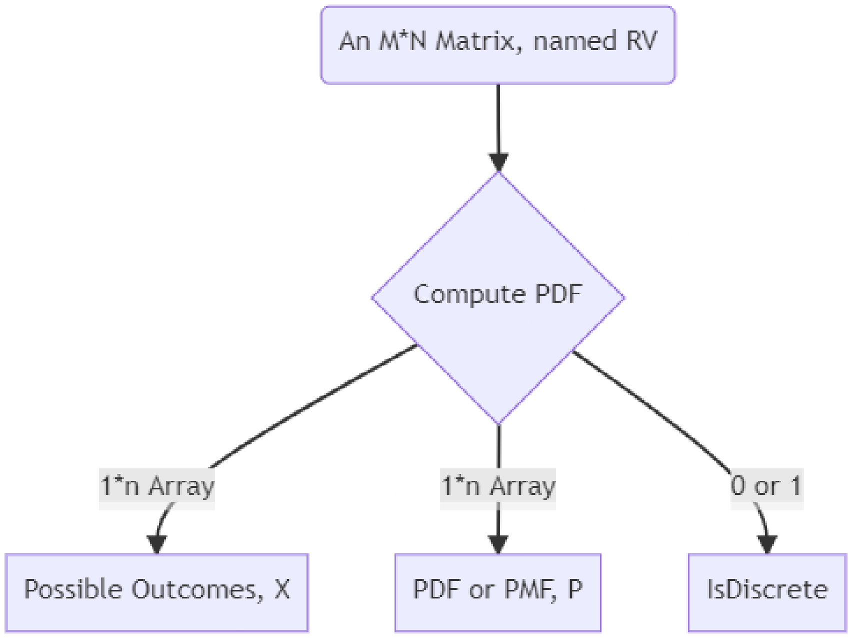
Assignment-2

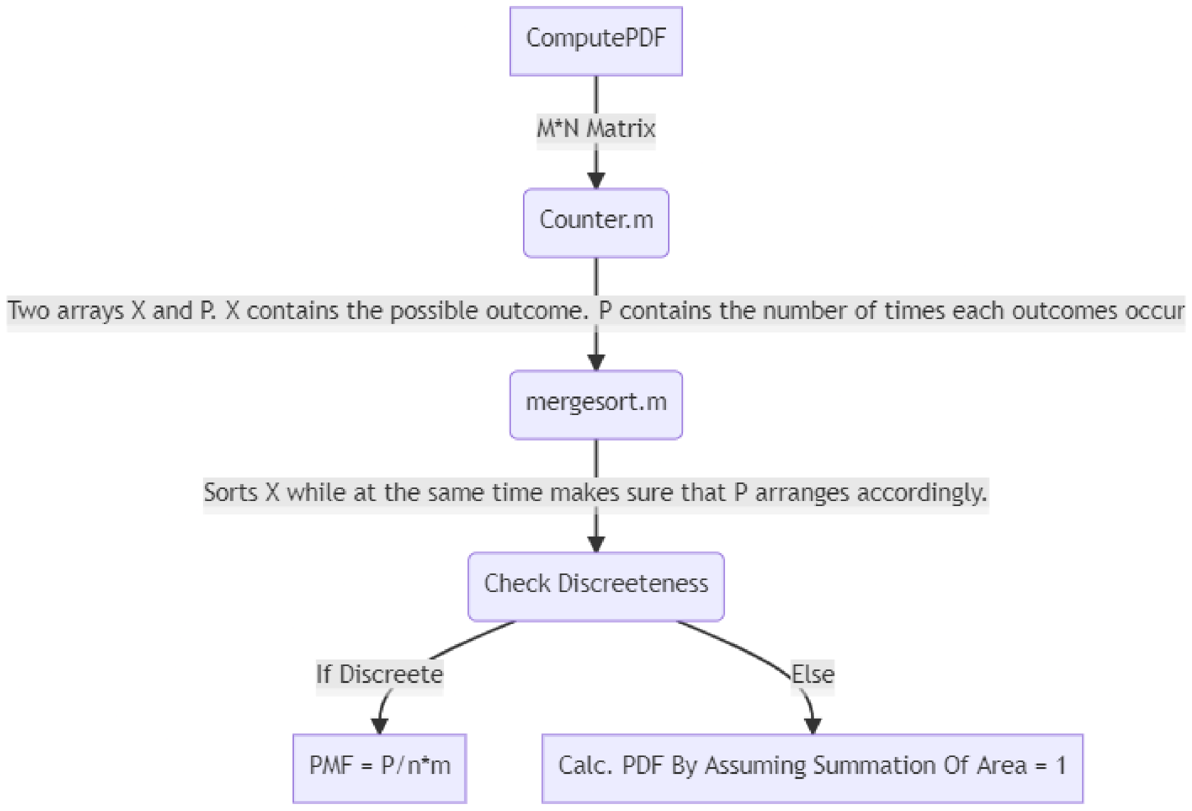
Created	@February 21, 2022 5:05 AM
Course	MA202: Mathematics-IV
Semester	https://www.notion.so/5445d7d64ff64934ac8ed4c15d920db1

What does the ComputePDF function do?



ComputePDF takes an $m * n$ matrix as the input. The values in that matrix is the population of a random variable (Let's say, X). The function gives a 1d array containing all possible outcomes of the random variable X, along with another 1d array representing Probability Mass Function (Or Density Function) of the variable for each of those possible outcomes at the same index. The third output that we get is whether X is discrete or continuous in terms of 1 or 0 respectively.

How does it work?


ComputePDF
M*N Matrix
Counter.m
Two arrays X and P. X contains the possible outcome. P contains the number of times each outcomes occur
mergesort.m
Sorts X while at the same time makes sure that P arranges accordingly.
Check Discreeteness
If Discreete
Else
PMF = P/n*m
Calc. PDF By Assuming Summation Of Area = 1

A Brief diagram of how the ComputePDF.m file works?

Here, It should be noted that files, “Counter.m” and “Mergesort.m” were written by me.

Now let’s look at the logic behind each of the sub-blocks,

Counter.m

Input: The Matrix RV.
Output: X & P (P contains number of times each outcome occured.)
What Do We do:

1. We first create 2 arrays, X&P Of Length m*n. Each of the element of the arrays are zero.
2. Then parse through each element of RV using loops.
    a. Check whether it is already there in X. If yes, then increase the number at that index by 1 in P.
    b. If not, then change element after the last Non-Zero location to the RV’s element. And change the number in P at respective location to 1.
3. P and X are our output arrays. (Returned to the computePDF function)

mergesort.m

Input: X & P (The same arrays that were received by ComputePDF from Counter)
Output: X & P (But X is sorted in ascending order, and P sorted accordingly)

I used the normal general algorithm for Merge sort. The only reason I couldn't use the in-Built function for sorting was that I needed to index the P accordingly.

Checking Discreteness

Suppose we have taken an $M * N$ matrix, and we get K unique outcomes for the random variables X.

- Now, if K is considerably less than $m * n$, then X has to be discrete, because we would have spotted distinct but close outcomes for continous random variables. We can say that those distinct close outcomes collapsed to the closest outcomes for discrete random variables.
- Also, if $m * n$ is considerably less than difference between maximum and minimum values of X, then the variable once again can be assumed to be discrete. This is because due to low resolution of the random variable X.

▼ Rough (The Experimented Space)

First Try

My way to proceed here was that the values of P for discrete variables wouldn't be continuous. So although they will be closely placed, there will be sudden increases and decreases in slope with magnitude being close to +ve and -ve infinity.

- Scale the values first (so that they fit in)
- Check the slope change of each successive interval in the array.
- If the slope change is greater than a particular value (I used a conservative and safe value of 100), then count that slope as discreet, otherwise continuous.
- Get the discreet percentage by counting number of discreet slopes.
- If percentage is greater than a certain value (I used 50), then the variable is discrete, otherwise continuous.

The calculations for that were as follows,

$$\frac{(p_{n+1} - p_n) * average(x)}{(x_{n+1} - x_n) * average(p)} > 100 \implies (p_{n+1} - p_n) * \bar{x} - 100(x_{n+1} - x_n) * \bar{p} > 0$$

$$\left(\begin{bmatrix} p_2 \\ p_3 \\ . \\ . \\ . \\ p_n \end{bmatrix} - \begin{bmatrix} p_1 \\ p_2 \\ . \\ . \\ . \\ p_{n-1} \end{bmatrix} \right) \bar{x} - 100\bar{p} * \left(\begin{bmatrix} x_2 \\ x_3 \\ . \\ . \\ . \\ x_n \end{bmatrix} - \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_{n-1} \end{bmatrix} \right) > 0 = A(\text{assume})$$

If the above represents A. Then, applying unit step function will convert A into a matrix of 1's and 0's. After that, the percentage of non-zero elements in the column vector represents discreteness of the given data-set.

Also,

$$[\bar{x} * \text{sum}(P)] = P'X$$

$$\bar{p} = \frac{\text{sum}(P)}{\text{n}(P)}$$

Second Try

The basic difference between continuous and discrete random variables is that, in discrete random variables, the difference between successive values is a finite multiple of an arbitrary constant. You can't find any such multiple for the continuous random variable. In other words,

$$\text{if we define, } x_{dif} = \begin{bmatrix} x_2 \\ x_3 \\ . \\ . \\ . \\ x_n \end{bmatrix} - \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_{n-1} \end{bmatrix}$$

Then,

- Let's take the GCD of all the elements of column vector x_{dif} and call it K. Dividing the vector by K should yield a finite vector.
- But this approach is a bit hard to implement in matlab. Also, there are some obvious problems with the approach.
- But it leads to one better thing. We can say that standard deviation of

Calculating PMF and PDF

The calculation for PMF is fairly straight forward. You know the freqeuncies, so divide them by summation of frequencies, which in our case happened to be M*N. But for the PDF...

We know that summation of area of the PDF graph needs to be one. We also know, that multiplying P by **some scalar** (assume **k**) will gives us the probability density for resepetive positions. We also have the x-axis divided into roughly uniform sections (sorted X), with P_i known for each.

So I did the following calculations,

$$\text{Let, } pdf = k * P$$

$$\text{Area of 1 Trapezium} = \frac{(x_{n+1} - x_n)(k * p_{n+1} + k * p_n)}{2}$$

$$\text{summation} = k * \sum_{n=1}^{N-1} \frac{(x_{n+1} - x_n)(p_{n+1} + p_n)}{2} = 1$$

$$\Rightarrow k * \sigma = 1$$

▼ Rough

$$\implies k * \left(\sum_{n=2}^{N} x_n p_n - \sum_{n=1}^{N-1} x_n p_n + \sum_{n=1}^{N-1} x_{n+1} p_n - \sum_{n=1}^{N-1} x_n p_{n+1} \right) = 1$$

$$k * \left(x_N (p_N + p_{N-1}) - x_1 (p_1 + p_2) + \sum_{n=2}^{N-1} x_n (p_{n-1} - p_{n+1}) \right) = 1$$

With this we get k. And then we only need to multiply p by k.

Part -2: Testing the Function

Section: 1

Distribution: Discrete Uniform

Range: 1-100

Graphs:

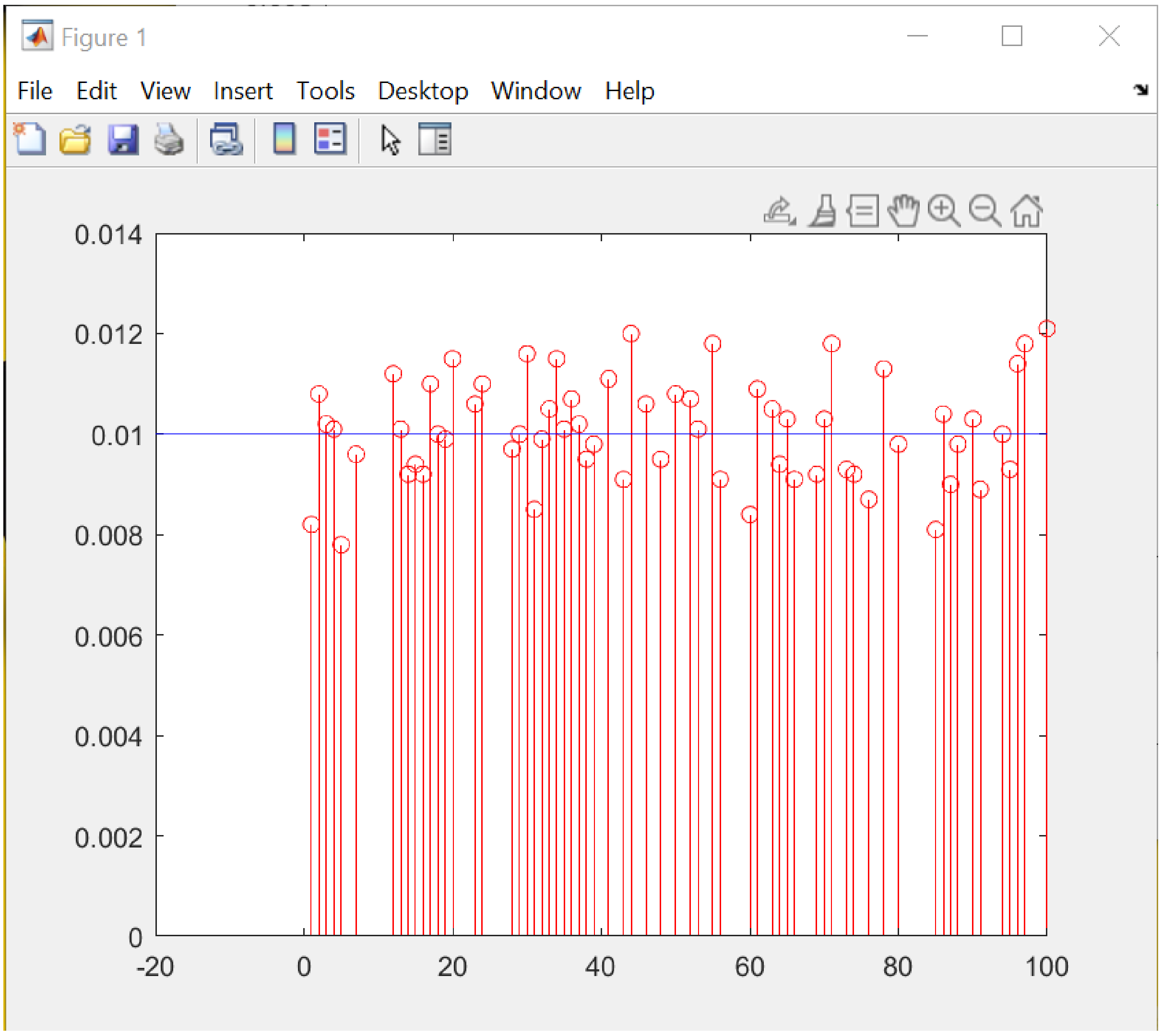
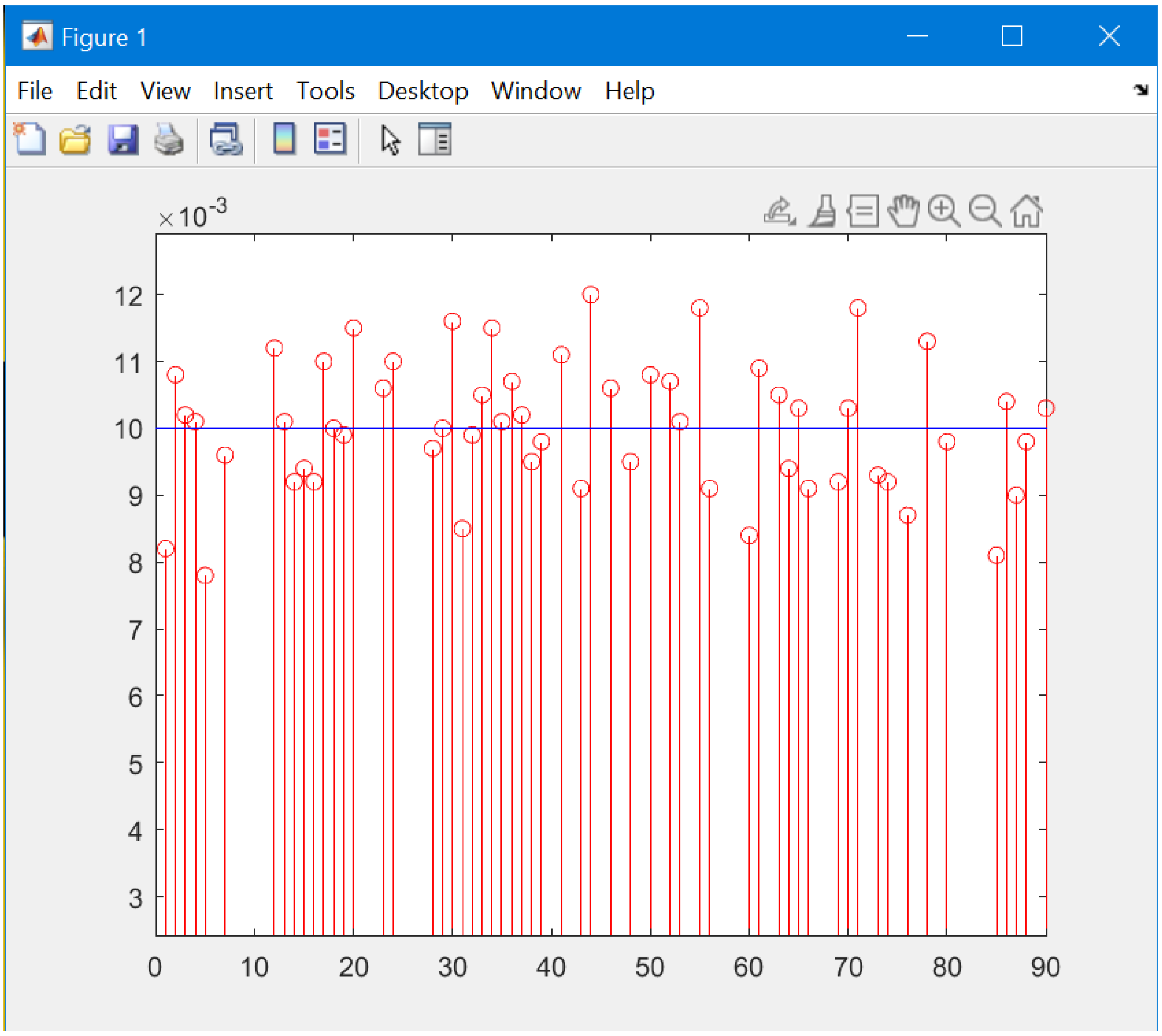


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
-20
0
20
40
60
80
100

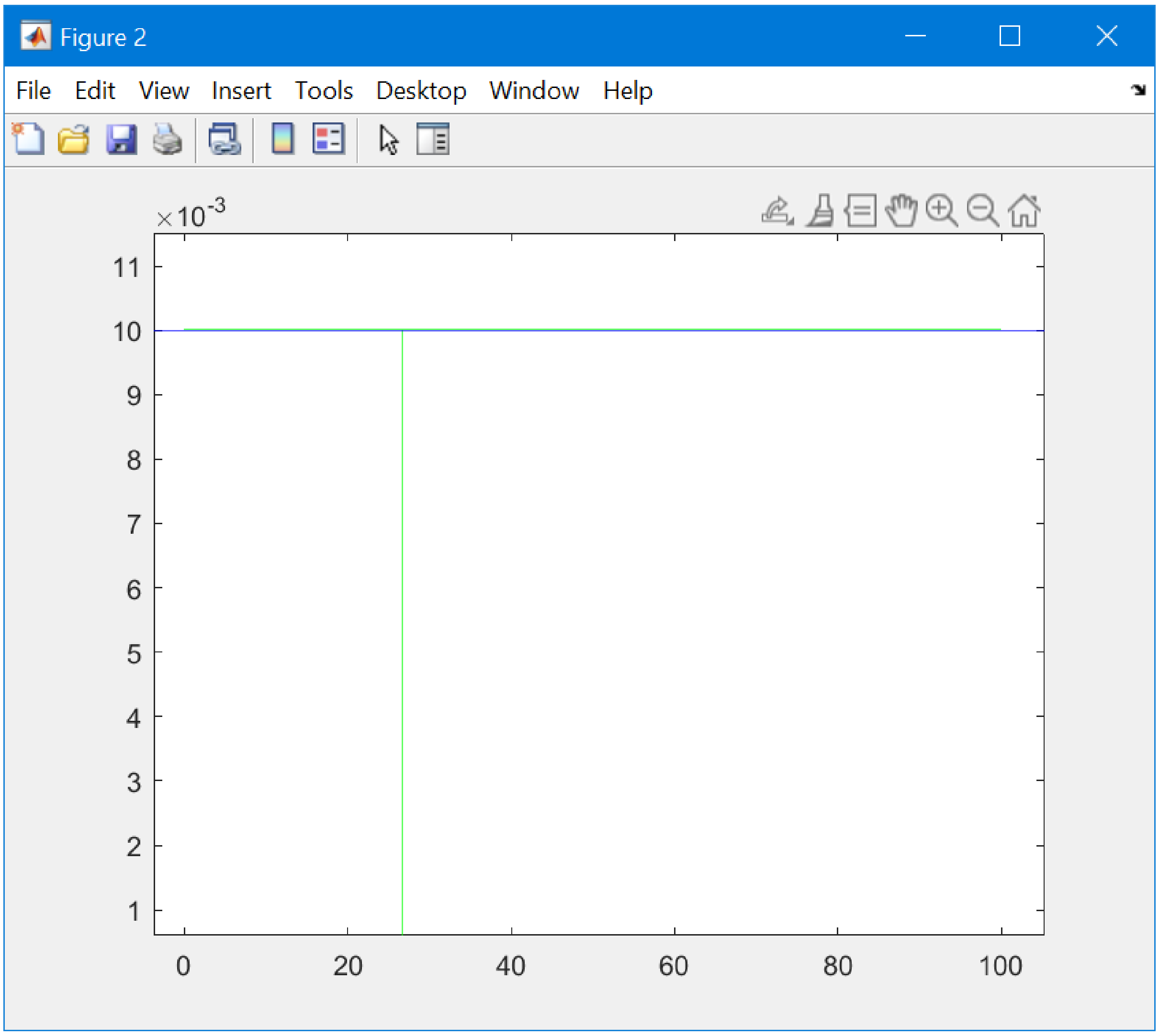


Section: 2

Distribution: Continuous Uniform

Range: 0-100

Graphs: Both data and fplot overlap.



Section: 3

Distribution: Continuous Exponential

Mean: 5

Graphs: Couldn’t do the fplot. The data however succeeds in showing the decrease.

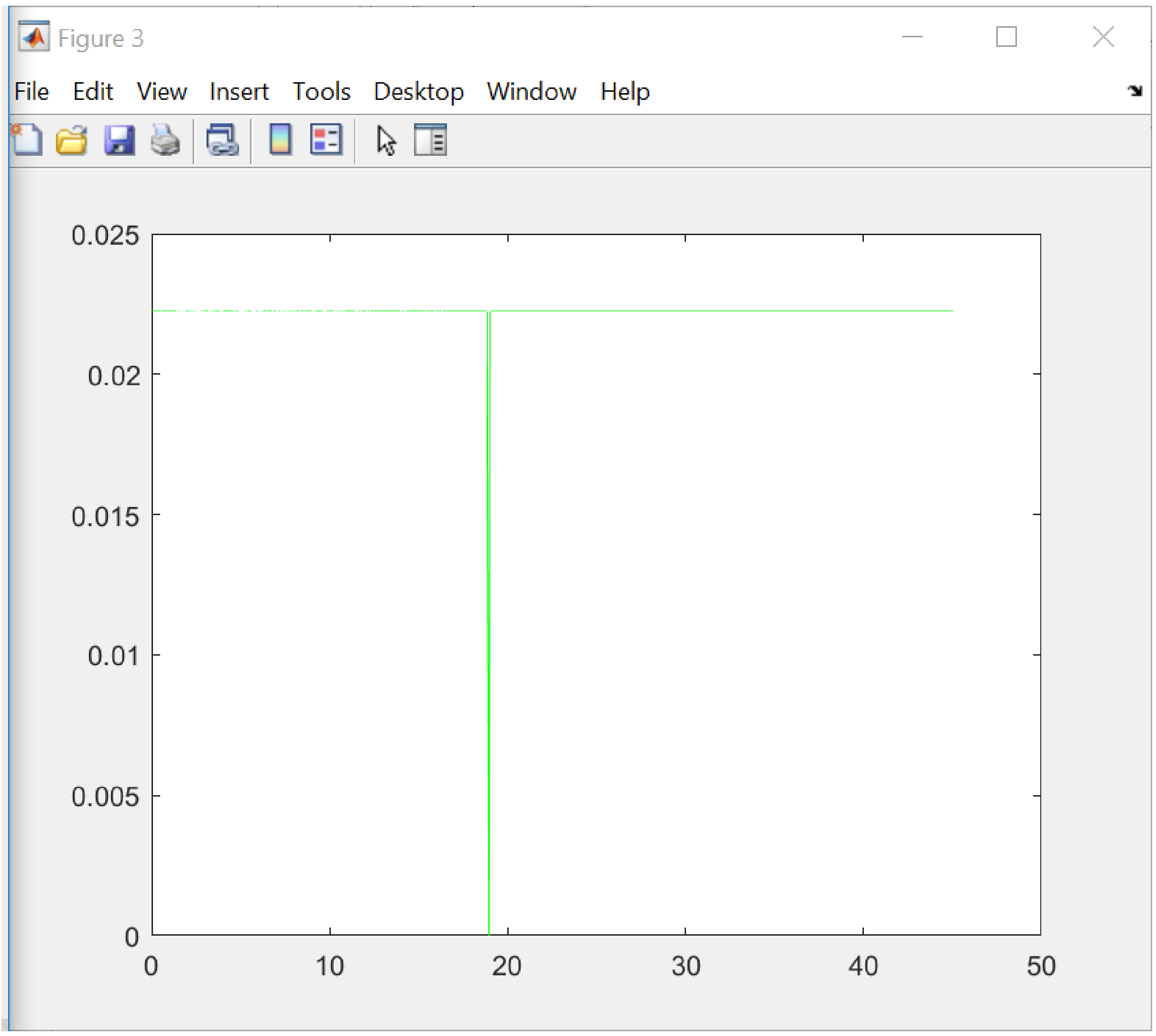


Figure 3
File Edit View Insert Tools Desktop Window Help
0.025
0.02
0.015
0.01
0.005
0
0
10
20
30
40
50