

LEARNING MATLAB WORKSHOP

This guide will show some fundamentals of Matlab using a concrete example all engineering students at McGill are familiar with. Samosas. Do not be intimidated by the length of this document; it is mostly pictures. The purpose of this guide is to teach you just enough to get you on your way to starting your Matlab projects and assignments. This is NOT an in depth look at all the functionality of Matlab and by no means a substitute for reading the help references.

CONTENTS

Before Beginning	3
Reference Materials	3
Matlab Version	4
The beginning	5
Desktop.....	5
Command window.....	6
Clearing the screen.....	6
Strings	7
Calling functions	8
Asking for help.....	8
Flow Control	9
Vectors.....	10
Vector Indexing	10
Vector Indexing Many elements	10
Vector Scaling.....	11
Built In Functions on vectors.....	11
Vector masking.....	11
Chaining Commands.....	13
Functions returning more than one value	13

Matrix manipulation	14
Matrix indexing.....	15
Getting one element	15
Getting a row or column	15
Matrix Scaling	16
Matrix Multiplication	17
Element-wise Multiplication.....	18
Working with scripts.....	19
Script Window	20
Comments	20
Running a script.....	20
2d Plotting	22
Writing our own functions.....	23

BEFORE BEGINNING



Figure 1 <http://riteshjsr.files.wordpress.com/2011/10/samosa1.jpg>

This Tutorial assumes you know the basics of linear algebra, that is, matrix addition and multiplication. If not you can read a book but for this workshop check out the Khan Academy videos as a primer:

<http://www.youtube.com/watch?v=xyAuNHPsq-g&list=PL835D2252574274C5>

REFERENCE MATERIALS

Questions Sheet for workshop:

https://docs.google.com/document/d/1TvTGfKoyYG2jKIIQgucl_VEFreWuv8zpENZ07uAOK3l/edit

Asnwers Sheet for workshop:

https://docs.google.com/spreadsheets/ccc?key=0AkZTslY2c9jwdERydFNPV8zZ0pwb0xqeE5HX1F0anc&usp=drive_web#gid=0

A very good matlab reference can be found on the mathworks website here:

http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf

For examples and libraries checkout the Matlab file exchange.

<http://www.mathworks.com/matlabcentral/fileexchange/>

For some more tutorials and tips checkout:

<http://www.matlabtips.com/>

Shabbir Hussain
COMP Tutor EPTS: epts.comp@mcgilleus.ca

MATLAB VERSION

This tutorial will be using the MATLABR2013a version.

THE BEGINNING

Imagine a very lucrative campus samosa business in a university similar to McGill. To analyze her business, the boss has her accountants use matlab. You have been hired by the business to analyze some experiments and find scrutinize sources of revenue. Consider this your training.

DESKTOP

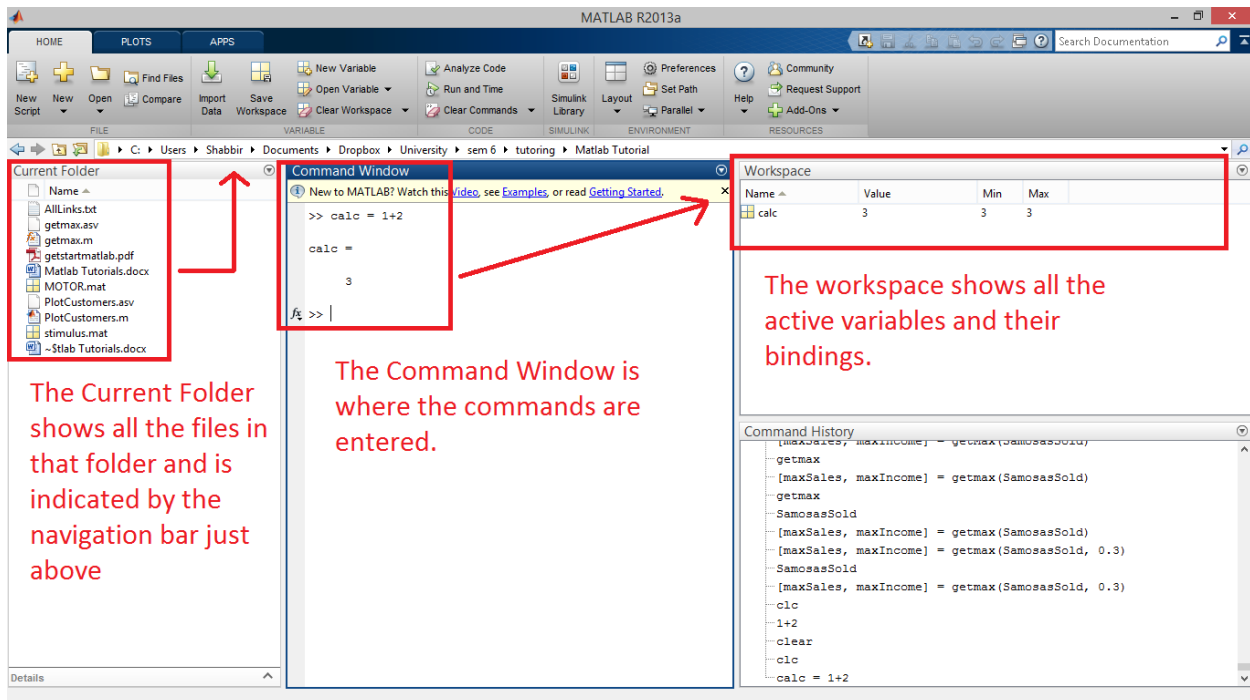


Figure 2 Desktop Window

When first opening Matlab you will see a screen called the desktop screen. This is where all the actions will take place. In the above example we've created a variable called 'x' and put the value 3 inside it.

COMMAND WINDOW

We can use Matlab to do simple calculations. In the following example, we've created a variable called 'SamosasSold' and put the number 150 in it to represent 150 samosas sold. (A variable is simply a placeholder for a number and so from now on the word 'SamosasSold' will represent the number 150)

The Price per samosa is 30¢ to keep track of it we create a variable called Price PerSamosa and put the value 0.30 in it.

To calculate income we simply multiplied the two variables and put the new value inside a variable called 'income'.

```
>> SamosasSold = 150

SamosasSold =

    150

>> PricePerSamosa = 0.30;
>> Income = SamosasSold * PricePerSamosa

Income =

    45

fx >>
```

Figure 3 Doing Simple calculations

CLEARING THE SCREEN

If you want Matlab to forget any variables type

```
fx >> clear
```

If you want your command window to be blank type

```
fx >> clc
```

STRINGS

We can store more than just numbers. We can also store strings.

```
>> DayOfTheWeek = 'Monday'  
  
DayOfTheWeek =  
  
Monday
```

Figure 4 Storing strings

Strings are stored using single quotes.

CALLING FUNCTIONS

Strings by themselves aren't as interesting as numbers in Matlab. However, at the samosa shop we need to keep track of the day of the week since we offer specials on Tuesdays. We can check if it's Tuesday by calling a built in function that compares strings.

```
DayOfTheWeek =  
Monday  
  
>> IsItTuesday = strcmp(DayOfTheWeek, 'Tuesday')  
  
IsItTuesday =  
  
0
```

Figure 5 Comparing days of the week

A function takes in certain inputs and spits out numbers called the outputs in the follow format

Outputs = functionName (inputs)

Later we will see how to make our own functions. In the above example, we called strcmp with the input as the variable 'DayOfTheWeek' and the literal 'Tuesday'. The output is 0 (false) which means that the two inputs are not equal. (The output would be 1 if it were true).

WARNING: This zero is not the number zero it is the logical value false. The distinction is important in some cases.

ASKING FOR HELP

There is a built in function called help which gives information about built in functions when called.

```
>> help strcmp  
strcmp Compare strings.  
TF = strcmp(S1,S2) compares the strings S1 and S2 and returns logical 1  
(true) if they are identical, and returns logical 0 (false) otherwise.  
  
TF = strcmp(S,C), compares string S to each element of cell array C,  
where S is a character vector (or a 1-by-1 cell array) and C is a cell  
array of strings. The function returns TF, a logical array that is the
```

Figure 6 Calling the Help on a built in method

FLOW CONTROL

Sometimes we want to have some decision making within our program. For that we will use flow control.

In our business the price of samosas varies depending on the day. In order to calculate the correct income we need to get the correct price.

```
>> ItIsTuesday = strcmp(DayOfTheWeek, 'Tuesday');  
>> if(ItIsTuesday)  
    PricePerSamosa = 0.1  
else  
    PricePerSamosa = 0.3  
end  
  
PricePerSamosa =  
  
    0.3000
```

Figure 7 Price based on day

Here we've used flow control to set the value of the price of samosas depending on the day. The format flow control statements are as follows:

```
if(condition)  
    do something  
elseif(condition)  
    do something  
else  
    do something  
end  
  
while(condition)  
    do something  
end  
  
for i=1:10  
    do something  
end
```

Figure 8 Format of flow control statements

All the flow control statements must finish with the keyword end. The other two flow control statements 'while' and 'for' are called loops. While loops 'do something' over and over as long as the condition is true. For loops 'do something' over and over for a fixed number of times; in the example the For loop executes 10 times. Loops are not as important in Matlab as they are in other programming languages.

VECTORS

We can do better than just calculating values one at a time. We can use vectors to hold a list of numbers. Let's calculate the income from three days of selling samosas.

```
>> PricePerSamosa = 0.30;  
>> SamosasSold = [ 150 120 89];
```

Figure 9 declaring vectors

The variable 'SamosasSold' now holds a vector where each entry represents the number of samosas sold on that particular day.

VECTOR INDEXING

To see the number of samosas sold on the very first day type in

```
>> SamosasSold = [ 150 120 89];  
>> SamosasSold(1)  
  
ans =  
  
    150  
  
fx >>
```

Figure 10 Vector Indexing

This returns the very first element of the vector.

VECTOR INDEXING MANY ELEMENTS

To see the samosas sold on the second and last day

```
>> SamosasSold = [150 120 89];  
>> Day2to3 = SamosasSold(2:3)  
  
Day2to3 =  
  
    120    89
```

Figure 11 Multiple indexing

By putting a vector as the index we can index a vector and get a sub vector.

VECTOR SCALING

To calculate the profit per day we need to multiply the samosas sold each day by the price.

```
>> IncomePerDay = SamosasSold * PricePerSamosa

IncomePerDay =

    45.0000    36.0000    26.7000
```

Figure 12 Vector scaling

Matlab is smart, it understands vector scaling from linear algebra and does just that in order to calculate income per day.

BUILT IN FUNCTIONS ON VECTORS

To get the total income from all three days we can use a built in function from Matlab

```
>> TotalIncome = sum(IncomePerDay)

TotalIncome =

    107.7000
```

Figure 13 Calling sum

This function called sum adds up all the elements of the vector 'IncomePerDay' and puts the number into 'TotalIncome'.

VECTOR MASKING

Sometimes the employees of the samosa shops get lazy and forget to report the sales numbers. When this happens our income vector looks like.

```
>> IncomePerDay = [45 0 35 0 12 23]

IncomePerDay =

    45     0    35     0    12    23
```

Figure 14 Missing income days

Each entry in the vector represents the income per day except where there is zero; the zeroes are days when the employee at the samosa stand forgot to punch in the sales number.

If we try finding the average income per day with this vector we get:

```
>> AverageIncomePerDay = mean(IncomePerDay)

AverageIncomePerDay =

    19.1667
```

Figure 15 Getting the wrong average

Logically we know this result is wrong because those zeroes are biasing the results. We need to create a mask.

```
IncomePerDay =

    45     0    35     0    12    23

>> mask = IncomePerDay>0

mask =

     1     0     1     0     1     1
```

Figure 16 Creating a vector mask

This mask is a vector where each value is set to 1 (or true) when the income is greater than zero. We can use this mask to get all the non-zero values of Income as follows:

```
>> MaskedIncome = IncomePerDay(mask)

MaskedIncome =

    45    35    12    23
```

Figure 17 masking away zeroes

Using the mask we have indexed the original Income vector and got a new vector with only the non-zero income values. We can now compute the actual Average Income of samosa sales.

```
>> mean(MaskedIncome)

ans =

    28.7500
```

Figure 18 Actual Income Per day

CHAINING COMMANDS

We could have also chosen to streamline the whole process of masking and finding the mean in one single line.

```
>> mean(IncomePerDay(IncomePerDay>0))  
  
ans =  
  
28.7500
```

Figure 19 Chaining Commands into one line

The above line is equivalent of the three step process to mask values. Neither way is better. This example serves to illustrate all the possibilities.

FUNCTIONS RETURNING MORE THAN ONE VALUE

Sometimes a function can return more than one value. We can have a vector on the left hand side to hold all the different values.

```
>> SamosasSold = [150 120 89];  
>> [maxSamosasSold, DayWhenMaxSamosasSold] = max(SamosasSold)  
  
maxSamosasSold =  
  
150  
  
DayWhenMaxSamosasSold =  
  
1
```

Figure 20 getting more than one value

MATRIX MANIPULATION

Matrices allow us to manipulate many numbers at a time. In fact, vectors are just a special case of matrices.

The samosa business has three stands across campus and we need to be able to keep track of all the sales across each stand.

```
>> SamosasSold = [ 125 150 120; 121 150 143; 119 158 133]

SamosasSold =

    125    150    120
    121    150    143
    119    158    133
```

Figure 21 Matrix of samosa sales

Here we've created a matrix where the first column represents the sales from kiosk 1, the second from kiosk 2 and the third from kiosk 3. Each row represents the sales on day 1, day 2 and day 3 respectively.

	Kiosk 1 sales	Kiosk 2 sales	Kiosk 3 sales
Day 1	125	150	120
Day 2	121	150	143
Day 3	119	158	133

You may think of the value inside a matrix like those inside an excel spreadsheet.

MATRIX INDEXING

GETTING ONE ELEMENT

We now need to access the sales on day 2 at kiosk 3. We can do so using the following syntax:

```
>> RowNumber = 2; ColumnNumber=3;
>> SamosasSold(RowNumber,ColumnNumber)

ans =

    143
```

Figure 22 indexing a matrix

GETTING A ROW OR COLUMN

To get an entire row or column we place a colon on that index. Here we are trying to find all the sales of kiosk 1, the result is a **Column Vector**

```
>> Kiosk1Column=1;
>> SamosasSoldByKiosk1 = SamosasSold(:,Kiosk1Column)

SamosasSoldByKiosk1 =

    125
    121
    119
```

Figure 23 Indexing a Column

Similarly if we wanted to get all the sales from day 3 we would write the following:

```
>> Day3Row=3;
>> SamosasSoldOnDay3 = SamosasSold(Day3Row,:)

SamosasSoldOnDay3 =

    119    158    133
```

Figure 24 Indexing a row

The result is a **Row Vector**.

MATRIX SCALING

We can get the Revenue on each day by each kiosk if we multiplied the sales by the price of samosas.

```
>> PricePerSamosa = 0.3;  
>> IncomeMatrix = SamosasSold * PricePerSamosa  
  
IncomeMatrix =  
  
    37.5000    45.0000    36.0000  
    36.3000    45.0000    42.9000  
    35.7000    47.4000    39.9000
```

Figure 25 Income Matrix

We've multiplied all the values in our matrix by a scalar to get a new matrix which shows the income on each day by each kiosk. (Matlab knows that a scalar times a matrix is matrix scaling).

MATRIX MULTIPLICATION

The price at kiosk 2 is not the same as the other kiosks. We need to use a matrix multiplication to find the Income per day.

```
>> PricePerSamosa = 0.3;PriceAtKiosk2=0.2;  
>> PriceMatrix=[PricePerSamosa; PriceAtKiosk2; PricePerSamosa]  
  
PriceMatrix =  
  
    0.3000  
    0.2000  
    0.3000  
  
>> IncomePerDay= SamosasSold * PriceMatrix  
  
IncomePerDay =  
  
    103.5000  
    109.2000  
    107.2000
```

Figure 26 Matrix Multiplication

We've created a Column Vector (3x1 matrix) and used matrix multiplication to get the income Per day.

WARNING: Dimensions must match just like in Linear algebra.

ELEMENT-WISE MULTIPLICATION

Recall that there is a samosa sale on Tuesdays (day 2). We want to have each element in the matrix multiplied by a unique number.

```
>> Reg=0.3;Kiosk2=0.2;Tues=0.1;
>> Row1= [Reg Kiosk2 Reg];
>> Row2 =[Tues Tues Tues];
>> Row3 = Row1;
>> PriceMatrix = [Row1; Row2; Row3]

PriceMatrix =

    0.3000    0.2000    0.3000
    0.1000    0.1000    0.1000
    0.3000    0.2000    0.3000

>> IncomeMatrix = SamosasSold .* PriceMatrix

IncomeMatrix =

    37.5000    30.0000    36.0000
    12.1000    15.0000    14.3000
    35.7000    31.6000    39.9000
```

Figure 27 Element wise Multiplication

Like in matrix addition where addition takes places element by element this is Element wise multiplication where each element in the 'SamosasSold' Matrix is multiplied to each element in the 'PriceMatrix'. This is done through the special operator '.*'. The dot is what makes it element wise multiplication rather than matrix multiplication.

WARNING: This is not the matrix multiplication from linear algebra

WORKING WITH SCRIPTS

If we want to keep code for reuse we need to put it in a script. Open a new script.

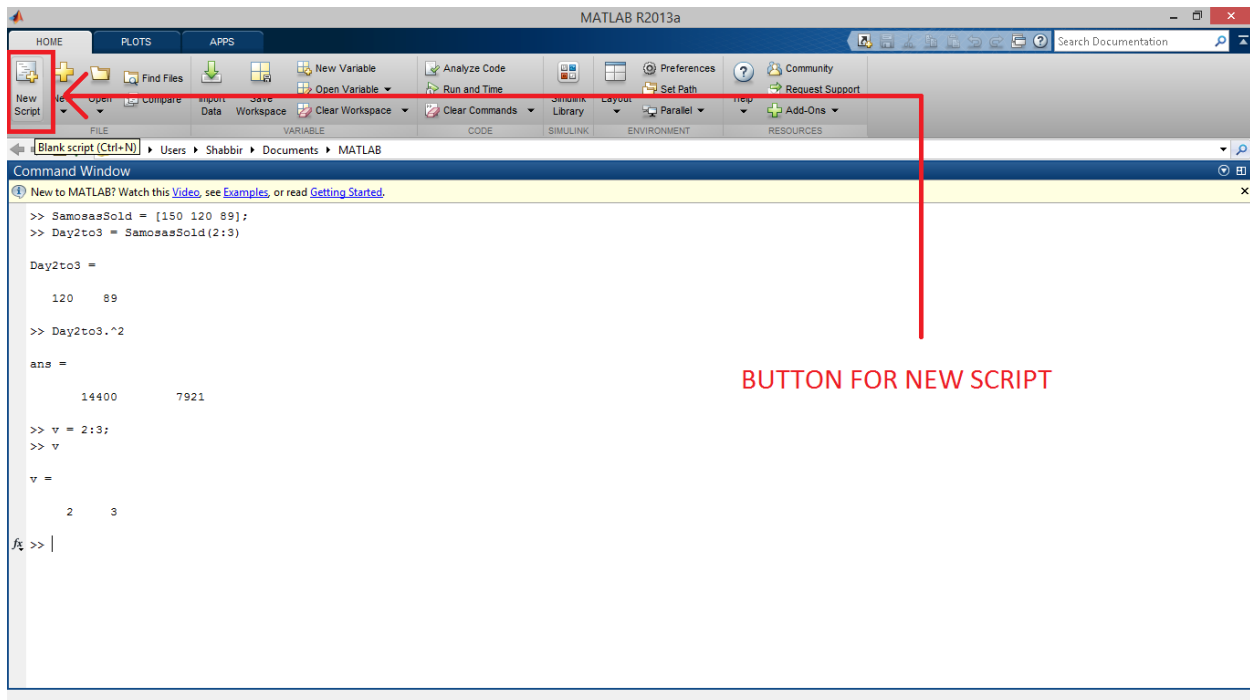


Figure 28 new script Navigation

SCRIPT WINDOW

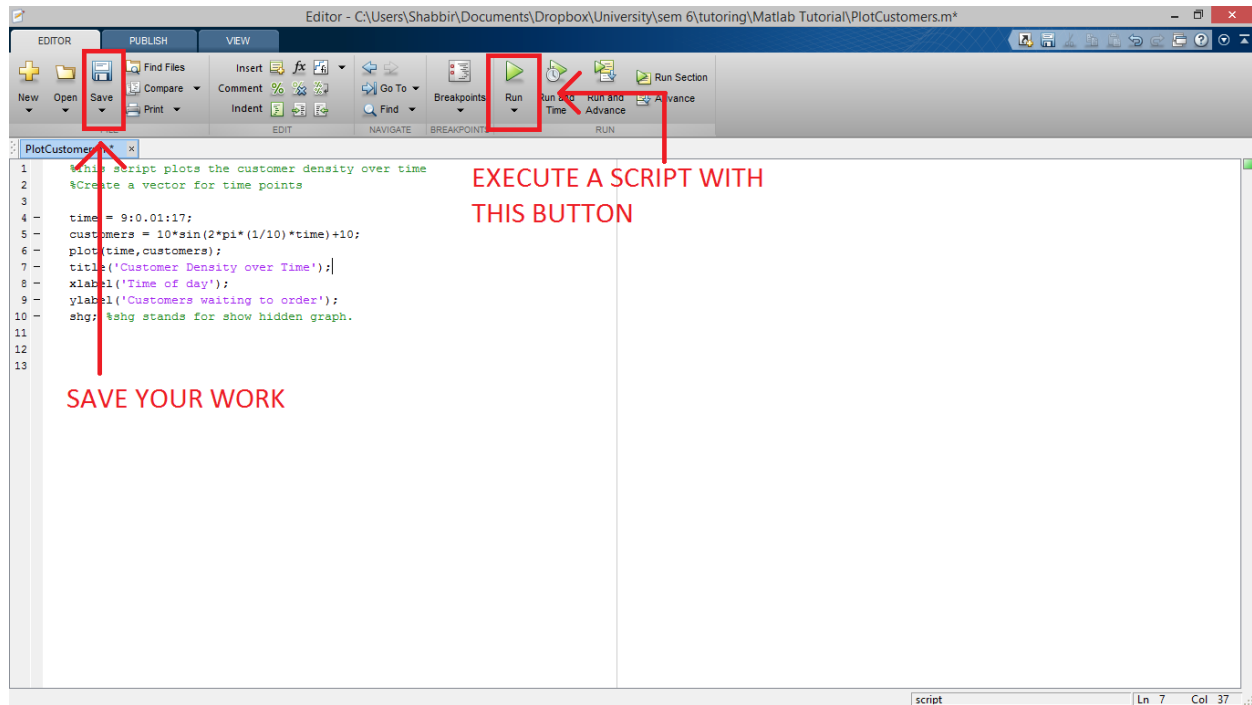


Figure 29 Script window

The following window is used for writing scripts. Scripts don't execute right away like commands. We use scripts to write lots of commands and then execute them at a later time.

COMMENTS

Comments are like commands that do nothing. A comment starts with the percent symbol.

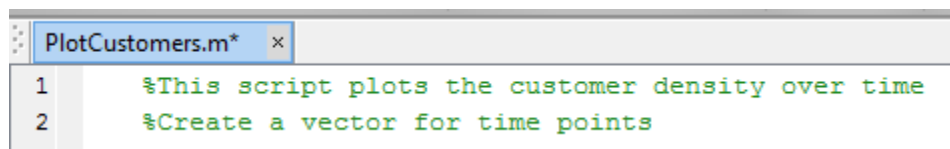


Figure 30 Comments on a script

Add comments to scripts to keep track of your thoughts.

RUNNING A SCRIPT

You may get some alert messages when running your scripts. If you get the following prompt:

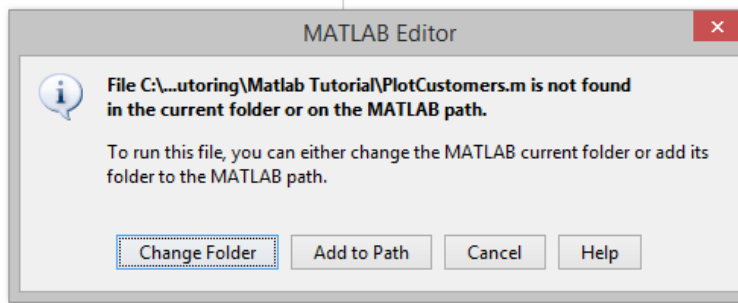


Figure 31 Change folder prompt

Choose the Change Folder option to continue

2D PLOTTING

Research has showed us that the number of customers waiting in line at a kiosk at any given time can be modeled by a sine wave.

```
PlotCustomers.m ×
1 %This script plots the customer density over time
2 %Create a vector for time points
3
4 time = 9:0.01:17;
5 customers = 10*sin(2*pi*(1/10)*time)+10;
6 plot(time,customers);
7 title('Customer Density over Time');
8 xlabel('Time of day');
9 ylabel('Customers waiting to order');
10 shg; %shg stands for show hidden graph.
11
```

Figure 32 plotting

Use the plot function to plot 2d functions.

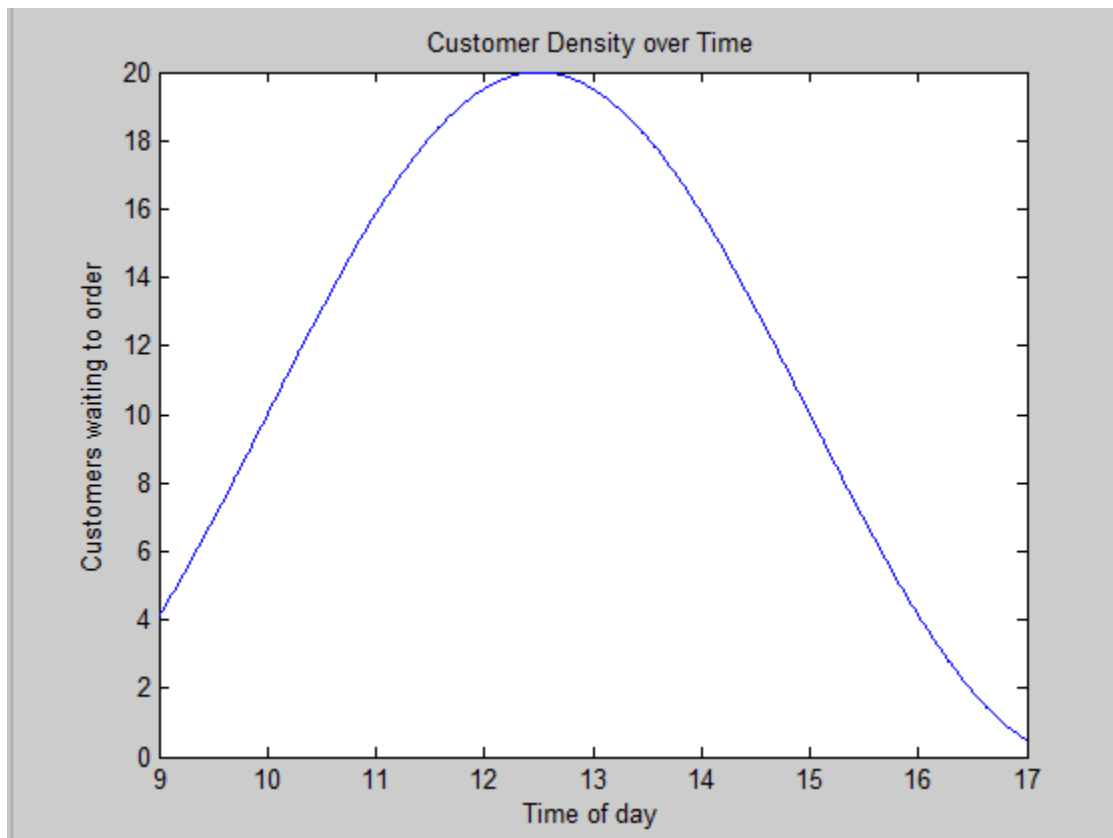


Figure 33 plot of customers waiting

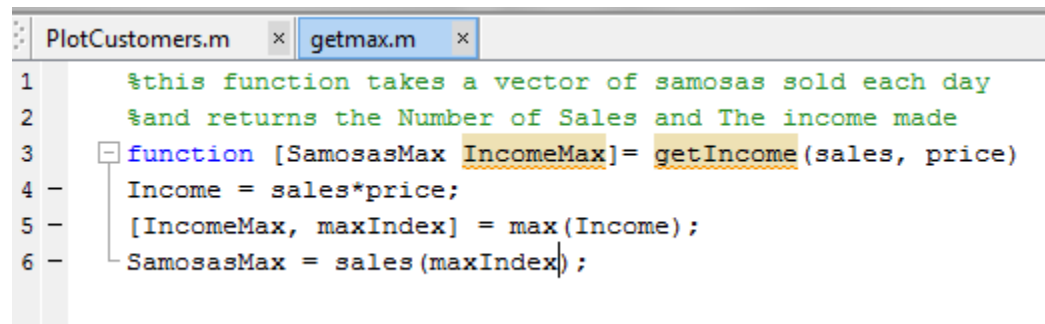
WRITING OUR OWN FUNCTIONS

Each program can be divided into independent tasks. Using functions allows us to breakdown the code into logical segments and also reuse code for tasks that are done many times.

A Function has the following syntax:

```
function [output_parameter_list] = function_name(input_parameter_list)
```

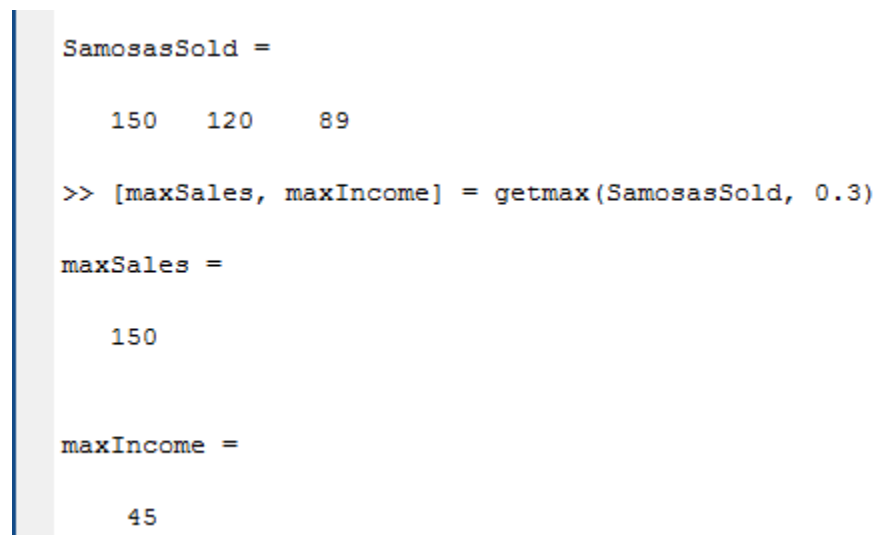
We must write functions inside scripts.

The image shows a MATLAB script editor window with two tabs: 'PlotCustomers.m' and 'getmax.m'. The 'getmax.m' tab is active, displaying a function definition. The function is named 'getmax' and takes two inputs, 'sales' and 'price'. It calculates the income for each sales entry, finds the maximum income, and returns the corresponding sales value and the maximum income. The code is as follows:

```
1 %this function takes a vector of samosas sold each day
2 %and returns the Number of Sales and The income made
3 function [SamosasMax IncomeMax]= getIncome(sales, price)
4 - Income = sales*price;
5 - [IncomeMax, maxIndex] = max(Income);
6 - SamosasMax = sales(maxIndex);
```

Figure 34 script to find max

As long as the file is in your Current Folder (see Desktop section) then the command window will recognize your function.

The image shows a MATLAB command window. It starts by defining a vector 'SamosasSold' with values 150, 120, and 89. Then, it calls the 'getmax' function with 'SamosasSold' and a price of 0.3. The output shows 'maxSales' as 150 and 'maxIncome' as 45.

```
SamosasSold =
    150    120     89

>> [maxSales, maxIncome] = getmax(SamosasSold, 0.3)

maxSales =
    150

maxIncome =
     45
```

Figure 35 Script being used