MATH 1070, Introduction to Computational Science

Lab Session 2 – MATLAB programming

Getting started

This lab runs on the PCs in the CLC, under the M/S Windows operating systems. If you are not already familiar with: finding and opening files, launching applications, etc. in windows; then you should take a few minutes to familiarize yourself with the computers in the lab – and feel free to ask for help.

Next, you should launch MATLAB, from the START -> APPLICATIONS menu on the lower LHS of the screen. Look for the MATLAB HELP menu, and browse around it, to get a feel for how its structured. This will be a key resource for you. Also look for, and use, the document file "SimpleMatLabCmds" on the MATH1070 web site.

Today's lab

In Lab 1, you learned how to get started with MATLAB. In this tutorial you will attempt some simple calculations in MATLAB.

In order to successfully solve a problem in MATLAB (or any other programming language for that matter), you need to:

Learn the **exact** rules for writing MATLAB statements;
Develop a logical plan of attack to solve the problem at hand.

In the first lab we used the MATLAB command window to enter a few MATLAB statements. However this is not efficient for writing longer programs (more than 15 lines). MATLAB provides a text editor to write all the statements forming your code in one file: this code is called an **m file**. All you need then is to execute this file: each statement in the m file will be read and executed in turn in a similar way to what you did when you used the MATLAB command window.
By the end of this laboratory tutorial you should be comfortable with the following:

- writing and using ".m" files;
- writing Loops to perform repetitive calculations in MATLAB;
  - using the "*for*" command; "*while*" command;
- using conditional test statements such as "if"; and
- basic plotting of numerical data in 2-D.

Try the following exercises.

1. Creating a simple MATLAB program

You have $1000 dollars in the bank. Interest is compounded at the rate of 9% per year. How much money do you have in the bank after a year?

a). This is a simple problem: a1) you need to think about how the concept of "interest" works and/or recall the formula for compound interest; and a2) think about the structure of the program before writing code.

b). A possible structure for your program could be:

Get the data (principal and interest)

Calculate the interest

Add the interest to principal

Display the new balance

Let's write the program! From the MATLAB desktop select **File** -> **New** -> **M-file** or click on the open new document icon (new page icon). A text editor will appear: you can now write MATLAB statements and save them in the file. Enter the following statements:

```
balance = 1000;
rate = 0.09;
interest = rate * balance;
balance = balance + interest;
disp('New balance: ');
disp(balance);
```

Select **File** -> **Save** from the menu bar or click on the Save icon (floppy disk icon): a window appears, select a directory (you might need to create a new Dir for this class, to keep your work organized) and enter the file name: **intcalc.m**. The filename **must** have the extension **.m**. The editor window has the name **intcalc.m**. This MATLAB program with the .m extension is called a **script** file and more commonly an **m-file**.

To run your program you can either:

In the editor window, open the **intcalc.m** file if it not open; select **Debug** -> **Run** from the menu bar

In the command window, type the name of the script without the .m extension: **`intcalc`**.

NB: when you run a script MATLAB may complain that it cannot find the script. It will offer you the option to change the MATLAB current directory or to add the directory to the top or bottom of the path. There are several ways you can deal with this, but for the time being click on Change the MATLAB current directory.

If you want to list the contents of the **intcalc.m** file in the command window, type:

```
type intcalc
```

Congratulations. You now have coded and run your first MATLAB program.

c). Spend some time understanding how the program works.

When you run your script, MATLAB read the first statement and translates it into a language that your CPU understands then the instruction is immediately carried out. Then MATLAB moves to the next instruction and so on. MATLAB is called an **interpreted** language; this is different to a **compiled** language. Interpreted languages include **java**. Compiled languages include **FORTRAN**, **C** and **C++**. Needless to see that these interpreted statements are carried in order, from the top down. For this reason and others, interpreted languages are less efficient than compiled.

The statements in the script are interpreted as follows:

Put the number **`1000`** into the variable **`balance`**
Put the number **`0.09`** into the variable **`rate`**
Multiply the content of **`rate`** with the content of **`balance`** and put the answer into the variable **`interest`**
Add the contents of **`balance`** to the contents of **`interest`** and put the answer into **`balance`**
Display in the command window the message given in single quote
Display the contents of **`balance`** in the command window

After running, the variables used have the following values:

`balance`: 1090
`rate`: 0.09
`interest`: 90

d). Run the program as it stands. Now change the first statement in the program to read:

```
balance = 2000:
```

Do you understand what happens?

e). Leave out the line:

```
balance = balance + interest;
```

and re-run the program. Do you understand what happens?

f). Rewrite the program so that the original value of **`balance`** is not lost.

Exercise 2

Write two script files (one for each question below) to do the following.

a) Sum all the odd numbers from 1 to 100.
b) Sum all the squares from 1 to 20^2.

Exercise 3

The purpose of this exercise is to show you that you should not use a **`for`** loop if you can "vectorise" the calculation! You will evaluate $\sum_{n=1}^{100000} n$ by using a **`for`** loop and then by "vectorising" the code using the in-built function **`sum,`** which adds up all the elements of its array (or vector) argument. To do this you first need to load an array (and choose a name) with the 100,000 integers that you plan to sum. You can time taken to do the calculation in each case with the functions, **`clock`** and **`etime,`** to compare the efficiency of each method.

a) **`for`** loop

```
t0=clock;
s=0;
for …… % create your loop here

etime(clock,t0)
```

b) vectorising

```
t0=clock;
s=0;
……   % load your vector here
etime(clock,t0)
```

Also try the **tic .. toc** commands in MATLAB.

Exercise 4

a) work out by hand what is the output of the following script when n = 4.

```
n=input('enter n: ')
s=0;
for k=1:n
        s=s+1/(k^2);
end
disp(sqrt(6*s))
```

b) rewrite the script using vectors and array operations.

Exercise 5

The purpose of this exercise is to get you familiar with **if**, which performs a logical test before doing the calculation. You will use the **rand** function to calculate a random number in the range, 0-1.

Type the commands:

```
r = rand
if r > 0.5 disp(‘  it is greater than ½ ’), end
```

Cut and paste the code fragment to execute it a few times, to see what happens. How else would you do this more efficiently? Note that each **if** is balance by an **end**. Look in the HELP menu for more examples.

Recall the solution of a quadratic equation – you might want to test the argument (>0) prior to taking the square root …. to avoid errors, or “software exceptions” which can “hang” the program.

If you run out of time, the next 2 exercises can be done in Lab 3.

Exercise 6

Now you will try combined **if** .. **elseif** constructions. Which combine several logical tests. Evaluate the given code fragments for each of the cases indicated and use MATLAB to check your answers.

a)
```
if T < 30
        h = 2*T + 1
elseif T < 10
        h = T - 2
else
        h = 0
end
```

i. T = 50 h = ?
ii. T = 15 h = ?
iii. T = 0 h = ?

b)
```
if 0 < x < 10
        y = 4*x
elseif 10 < x < 40
        y = 10*x
else
        y = 500
end
```

i. x = -1 y = ?
ii. x = 5 y = ?
iii. x = 30 y = ?
iv. x = 100 y = ?

A related construct is **while** (eg. a > b) – look this up in the MATLAB HELP menu.

Exercise 7

The electricity accounts of residents in a small town are calculated as follows:

- If 500 units (of energy [eg. kW-hr]) or less are used the cost is 2 cents per unit;
- If more than 500, but not more than 1000 units are used, the cost is $10 for the first 500 units, and then 5 cents for every unit in excess of 500;
- if more than 1000 units are used, the cost is $35 for the first 1000 units, plus 10 cents for every unit in excess of 1000;
- a basic services fee of $5 is charged, no matter how much electricity is used.

Write a program which enters the following five consumption cases into a vector, and uses a **`for`** loop to calculate and display the total charge for each case of: 200, 500, 700, 1000 and 1500 units.