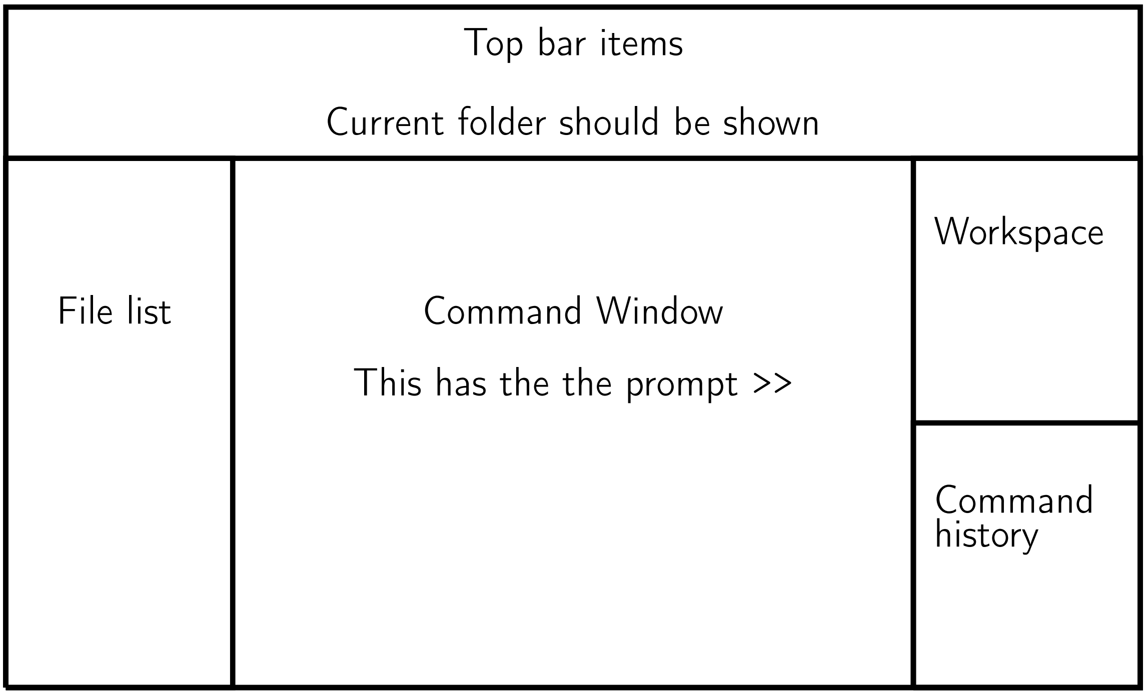
# MA1710: Week 2

# `for`, `if`, `else`, `break`

## Getting started in session 2

With the Windows 7 operating system that we have in our Labs start Matlab as you did last week by clicking the mouse pointer on `Start` (bottom left hand corner), typing `Matlab` in the search box and selecting `MATLAB` from the options offered. When this is complete the layout should correspond to the following.



1. If the version is updated or you have your own newer version then the default may be slightly different, e.g. more items in the top bar and possibly the command history not

being shown, but it will at least approximately conform to what is shown above. If you have changed from the default layout and you wish to return to the default layout then with our versions click on `Desktop` in the top bar part and select `Default`.

2. In the list of the files shown in the panel on the left do you see the names of files that you created in earlier sessions? If the answer is yes then you are ready for the next section.

   If not then try typing the following in the command window.

   ```
   cd h:\Matlab_Level1
   ```

   If that fails because you do not have such a folder then type

   ```
   mkdir h:\Matlab_Level1
   cd h:\Matlab_Level1
   ```

   If you still have problems then ask for help and I suggest that you also study the material for session 1.

## 2.1 Introductory comments

In the first session you started to use Matlab, you converted some mathematical expressions into one or more Matlab statements and you used the Matlab editor so that you could create, store and run a group of statements. You get more practice with all of these in this session and you are also introduced to `if`, `if–else` and `for`–loops. The syntax of these is properly given in subsection 2.3 but first we would like you to try a few short examples. In each case you should create a file which contains the lines listed and run the command by typing something in the command window (this is the window containing the prompt >>).

## 2.2 Experimenting with `for`–loops

At the prompt >> type

```
edit sess2a.m
```

In the edit window enter the following 3 lines.

```
for k=1:5
  disp('What does this do?')
end
```

Save the file (you can click on the diskette symbol to do this). One way of executing the commands is to type the following at the command prompt.

```
sess2a
```

Note that the file is saved with a name ending in `.m` and in the command window the command to type has the `.m` removed. If you have done this correctly then what you should see in the command window is the following.

```
What does this do?
What does this do?
What does this do?
What does this do?
What does this do?
```

Now change `k` in the first line so that the file is as follows.

```
for j=1:5
  disp('What does this do?')
end
```

Save this version as `sess2b.m` and run again. Further change so that the file contains the following and save as `sess2c.m`.

```
for i=2015:-1:2011
  disp('What does this do?')
end
```

You run `sess2b.m` by typing `sess2b` at the command prompt and you run `sess2c.m` by typing `sess2c` at the command prompt. All 3 versions give the same output.

Next create `sess2d.m` to contain the following.

```
for k=1:5
  fprintf('k=%d\n', k)
end
```

Type `sess2d` at the command prompt which should generate the following.

```
k=1
k=2
k=3
k=4
k=5
```

The use of the command `fprintf` here is to have more control on the output. The string `'k=%d\n'` contains some things that are displayed as given and somethings which are formatting instructions for the quantities listed after the string. In this example `%d` is the formatting instruction for the display of `k` and it is for displaying the value of `k` using decimal digits. The part `\n` is the new line character. Other examples in this session use `fprintf` to neatly format the output. Edit the file to generate

```
for j=1:5
  fprintf('j=%d\n', j)
end
```

Save this as sess2e.m and run the commands. Further edit to create the following and save as sess2f.m and run the commands.

```
for i=2015:-1:2011
  fprintf('i=%d\n', i)
end
```

The outputs from sess2d, sess2e and sess2f differ with k, j or i displayed and in the case of sess2f the display is as follows.

```
i=2015
i=2014
i=2013
i=2012
i=2011
```

To explain what is happening here, in all cases there are 5 things shown with a variable called k, j or i (depending on the example) taking each value from a list $1, 2, 3, 4, 5$ or from a list $2015, 2014, 2013, 2012, 2011$. It is the part 1:5 which means the first case and it is the part 2015:-1:2011 which means the second case. In the version with just two numbers separated by the colon : you start at the first number and increment in steps of 1 until the second number is reached. In the version with three numbers separated by two colons the middle number is the increment and this can be any number other than 0. In the example involving the variable i the increment is -1. As a comment here, if you put 2015:2011 instead of what is given above then the list of values to consider is zero because the list is empty. There will be more about lists and arrays in the next session and it worth pointing out here that in this context a list is not actually created as Matlab knows how to step through the equally spaced values.

## 2.3 The for–loop syntax

```
for variable_name=list_of_values
  Instructions to do for each value in the list.
  The instructions typically use variable_name.
end
```

In the previous examples k, j or i were used as the variable name. The keyword end is the statement which indicates the end of the block and you will see this in several places in Matlab to indicate the end of a block. It is advisable for readability to uniformly indent the block of statements to quickly determine where a block starts and ends.

## 2.4 Examples to evaluate sums

Suppose that we wish to compute

$$\sum_{n=1}^{1000} \frac{1}{n^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{1000^2}.$$

Very few Matlab statements are needed to compute this and before these are given we consider briefly how we would set about doing this by hand calculation. We would start by calculating

$$1 + \frac{1}{2^2}.$$

We would then add the next term so that we have

$$1 + \frac{1}{2^2} + \frac{1}{3^2} = \left(1 + \frac{1}{2^2}\right) + \frac{1}{3^2}.$$

That is we use the previously stored answer and add the next term. This process of adding the next term to the previous result continues until all the terms are used. In Matlab you can have the following statements.

```
s=0;
for n=1:1000
  s=s+1.0/(n*n);
end
```

The variable called `s` starts at `0` and keeps changing as it successively stores the values

$$\sum_{r=1}^{n} \frac{1}{r^2}, \quad n = 1, 2, \ldots, 1000.$$

Once the loop has finished `s` stores the required value. Try the above, by creating the instructions in the editor, and display the final value of `s`. You can display the final value of `s` by adding the statement

```
s
```

If you want more control over the format of the output then you can put instead

```
fprintf('s=%24.16e\n', s);
```

With the second case the display is as follows.

```
s=   1.6439345666815615e+00
```

Here the part `%24.16e` is for the format of `s` with the letter `e` being for the scientific format to be used, the number is to be displayed in a width of 24 with 16 digits shown after the decimal point.

### Exercise 2.4.1

Write Matlab instructions to compute the following sum.

$$\sum_{n=1}^{500} \frac{1}{(2n-1)^2} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + \cdots + \frac{1}{999^2}.$$

You should get the following.

```
s=   1.2332005503028372e+00
```

## 2.5 Computing a product using a `for`–loop

In your previous study you should have been introduced to the factorial which is as follows.

$$\begin{aligned} 0! &= 1, \qquad \text{(by definition)}, \\ 1! &= 1, \\ 2! &= 2, \\ 3! &= 2 \times 3 = 6, \\ 4! &= 2 \times 3 \times 4 = 24, \\ &\vdots \end{aligned}$$

Mathematically we have

$$(n+1)! = (n+1)n!, \quad n = 0, 1, 2, \ldots .$$

To compute this in Matlab using a `for`–loop when $n = 10$ can be done with the following statements.

```
n=10;
p=1;
for r=2:n
  p=r*p;
end
```

Create these instructions in the editor and run the program and display the value of p. You should get 3628800 for 10!.

The mechanism here for computing the product is similar to the mechanism for computing the sum with addition replaced by multiplication and with the initial value set to 1 rather than 0. Matlab actually has a function called `factorial` and you can try using this in the command window by typing

```
factorial(10)
```

There will be more on loops later in this section and these will be used with other statements which we first need to introduce.

## 2.6 The `if`–statement and the `if–else` construction

There are many situations in which we only want to do something if a condition is true and in Matlab this can be done using an `if`–statement. More generally we may wish to do something if a condition is true and something else if the condition is false and in Matlab this is achieved with an `if–else` construction. We start with some short examples and later say more about the test condition and the syntax.

### Example: The quadratic equation formula

Most programming texts use the quadratic equation formula for solving a quadratic equation near the start of a course and this is used here to illustrate the use of `if`–statements. (The case of implementing the quadratic formula was also at the end of the week 1 session.) The context is the quadratic equation

$$ax^2 + bx + c = 0, \quad a \neq 0,$$

and the formula for the solutions is

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Assuming that $a$, $b$ and $c$ are real numbers and only real roots are wanted then as part of a program we can have the following statements.

```
d=b*b-4*a*c;
if d>=0
  disp('The quadratic has real roots')
end
```

The variables `a`, `b` and `c` will need to be already be in the Matlab workspace for this to run. Create a file called `sess2_quad1.m` which contains these statements and appropriately add statements to consider the case $a = 1$, $b = -5$, $c = 4$.

The test here is the part `d>=0` and the outcome is 1 denoting true or 0 denoting false. To separately test the condition part try typing the following in the command window.

```
d=3
d>=0
d<0
x=-3
x>=0
x<=-3
```

This will generate the following output.

```
d =
     3
ans =
     1
ans =
     0
x =
    -3
ans =
     0
ans =
     1
```

It is the answers of 0 or 1 which are the outcomes of the tests.

If we want an action to happen if the condition is false then the previous example can be changed to the following.

```
d=b*b-4*a*c;
if d>=0
  s=sqrt(d);
  x1=(-b-s)/(2*a);
  x2=(-b+s)/(2*a);
else
  fprintf('d=%e,', d);
  fprintf(' the quadratic has complex roots\n');
end
```

Create a file containing these statements and add additional statements to run it in the cases

$$x^2-5x+4=(x-1)(x-4)=0$$

and

$$x^2-5x+7=(x-5/2)^2+(3/4).$$

## 2.7 The logical operators

The examples so far have have illustrated some of the logical operators and for a complete list we have the following.

| | |
|---|---|
| < | Less than |
| > | Greater than |
| <= | Less than or equal |
| >= | Greater than or equal |
| == | True if the items are the same |
| ~= | True if the items are different |

In several of these we need two characters, e.g. >= whereas in mathematics we write the symbol ≥. Also take care to distinguish the assignment operation d=3 from the comparison operation d==3.

## 2.8 Combining tests

We can create tests by combining other tests using `&&` (logical and) and `||` (logical or). As an example, in mathematics we often write $x \in (a, b]$ as shorthand for $a < x \le b$. There are two conditions here and in Matlab we need to put

```
test=a<x && x<=b
```

In this context the variable `test` is set to 0 if `x` is not in the interval and 1 if `x` is in the interval. If we want the opposite outcome then we can have one of the following.

```
test1=x<=a || x>b
test2=~(a<x && x<=b)
```

I would choose the `test1` case which uses the logical or operation and is true if at least one of the tests is true. The second possibility involves reversing the outcome of the test in the round brackets.

## 2.9 The `if`–block syntax

```
if  logical_condition
  Statements to do if the condition is true.
end
```

Note that again the keyword `end` denotes the end of the block and thus again I recommend that the statements in a block are uniformly indented to easily see where a block starts and ends.

## 2.10 The `if–else` construction syntax

```
if  logical_condition
  Statements to do if the condition is true.
else
  Statements to do if the condition is false.
end
```

With this set-up exactly one of the block of statements is executed.

## 2.11 The break statements in loops

In the for–loop examples presented so far the number of times the statements in the loop were repeated is the number of entries in the list, e.g. 1:5 and 2015:-1:2011 both have 5 entries and n=1:1000 has 1000 entries. To leave a loop before all the entries have been considered can be done using a break statement. We would usually do this after some testing and this is illustrated in the following examples.

### Factorials and a break statement

The factorials are $0! = 1$, $1! = 1$, $2! = 2$, $3! = 6$, $4! = 24$, ,... and these grow rapidly in magntitude. If we wish to just show these numbers until you first exceed $10^{12}$ then we can do the following.

```
for n=1:30
  v=factorial(n);
  fprintf('n=%2d, n!=%14d=%22.14e\n', n, v, v);
  if v>=1e12
    break;
  end
end
```

Create a file which contains these lines and run it and check that you get the following output.

```
n= 1, n!=             1=  1.00000000000000e+00
n= 2, n!=             2=  2.00000000000000e+00
n= 3, n!=             6=  6.00000000000000e+00
n= 4, n!=            24=  2.40000000000000e+01
n= 5, n!=           120=  1.20000000000000e+02
n= 6, n!=           720=  7.20000000000000e+02
n= 7, n!=          5040=  5.04000000000000e+03
n= 8, n!=         40320=  4.03200000000000e+04
n= 9, n!=        362880=  3.62880000000000e+05
n=10, n!=       3628800=  3.62880000000000e+06
n=11, n!=      39916800=  3.99168000000000e+07
n=12, n!=     479001600=  4.79001600000000e+08
n=13, n!=    6227020800=  6.22702080000000e+09
n=14, n!=   87178291200=  8.71782912000000e+10
n=15, n!= 1307674368000=  1.30767436800000e+12
```

As the output shows $15! > 10^{12}$ and the loop ends. Note that to write $10^{12}$ we use the scientific notation 1e12. The fprintf instruction here is a bit more complicated than the earlier examples with 3 format specifier in the string to get the alignment in the output. The instructions %2d and %14d are for integers which are right justified in a width of 2 and 14 respectively (the d standards for decimal digits as in the earlier examples). The instruction

`%22.14e` is for scientific notation with a width of 22 and with 14 digits shown as was used in an earlier example (the `e` is for the mantissa exponent way of representing numbers in a scientific format).

## 2.12 Summary

In this session you have been introduced to the following.

- `for` loops for repeating statements usually a fixed number of times.
- The colon notation in examples such as `1:5` and `2015:-1:2011`.
- The logical comparison operators and examples of constructing tests.
- `if`-blocks and `if–else` blocks.
- The `break` statement for leaving a loop.
- Several examples of using `fprintf` to create aligned output.

## 2.13 Further exercises

### Exercise 2.13.1

In one of your early `MA1710` sessions you were asked to gives the values of sin($x$), cos($x$) and tan($x$) for $x = k\pi/12$ for the 17 values $k = -12, -10, -9, -8, -6, -4, -3, -2, 0, 2, 3, 4, 6, 8, 9, 10, 12$. The following skeleton program nearly does this. Create a file containing these instruction and add additional instructions to attempt to create the output shown below.

```
L=[-12 -10 -9 -8 -6 -4 -3 -2 0 2 3 4 6 8 9 10 12];
fprintf('%3s %10s %10s %10s\n', ...
        'k', 'sin(x)', 'cos(x)', 'tan(x)');
for k=L
  x=(pi/12)*k;
  c=cos(x);
  s=sin(x);
  % .. statements needed
end
```

The aim is to create the following output.

```
  k     sin(x)     cos(x)     tan(x)
-12 -0.0000000 -1.0000000  0.0000000
-10 -0.5000000 -0.8660254  0.5773503
 -9 -0.7071068 -0.7071068  1.0000000
 -8 -0.8660254 -0.5000000  1.7320508
 -6 -1.0000000  0.0000000        Inf
 -4 -0.8660254  0.5000000 -1.7320508
 -3 -0.7071068  0.7071068 -1.0000000
 -2 -0.5000000  0.8660254 -0.5773503
  0  0.0000000  1.0000000  0.0000000
  2  0.5000000  0.8660254  0.5773503
  3  0.7071068  0.7071068  1.0000000
  4  0.8660254  0.5000000  1.7320508
  6  1.0000000  0.0000000        Inf
  8  0.8660254 -0.5000000 -1.7320508
  9  0.7071068 -0.7071068 -1.0000000
 10  0.5000000 -0.8660254 -0.5773503
 12  0.0000000 -1.0000000 -0.0000000
```

To get exactly what is shown above you need an `if` block to carefully deal with the cases when $|\tan(x)|$ is $\infty$. The numbers are written with the format `%10.7f` which means that they are in a width of 10 and 7 digits are shown after the decimal point. The part `f` is for a floaing point representation but not using the scientific notation and is suitable when you know that the numbers are not too small and not too large in magnitude. Another feature of this example is that one statement below is spread over two lines and this is achieved by putting `...` at the end of the line to be continued.

```
fprintf('%3s %10s %10s %10s\n', ...
       'k', 'sin(x)', 'cos(x)', 'tan(x)');
```

The parts `%3s` and `%10s` are for displaying strings, which are right justified' in a width of 3 and 10 respectively. The widths used are the same as for the numbers so that the header parts are aligned properly with the numbers.

## Exercise 2.13.2

Let $n \ge 0$ and let

$$s_n = \sum_{k=0}^{n} \frac{1}{k!} = 1 + 1 + \frac{1}{2!} + \cdots + \frac{1}{n!}.$$

Create a m-file containing Matlab statements to compute $s_{17}$ and $s_{18}$ and show the result (you can use variable names of `s17` and `s18`). If your variable names are `s17` and `s18` then compute `s18-s17`. Can you explain the answer? In the case of `s17` you can aim to show the following.

```
s17=  2.7182818284590455e+00
```

## Exercise 2.13.3

If $r > 0$ denotes the interest rate on savings over a period of time and interest is applied discretely then after 1 unit of time the starting amount has increased by the factor $(1+r)$ and after $m$ units of time the starting amount has increased by the factor $(1+r)^m$. Now as

$$1+r < \left(1+\frac{r}{2}\right)^2 = 1+r+\frac{r^2}{4}$$

it follows that if we replace $r$ by $r/2$ and we replace $m$ by $2m$ then the amount is greater. To investigate this difference consider the case $r = 0.1$ and $m = 10$ and write statements to compute and display

$$\left(1+\frac{r}{n}\right)^{nm}, \quad n = 1, 2, 2^2, \ldots, 2^{10}.$$

The numbers in your output should correspond to the following.

```
   1  2.593742460100
   2  2.653297705144
 ...  ..............
1024  2.718149111732
```

The output is as follows.

```
   1  2.593742460100
   2  2.653297705144
   4  2.685063838390
   8  2.701484940753
  16  2.709835576308
  32  2.714046643708
  64  2.716161207948
 128  2.717220759477
 256  2.717751104075
 512  2.718016418768
1024  2.718149111732
```

The numbers are converging to $e$.