MA1710: Week 3

Control statements and the creation of vectors

3.1 Introduction to the week 3 session

We observed that there was more material in the week 2 session than most students could do during the hour and thus we start this session with a quick revision of what most students tried, there is further material on the control statements of `for`, `if` and `break` and then we move on to vectors. There is a connection between the previous parts and the new material in that with `for` loops the variable takes in turn each value in a list of values which is also described as vector of values.

3.2 Revision from week 2

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
```

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
```

Example of computing a sum using a for–loop

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

One way to compute s is to use the colon notation and use an increment of 2.

```
s=0;
for n=1:2:999
  s=s+1/(n*n);
end
fprintf('s=%24.16e\n', s);
```

3.3 The `break` statements in loops

In the `for`–loop examples presented in the week 2 notes the number of times the statements in the loop was repeated was 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 example.

Factorials and a `break` statement

The factorials are $0! = 1$, $1! = 1$, $2! = 2$, $3! = 6$, $4! = 24$, ,... and these grow rapidly in magnitude. 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 called `sess3_fact.m` 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 in the week 2 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 week 2 examples). The instruction `%22.14e` is for scientific notation with a width of 22 and with 14 digits shown as was used in week 2 (the `e` is for the mantissa exponent way of representing numbers in a scientific format).

3.4 Introduction to vectors

The basic type in Matlab is a matrix with every entry of the matrix being of type `double` which means about 16 decimal digit accuracy for each entry. The variables used in previous Matlab sessions are considered as 1×1 matrices and are hence a special case of a matrix. Vectors, which we consider next in this session, are also a special case of a matrix in that there is just one row or just one column. In the following we consider various ways of creating vectors and we discuss some operations with vectors.

Vectors will be used in a subsequent session as part of the process to create two-dimensional plots when we plot a vector of x values against a vector of y values and it will be used in your first project to store the history of values of various quantities.

3.5 Creating vectors using [and] notation

In the Matlab editor create a file called `sess3a.m` containing the following lines and run the script file.

```
clear
a=pi
b=[5.1, 4, 3.3, -2.42, 1]
c=[5; 4; 3]
d=[1, 2, 3, 4]'
whos
```

The output generated in the command window should be as follows.

```
a =
    3.1416
b =
    5.1000    4.0000    3.3000   -2.4200    1.0000
c =
     5
     4
     3
d =
     1
     2
     3
     4
  Name      Size             Bytes  Class     Attributes

  a         1x1                  8  double
  b         1x5                 40  double
  c         3x1                 24  double
  d         4x1                 32  double
```

The `clear` instruction clears the workspace and the `whos` instructions display information about what is in the current workspace. Here `a` is just a variable with the `whos` command indicating that the size is 1×1. `b` is a row vector, (a matrix with just one row) and `c` and `d` are column vectors (matrices with just one column). In these examples the entries are given between the square brackets `[` and `]`. To separate entries on a row you use a comma and/or one or more spaces. In the case of `c` the semi-colon means go to the next row and in the case of `d` the character `'` at the end of the statement means transpose. In the case of `d` we would have instead a row vector if the character `'` is removed.

> The column with the header `Bytes` in the output from `whos` gives the number of bytes used to store each quantity and for vector and matrices it is just 8 times the number of entries. One byte is 8 bits and thus one item of type `double` involves 64 bits with these bits used as follows. 52 bits are used for the mantissa of the number, 11 bits are used for the exponent and 1 bit is used for the sign of the number. As $2^{-52} \approx 2.2 \times 10^{-16}$ we get close to 16 decimal digit accuracy. In Matlab the variable `eps` stores 2^{-52} and this is referred to as machine precision.

3.6 Creating vectors with equally spaced entries

We usually use `[` and `]`, as above, to create vectors which do not have too many entries with larger vectors generated by other means dependent on the task being considered and there are many cases when we want a vector of equally spaced values. This can be done using the colon notation (as was used in `for`–loops in session 2) and with the Matlab function `linspace` and we discuss the use of both of these in this section.

Use the editor to create a script file called `sess3b.m` which contains the lines indicated below and run the script file.

```
a=0:9
b=9:-1:3
c=(2015:2018)'

d=0:0.2:1
d2=linspace(0, 1, 6)

e=0:0.5:pi
e2=0:pi/6:pi
e3=linspace(0, pi, 7)
```

You should get the following output in the command window.

```
a =
     0     1     2     3     4     5     6     7     8     9
b =
     9     8     7     6     5     4     3
c =
        2015
        2016
        2017
        2018
d =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
d2 =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
e =
         0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000
e2 =
         0    0.5236    1.0472    1.5708    2.0944    2.6180    3.1416
e3 =
         0    0.5236    1.0472    1.5708    2.0944    2.6180    3.1416
```

As in the `for`–loop examples when just two numbers are involved separated by a colon the step is 1 which is the case with `a=0:9` which generates a row vector of length 10. A step of 1 is also the case with the creation of `c` and the use of the round brackets and the character `'` gives the column vector.

When 3 values are given the middle value is the increment and some care is often needed to ensure that the outcome is what is wanted. In the case of `b` the increment is -1 and we get a row vector of length 7. Note that `d` and `d2` are the same and we have two convenient ways of creating the vector. If we only want to think about the number of points used then the `linspace` version is easier to use whereas if we wish to think more in terms of the spacing then the colon version would be the preferred version. Please note however that the increment has to be compatible with the 'start' and 'end' values to ensure that the 'end' value is one of the entries. This is illustrated in the examples for `e`, `e2` and `e3` with both `e` and `e2` containing the last value but the 7 entries in `e` do not including the end number of `pi`.

To summarise the syntax of using the colon notation or using `linspace` we have the following.

`a:b`	A row vector starting at `a` with an increment of 1. `b` is the upper bound.
`a:h:b`	A row vector starting at `a` with an increment of h. `b` is the bound (an upper bound if `h`> 0 or a lower bound if `h`< 0).
`linspace(a, b, n)`	A row vector starting at `a` and ending at `b` with in total `n` values. The spacing is thus `(b - a)/(n - 1)`.
`linspace(a, b)`	A row vector starting at `a` and ending at `b` with in total 100 values.

3.7 Using the entries of a vector

If `x` is a vector of length `n` then we can refer to the entries by putting `x(1)` for the first entry, by putting `x(2)` for the second entry, ..., by putting `x(n)` for the `n`th entry, We can also change individual entries. To illustrate this create a script file called `sess3c.m` which contains the following and run the script file.

```
x=0:0.2:1
x(3)
x(6)=x(6)+0.5
x(end)
```

The following output is generated in the command window.

```
x =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
ans =
    0.4000
x =
         0    0.2000    0.4000    0.6000    0.8000    1.5000
ans =
    1.5000
```

In this example the entry `x(3)` is displayed, the entry `x(6)` is changed with the entire contents of the modified `x` displayed and in the last statement the index of `end` refers to the last entry of the vector.

An error is shown if an invalid index is used. This is the case with the last two statements in the following.

```
x=0:0.2:1.2;
x(0)
x(3.4)
```

The message generated is the following.

```
Subscript indices must either be real positive integers or logicals.
```

3.8 Adding vectors, multiplying by a scalar and adding a scalar

In this section and the next we describe some of the manipulations that can be done with vectors.

Create the file `sess3d.m` containing the following and run the script file.

```
x=ones(1, 6)
y=2:7
z=x+y
x3=3*x
v=y-0.5
```

This generates the following output.

```
x =
     1     1     1     1     1     1
y =
     2     3     4     5     6     7
z =
     3     4     5     6     7     8
x3 =
     3     3     3     3     3     3
v =
    1.5000    2.5000    3.5000    4.5000    5.5000    6.5000
```

The function `ones` is a built-in function which creates a matrix of the shape given with every entry being 1 and this is used to create `x`. `x` and `y` have the same shape. We can add two vectors of the same dimensions (the statement `z=x+y`), we can multiply every entry by a scalar (the statement `x3=3*x`) and we can add or subtract a scalar from every entry (the statement `v=y-0.5`). The statement which creates `v` is an exception to the usual case of requiring the vectors involved to have the same size and is equivalent to putting

```
v=y-0.5*ones(1, 6)
```

3.9 Entry-wise operations and vectorised functions

We next consider what are known as entry-wise operations and do this by an example of evaluating the following quadratic at several different points. Let

$$y = x^2 - 3x + 2 = (x-2)(x-1).$$

As there are roots at 1 and 2 and the function has a minimum at $x = 3/2$ a suitable range for x is the interval $[0,3]$. To create a vector of x–values and a vector of the corresponding y–values based on what has been taught so far can be achieved with statements such as the following which you should create in a file with the name `sess3e.m`.

```
clear
x=0:0.25:3;
y=zeros(1, 13);
for k=1:13
  y(k)=x(k)^2-3*x(k)+2;
end
[x; y]'
```

The part `zeros(1, 13)` creates a row vector of length 13 with every entry set to 0 and its purpose here is to allocate space for the `y` entries which are set in the statements that follow. The script will still work if this was not done but it would be inefficient as the size of `y` will increase as each new value of `k` is considered. The part `[x; y]` creates a quantity with two

rows and the transpose symbol then converts this to a quantity with two columns so that the output generated is as follows.

```
ans =
         0    2.0000
    0.2500    1.3125
    0.5000    0.7500
    0.7500    0.3125
    1.0000         0
    1.2500   -0.1875
    1.5000   -0.2500
    1.7500   -0.1875
    2.0000         0
    2.2500    0.3125
    2.5000    0.7500
    2.7500    1.3125
    3.0000    2.0000
```

In the form given several things are quite specific to having 13 entries for both `x` and `y`. To make it easier to consider a different vector it helps to change it to the following.

```
x=0:0.25:3;
m=length(x);
y=zeros(1, m);
for k=1:m
  y(k)=x(k)^2-3*x(k)+2;
end
[x; y]'
```

Matlab has a much shorter way of generating the vector `y` which avoids completely the need for the `for`–loop and in this example we can shorten the script file to the following which you should create as the file `sess3f.m`.

```
x=0:0.25:3;
y=x.^2-3*x+2;
[x; y]'
```

Try this and check that you get the same output as before.

In Matlab the notation `x.^2` means create a vector of the same shape as `x` with every entry squared and it is an example of an entry-wise operation. As we know the roots of the quadratic an alternative script file is to have the following.

```
x=0:0.25:3;
y=(x-2).*(x-1);
[x; y]'
```

With this way of writing the expression `x-2` and `x-1` are both vectors of the same length and `.*` is the entry-wise operation of multiplying together the corresponding entries for the creation of `y`. Try this and check that you get the same output as before.

Similar to the entry-wise operations of `.^` and `.*` the built-in standard functions such as `sin`, `cos`, `exp` etc. act in a vectorised way, i.e. if the argument is a vector then they act individually on each entry to create an output of the same shape as the input. To test this create files `sess3g.m` and and `sess3gv.m` as follows. The file `sess3g.m` should contain the statements below containing a `for`–loop.

```
x=0:pi/6:pi;
m=length(x);
y=zeros(1, m);
for k=1:m
  y(k)=sin(x(k));
end
[x; y]'
```

The file `sess3gv.m` should contain the vectorised version.

```
x=0:pi/6:pi;
y=sin(x);
[x; y]'
```

Try both of these and check that you get the following output in the command window.

```
         0         0
    0.5236    0.5000
    1.0472    0.8660
    1.5708    1.0000
    2.0944    0.8660
    2.6180    0.5000
    3.1416    0.0000
```

3.10 Summary

In this session there has been a quick recap of `for`-loops, the `if–else` syntax and the use of the statement `break` to leave a loop early. You have then been introduced to a number of different ways of creating vectors and we summarize here a few of the key points.

- Use the brackets `[` and `]` to create a vector with all the entries given explicitly.
- For equally spaced entries the colon notation can be used or `linspace` can be used. The colon notation is likely to be better when all the entries are integers. When we do not have integers `linspace` ensures that the 'start' and 'end' values are in the list.

- The colon notation and `linspace` both create row vectors.
- Use `'` to transpose a vector.
- You refer to individual entries with round brackets, e.g. `x(3)`.
- If you need to create a vector entry–by–entry then it is advisable to pre-allocate the size first and typically this is done using the `zeros` function.
- Examples have been given of using the entry-wise operations and the vectorised version of one of the standard functions.

3.11 Further exercises

Exercise 3.11.1

The normal distribution function in the case of a mean of 0 and a variance of 1 is given by

$$y = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

By any means create and display a vectors of 21 equally spaces x values in $[-3, 3]$ together with the corresponding y values.

Exercise 3.11.2

Create the following script file and run it.

```
t=0:15:360;
x=pi*t/180;

disp('Radian version')
[sin(x); cos(x); tan(x)]'

disp('Degrees version')
[sind(t); cosd(t); tand(t)]'
```

Why do you think the display in the radians version is different from the display in the degrees version?

Exercise 3.11.3

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
```