Matlab for Numerical Algorithms

Vectors

A vector is a one-dimensional array of numbers. A row vector is written horizontally; a column vector is written vertically.
In Matlab, vectors are defined by writing the components inside square brackets. For row vectors, the elements are separated by commas or spaces, e.g. `[5 3 -2 4 -1 0 2]`. The components in column vectors are separated by semicolons, e.g. `[3;-1;0;2;9]`.

We can refer to a component of a vector by putting the component number in round brackets. For example, if `v` is the vector `[5 0 2 -4 1 0 3]`, then `v(5)` is the 5th component which is 1.

Matrices

A matrix is a two-dimensional array of numbers. We write these in Matlab inside square brackets, entering one (horizontal) row at a time, just like a row vector, and we put a semi-colon at the end of each row. For example, the matrix

$$C = \begin{bmatrix} -2 & 5 & 1 \\ 1 & 0 & 4 \end{bmatrix}$$

is entered into Matlab using `C=[-2 5 1;1 0 4]`.

We can refer to a single row or column of a matrix. To find the second row of matrix `C`, we write `C(2,:)`, which gives `[1 0 4]`. It finds all the components in `C` whose first subscript is 2. Similarly the third column of `C` is referred to as `C(:,3)`.

`for` loops

These are used to repeat sections of code when the number of repeats is known before the code is run.

Examples include:

- Do 5 iterations. We know in advance that the section will be repeated 5 times.

```
x=3;
for i=1:5
    x=(x+2/x)/2;
    disp(x)
end
```

- Subtract the first row of a matrix from the second row - assume the matrix has `n` columns. When this is performed, the value for `n` will be known - though it won't be known when the code is written.

```
A=[1 5 3 5; 6 4 -2 5;6 0 -5 2];
[m,n]=size(A)
for i=1:n
    A(2,i)=A(2,i)-A(1,i);
end
disp(A)
```

`while` loops

These are used to repeat sections of code when the number of repeats depends on the results found/calculated in the loop.

Examples include:

- Prompt the user to input a positive integer and keep prompting until they do.

```
n=input('Enter a positive number ');
while n<=0
    n=input('Enter a positive number ');
end
```

  The number of repeats depends on how long it takes the user to enter a positive number.

- Perform iterations until the result is less than 1.

```
x=input('Enter the value of x ');
m=0.8;
while x>=1
    x=x*m;
    disp(x)
end
```

  In this case we keep multiplying by `m` until `x` becomes less than 1. The number of repeats depends on how many times we need to do this to get below 1.

Functions

Functions are also used to repeat sections of code. They may be used for standard methods that we want to use often. A function is usually defined by writing it in a separate Matlab file.

A function will be of the form

`function` *output parameter(s)* = *function name* (*input parameter(s)*)
% Write in here some comments about what the function does
Matlab statements to calculate the output parameters values from the input parameter values.

The output parameters represent the values that you want calculated in the function; the input parameters represent the values that are available to the function. If there is more than one output parameter, the output parameters are written in square brackets.

For example, we could write a Matlab function to find the volume and surface area of a cylinder with given height and diameter. The information used is the diameter and the height, so these will be the input parameters. The values to be calculated are the volume and surface area; these will be the output parameters.

```
function [vol,area]=mycylinder(d,h)
% calculates the volume (vol) and surface area (area) of
% cylinder whose diameter is d and height h
r=d/2;
vol=pi*r^2*h;
area=2*pi*r^2+2*pi*r*h;
```

To use the function in a script file or in the Command Window, we give the variable names for the results in the output parameters, and the values (or variables containing them) in the input parameters. For example to find the volume and surface area of a cylinder whose diameter is 4 and height 5, we would write
`[v,a]=mycylinder(4,5)`
If the diameter and height of the cylinder we are interested in are contained in variables `diam` and `ht`, we write
`[v,a]=mycylinder(diam,ht)`

Suppose we want to make a table of the volume and surface area of cylinders whose height is given and whose diameter ranges from 1 up to some value that the user supplies. We could write the following script file.

```
h=input('What is the height of the cylinders? ');
max_d=input('What is the maximum diameter required? ');
disp('Height    Diameter    Volume     Surface Area')
for k=1:max_d
    [v,a]=mycylinder(h,k);
    disp(sprintf(' %7.2f     %2d      %7.2f        %7.2f',h,k,v,a))
end
```

Sample output is

```
What is the height of the cylinders? 5
What is the maximum diameter required? 6
Height    Diameter    Volume     Surface Area
    5.00      1        19.63          54.98
    5.00      2        39.27          70.69
    5.00      3        58.90          86.39
    5.00      4        78.54         102.10
    5.00      5        98.17         117.81
    5.00      6       117.81         133.52
```

Examples

1. One way to estimate the square root of 2 is to use the recurrence

$$x_n = \frac{1}{2}(x_{n-1}\frac{2}{x_{n-1}}), \quad n = 1, 2, \ldots, \quad x_0 = 3.$$

   This is done in the `for` loop section above as

   ```
   x=3;
   for i=1:5
       x=(x+2/x)/2;
       disp(x)
   end
   ```

   This will display each value as it is calculated. If we don't need to see these intermediate values, we can move the `disp` outside the loop and put it last. If we want to keep these intermediate values for some reason then we can use a vector as below.

   ```
   x=[3];
   for i=1:5
       x(i+1)=(x(i)+2/x(i))/2;
   end
   disp(x)
   ```

2. Use the iteration in Example 1, until the difference between the iterates is less than 0.005. If we do not need *all* the iterates, then we can just keep the current one (called `x`) and the previous one (called `x_old`) so we can find the difference.

   ```
   x=3; x_old=x+1;  % x_old is set to x+1 so that abs(x-x_old) > =0.005 on 1st time through loop
   while abs(x-x_old)>=0.005
       x_old=x;
       x=(x+2/x)/2;
   end
   disp(x)
   ```

   Notice the use of `abs` in the Boolean expression. Sometimes the iterates are decreasing and so the difference is negative.

3. For the matrix `A`, find the component in the first column with the maximum magnitude, and which row it is in.

```
A=input('Enter the matrix ');
[maxabs,n]=max(abs(A(:,1)));
disp(sprintf('Max magnitude is %0.5f in row %d',maxabs,n))
```

Vectorisation

Vectorisation can be used in Matlab to speed up calculations involving vectors and matrices. Often this replaces `for` loops.
The example on subtracting the first row of a matrix from the second row, as in the section on `for` loops, can be written using vectorisation as:

```
A=[1 5 3 5; 6 4 -2 5;6 0 -5 2];
disp(A)
A(2,:)=A(2,:)-A(1,:);
disp(A)
```

Exercises

1. Write a Matlab function to perform the iterations in Example 1. The input parameters will be `x0` (the initial value of `x`) and `n` (the number of iterations to be performed).
2. Write a Matlab script file that prompts the user to enter the initial value for `x` and how many iterations are required, calls the function in Exercise 1 and then displays the final result.
3. When we perform Gaussian elimination, we first subtract a multiple of the first row from the second row. We choose this multiple so that the first element in the second row is zero after this subtraction.

   Write a Matlab script file which

   - prompts the user to enter a matrix
   - finds how many columns in the matrix
   - checks that the first element in the first row is non-zero
   - checks that the first element in the second row is non-zero
   - finds the multiple needed
   - uses a `for` loop to subtract that multiple of the first row from the second row so as to make the first element in the second row zero.
4. Repeat Exercise 3 using vectorisation.
5. You are given the following function for the secant method.

```
function [x,fx]=secant(xkm1,xkm2,N)
% to perform N iterations of the secant method starting with the
% values xkm1 and xkm2.
fkm1 = f(xkm1);
fkm2 = f(xkm2);

for k = 1:N
   x    = xkm1 - (xkm1-xkm2)/(fkm1-fkm2)*fkm1;
   xkm2 = xkm1;
   xkm1 = x;
   fkm2 = fkm1;
   fkm1 = f(x);
end
fx=fkm1;
```

   Write a script file, and any other files you need, to estimate a zero of $f(x) = x^2 - e^{-x}$.