MA2507 Computing Mathematics Laboratory: Week 5

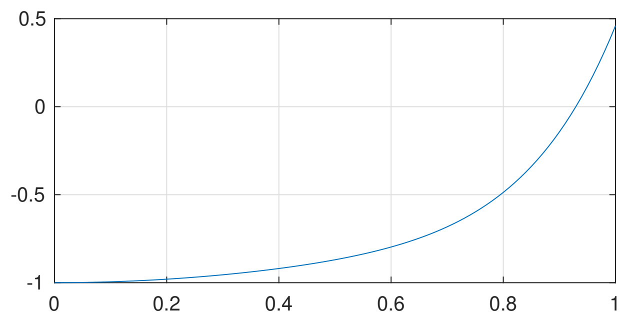
1. Functions. When we execute a MATLAB script file, all variables, vectors and matrices that appear in the script file go into the main memory of MATLAB. In order to keep a clean account of the memory space and to have a clean control for input and output, it is extremely useful to use functions. Usually, a function with the name `funcname` must be stored in the file called `funcname.m` and it should be in the current working folder. Here is a function for $f(x) = x^7 - \cos(x)$ and it is saved as `f.m`.

```
function y=f(x)
y = x.^7 - cos(x);
end
```

In the folder where `f.m` is stored, we can simply evaluate it in the MATLAB main command window, or use the function in any script file. For example, the following lines

```
>> x = 0:0.01:1;
>> plot(x, f(x)); grid
```

produce the figure below. We can see that the function has a zero between 0.9 and 1.



If the function is only useful for a particular script, we can put the function below the main script (in the more recent versions of MATLAB). Let us try to find the zero of function $f(x)$ by Newton's method. For that we need another function for its derivative. We have the following script file:

```
% main script file: Newton's method
format long
x = 0.9;
rat = 1;
while rat > 1.0e-12
    dx = f(x)/fp(x);
    x = x - dx;
    rat = abs(dx/x);
end
x

% function below the main script
function y=f(x)
y = x.^7 - cos(x);
```

```
end

% another function below the main script
function z=fp(x)
z = 7*x.^6 + sin(x);
end
```

The above gives $x = 0.929273104148150$. Since we still have `f.m` in the folder, we can also use MATLAB internal function `fzero` to find the solution:

```
>> fzero(@f,0.9)
ans =
   0.929273104148150
```

Notice that the first input of `fzero` is the name of a function. This is made possible by adding the symbol `@` in front of the name, otherwise (i.e, if you do not add `@`), MATLAB would think that you are trying to run the function without the input. We can write a program for Newton's method that also inputs functions.

```
% a very short main script
format long
mynewt(@f, @fp, 0.9)

% a very general Newton's method that inputs functions
function sol= mynewt(F,Fp,x)
rat = 1;
while rat > 1.0e-10
    dx = feval(F,x)/feval(Fp,x);
    x = x - dx;
    rat = abs(dx/x);
end
sol = x;
end

% a function f(x)
function y=f(x)
y = x.^7 - cos(x);
end

% the derivative of f(x)
function z=fp(x)
z = 7*x.^6 + sin(x);
end
```

Newton's method requires the derivative of $f(x)$. Secant method is useful, since it does not need the derivative, but it needs two initial guesses. Given x_1 and x_2, the secant method gives x_3, x_4, ..., by

$$x_{j+1} = x_j - \frac{f(x_j)(x_j - x_{j-1})}{f(x_j) - f(x_{j-1})}, \quad j = 2, 3, ...$$

To have an efficient code, we do not use a vector x. We implement the first step for computing x_3, after that, we swap the variables, so that x_4, x_5, ..., are all called x_3. Meanwhile, we save and swap $f(x_j)$, so that f is only evaluated once in each iteration. Here is the complete program.

```
% a very short main script
format long
x = mysecm(@f,0.8,0.9)

% a very general function for the secant method
function sol= mysecm(F,x1,x2)
rat = 1;
y1 = feval(F,x1);
y2 = feval(F,x2);
while rat > 1.0e-10
    dx = y2*(x2-x1)/(y2-y1);
    x3 = x2 - dx;
    y3 = feval(F,x3);
    rat = abs(dx/x3);
    x1=x2;
    x2=x3;
    y1=y2;
    y2=y3;
end
sol = x3;
end

% a function below
function y=f(x)
y = x^7 - cos(x);
end
```

2. Barnsley fern: We can use a random variable (uniform in [0,1]), and the "if ... elseif ... else ... end" statement to perform different tasks with different probabilities. Here is an example for generating the so-called Barnsley fern. Given four 2×2 matrices A_1, A_2, A_3, A_4 and four vectors b_1, b_2, b_3 and b_4 below

```
A1 = [0.85, 0.04; -0.04, 0.85]
A2 = [0.2, -0.26; 0.23, 0.22]
A3 = [-0.15, 0.28; 0.26, 0.24]
A4 = [0, 0; 0, 0.16]
b1 = [0; 1.6]
b2 = b1
b3 = [0; 0.44]
b4 = [0; 0]
```

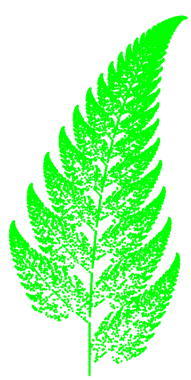
and the zero staring point $(x_1, y_1) = (0, 0)$, we can generate points (x_j, y_j) using

$$\begin{bmatrix} x_{j+1} \\ y_{j+1} \end{bmatrix} = A_m \begin{bmatrix} x_j \\ y_j \end{bmatrix} + b_m,$$

where the choice of integer m follows the following probability: $P(m=1)=0.85$, $P(m=2)=0.07$, $P(m=3)=0.07$ and $P(m=4)=0.01$. Notice that $0.85+0.07=0.92$ and $0.85+0.07+0.07=0.99$. We calculate 20000 points using the following program.

```
m=20000;
x=zeros(2,m);
for j=2:m
    a=rand;
    if a < 0.85
        A = [0.85, 0.04; -0.04, 0.85];
        b = [0; 1.6];
    elseif a < 0.92
        A = [0.2, -0.26; 0.23, 0.22];
        b = [0; 1.6];
    elseif a < 0.99
        A = [-0.15, 0.28; 0.26, 0.24];
        b = [0; 0.44];
    else
        A = [0, 0; 0, 0.16];
        b = [0; 0];
    end
    x(:,j)= A*x(:,j-1)+b;
end
plot(x(1,:),x(2,:),'g.')
axis equal
axis off
```

I have stored the points in a matrix with two rows and plot the points as green dots. The last two lines make sure the same scaling is used for x and y, and the box for the plot is removed. The above program gives the following figure.



3. Creepy animal: Here is another example (last year's test question) using `rand` to control different probabilities. Starting from the origin $(x_1, y_1) = (0,0)$, generate 10000 points by the following rule: (a) with probability 0.4, $(x_{j+1}, y_{j+1}) = (x_j, y_j) + (0,1)$ (move up by distance 1); (b) with probability

0.25, $(x_{j+1}, y_{j+1}) = (x_j, y_j) + (\sqrt{3}/2, 1/2)$ (move up-right by distance 1); (c) with probability 0.25, $(x_{j+1}, y_{j+1}) = (x_j, y_j) + (-\sqrt{3}/2, 1/2)$ (move up-left by distance 1); (d) with probability 0.1, reset (x_{j+1}, y_{j+1}) to the origin. Plot the 10000 points as "*".

We can keep all x_j and y_j in a 10000×2 matrix, called it `x`. The index j refers to j-th row. Here is the MATLAB script file.

```
x(1,:) = [0 0];
for j=2:10000
    p = rand;
    if p < 0.40
        x(j,:) = x(j-1,:) + [0,1];
    elseif p <0.65
        x(j,:) = x(j-1,:) + [sqrt(3)/2, 0.5];
    elseif p < 0.9
        x(j,:) = x(j-1,:) + [-sqrt(3)/2, 0.5];
    else
        x(j,:) = [0 0];
    end
end
plot(x(:,1), x(:,2), '*')
axis equal
axis off
```

We run the above program and get a typical figure like the one below.