CAPE Laboratory Assignment-1

Instructor: Dr Debashis Sarkar

18CH10071, Anshuman Agrawal

Q.1

Problem Statement

The following Eigenvalue Equation is obtained during solution of an unsteady state diffusion equation by Separation of Variables.

$$\sin\lambda - \lambda\cos\lambda = 0$$

There are many roots for the above equation and $\lambda = 0$ is one of them. Find the smallest positive root by any numerical zero-finding method of your choice. Also, solve the problem using *fzero* function of MATLAB and compare two results in a table.

MATLAB Code

Method used: Newton – Raphson

```
x=4;
tol=0.0001;
counter=0;
while true
    f_x=sin(x)-x*cos(x);
    f_dash_x=x*sin(x);
    x_next=x-f_x/f_dash_x;
    counter=counter+1;
    fprintf('Iteration %d: %d\n',counter,x_next);
    if abs(x_next-x)<=tol
        x=x_next;
        break;
    else
        x=x_next;
        continue;
    end
end
```

Code using *fzero* function

```
x0=6;
fun=@(x) sin(x)-x*cos(x);
sol=fzero(fun,x0);
fprintf('Root: %d\n',sol);
```

Results

Initial Guess	Solution using Newton Raphson	Iterations in Newton Raphson	Solution using *fzero*
1	1.883e-4 → 0	21	-9.015e-9 → 0
2	1.952e-4 → 0	22	-6.293e-9 → 0
3	-4.493409	4	4.493409
4	4.493409	4	4.493409
5	4.493409	3	4.493409
6	1.707e-4	23	4.493409

From the above results and analysis, it is observed that 4.493409 is the positive solution occurring most frequently while considering 6 initial guesses. Therefore, the smallest positive root of given eigen value equation is **4.493409**.

Conclusion

The given function of the eigen value equation is clearly an odd function due to which negative of the desired solution is also obtained. However, from our analysis, it is clear that *fzero* function is more likely to give us the desired solution that the Newton – Raphson method for similar initial guesses. However, while coding using Newton – Raphson we have more flexibility in deciding the accuracy of the desired solution. Newton – Raphson might appear to be a lengthy method here as compared to *fzero* but it is actually taking lesser computational time than *fzero* when timed using the MATLAB Profiler for the same initial guesses. This interesting observation is pretty logical if one considers the algorithm for *fzero* which is a combination of several methods hence, more complexity.

Q.2

Problem Statement

The well-known Colebrook-White equation for flow friction is an implicit equation.

$$\frac{1}{\sqrt{f}} = -2 \cdot \log_{10}\left(\frac{2.51}{Re} \cdot \frac{1}{\sqrt{f}} + \frac{\frac{\varepsilon}{D}}{3.71}\right)$$

Determine the flow friction factor (*f*) for the following two cases using Bisection Method.

(i) $Re = 2.3 \times 10^5$ and $\frac{\varepsilon}{D} = 10^{-4}$

(ii) $Re = 4.6 \times 10^7$ and $\frac{\varepsilon}{D} = 0.037$

Also, solve the problem using *fzero* function of MATLAB and compare two results.

In literature, many explicit equations have been proposed to approximate the Colebrook - White implicit equation. Recently, Praks and Brkic (2020) have proposed the following relation.

$$\frac{1}{\sqrt{f}} = 0.8686\left(B - C + \frac{C}{X - 0.5564C + 1.207}\right)$$

Where

X = A + B

$$A = \frac{(Re)\varepsilon}{8.0884D}, \quad B = \ln(Re) - 0.7794, \quad C = \ln(X)$$

Determine the flow friction factor (*f*) for the above two cases using the explicit relation and report the Relative Error (%) compared with the implicit Colebrook-White equation.

MATLAB Code

Bisection Method

```
Re=2.3*10^5;
epsilon_D=10^-4;
x0=0.01;
x1=0.05;
tol=10^-9;
counter=0;

while true
```

```
    x2=(x0+x1)/2;

f0=(1/sqrt(x0))+2*log10((2.51/Re)*(1/sqrt(x0))+epsilon_D/
3.71);

f2=(1/sqrt(x2))+2*log10((2.51/Re)*(1/sqrt(x2))+epsilon_D/
3.71);
    if f0*f2<0
        x1=x2;
    else
        x0=x2;
    end
    if abs(x1-x0)<=tol
        break;
    end
end
```

Code using *fzero* function

```
global Re epsilon_D;
Re = 2.3*10^5;
epsilon_D = 10^-4;
x0=0.01;
sol=fzero(@fun,x0);
fprintf('Root: %d\n',sol);

function y=fun(x)
global Re epsilon_D;
y=(1/sqrt(x))+2*log10((2.51/Re)*(1/sqrt(x))+epsilon_D/3.7
1);
end
```

Explicit Equation Code

```
Re = 2.3*10^5;
epsilon_D = 10^-4;

A = Re*epsilon_D/8.0884;
B = log(Re)-0.7794;
X = A+B;
C = log(X);

f = 1/((0.8686*(B-C+C/(X-0.5564*C+1.207)))^2);
```

*For the second case, values of Re and epsilon_D in the above code just need to be replaced.

Initial guesses taken for the 2nd case:

1. Bisection Method: 0.01 and 0.1
2. *fzero*: 0.05

Results

	Friction Factor computed using Colebrook-White equation		Friction Factor computed using Explicit Relation (C)	Relative Error (%)
Case	**Bisection Method (A)**	***fzero* (B)**		$\left\|\frac{C-B}{B}\right\| \times 100$
$Re = 2.3 \times 10^5$ and $\frac{\varepsilon}{D} = 10^{-4}$	0.0161	0.01605096	0.0161	0.305
$Re = 4.6 \times 10^7$ and $\frac{\varepsilon}{D} = 0.037$	0.0624	0.0624274	0.0624	0.044

Conclusion

The *fzero* function has much higher precision than the explicit relation or the bisection method. When compared with the actual value of friction factor by plotting the function of the Colebrook-White equation, *fzero* turns out to be much more accurate than both the other options. In terms of complexity, Bisection Method gives the best results when checked through the MATLAB profiler by taking the least computation time. Other methods involve raw coding with the restriction of significant digits thus, we are not able to generate as accurate solutions on MATLAB as *fzero*.