EM215: Numerical Methods

Lab Assignment 2

WIJERATHNE E.S.G. – E/18/397

(1) (a) Estimate the first derivative of the function,

$$f(x) = - 0.1x^4 - 0.5x^2 - 0.5x + 1.2$$

at x= 0.5 with forward, backward, and centered difference approximation at x= 0.5. In each case, start with a step size of 1 and halve the step size and repeat calculations until the step size reaches $1/2^{15}$. Given that f' (0.5) = −1.05, tabulate the values: step size, derivative approximations using the three methods and the absolute errors of the estimates of the three methods. Plot the errors versus step size, in log-log scale, for the three methods. Explain how you identify the orders of errors in the methods using the gradients of the graphs drawn.

- Data calculated from each method is as follows

Step Size(h)	Forward difference approximation		Backward difference approximation		Centered difference approximation	
	f'(0.5)	Absolute error(e)	f'(0.5)	Absolute error(e)	f'(0.5)	Absolute error(e)
1.0000000000	-2.0000000000	0.9500000000	-0.5000000000	0.5500000000	-1.2500000000	0.2000000000
0.5000000000	-1.4375000000	0.3875000000	-0.7625000000	0.2875000000	-1.1000000000	0.0500000000
0.2500000000	-1.2265625000	0.1765625000	-0.8984375000	0.1515625000	-1.0625000000	0.0125000000
0.1250000000	-1.1345703125	0.0845703125	-0.9716796875	0.0783203125	-1.0531250000	0.0031250000
0.0625000000	-1.0914306641	0.0414306641	-1.0101318359	0.0398681641	-1.0507812500	0.0007812500
0.0312500000	-1.0705108643	0.0205108643	-1.0298797607	0.0201202393	-1.0501953125	0.0001953125
0.0156250000	-1.0602054596	0.0102054596	-1.0398921967	0.0101078033	-1.0500488281	0.0000488281
0.0078125000	-1.0550903797	0.0050903797	-1.0449340343	0.0050659657	-1.0500122070	0.0000122070
0.0039062500	-1.0525421202	0.0025421202	-1.0474639833	0.0025360167	-1.0500030518	0.0000030518
0.0019531250	-1.0512702949	0.0012702949	-1.0487312309	0.0012687691	-1.0500007629	0.0000007629
0.0009765625	-1.0506349565	0.0006349565	-1.0493654250	0.0006345750	-1.0500001907	0.0000001907
0.0004882812	-1.0503174305	0.0003174305	-1.0496826649	0.0003173351	-1.0500000477	0.0000000477
0.0002441406	-1.0501587033	0.0001587033	-1.0498413205	0.0001586795	-1.0500000119	0.0000000119
0.0001220703	-1.0500793487	0.0000793487	-1.0499206573	0.0000793427	-1.0500000030	0.0000000030
0.0000610352	-1.0500396736	0.0000396736	-1.0499603279	0.0000396721	-1.0500000007	0.0000000007
0.0000305176	-1.0500198366	0.0000198366	-1.0499801638	0.0000198362	-1.0500000002	0.0000000002

Table 01: step size, derivative approximations using the three methods and the absolute errors of the estimates of the three methods

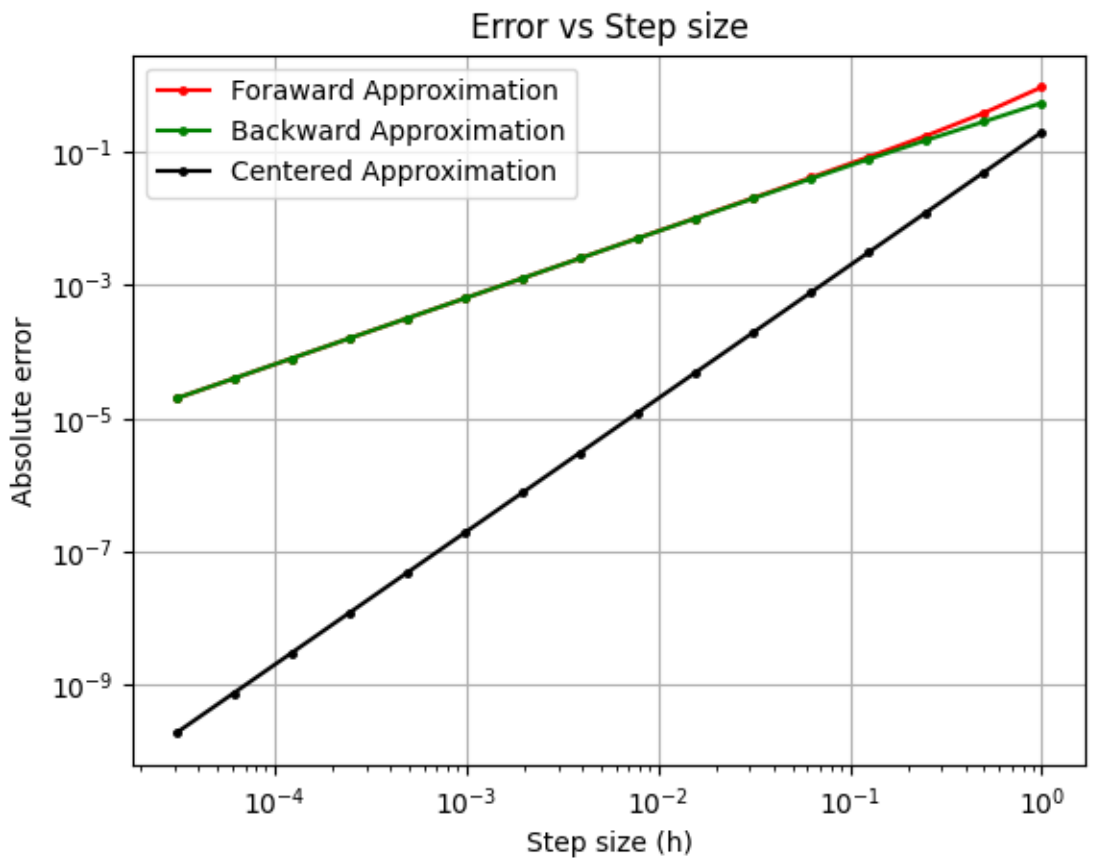

Figure 01: Graph of errors versus step size, in log-log scale

```python
import numpy as np
import matplotlib.pyplot as plt

# given function
fn = (lambda x: -0.1*(x**4)- 0.5*(x**2)-0.5*x+1.2)

# initial values
x0 = 0.5
h = 1
f_dash = -1.05  # f'(0.5)

# arrays to store data
H = np.array([])
fwd_Df = np.array([])
fwd_E  = np.array([])
bkd_Df = np.array([])
bkd_E  = np.array([])
ctr_Df = np.array([])
ctr_E  = np.array([])

# loop to calculate approximations and errors
while(h>= 1/2**15):

    # forrward difference approximation calculations
    df      = (fn(x0+h)-fn(x0))/h
    e       = abs(f_dash-df)
    fwd_Df  = np.append(fwd_Df, df)
    fwd_E   = np.append(fwd_E,e)

    # backward difference approximations calculations
    df      = (fn(x0)-fn(x0-h))/h
    e       = abs(f_dash-df)
    bkd_Df  = np.append(bkd_Df, df)
    bkd_E   = np.append(bkd_E,e)

    # centered difference approximations calculations
    df      = (fn(x0+h)-fn(x0-h))/(2*h)
    e       = abs(f_dash-df)
    ctr_Df  = np.append(ctr_Df, df)
    ctr_E   = np.append(ctr_E,e)
    #print("%.10f" %e) ---> for tabluation purpose
    # printing values in above way

    H       = np.append(H,h)
    h       = h/2

# printing plots
plt.plot(H,fwd_E, marker=".", ms =5, color="red")
plt.plot(H,bkd_E, marker=".", ms =5, color="green")
plt.plot(H,ctr_E, marker=".", ms =5, color="black")
plt.title("Error vs Step size")
plt.legend(["Foraward Approximation","Backward Approximation", "Centered
Approximation"])
plt.xlabel("Step size (h)")
plt.ylabel("Absolute error")
plt.xscale("log")
plt.yscale("log")
plt.grid()
plt.show()
```

Figure 02: python code for Activity 1 part a

- Each curve shows linear gradient. Hence the error is of order h (step size) for forward, backward, and centered difference approximations. The slop of the centered difference approximation is higher than the other approximations' curves. Therefore, the rate of absolute error is more in centered different approximation. As shown in Figure 01 centered difference approximation can achieve minimum absolute error when compared with forward and backward approximations curves.

(b) Estimate the second derivative of the function given in part (a) above at x= 0.5. Use the centered difference approximation and repeat the calculations as before until the step size reaches $1/2^{17}$ Given that f''(0.5) = −1.3, tabulate the values: step size, the second derivative approximation and the absolute error. As earlier, plot the error versus step size in log-log scale. Explain the behavior of the graph.

Step Size(h)	Centered difference approximation	
	f''(0.5)	Absolute error(e)
1.0000000000	-1.5000000000	0.2000000000
0.5000000000	-1.3500000000	0.0500000000
0.2500000000	-1.3125000000	0.0125000000
0.1250000000	-1.3031250000	0.0031250000
0.0625000000	-1.3007812500	0.0007812500
0.0312500000	-1.3001953125	0.0001953125
0.0156250000	-1.3000488281	0.0000488281
0.0078125000	-1.3000122070	0.0000122070
0.0039062500	-1.3000030518	0.0000030518
0.0019531250	-1.3000007629	0.0000007629
0.0009765625	-1.3000001907	0.0000001907
0.0004882812	-1.3000000478	0.0000000478
0.0002441406	-1.3000000119	0.0000000119
0.0001220703	-1.3000000045	0.0000000045
0.0000610352	-1.3000000119	0.0000000119
0.0000305176	-1.3000000715	0.0000000715
0.0000152588	-1.3000001907	0.0000001907
0.0000076294	-1.3000011444	0.0000011444

Table 02: Centered difference approximation for the second derivative

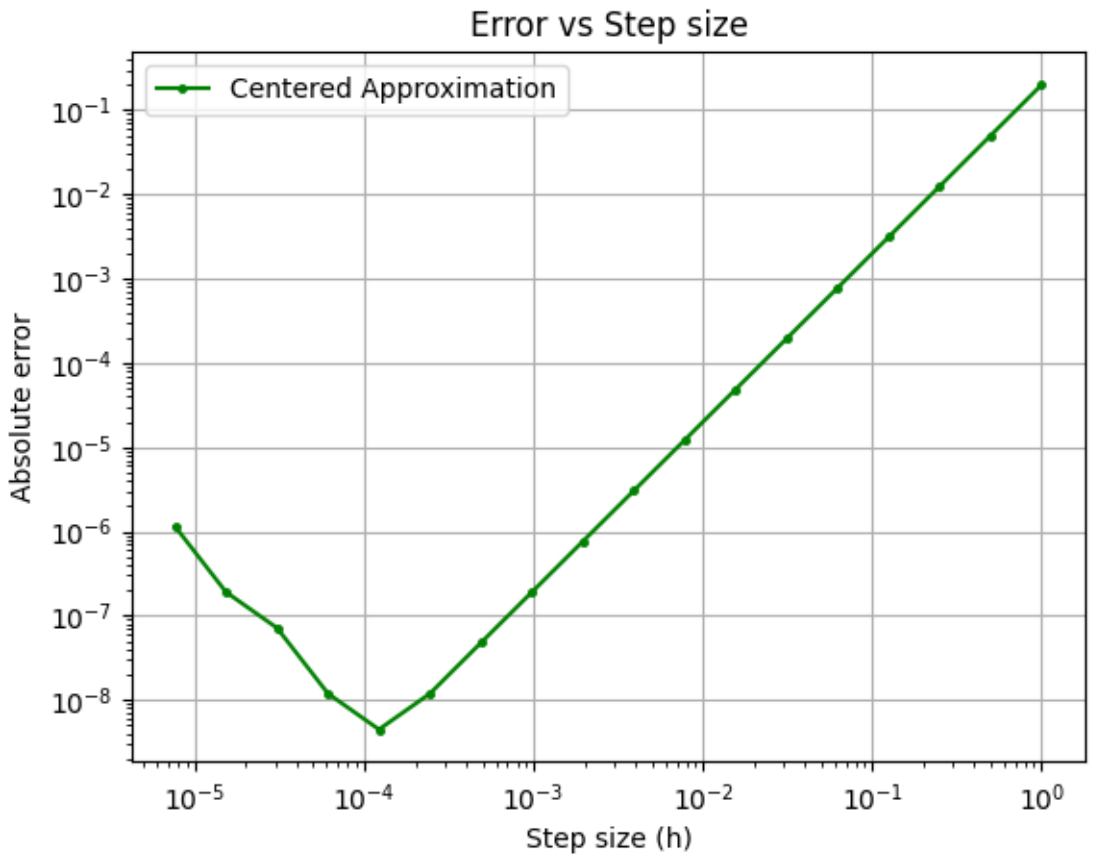

Figure 03: graph of error vs step size

- As seen in Figure 03 when the step size decreases the error linearly decreases. However, after a certain point decreasing the step size causes error to increase. This maybe occurred due to the fact that trying to decrease step size in order to minimize truncation error is causing an increase in round off error after a certain point. The total numerical error is the summation of the truncation and round-off errors. Generally, as the truncation errors decrease the round off errors increase. The optimal step size for a given computation would be the one that minimizes the sum of the two errors.

```
import numpy as np
import matplotlib.pyplot as plt

# given function
fn = (lambda x: -0.1*(x**4)- 0.5*(x**2)-0.5*x+1.2)

# initial values
x0 = 0.5
h =1
f_2dash = -1.3

# arrays to store calculateda values
H      = np.array([])
ctr_Df = np.array([])
ctr_E  = np.array([])

# loop to calculate centered difference approximations
while(h>= 1/2**17):
    df      = (fn(x0+h)-2*fn(x0)+fn(x0-h))/h**2
    e       = abs(f_2dash-df)
    ctr_Df  = np.append(ctr_Df, df)
    ctr_E   = np.append(ctr_E,e)
    H       = np.append(H,h)
    h       = h/2


# plotting the graphs
plt.plot(H, ctr_E, marker=".", ms =5, color="green")
plt.title("Error vs Step size")
plt.legend(["Centered Approximation"])
plt.xlabel("Step size (h)")
plt.ylabel("Absolute error")
plt.xscale("log")
plt.yscale("log")
plt.grid()
plt.show()
```

Figure 03: Python code for Activity 1 part b

(2) Consider the graphical method to approximate the roots of f(x) = 4 log (x) − x = 0 Considering the graphs of,

$f_1(x) = x$

$f_2(x) = 4 \log(x)$

$f_2'(x) = 4/x$

explain how/why the fixed-point iteration converges/diverges for starting values,

$x_0 < x_{r1}$, $x_{r1} < x_0 <= 4$ and $x_0 > 4 x_{r1} < x_{r2}$ are the two roots of f(x) = 0.

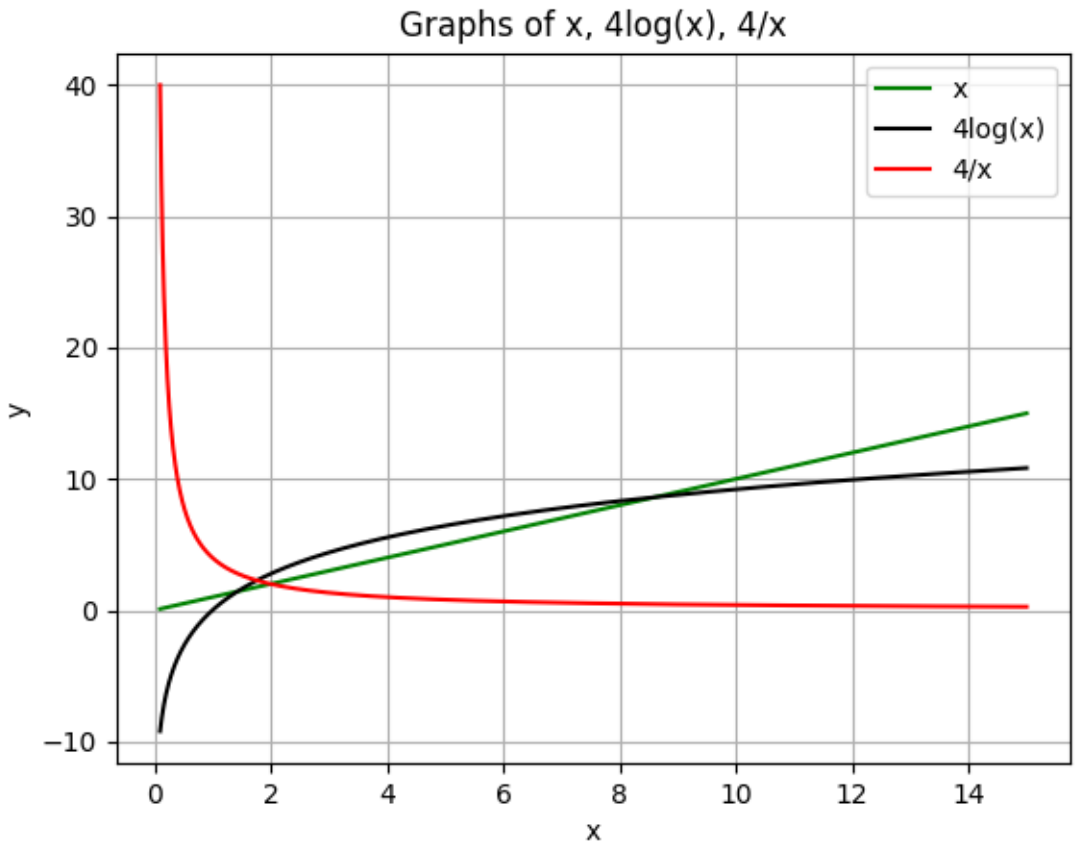

Figure 04: Graphs of $f_1(x) = x$, $f_2(x) = 4 \log(x)$, $f_2'(x) = 4/x$

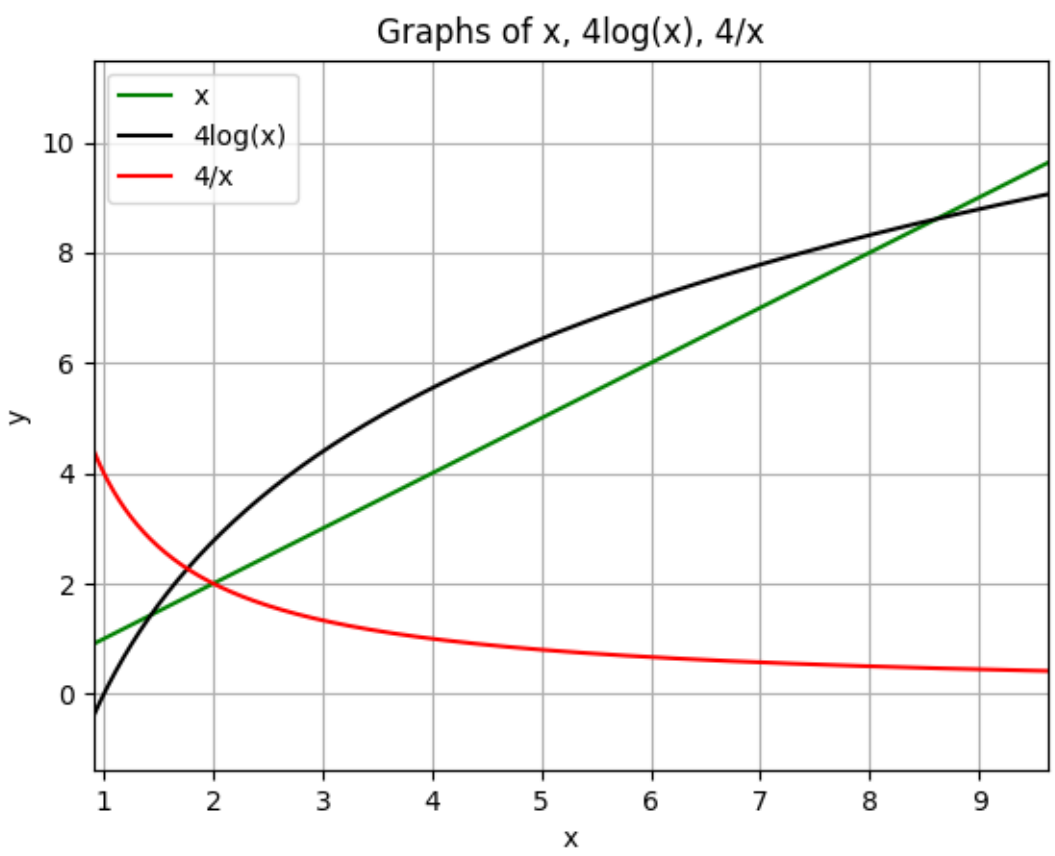

Figure 05: zoomed view of Figure 4

* for $x_0 < x_{r1}$,
consider interval where $x \in (0, x_{r1}]$
$f_2'(x) = 4/x$ is continuous in this interval. According to the figure as $|f_2'(x)| > 1$ for this interval.
Therefore, there does not exist a constant k such that

$$|f_2'(x)| \leq k < 1 \quad \text{where} \quad x \in (0, x_{r1}]$$

∴ does not satisfy fixed point theorem.
∴ sequence of iterates diverges.

* for $x_n < x_0 \leq 4$,
There also does not exist a constant k such that

$$|f_2'(x)| \leq k < 1 \quad \text{where} \quad x \in [x_{r1}, 4)$$

∴ does not satisfy fixed point theorem.
∴ sequence of iterates diverges.

* for $x_0 > 4$,
when $4 < x_0 \leq x_{r2}$ As seen in figure 05 there exist a constant k such that

$$|f_2'(x)| \leq k < 1 \quad \text{where} \quad x > 4$$

∴ In this interval follows fixed point theorem.
∴ sequence of iterates converges.

```python
import numpy as np
import matplotlib.pyplot as plt

# functions
f1x = np.linspace(0.1,15,2000)
f2x = 4 * np.log(f1x)
f2_dashx = 4/ f1x

# printing graphs
plt.plot(f1x, f1x , color = "green")
plt.plot(f1x, f2x, color = "black")
plt.plot(f1x, f2_dashx, color = "red")
plt.legend(["x","4log(x)","4/x"])
plt.xlabel("x")
plt.ylabel("y")
plt.title("Graphs of x, 4log(x), 4/x")
plt.grid()
plt.show()
```

Figure 06: python code for Activity 2