BOĞAZIÇI UNIVERSITY

NONLINEAR MODELS IN OPERATIONS RESEARCH
IE 440

Homework 1

Authors:
M. Akın Elden
Yunus Emre Karataş
Y. Harun Kıvrıl
Sefa Kayraklık

21 October 2019

Department of Industrial Engineering
Boğaziçi University

1 Introduction

The project is implemented using Python as the programming language. The given function is plotted between the interval [-3,9] and then its first and second order derivatives are taken using ”sympy” package.
The graph of the function:

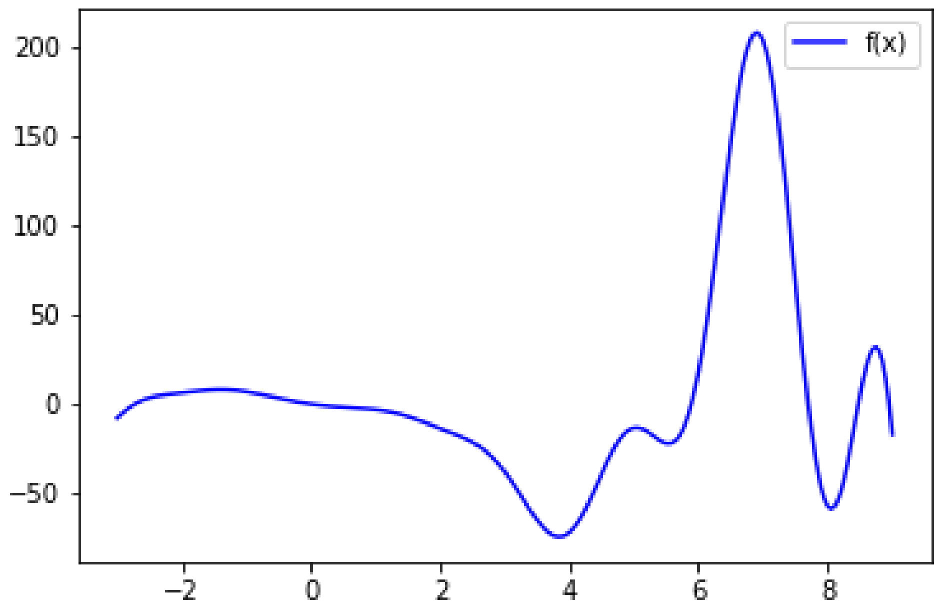

Figure 1: The graph of the given function

The source code used to import required dependencies, plot the graph, taking the derivatives of the function and calculating convergence rate of a result set:

```python
import numpy as np
from sympy import Symbol, cos, sin, lambdify
import matplotlib.pyplot as plt
import pandas as pd

x = Symbol('x')
function = x**3*cos(x)**2*sin(x)+3*x**2*cos(x)-5*x
first_deriv = function.diff(x)
second_deriv = first_deriv.diff(x)

first_deriv
# -2*x**3*sin(x)**2*cos(x) + x**3*cos(x)**3 + 3*x**2*sin(x)*cos(x)**2 - 3*x**2*sin(x) +
    6*x*cos(x) - 5

```

```python
second_deriv
# 2*x**3*sin(x)**3 - 7*x**3*sin(x)*cos(x)**2 - 12*x**2*sin(x)**2*cos(x) + 6*x**2*cos(x)
    **3 - 3*x**2*cos(x) + 6*x*sin(x)*cos(x)**2 - 12*x*sin(x) + 6*cos(x)

f = lambdify(x, function, 'numpy')
df = lambdify(x, first_deriv, 'numpy')
ddf = lambdify(x, second_deriv, 'numpy')

def plotGraphWithLines(x_pts=[], colors=[], labels=[]):
    t1 = np.arange(-3, 9.05, 0.05)
    plt.figure()
    plt.plot(t1, f(t1), 'b-', label='f(x)')
    for i in range(len(x_pts)):
        plt.axvline(x_pts[i], color=colors[i], label=labels[i])
    plt.legend()

plotGraphWithLines()
# Output: Graph of the function (Figure 1)
```

Since the convergence rate is calculated for all methods, a function is defined to calculate the convergence rate of a given result set:

```python
def c_rate(x, degree=1):
    res = [None]
    for i in range(1,(len(x) -1)):
        res.append(np.abs(x[i+1] - x[i])/np.abs(x[i] - x[i-1])**degree)
    return res
```

2 Bisection Method

Bisection method is used to find a local minimum (or maximum) point in a given interval. The method evaluates the function at the midpoint of the interval and compares it with the value found by evaluating the function at a point shifted by ε ahead. According to the comparison it removes the left or right half of the interval at each iteration until the interval becomes as small as ε. At each iteration, 2 function evaluation is done: $f(x)$ and $f(x+\varepsilon)$. Since the interval is halved at each iteration, the theoretical convergence rate is 0.5.

The source code used to implement the method :

```python
def BisectionMethod(a=-3,b=9,epsilon=0.001) :
    iteration=0
    res = []
    while (b - a) >= epsilon:
        x_1 = (a + b) / 2
        fx_1 = f(x_1)
        res.append([iteration, a, b, x_1, fx_1])
        if f(x_1 + epsilon) <= fx_1:
            a = x_1
        else:
            b = x_1
        iteration+=1
    x_star = (a+b)/2
    fx_star = f(x_star)
    res.append([iteration, a, b, x_star, fx_star])
    result_table = pd.DataFrame(res, columns=['iteration' ,'a', 'b', 'x', 'f(x)'])
    result_table['c_rate'] = pd.Series(c_rate(result_table.x))
    result_table["log_c_rate"] = -np.log(result_table.c_rate)
    return x_star, fx_star, result_table
```

Four different results obtained using four different parameter sets:

2.1 Bisection Model 1

The parameters used : a = -3, b = 9, ε = 0.001

Graph of the function :

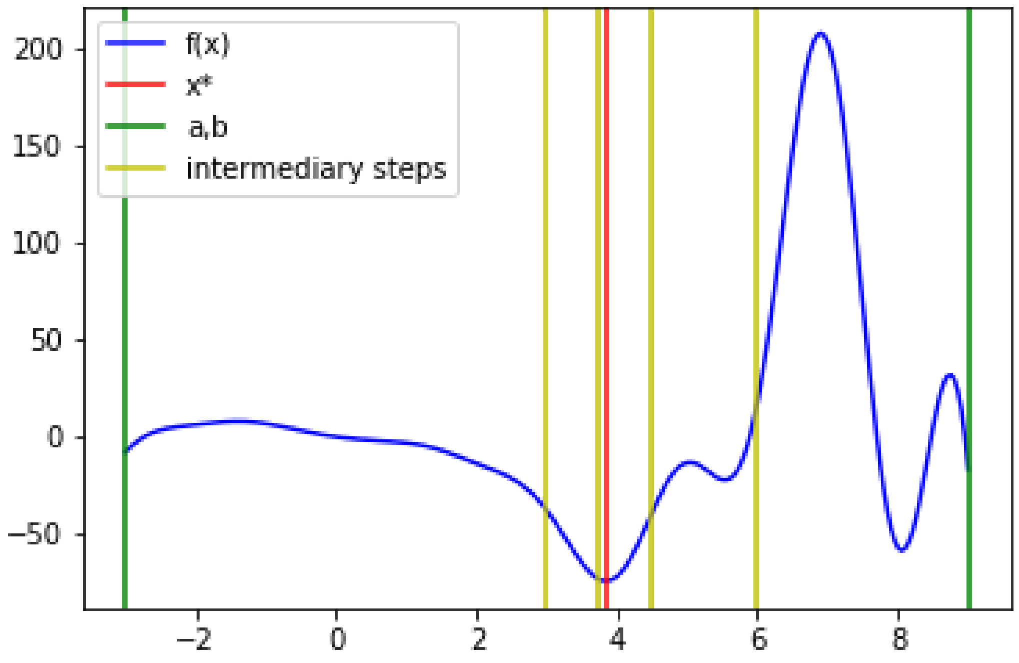

Figure 2: The graph with given parameters a and b,the found value x* and some intermediary steps

Results of the model :

Iteration	a	b	x	$f(x)$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert}$	$-log\lvert x^{(k+1)}-x^{(k)}\rvert$ $+log\lvert x^{(k)}-x^{(k-1)}\rvert$
0	-3.0000	9.0000	3.0000	-37.9954	NaN	NaN
1	3.0000	9.0000	6.0000	18.0566	0.5	0.6931
2	3.0000	6.0000	4.5000	-39.2640	0.5	0.6931
3	3.0000	4.5000	3.7500	-73.6618	0.5	0.6931
4	3.7500	4.5000	4.1250	-66.8585	0.5	0.6931
5	3.7500	4.1250	3.9375	-73.5793	0.5	0.6931
6	3.7500	3.9375	3.8438	-74.4355	0.5	0.6931
7	3.7500	3.8438	3.7969	-74.2444	0.5	0.6931
8	3.7969	3.8438	3.8203	-74.3901	0.5	0.6931
9	3.8203	3.8438	3.8320	-74.4255	0.5	0.6931
10	3.8320	3.8438	3.8379	-74.4337	0.5	0.6931
11	3.8379	3.8438	3.8408	-74.4354	0.5	0.6931
12	3.8408	3.8438	3.8423	-74.4356	0.5	0.6931
13	3.8408	3.8423	3.8416	-74.4356	0.5	0.6931
14	3.8416	3.8423	3.8419	-74.4356	NaN	NaN

$$x^* = 3.8419$$
$$f(x^*) = -74.4356$$

The source code to apply parameters :

```python
a=-3
b=9
x_star, fx_star, res = BisectionMethod(a,b,0.001)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
# Output : Figure 2
x_star
# 3.8419189453125
fx_star
# -74.43561182356329
```

The conclusion :

Algorithm found a local minimum point successfully and it is also the global minimum point in the interval. The local convergence rate is calculated as 0.5 for each iteration.

2.2 Bisection Model 2

The parameters used : a = -2, b = 8, ε = 0.001

Graph of the function :

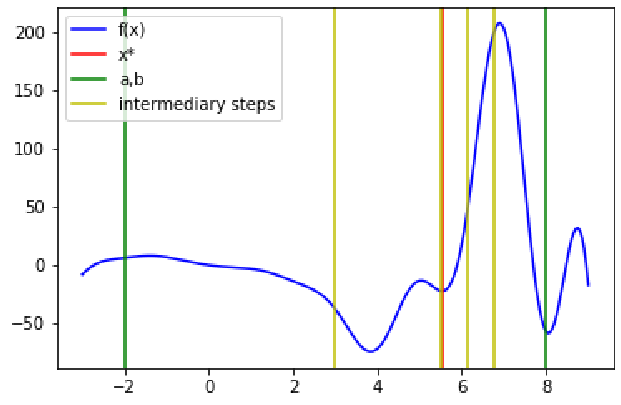

Figure 3: The graph with given parameters a and b, the found value x* and some intermediary steps

Results of the model :

Iteration	a	b	x	$f(x)$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert}$	$-log\lvert x^{(k+1)}-x^{(k)}\rvert$ $+log\lvert x^{(k)}-x^{(k-1)}\rvert$
0	-2.0000	8.0000	3.0000	-37.9954	NaN	NaN
1	3.0000	8.0000	5.5000	-22.1401	0.5	0.6931
2	5.5000	8.0000	6.7500	198.6890	0.5	0.6931
3	5.5000	6.7500	6.1250	45.2180	0.5	0.6931
4	5.5000	6.1250	5.8125	-9.4695	0.5	0.6931
5	5.5000	5.8125	5.6562	-20.1790	0.5	0.6931
6	5.5000	5.6562	5.5781	-22.0408	0.5	0.6931
7	5.5000	5.5781	5.5391	-22.2804	0.5	0.6931
8	5.5000	5.5391	5.5195	-22.2538	0.5	0.6931
9	5.5195	5.5391	5.5293	-22.2785	0.5	0.6931
10	5.5293	5.5391	5.5342	-22.2823	0.5	0.6931
11	5.5342	5.5391	5.5366	-22.2821	0.5	0.6931
12	5.5342	5.5366	5.5354	-22.2824	0.5	0.6931
13	5.5342	5.5354	5.5348	-22.2824	0.5	0.6931
14	5.5342	5.5348	5.5345	-22.2824	NaN	NaN

$$x^* = 5.5345$$
$$f(x^*) = -22.2824$$

The source code to apply parameters :

```python
a=-2
b=8
x_star, fx_star, res = BisectionMethod(a,b,0.001)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
# Output : Figure 3
x_star
# 5.53448486328125
fx_star
# -22.28237461176161
```

The conclusion :
Algorithm found a local minimum point but it isn't the global minimum point in the interval. Because at the second iteration, the algorithm evaluated the function at descending side of a local minimum so the interval's left side is removed which also contains the global minimum of the interval. Then the algorithm found another local minimum. The local convergence rate is calculated as 0.5 for each iteration.

2.3 Bisection Model 3

The parameters used : a = 2, b = 9, $\varepsilon = 0.001$
Graph of the function :

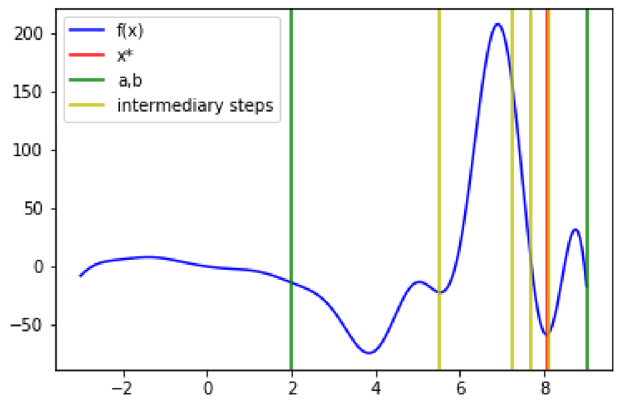

Figure 4: The graph with given parameters a and b, the found value x* and some intermediary steps

Results of the model :

Iteration	a	b	x	$f(x)$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert}$	$-log\lvert x^{(k+1)}-x^{(k)}\rvert$ $+log\lvert x^{(k)}-x^{(k-1)}\rvert$
0	2.0000	9.0000	5.5000	-22.1401	NaN	NaN
1	5.5000	9.0000	7.2500	154.4712	0.5	0.6931
2	7.2500	9.0000	8.1250	-56.6057	0.5	0.6931
3	7.2500	8.1250	7.6875	3.2458	0.5	0.6931
4	7.6875	8.1250	7.9062	-47.9814	0.5	0.6931
5	7.9062	8.1250	8.0156	-57.9338	0.5	0.6931
6	8.0156	8.1250	8.0703	-58.6388	0.5	0.6931
7	8.0156	8.0703	8.0430	-58.6375	0.5	0.6931
8	8.0430	8.0703	8.0566	-58.7251	0.5	0.6931
9	8.0430	8.0566	8.0498	-58.7031	0.5	0.6931
10	8.0498	8.0566	8.0532	-58.7196	0.5	0.6931
11	8.0532	8.0566	8.0549	-58.7237	0.5	0.6931
12	8.0549	8.0566	8.0558	-58.7247	0.5	0.6931
13	8.0558	8.0566	8.0562	-58.7250	NaN	NaN

$$x^* = 8.0562$$
$$f(x^*) = -58.7250$$

The source code to apply parameters :

```python
a=2
b=9
x_star, fx_star, res = BisectionMethod(a,b,0.001)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
# Output : Figure 4
x_star
# 8.05621337890625
fx_star
# -58.72499046391924
```

The conclusion :

Algorithm found a local minimum point but it isn't the global minimum point in the interval. Because at the first iteration, the algorithm evaluated the function at descending side of a local minimum so the interval's left side is removed which also contains the global minimum of the interval. In the next iteration algorithm evaluated the function at the right side of the local max and removed the left interval. So it found another local minimum at x=8.0562. The local convergence rate is calculated as 0.5 for each iteration.

2.4 Bisection Model 4

The parameters used : a = 2.1, b = 9, ε = 0.001
Graph of the function :

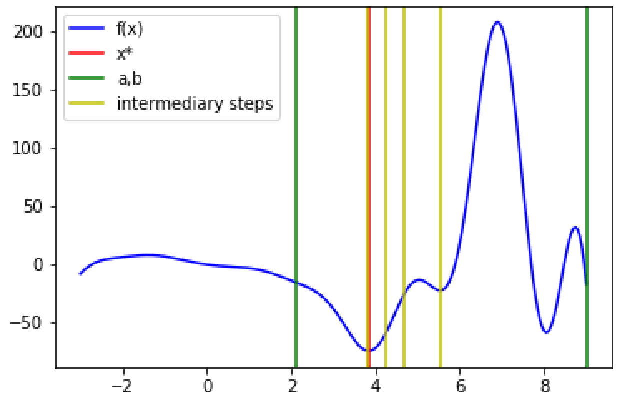

Figure 5: The graph with given parameters a and b, the found value x* and some intermediary steps

Results of the model :

Iteration	a	b	x	$f(x)$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert}$	$-log\lvert x^{(k+1)}-x^{(k)}\rvert$ $+log\lvert x^{(k)}-x^{(k-1)}\rvert$
0	2.1000	9.0000	5.5500	-22.2543	NaN	NaN
1	2.1000	5.5500	3.8250	-74.4073	0.5	0.6931
2	3.8250	5.5500	4.6875	-25.1417	0.5	0.6931
3	3.8250	4.6875	4.2562	-58.6512	0.5	0.6931
4	3.8250	4.2562	4.0406	-70.6875	0.5	0.6931
5	3.8250	4.0406	3.9328	-73.6622	0.5	0.6931
6	3.8250	3.9328	3.8789	-74.3111	0.5	0.6931
7	3.8250	3.8789	3.8520	-74.4273	0.5	0.6931
8	3.8250	3.8520	3.8385	-74.4341	0.5	0.6931
9	3.8385	3.8520	3.8452	-74.4349	0.5	0.6931
10	3.8385	3.8452	3.8418	-74.4356	0.5	0.6931
11	3.8418	3.8452	3.8435	-74.4355	0.5	0.6931
12	3.8418	3.8435	3.8427	-74.4356	0.5	0.6931
13	3.8418	3.8427	3.8423	-74.4356	NaN	NaN

$$x^* = 3.8423$$
$$f(x^*) = -74.4356$$

The source code to apply parameters :

```
a=2.1
b=9
x_star, fx_star, res = BisectionMethod(a,b,0.001)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
# Output : Figure 5
x_star
# 3.8422668457031253
fx_star
# -74.43563736033886
```

The conclusion :
Just changing the a value by 0.1 compared to the previous model, the algorithm found the global minimum of the interval. Because at the first iteration, the algorithm evaluated the function at ascending side of the local minimum not descending side of it, so the interval's right side is removed. At the end, global minimum of the interval is located. The local convergence rate is calculated as 0.5 for each iteration. So these examples show that Bisection method locally converges and it doesn't guarantee the global minimum.

3 Golden Section Method

The source code used to implement the method :

```
def GoldenSection(a,b,epsilon):
    golden_ratio = (1+np.sqrt(5))/2
    gama = 1/golden_ratio
    iteration = 0
    x_1 = b - gama*(b-a)
    x_2 = a + gama*(b-a)
    fx_1 = f(x_1)
    fx_2 = f(x_2)
    res = [[iteration, a, b, x_1, x_2, fx_1, fx_2]]
    while (b-a) >= epsilon:
        iteration+=1
        if(fx_1 >= fx_2):
            a = x_1
            x_1 = x_2
            x_2 = a + gama*(b-a)
            fx_1 = fx_2
            fx_2 = f(x_2)
        else:
            b = x_2
            x_2 = x_1
            x_1 = b - gama*(b-a)
            fx_2 = fx_1
            fx_1 = f(x_1)
        res.append([iteration, a, b, x_1, x_2, fx_1, fx_2])
    result_table = pd.DataFrame(res, columns = ['iteration', 'a', 'b', 'x', 'y', 'f(x)',
        'f(y)'])
    result_table['c_rate'] = pd.Series(c_rate((result_table.a + result_table.b)/2))
    result_table["log_c_rate"] = -np.log(result_table.c_rate)
    x_star = (a+b)/2
    fx_star = f(x_star)
    return x_star, fx_star, result_table
```

Four different results obtained through four different parameters:

3.1 Golden Section Model 1

The parameters used : a = -3, b = 9, $\varepsilon = 0.001$

Graph of the function :

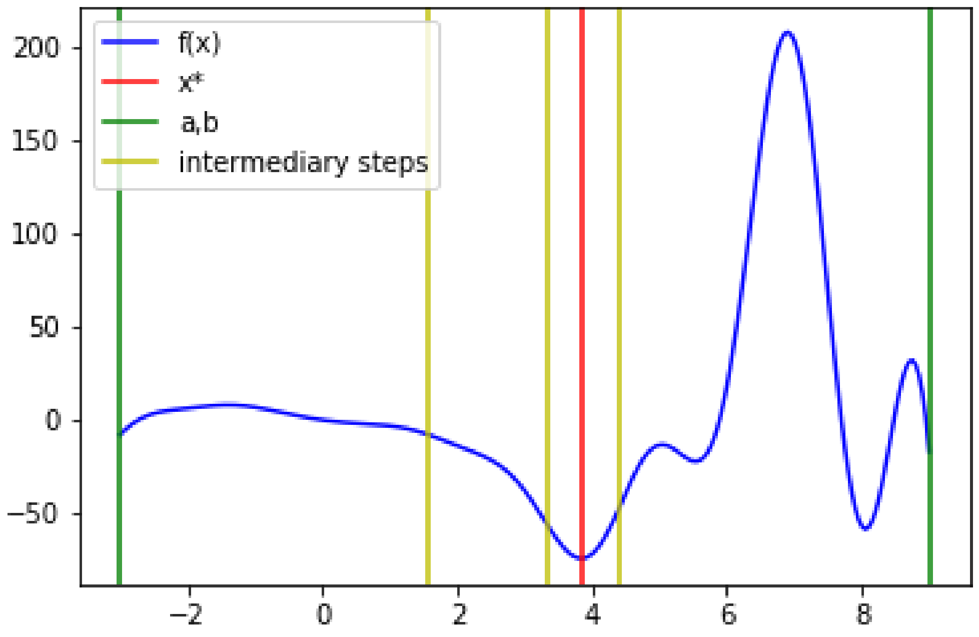

Figure 6: The graph with given parameters a,b and x* value found with model

Results of the model :

Iteration	*a*	*b*	*x*	*y*	*f(x)*	*f(y)*	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert}$	$-log\lvert x^{(k+1)}-x^{(k)}\rvert + log\lvert x^{(k)}-x^{(k-1)}\rvert$
0	-3.0000	9.0000	1.5836	4.4164	-8.0136	-46.1592	NaN	NaN
1	1.5836	9.0000	4.4164	6.1672	-46.1592	55.7148	0.6180	0.4812
2	1.5836	6.1672	3.3344	4.4164	-56.2497	-46.1592	0.6180	0.4812
3	1.5836	4.4164	2.6656	3.3344	-25.4190	-56.2497	0.6180	0.4812
4	2.6656	4.4164	3.3344	3.7477	-56.2497	-73.6230	0.6180	0.4812
5	3.3344	4.4164	3.7477	4.0031	-73.6230	-71.9743	0.6180	0.4812
6	3.3344	4.0031	3.5898	3.7477	-69.0728	-73.6230	0.6180	0.4812
7	3.5898	4.0031	3.7477	3.8452	-73.6230	-74.4349	0.6180	0.4812
8	3.7477	4.0031	3.8452	3.9055	-74.4349	-74.0604	0.6180	0.4812
9	3.7477	3.9055	3.8080	3.8452	-74.3257	-74.4349	0.6180	0.4812
10	3.8080	3.9055	3.8452	3.8683	-74.4349	-74.3734	0.6180	0.4812
11	3.8080	3.8683	3.8310	3.8452	-74.4234	-74.4349	0.6180	0.4812
12	3.8310	3.8683	3.8452	3.8540	-74.4349	-74.4232	0.6180	0.4812
13	3.8310	3.8540	3.8398	3.8452	-74.4350	-74.4349	0.6180	0.4812
14	3.8310	3.8452	3.8364	3.8398	-74.4322	-74.4350	0.6180	0.4812
15	3.8364	3.8452	3.8398	3.8419	-74.4350	-74.4356	0.6180	0.4812
16	3.8398	3.8452	3.8419	3.8432	-74.4356	-74.4356	0.6180	0.4812
17	3.8398	3.8432	3.8411	3.8419	-74.4355	-74.4356	0.6180	0.4812
18	3.8411	3.8432	3.8419	3.8424	-74.4356	-74.4356	0.6180	0.4812
19	3.8419	3.8432	3.8424	3.8427	-74.4356	-74.4356	0.6180	0.4812
20	3.8419	3.8427	3.8422	3.8424	-74.4356	-74.4356	NaN	NaN

$$x^* = 3.8423$$
$$f(x^*) = -74.4356$$

The source code to apply parameters :

```python
a=-3
b=9
epsilon=0.001
x_star, fx_star, res = GoldenSection(a,b,epsilon)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,3)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
plt.savefig('golden_3.png')
round(x_star,4),round(fx_star,4)
```

The conclusion :
When we evaluate the function over its own domain, Golden Section Method converges to global minimum.The local convergence rate is 0.618 and there are 20 iterations made to find optimal solution.Moreover,this algorithm is more costly compared to Bisection Method because it evaluates 2 function per iteration.These outputs verify our theoretical knowledge about the Golden-Section Method.

3.2 Golden Section Model 2

The parameters used : a = 3.5, b = 9, $\varepsilon = 0.001$
Graph of the function :

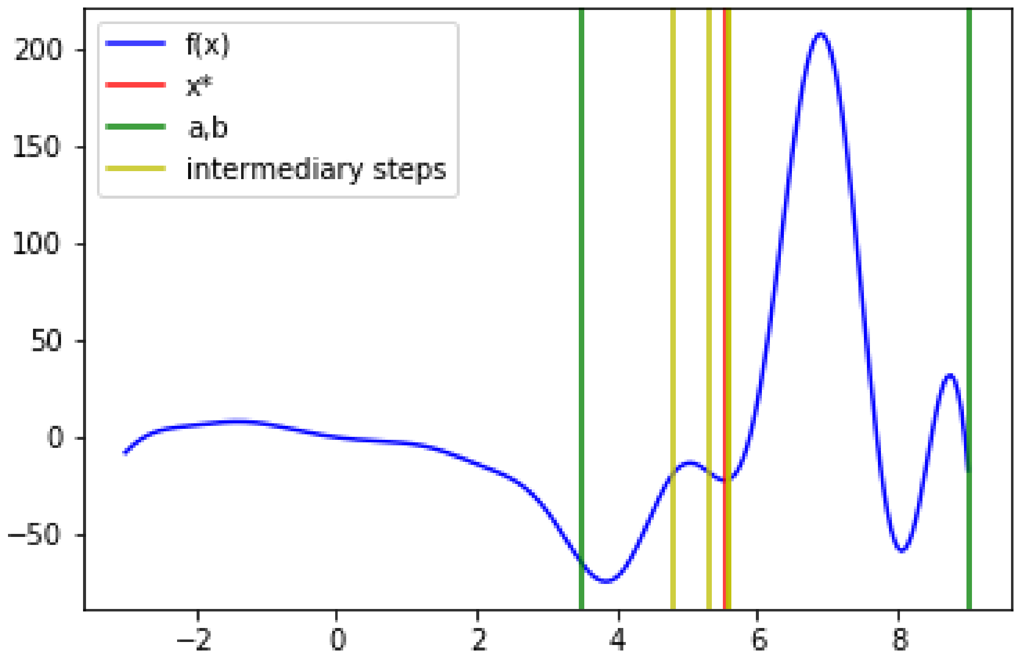

Figure 7: The graph with given parameters a,b and x* value found with model

Results of the model :

Iteration	a	b	x	y	$f(x)$	$f(y)$	$\frac{\|x^{(k+1)}-x^{(k)}\|}{\|x^{(k)}-x^{(k-1)}\|}$	$-log\|x^{(k+1)}-x^{(k)}\| + log\|x^{(k)}-x^{(k-1)}\|$
0	3.5000	9.0000	5.6008	6.8992	-21.7028	208.4521	NaN	NaN
1	3.5000	6.8992	4.7984	5.6008	-18.8717	-21.7028	0.6180	0.4812
2	4.7984	6.8992	5.6008	6.0967	-21.7028	38.5323	0.6180	0.4812
3	4.7984	6.0967	5.2943	5.6008	-17.7050	-21.7028	0.6180	0.4812
4	5.2943	6.0967	5.6008	5.7902	-21.7028	-11.6387	0.6180	0.4812
5	5.2943	5.7902	5.4837	5.6008	-21.9848	-21.7028	0.6180	0.4812
6	5.2943	5.6008	5.4114	5.4837	-20.7522	-21.9848	0.6180	0.4812
7	5.4114	5.6008	5.4837	5.5285	-21.9848	-22.2772	0.6180	0.4812
8	5.4837	5.6008	5.5285	5.5561	-22.2772	-22.2263	0.6180	0.4812
9	5.4837	5.5561	5.5114	5.5285	-22.2164	-22.2772	0.6180	0.4812
10	5.5114	5.5561	5.5285	5.5390	-22.2772	-22.2804	0.6180	0.4812
11	5.5285	5.5561	5.5390	5.5455	-22.2804	-22.2686	0.6180	0.4812
12	5.5285	5.5455	5.5350	5.5390	-22.2824	-22.2804	0.6180	0.4812
13	5.5285	5.5390	5.5325	5.5350	-22.2816	-22.2824	0.6180	0.4812
14	5.5325	5.5390	5.5350	5.5365	-22.2824	-22.2821	0.6180	0.4812
15	5.5325	5.5365	5.5340	5.5350	-22.2823	-22.2824	0.6180	0.4812
16	5.5340	5.5365	5.5350	5.5356	-22.2824	-22.2824	0.6180	0.4812
17	5.5340	5.5356	5.5346	5.5350	-22.2824	-22.2824	0.6180	0.4812
18	5.5346	5.5356	5.5350	5.5352	-22.2824	-22.2824	NaN	NaN

$$x^* = 5.5351$$
$$f(x^*) = -22.2824$$

The source code to apply parameters :

```python
a=3.5
b=9
epsilon=0.001
x_star, fx_star, res = GoldenSection(a,b,epsilon)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
plt.savefig('golden_1.png')
round(x_star,4),round(fx_star,4)
```

The conclusion :
When we evaluate the function over the domain [3.5,9], Golden Section Method diverges to the closest local minimum point of the function.F(y) value is lower than F(x) value in the first iteration,and therefore the part that includes global min is removed in the first iteration.Moreover, convergence rate is the same as the previous model, so we can conclude that converge rate does not depend on the searched domain.

3.3 Golden Section Model 3

The parameters used : a = -3, b = 9, ε = 0.669
Graph of the function :

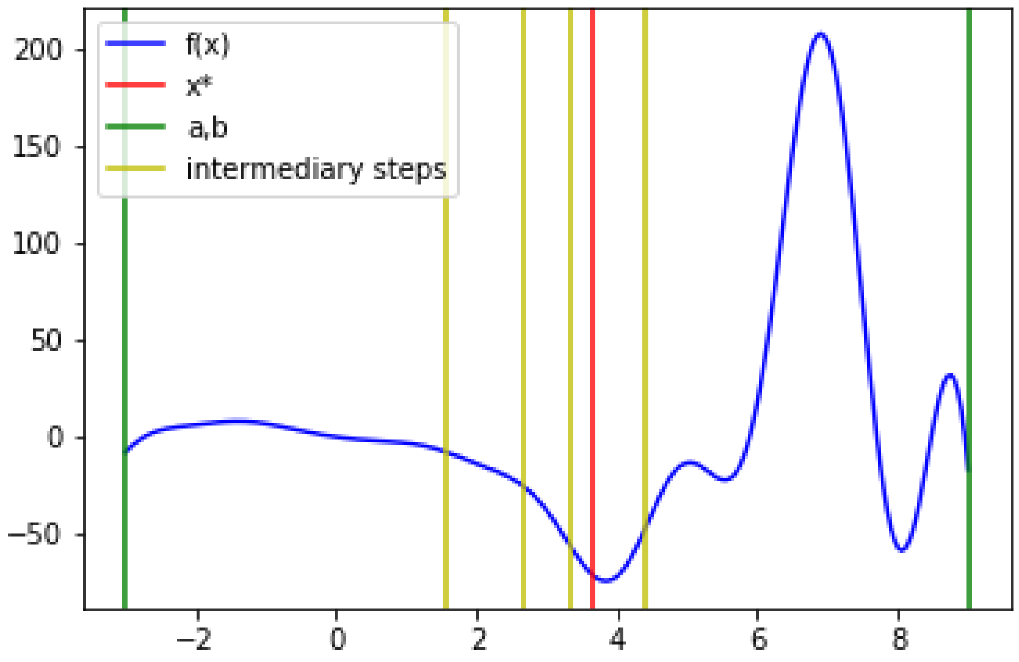

Figure 8: The graph with given parameters a,b and x* value found with model

Results of the model :

Iteration	a	b	x	y	$f(x)$	$f(y)$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert}$	$-log\lvert x^{(k+1)}-x^{(k)}\rvert$ $+log\lvert x^{(k)}-x^{(k-1)}\rvert$
0	-3.0000	9.0000	1.5836	4.4164	-8.0136	-46.1592	NaN	NaN
1	1.5836	9.0000	4.4164	6.1672	-46.1592	55.7148	0.6180	0.4812
2	1.5836	6.1672	3.3344	4.4164	-56.2497	-46.1592	0.6180	0.4812
3	1.5836	4.4164	2.6656	3.3344	-25.4190	-56.2497	0.6180	0.4812
4	2.6656	4.4164	3.3344	3.7477	-56.2497	-73.6230	0.6180	0.4812
5	3.3344	4.4164	3.7477	4.0031	-73.6230	-71.9743	0.6180	0.4812
6	3.3344	4.0031	3.5898	3.7477	-69.0728	-73.6230	NaN	NaN

$$x^* = 3.6687$$
$$f(x^*) = -71.7956$$

The source code to apply parameters :

```python
a=-3
b=9
epsilon=0.669
x_star, fx_star, res = GoldenSection(a,b,epsilon)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
plt.savefig('golden_2.png')
round(x_star,4),round(fx_star,4)
```

The conclusion :

When we evaluate the function with epsilon value=0.669, it diverges slightly from the global min because the accuracy of the algorithm is not enough to detect small changes in the value of the function.However,it takes less iteration to converge,so we can conclude that Golden Section method gives good enough solution with less iteration if we increase the epsilon value.

3.4 Golden Section Model 4

The parameters used : a = 3.80, b = 8, ε = 0.001
Graph of the function :

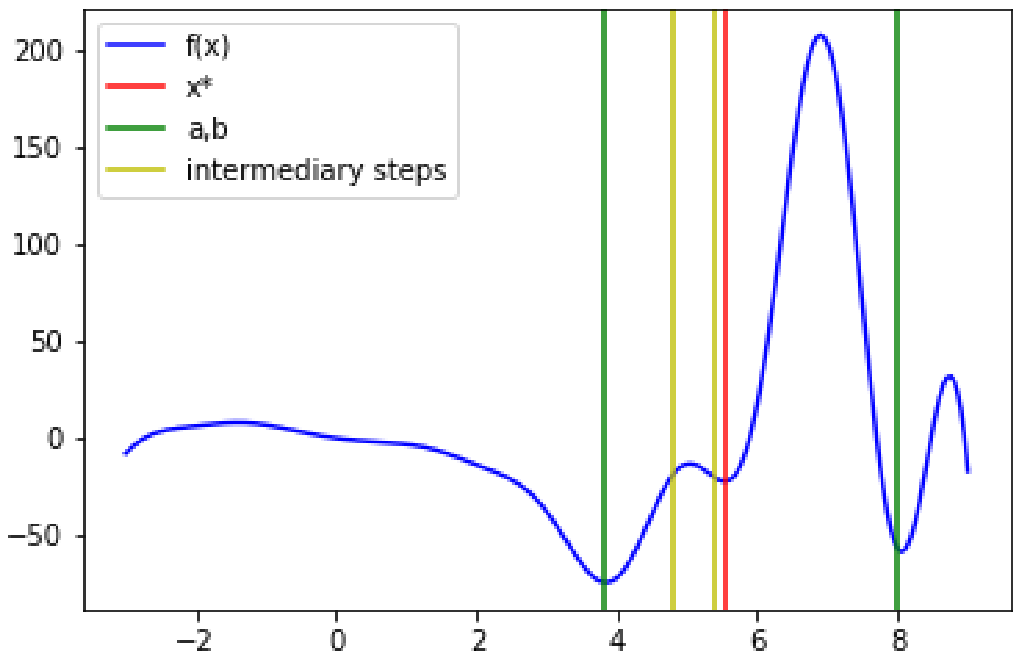

Figure 9: The graph with given parameters a,b and x* value found with model

Results of the model :

Iteration	a	b	x	y	$f(x)$	$f(y)$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert}$	$-log\lvert x^{(k+1)}-x^{(k)}\rvert$ $+log\lvert x^{(k)}-x^{(k-1)}\rvert$
0	3.8000	8.0000	5.4043	6.3957	-20.5928	118.9759	NaN	NaN
1	3.8000	6.3957	4.7915	5.4043	-19.2000	-20.5928	0.6180	0.4812
2	4.7915	6.3957	5.4043	5.7830	-20.5928	-12.2987	0.6180	0.4812
3	4.7915	5.7830	5.1702	5.4043	-14.6251	-20.5928	0.6180	0.4812
4	5.1702	5.7830	5.4043	5.5489	-20.5928	-22.2583	0.6180	0.4812
5	5.4043	5.7830	5.5489	5.6383	-22.2583	-20.7867	0.6180	0.4812
6	5.4043	5.6383	5.4937	5.5489	-22.0858	-22.2583	0.6180	0.4812
7	5.4937	5.6383	5.5489	5.5831	-22.2583	-21.9804	0.6180	0.4812
8	5.4937	5.5831	5.5278	5.5489	-22.2761	-22.2583	0.6180	0.4812
9	5.4937	5.5489	5.5148	5.5278	-22.2338	-22.2761	0.6180	0.4812
10	5.5148	5.5489	5.5278	5.5359	-22.2761	-22.2823	0.6180	0.4812
11	5.5278	5.5489	5.5359	5.5409	-22.2823	-22.2782	0.6180	0.4812
12	5.5278	5.5409	5.5328	5.5359	-22.2818	-22.2823	0.6180	0.4812
13	5.5328	5.5409	5.5359	5.5378	-22.2823	-22.2815	0.6180	0.4812
14	5.5328	5.5378	5.5347	5.5359	-22.2824	-22.2823	0.6180	0.4812
15	5.5328	5.5359	5.5340	5.5347	-22.2823	-22.2824	0.6180	0.4812
16	5.5340	5.5359	5.5347	5.5351	-22.2824	-22.2824	0.6180	0.4812
17	5.5347	5.5359	5.5351	5.5354	-22.2824	-22.2824	0.6180	0.4812
18	5.5347	5.5354	5.5350	5.5351	-22.2824	-22.2824	NaN	NaN

$$x^* = 5.5351$$
$$f(x^*) = -22.2824$$

The source code to apply parameters :

```python
a=3.80
b=8
epsilon=0.001
x_star, fx_star, res = GoldenSection(a,b,epsilon)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,2)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
plt.savefig('golden_4.png')
round(x_star,4),round(fx_star,4)
```

The conclusion :

We shorten the search interval compared to Model 2 and evaluate the function over the domain

[3.80,8].However,we fail to reach global minimum and it diverges to the closest local minimum point again.Even if Golden Section algorithm removes the right side of the interval correctly in the first iteration,the part that includes global minimum is removed in the second iteration because F(y) is smaller than F(x) in that iteration.

4 Newton's Method

The Newton's method finds the local extremum by approximating the given function using its first and second derivatives with the assumption of the continuous twice differentiability of it. First, it determines the stationary point of the function which is approximated according to the given initial point. Then it updates the initial point with the found stationary point. It repeats these steps until the difference between the stationary point and the point that is used to approximate the function is lower than a given ε. It has a quadratic convergence rate so it can reach the local extremum with a few iteration; and it uses two function evaluations per iteration.
Since in the homework, the local minimum point is asked to find, the second derivative of the found extremum point is check whether it is negative or positive. If it is positive, the extremum point is local minimum, if negative, the extremum point is local maximum.

The source code used to implement the method :

```python
def NewtonsMethod(a, b, x_0, epsilon):
    iteration = 0
    res = []
    while True:
        dfx0 = df(x_0)
        ddfx0 = ddf(x_0)
        x_1 = x_0-dfx0/ddfx0
        res.append([iteration, x_0, f(x_0), dfx0, ddfx0])
        iteration +=1
        if abs(x_0-x_1)<epsilon:
            break
        else:
            x_0 = x_1
        if x_1<a or x_1>b:
            print("Error: The Newton\'s method is not able to find any local minimum in
                the given range")
            break
    res.append([iteration, x_1, f(x_1), df(x_1), ddf(x_1)])
    result_table = pd.DataFrame(res, columns = ['iteration', 'x', 'f(x)', "f'(x)", "f''(
        x)"])
    result_table['c_rate'] = pd.Series(c_rate(result_table.x, 2))
    x_star = x_1
    fx_star = f(x_1)
    return x_star, fx_star, result_table
```

The source code to apply parameters :

```
newton_par = [[3.5, 0.001], [6.6, 0.0001], [7.5, 0.001], [7.6, 0.005]]
a= -3
b= 9
for par in newton_par:
    x_star, fx_star, res = NewtonsMethod(a, b, par[0], par[1])
    lines = [x_star,par[0],a,b]
    colors = ['r','y','g','g']
    labels = ['x*','x0','a,b','']
    plotGraphWithLines(lines,colors,labels, 'newton' + str(par[0]))
    print(res.to_latex(index=False,float_format='%.4f'))
    print(round(x_star,4),round(fx_star,4))
```

Four different results obtained through four different parameters:

4.1 Newton's Model 1

The parameters used : a = -3, b = 9, x0 = 3.5, ε = 0.001

Graph of the function :

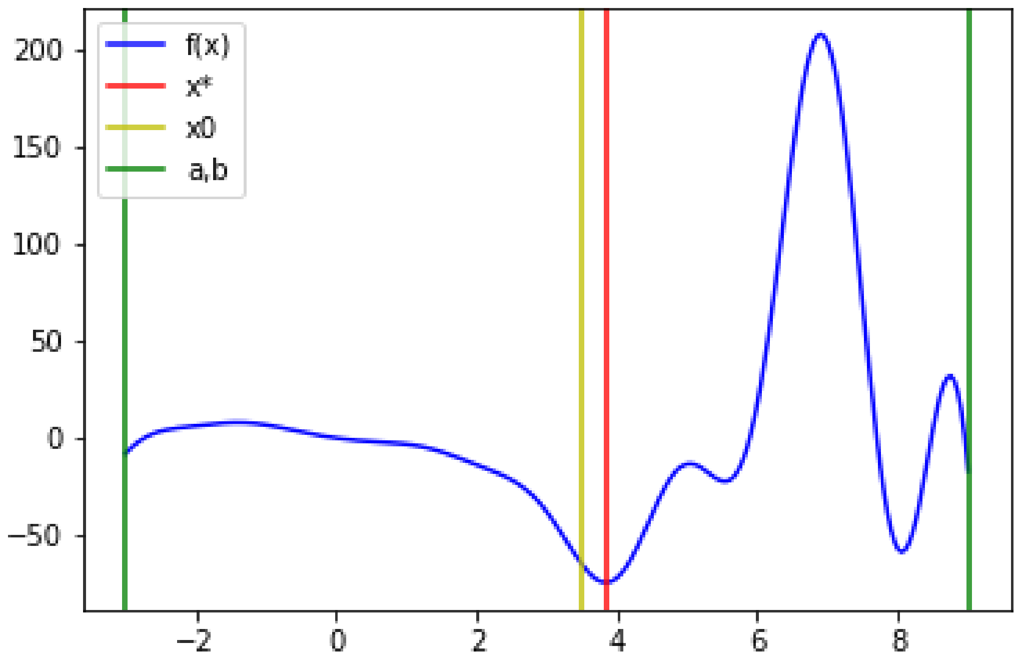

Figure 10: The graph with given parameters a,b and x* value found with model

Results of the model :

Iteration	$x^{(k)}$	$f_k(x^{(k)})$	$f_k^{'}(x^{(k)})$	$f_k^{''}(x^{(k)})$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert^2}$
0	3.5000	-65.1040	-48.4085	82.2707	NaN
1	4.0884	-68.6732	46.4354	175.4698	0.7644
2	3.8238	-74.4032	-3.4597	183.3522	0.2694
3	3.8426	-74.4356	0.0283	186.2723	0.4265
4	3.8425	-74.4356	0.0000	186.2507	NaN

$$x^* = 3.8425$$
$$f(x^*) = -74.4356$$
$$f^{''}(x^*) = 186.2507$$

The conclusion :
The Newton's method can find the local minimum point of the given function since the initial point is close enough to find the local minimum point and the function is convex in the range [3 5]. It can be seen that it is also the global minimum point in the given range [-3 9], by chance. And since its second derivative is positive, it is a local minimum point.

4.2 Newton's Model 2

The parameters used : a = -3, b = 9, x0 = 6.6, ε = 0.001
Graph of the function :

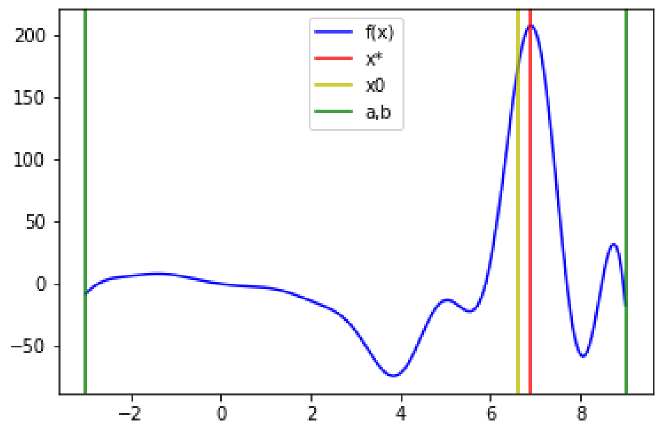

Figure 11: The graph with given parameters a,b and x* value found with model

Results of the model :

Iteration	$x^{(k)}$	$f_k(x^{(k)})$	$f_k^{'}(x^{(k)})$	$f_k^{''}(x^{(k)})$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert^2}$
0	6.6000	172.0501	222.3206	-504.7001	NaN
1	7.0405	199.4615	-127.9609	-896.2071	0.7358
2	6.8977	208.4499	2.1716	-901.3016	0.1182
3	6.9001	208.4525	-0.0014	-902.4538	0.2665
4	6.9001	208.4525	-0.0000	-902.4531	NaN

$$x^* = 6.9001$$
$$f(x^*) = 208.4525$$
$$f^{''}(x^*) = -902.4531$$

The conclusion :

The Newton's method can find the local maximum point of the given function since the initial point is close enough to find the local maximum point and the function is concave in the range [6.5 7.5]. It can be seen that it is also the global maximum point in the given range [-3 9], by chance. And changing the ε makes the result more accurate but since the precision to represent the numbers is not so large, the ε is not needed to be so much small.

The Newton's method is not able to find a local minimum point of the given function with the initial point x_0 = 3.5, since the second derivative of the extremum point is negative.

4.3 Newton's Model 3

The parameters used : a = -3, b = 9, x0 = 7.5, ε = 0.001
Graph of the function :

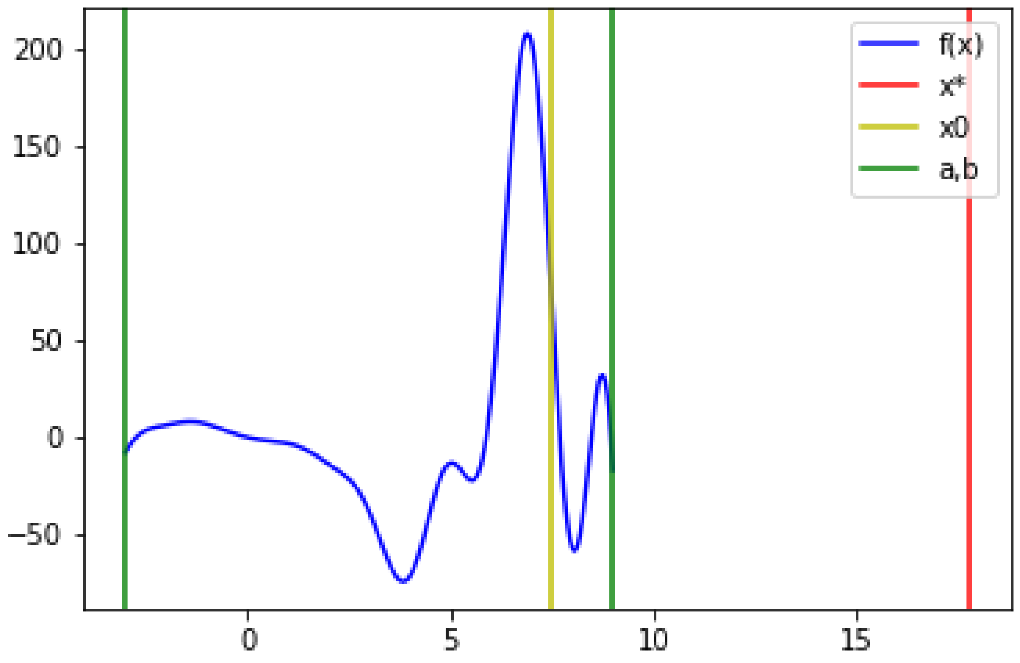

Figure 12: The graph with given parameters a,b and x* value found with model

Results of the model :

Iteration	$x^{(k)}$	$f_k(x^{(k)})$	$f_k'(x^{(k)})$	$f_k''(x^{(k)})$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert^2}$
0	7.5000	68.5427	-368.4296	35.9345	None
1	17.7528	-694.5034	-2801.9112	-2065.8252	NaN

The Newton's method is not able to find any local minimum in the given range

The conclusion :
The Newton's method can not find a local extremum in the given range since the stationary point of the first approximated function is not in the given range, so the algorithm exceeds the boundaries. Because in the question description it is asked to find a local minimum point in the given range, it can be said that the method fails to find a local minimum point with the initial point $x_0 = 7.5$.

4.4 Newton's Model 4

The parameters used : a = -3, b = 9, x0 = 7.6, ε = 0.005

Graph of the function :

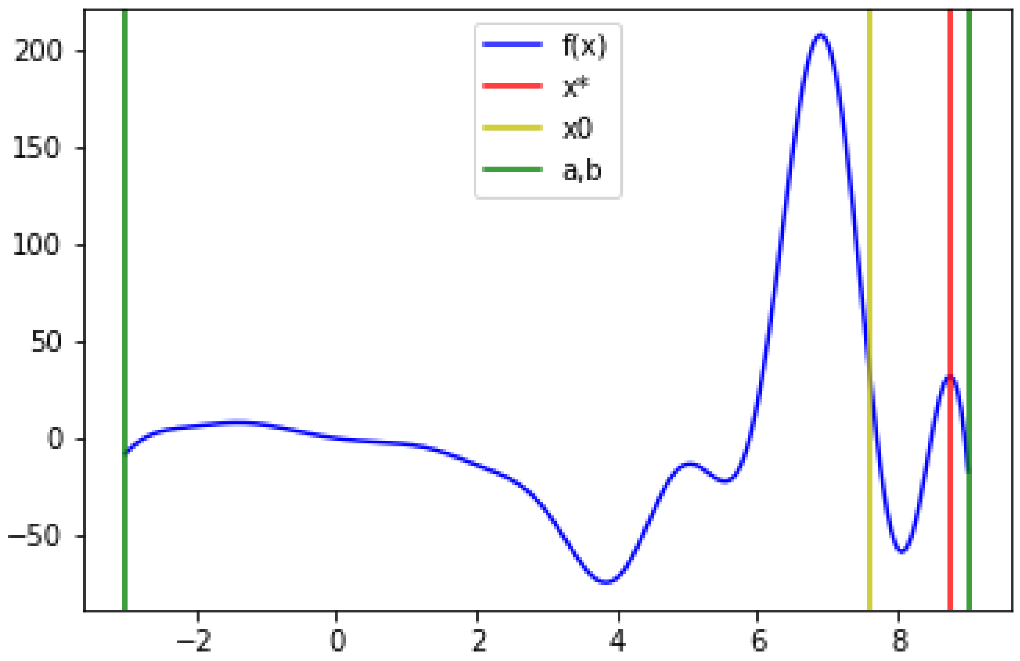

Figure 13: The graph with given parameters a,b and x* value found with model

Results of the model :

Iteration	$x^{(k)}$	$f_k(x^{(k)})$	$f_k'(x^{(k)})$	$f_k''(x^{(k)})$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert^2}$
0	7.6000	32.3625	-350.3796	323.1895	NaN
1	8.6841	30.3087	63.9051	-1048.9343	0.0518
2	8.7451	32.1242	-6.3893	-1253.9098	1.3728
3	8.7400	32.1406	-0.0407	-1237.9096	1.2653
4	8.7399	32.1406	-0.0000	-1237.8057	NaN

$$x^* = 8.7399$$

$$f(x^*) = 32.1406$$

$$f''(x^*) = -1237.8057$$

The conclusion :

Since the Newton's method that is used in the homework uses second order approximation[1], it is not able to approximate the function with the initial point $x_0 = 7.6$, properly. So, it jumps the closest local extremum point in the first iteration and converges to an other extremum point. The Newton's method is not able to find a local minimum point of the given function with the initial point $x_0 = 7.6$, since the second derivative of the extremum point is negative.

5 Secant Method

The source code used to implement the method :

```python
def SecantMethod(x_0, x_1, epsilon, a=-3, b=9):
    iteration = 0
    res = [[iteration, x_0, f(x_0), df(x_0)]]
    iteration += 1
    dfx0 = df(x_0)
    while True:
        dfx1 = df(x_1)
        x_next = x_1 - dfx1 / (dfx1 - dfx0) * (x_1 - x_0)
        res.append([iteration, x_1, f(x_1), dfx1])
        iteration +=1
        if abs(x_next-x_1)<epsilon:
            break
        x_0 = x_1
        dfx0 = dfx1
        x_1 = x_next
        if x_next<a or x_next>b:
            print("Error: The Secant method is not able to find any local minimum in the
                given range")
            break
    res.append([iteration, x_next, f(x_next), df(x_next)])
    result_table = pd.DataFrame(res, columns = ['iteration', 'x', 'f(x)', "f'(x)"])
    result_table['c_rate'] = pd.Series(c_rate(result_table.x, 1.618))
    x_star = x_next
    fx_star = f(x_next)
    return x_star, fx_star, result_table
```

The source code to apply parameters :

```python
secant_par = [[6 ,7, 0.001], [1, 2, 0.005], [3, 4, 0.0001]]
a= -3
b= 9
for par in secant_par:
    x_star, fx_star, res = SecantMethod(par[0],par[1],par[2])
    lines = [x_star,a,b]
    colors = ['r','g','g']
    labels = ['x*','a,b','']
    for i in range(min(len(res.x)-1,2)):
```

[1] $f_k(x) = c + bx + ax^2$ is used to approximate

```
        lines.append(res.x[i])
        colors.append('y')
        labels.append('')
    labels[-1] = 'intermediary steps'
    plotGraphWithLines(lines,colors,labels, 'secant' + str(par[0]))
    print(res.to_latex(index=False,float_format='%.4f'))
    print(round(x_star,4),round(fx_star,4))
```

Three different results obtained through three different parameters:

5.1 Secant Model 1

The parameters used : $x_0 = 6$, $x_1 = 7$, $\varepsilon = 0.001$

Graph of the function :

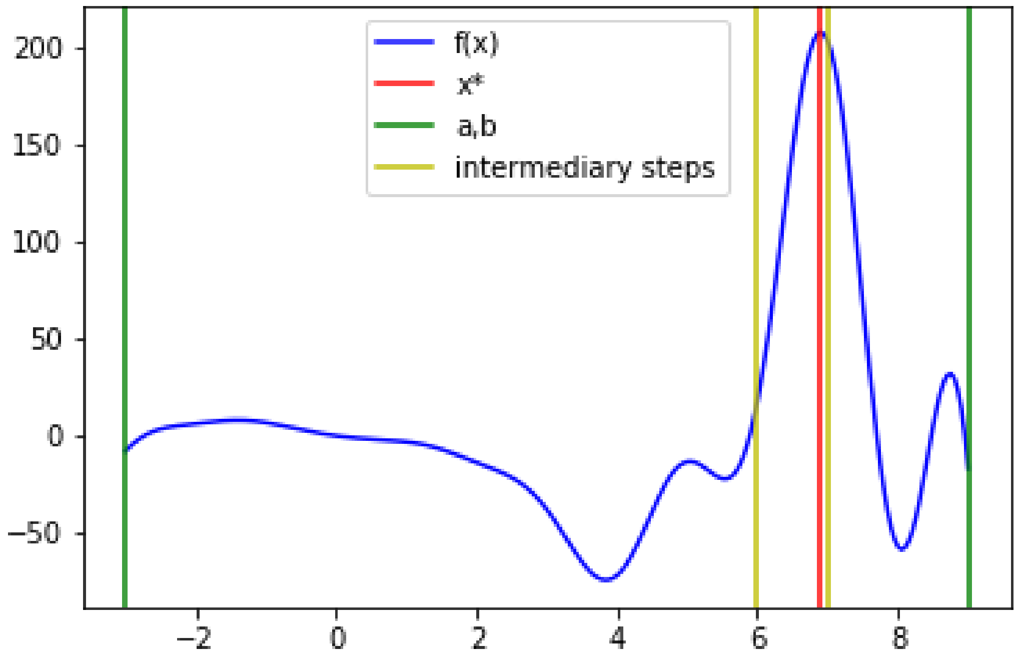

Figure 14: The graph with given parameters a,b and x* value found with model

Results of the model :

Iteration	$x^{(k)}$	$f_k(x^{(k)})$	$f_k'(x^{(k)})$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert^{1.618}}$
0	6.0000	18.0566	190.7426	NaN
1	7.0000	203.9035	-91.2780	0.3237
2	6.6763	187.4117	178.3316	1.3282
3	6.8904	208.4101	8.7340	0.1335
4	6.9014	208.4517	-1.1926	1.9477
5	6.9001	208.4525	0.0031	0.1579
6	6.9001	208.4525	0.0000	NaN

$$x^* = 6.9001$$
$$f(x^*) = 208.452$$

The conclusion :
The Secant Method is applied for starting points $x_0 = 6$ and $x_1 = 7$. The method was successful to find a local maximum in the given interval by estimating the second derivative. However it is asked to find a local minimum in this task. This shows that secant method can be both converged to a local minumum or maximum.

5.2 Secant Model 2

The parameters used : $x_0 = 1$, $x_1 = 2$, $\varepsilon = 0.005$
Graph of the function :

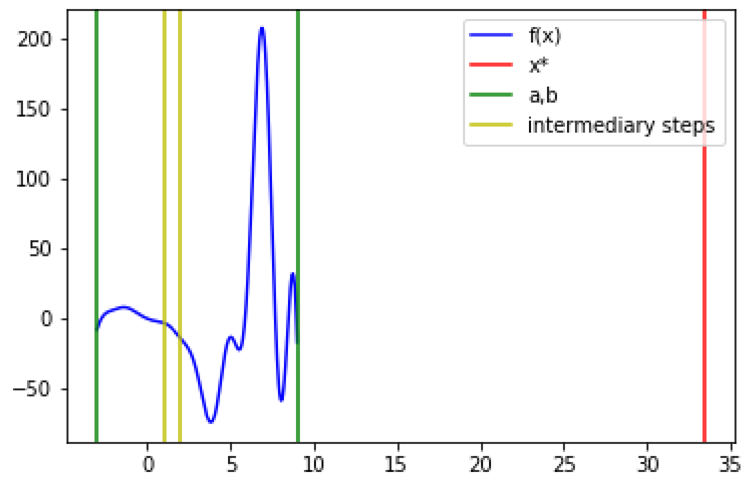

Figure 15: The graph with given parameters a,b and $x_{termination}$ value found with model

Results of the model :

Iteration	$x^{(k)}$	$f_k(x^{(k)})$	$f'_k(x^{(k)})$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert^{1.618}}$
0	1.0000	-3.1334	-4.1531	NaN
1	2.0000	-13.7340	-14.0870	1.4181
2	0.5819	-1.9853	-2.2364	0.1521
3	0.3143	-1.2810	-3.1942	7.5316
4	1.2068	-4.2703	-7.0007	1.9731
5	-0.4346	2.7152	-7.3577	15.1765
6	33.4009	4010.5291	20052.2319	NaN

The Secant method is not able to find any local minimum in the given range

The conclusion :
The method has failed to find an local extremum with the starting points 1 and 2 in the given domain. At the 6th iteration method jumped to a intermediate state with $x = 33.4009$ which is outside of our interval. It is assumed that no information outside the interval is available. Therefore it is concluded that the method is failed. Another option to terminate model could be choose the closest border of the domain as a local extrema when the x is outside the interval.

5.3 Secant Model 3

The parameters used : $x_0 = 3$, $x_1 = 5$, $\varepsilon = 0.0001$
Graph of the function :

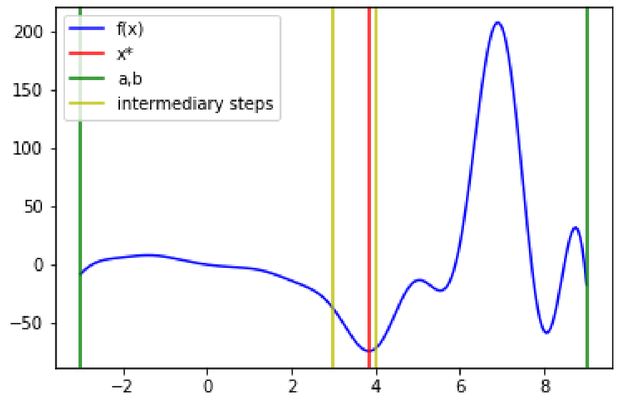

Figure 16: The graph with given parameters a,b and x* value found with model

Results of the model :

Iteration	$x^{(k)}$	$f_k(x^{(k)})$	$f'_k(x^{(k)})$	$\frac{\lvert x^{(k+1)}-x^{(k)}\rvert}{\lvert x^{(k)}-x^{(k-1)}\rvert^{1.618}}$
0	3.0000	-37.9954	-48.0286	NaN
1	4.0000	-72.0689	30.1653	0.3858
2	3.6142	-70.0011	-36.5129	0.9865
3	3.8255	-74.4088	-3.1468	0.2465
4	3.8454	-74.4349	0.5428	1.6545
5	3.8425	-74.4356	-0.0037	0.2491
6	3.8425	-74.4356	-0.0000	NaN

$$x^* = 3.8425$$
$$f(x^*) = -74.4356$$

The conclusion :
The Secant Method is applied for starting points $x_0 = 3$ and $x_1 = 4$. The method was successful to find a local minimum in the given interval by estimating the second derivative. With these initializing points the method was able to do our task properly. Finding different local extrema with different initiliazing points show that Secant method has local convergence.

6 Appendix

The complete script file:

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 12 13:46:53 2019
"""
import numpy as np
from sympy import Symbol, cos, sin, lambdify
import matplotlib.pyplot as plt
import pandas as pd

#%%

x = Symbol('x')
function = x**3*cos(x)**2*sin(x)+3*x**2*cos(x)-5*x
first_deriv = function.diff(x)
second_deriv = first_deriv.diff(x)

f = lambdify(x, function, 'numpy')
df = lambdify(x, first_deriv, 'numpy')
ddf = lambdify(x, second_deriv, 'numpy')

#%%
def plotGraphWithLines(x_pts=[], colors=[], labels=[], name="graph"):
    t1 = np.arange(-3, 9.05, 0.05)
    plt.figure()
    plt.plot(t1, f(t1), 'b-', label='f(x)')
    for i in range(len(x_pts)):
        plt.axvline(x_pts[i], color=colors[i], label=labels[i])
    plt.legend()
    plt.savefig("{0}.png".format(name))

plotGraphWithLines()
#%%
def c_rate(x, degree=1):
    res = [None]
    for i in range(1,(len(x) -1)):
        res.append(np.abs(x[i+1] - x[i])/np.abs(x[i] - x[i-1])**degree)
    return res
#%%
def BisectionMethod(a=-3,b=9,epsilon=0.001) :
    iteration=0
    res = []
    while (b - a) >= epsilon:
        x_1 = (a + b) / 2
        fx_1 = f(x_1)
        res.append([iteration, a, b, x_1, fx_1])
        if f(x_1 + epsilon) <= fx_1:
            a = x_1
        else:
            b = x_1
        iteration+=1
    x_star = (a+b)/2
    fx_star = f(x_star)
    res.append([iteration, a, b, x_star, fx_star])
    result_table = pd.DataFrame(res, columns=['iteration' ,'a', 'b', 'x', 'f(x)'])
    result_table['c_rate'] = pd.Series(c_rate(result_table.x))
    result_table["log_c_rate"] = -np.log(result_table.c_rate)
    return x_star, fx_star, result_table

#%%
def GoldenSection(a,b,epsilon):
    golden_ratio = (1+np.sqrt(5))/2
    gama = 1/golden_ratio
    iteration = 0
    x_1 = b - gama*(b-a)
    x_2 = a + gama*(b-a)
    fx_1 = f(x_1)
    fx_2 = f(x_2)
    res = [[iteration, a, b, x_1, x_2, fx_1, fx_2]]
    while (b-a) >= epsilon:
        iteration+=1
        if(fx_1 >= fx_2):
            a = x_1
            x_1 = x_2
            x_2 = a + gama*(b-a)
            fx_1 = fx_2
            fx_2 = f(x_2)
        else:
            b = x_2
            x_2 = x_1
            x_1 = b - gama*(b-a)
            fx_2 = fx_1
            fx_1 = f(x_1)
        res.append([iteration, a, b, x_1, x_2, fx_1, fx_2])
    result_table = pd.DataFrame(res, columns = ['iteration', 'a', 'b', 'x', 'y', 'f(x)',
         'f(y)'])
    result_table['c_rate'] = pd.Series(c_rate((result_table.a + result_table.b)/2))
    result_table["log_c_rate"] = -np.log(result_table.c_rate)
    x_star = (a+b)/2
    fx_star = f(x_star)
    return x_star, fx_star, result_table

#%%
def NewtonsMethod(a, b, x_0, epsilon):
    iteration = 0
    res = []
    while True:
        dfx0 = df(x_0)
        ddfx0 = ddf(x_0)
        x_1 = x_0-dfx0/ddfx0
        res.append([iteration, x_0, f(x_0), dfx0, ddfx0])
        iteration +=1
        if abs(x_0-x_1)<epsilon:
            break
        else:
            x_0 = x_1
        if x_1<a or x_1>b:
            print("Error: The Newton\'s method is not able to find any local minimum in
                the given range")
            break
    res.append([iteration, x_1, f(x_1), df(x_1), ddf(x_1)])
    result_table = pd.DataFrame(res, columns = ['iteration', 'x', 'f(x)', "f'(x)", "f''(
        x)"])
    result_table['c_rate'] = pd.Series(c_rate(result_table.x, 2))
    x_star = x_1
    fx_star = f(x_1)
    return x_star, fx_star, result_table
#%%
def SecantMethod(x_0, x_1, epsilon, a=-3, b=9):
    iteration = 0
    res = [[iteration, x_0, f(x_0), df(x_0)]]
    iteration += 1
    dfx0 = df(x_0)
    while True:
        dfx1 = df(x_1)
        x_next = x_1 - dfx1 / (dfx1 - dfx0) * (x_1 - x_0)
        res.append([iteration, x_1, f(x_1), dfx1])
        iteration +=1
        if abs(x_next-x_1)<epsilon:
            break
        x_0 = x_1
        dfx0 = dfx1
        x_1 = x_next
        if x_next<a or x_next>b:
            print("Error: The Secant method is not able to find any local minimum in the
                given range")
            break
    res.append([iteration, x_next, f(x_next), df(x_next)])
    result_table = pd.DataFrame(res, columns = ['iteration', 'x', 'f(x)', "f'(x)"])
    result_table['c_rate'] = pd.Series(c_rate(result_table.x, 1.618))
    x_star = x_next
    fx_star = f(x_next)
    return x_star, fx_star, result_table
#%%
'''
Bisection Model-1
'''
a=-3
b=9
x_star, fx_star, res = BisectionMethod(a,b,0.001)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
```

```
#print(res.to_latex(index=False,float_format='%.4f'))
#%%
'''
Bisection Model-2
'''
a=-2
b=8
x_star, fx_star, res = BisectionMethod(a,b,0.001)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
#print(res.to_latex(index=False,float_format='%.4f'))
#%%
'''
Bisection Model-3
'''
a=2
b=9
x_star, fx_star, res = BisectionMethod(a,b,0.001)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
#print(res.to_latex(index=False,float_format='%.4f'))
#%%
'''
Bisection Model-4
'''
a=2.1
b=9
x_star, fx_star, res = BisectionMethod(a,b,0.001)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
#print(res.to_latex(index=False,float_format='%.4f'))

#%%
```

```
'''
Golden Section Model -2
'''
a=3.5
b=9
epsilon=0.001
x_star, fx_star, res = GoldenSection(a,b,epsilon)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
#print(res.to_latex(index=False,float_format='%.4f'))
round(x_star,4),round(fx_star,4)
#%%
'''
Golden Section Model - 3
'''
a=-3
b=9
epsilon=0.669
x_star, fx_star, res = GoldenSection(a,b,epsilon)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,4)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
#print(res.to_latex(index=False,float_format='%.4f'))
round(x_star,4),round(fx_star,4)
#%%
'''
Golden Section Model 1
'''
a=-3
b=9
epsilon=0.001
x_star, fx_star, res = GoldenSection(a,b,epsilon)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,3)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
```

```
#print(res.to_latex(index=False,float_format='%.4f'))
round(x_star,4),round(fx_star,4)
#%%
'''
Golden Section Model -4
'''
a=3.80
b=8
epsilon=0.001
x_star, fx_star, res = GoldenSection(a,b,epsilon)
lines = [x_star,a,b]
colors = ['r','g','g']
labels = ['x*','a,b','']
for i in range(min(len(res.x)-1,2)):
    lines.append(res.x[i])
    colors.append('y')
    labels.append('')
labels[-1] = 'intermediary steps'
plotGraphWithLines(lines,colors,labels)
#print(res.to_latex(index=False,float_format='%.4f'))
round(x_star,4),round(fx_star,4)
#%%
'''
Secant Method
'''
secant_par = [[6 ,7, 0.001], [1, 2, 0.005], [3, 5, 0.0001]]
a= -3
b= 9
for par in secant_par:
    x_star, fx_star, res = SecantMethod(par[0],par[1],par[2])
    lines = [x_star,a,b]
    colors = ['r','g','g']
    labels = ['x*','a,b','']
    for i in range(min(len(res.x)-1,2)):
        lines.append(res.x[i])
        colors.append('y')
        labels.append('')
    labels[-1] = 'intermediary steps'
    plotGraphWithLines(lines,colors,labels, 'secant' + str(par[0]))
    print(res.to_latex(index=False,float_format='%.4f'))
    print(round(x_star,4),round(fx_star,4))
#%%
'''
Newton's Method
'''
newton_par = [[3.5, 0.001], [6.6, 0.0001], [7.5, 0.001], [7.6, 0.005]]
a= -3
b= 9
for par in newton_par:
    x_star, fx_star, res = NewtonsMethod(a, b, par[0], par[1])
    lines = [x_star,par[0],a,b]
    colors = ['r','y','g','g']
    labels = ['x*','x0','a,b','']
    plotGraphWithLines(lines,colors,labels, 'newton' + str(par[0]))
    #print(res.to_latex(index=False,float_format='%.4f'))
    print(round(x_star,4),round(fx_star,4),round(ddf(x_star),4))
```