Kaunas University of Technology
Faculty of Informatics

Numerical Methods and Algorithms

Engeneering Project 1

Student name, surname, academical group

Student **Zahi El Helou**, IFU-1

Position

Instructor **KRIŠČIŪNAS Andrius**

Kaunas, 2023

Part 1:

1. The functions f(x) and g(x):

Polynomial $f(x) = -0.63x^4 + 3.92x^3 - 7.95x^2 + 5.50x - 0.53$
Transcendantal g(x) = sin(x) (x 2 − 1)(x + 3) − 0,9; $-10 \le x \le 10$

Methods :

Number	**Method**
1	Bisection
2	Chords
3	Newton
4	Quasi-Newton

Solving the nonlinear equations:

a) polynomial $f(x) = 0$; b) Transcendental function $g(x) = 0$.

The calculations to get range for roots of function f(x):

We should multiply by -1 and our function would be:

$$f(x) = 0.63x^4 - 3.92x^3 + 7.95x^2 - 5.50x + 0.53$$

In order to start, we have to find the region of interest as well as the rough estimation:

Region of interest:

1 + 7.95 / 0.63 = 13.62

Rough estimation

[-13.6 ; 13.6]

Precise estimation Rpos and Rneg

$$Rpos = 1 + \sqrt[k]{\frac{B}{a^n}} = 1 + \sqrt[3]{\frac{5.5}{0.63}} = 3.05$$

$$Rneg = 1 + \sqrt[k]{\frac{B}{a^n}} = 1 + \sqrt[4]{\frac{0}{0.63}} = 1 + 0 = 1$$

So, in that case we have:

min(-13.6 ; -1) ≤ X ≤ max(3.05 ; 13.6)

In conclusion, the interval where the roots might be represent more precisely :

$$-1 \le x \le 3.05$$

2. Plotting and visualizing the functions f(x) and g(x)

F(x)

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

Defining fx as function

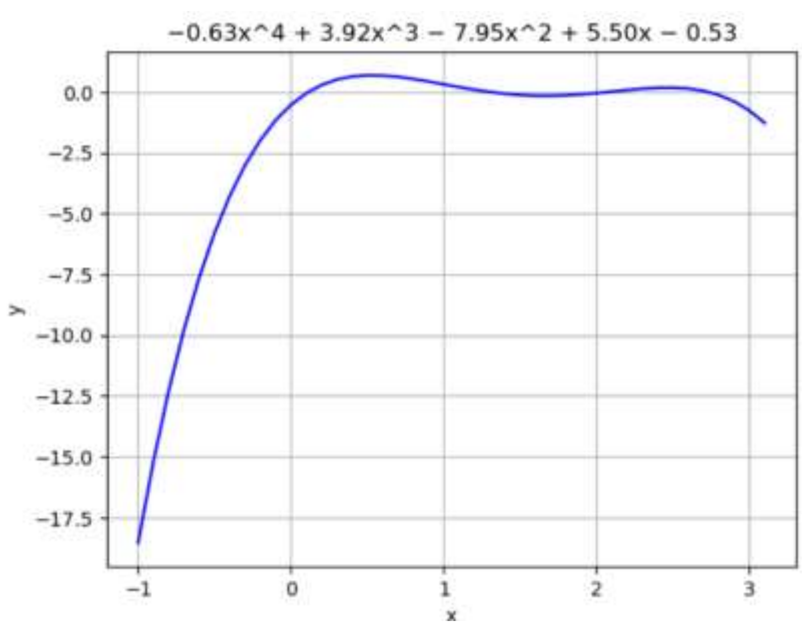
```
def fx(x):
  return -0.63*x**4 + 3.92*x**3 - 7.95*x**2 + 5.50*x - 0.53
```

Getting y values

```
dx= 0.1 #discretization step
x=np.arange(-1, 3.05+dx, dx)
y = fx(x)
```

Grafical results reperesentation

```
plt.title('-0.63x^4 + 3.92x^3 - 7.95x^2 + 5.50x - 0.53')
plt.xlabel("x");plt.ylabel("y")
plt.plot(x, y, 'b')
#plt.xlim([-10, 10])
#plt.ylim([-100, 10])
plt.grid()
```



g(x)

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

Defining fx as function

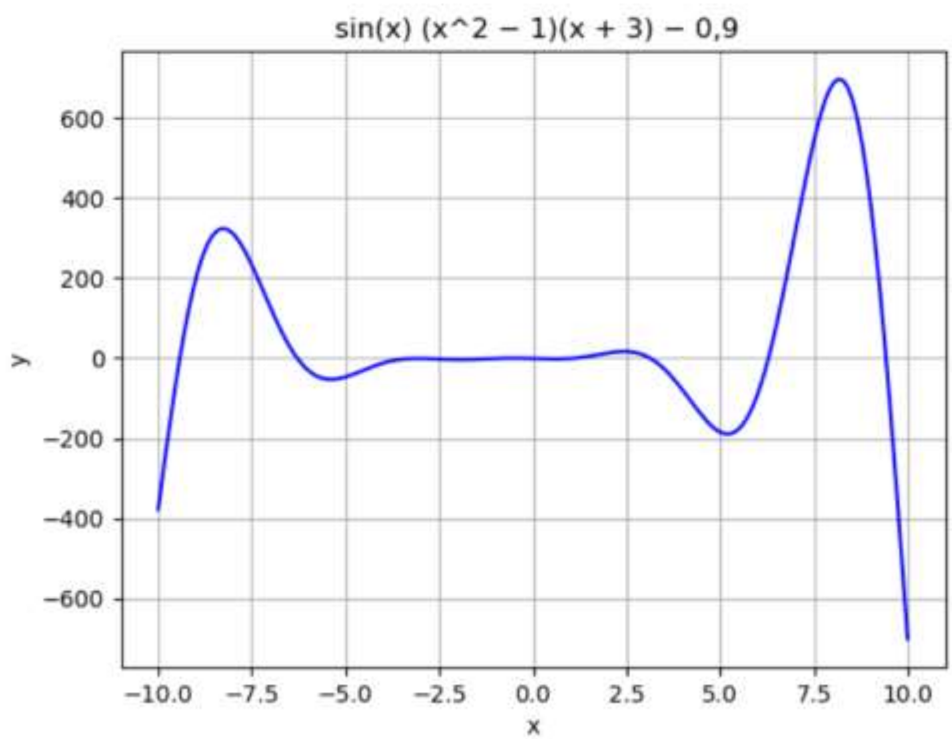
```
def fx(x):
  return np.sin(x) * (x**2 - 1) * (x + 3) - 0.9
```

Getting y values

```
dx = 0.1  # discretization step
x = np.arange(-10, 10 + dx, dx)
y = fx(x)
```

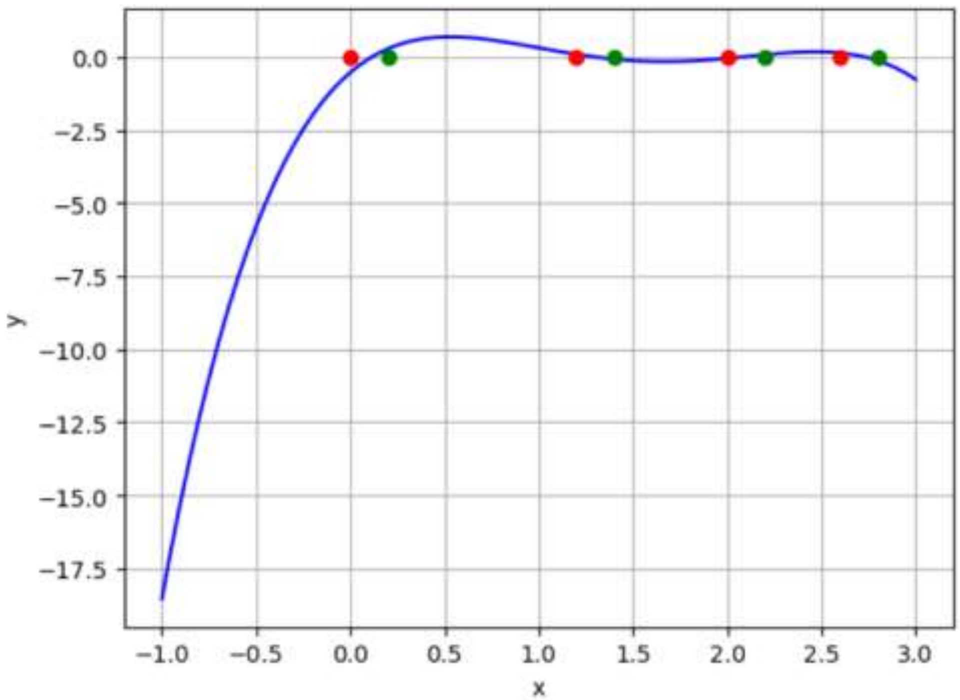
Grafical results reperesentation

```
plt.title(' sin(x) (x^2 - 1)(x + 3) - 0,9')
plt.xlabel("x");plt.ylabel("y")
plt.plot(x, y, 'b')
#plt.xlim([-10, 10])
#plt.ylim([-100, 10])
plt.grid()
```



3. Root Isolation Intervals:

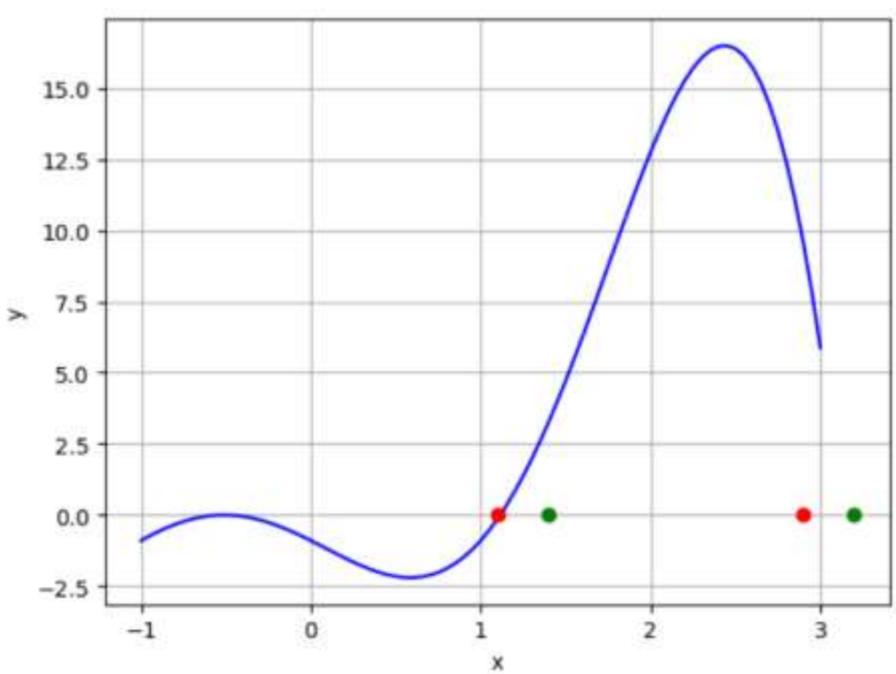
f(x)



Intervals:

```
Root : [-0.000 ;  0.200]
Root : [1.200 ;  1.400]
Root : [2.000 ;  2.200]
Root : [2.600 ;  2.800]
```

g(x)



Intervals:

```
Root : [1.100 ;  1.400]
Root : [2.900 ;  3.200]
```

4. The root values with acceptable tolerance (choose arbitrarily) using methods provided in Tables 1, 2

Bisection	Initial guess	defined root	Iterations count	Validation	Value of the function in calculated root
f(x)	[-0.100 ; 0.200]	0.11416085362434387	24	0.114161	-9.431455172403957e-09
	[1.100 ; 1.400]	1.2968826174736026	23	1.2968	-2.385675434979362e-09
	[2.000 ; 2.300]	2.082095605134964	23	2.0820	-6.430116439304356e-10
	[2.600 ; 2.900]	2.729083150625229	20	2.7290	-8.043918198197275e-09
g(f)	[1.100 ; 1.400]	1.1151468813419343	24	1.11514	-4.844328826969502e-09
	[2.900 ; 3.200]	3.1248262465000156	24	3.12482	7.553644931768133e-09

Chords	Initial guess	defined root	Iterations count	Validation	Value of thefunction in calculated root
f(x)	[-0.100 ; 0.200]	0.11416085711084732	15	0.1141	3.937045911506232e-09
	[1.100 ; 1.400]	1.296882621982338	9	1.29688	-9.984155502351655e-09
	[2.000 ; 2.300]	2.083725447512516	4	2.08372	-4.988676138850678e-12
	[2.600 ; 2.900	2.7290831406975147	16	2.7290	7.110832145329482e-09
g(f)	[1.100 ; 1.400]	1.1151468808487701	11	1.11514	-9.23415510722947e-09
	[2.900 ; 3.200]	3.124826246572521	8	3.12482	3.719185848183315e-09

NEWTON	Initial guess	defined root	Iterations count	Validation	Value of the function in calculated root
f(x)	2.000	2.0862068965517238	3	2.0862	0.002591902122351941
	2.500	3.885	8	3.885	-12.813324347393765
	4.500	3.841439308367364	8	3.84143	-11.693353413215553
	1.400	1.2788894575230254	4	1.27888	0.01537891499758648
g(f)	2.000	1.1316946908723453	4	1.13169	0.14986714796208556
	2.500	7.136458753928679	5	7.13645	380.41714294429767
	3.000	0.932345772815033	4	0.9323	3.1421861358869645
	3.500	3.199599249212919	5	3.19959	-4.220082149945739

Quasi-Newton	Initial guess	defined root	Iterations count	Validation	Value of thefunction in calculated root
f(x)	2.000	2.086206962817795	3	2.08620	0.002591943936335328
	2.500	3.8850240947253027	8	3.8850	-12.813963853468133
	3.000	2.7468445538457305	5	2.7468	-0.028394282384403224
	3.500	3.1365843830297298	7	3.1365	-1.505269251963919

g(f)	2.000	1.1316951765670218	4	1.13169	0.149871622031846
	2.500	7.136496281957376	5	7.13649	380.43513369510174
	3.000	3.1421862916198506	4	3.142186	-0.9323542649043278
	3.500	3.199599096135544	5	3.1995990	-4.220072964143578

Part 2

Task number	m, kg	t_1, s	v_1, m/s
6	90	3,5	30

$$v(t) = \frac{mg}{c}\left(1 - e^{-\left(\frac{c}{m}\right)t}\right)$$

Task number	m, Kg	t, s	v_t, m/s
6	90	3,5	30

$$v(t) = \frac{mg}{c}\left(1 - e^{-\left(\frac{c}{m}\right)t}\right)$$

$g = 9,8\ m/s^2$

m : mass

t time

coefficient of [illegible] : c

$$30 = \frac{90 \times 9,8}{c} \times \left(1 - e^{-\frac{c}{90} \times 3,5}\right)$$

$$30 = \frac{882}{c} \times \left(1 - e^{-\frac{c \times 3,5}{90}}\right)$$

$$30c = 882 \times \left(1 - e^{-\frac{c \times 3,5}{90}}\right)$$

$$30c = 882 - 882 \times e^{-\frac{c \times 3,5}{90}}$$

$$882 \times e^{-\frac{c \times 3,5}{90}} = 882 - 30c$$

$$e^{-\frac{c \times 3,5}{90}} = 1 - \frac{30c}{882}$$

$$-\frac{c \times 3,5}{90} = \ln\left(1 - \frac{30c}{882}\right)$$

$$c = -\left(\frac{90}{3,5}\right) \times \ln\left(1 - \frac{30c}{882}\right)$$

$$\Rightarrow 30c = 882\left(1 - e^{-\frac{3,5c}{90}}\right)$$

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

```python
# given function
def fx(x):
    return 30 - (x * 90 * 9.8 / x) * (1 - np.exp(-(x / 90) * 3.5))
```

Qiasi Newton

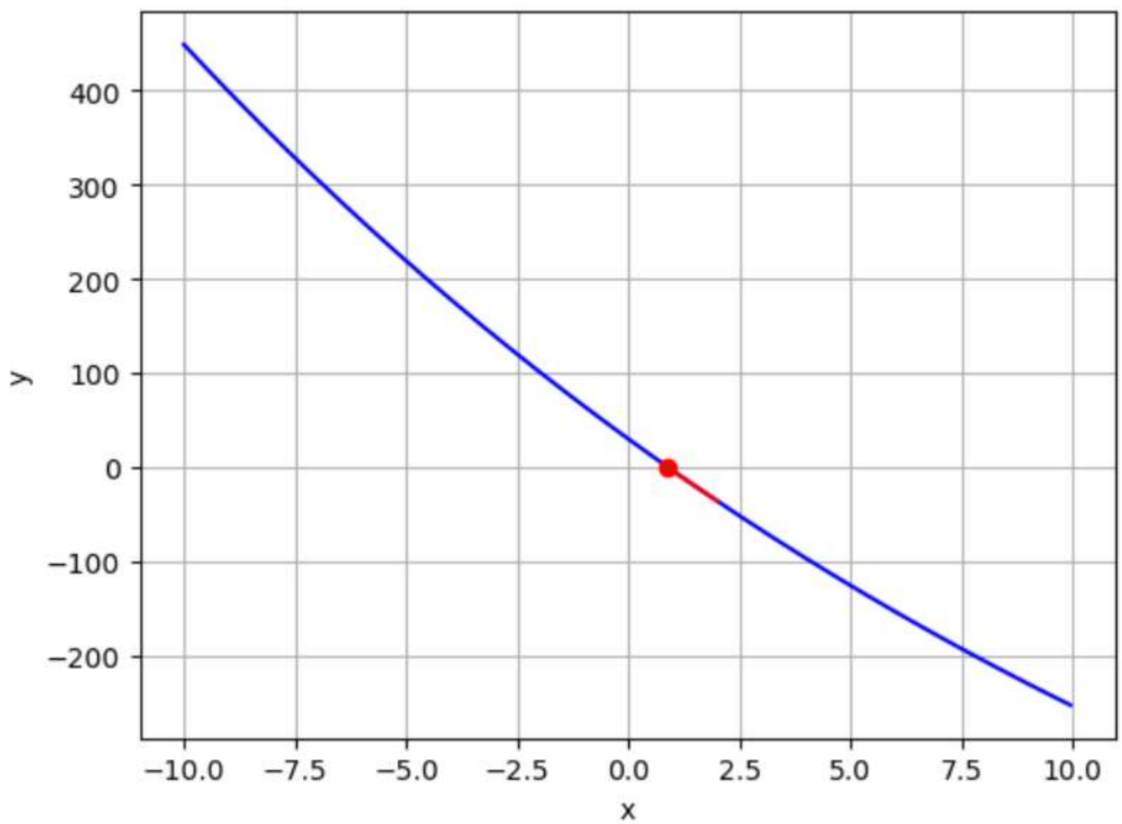
```
def dfx(x):
  h = 1e-6
  return (fx(x) - fx(x-h)) / h
```

```
eps = 1e-8
xi = xs
while np.abs(fx(xi)) > eps:
  xi_bef = xi

  xi = xi - (1 / dfx(xi)) * fx(xi)

  print("Fx = " + str(fx(xi)) + "  /  x = " + str(xi))

  plt.xlabel("x"); plt.ylabel("y"); plt.plot(x, y, 'b'); plt.grid()
  plt.plot([xi], [0], 'or')
  plt.plot([xi_bef, xi], [fx(xi_bef), 0], 'r-')
  plt.plot([xi, xi], [0, fx(xi)], 'g--')
  plt.show()
```



Initial guess	defined root	Iterations count	Validation	Value of the function in calculated root

1	0.8655441806582909	3	0.865544	0.8059279401449899