

**KAUNAS UNIVERSITY OF TECHNOLOGY**

**Faculty of Informatics**

**Department of Information Systems**

**Image Processing – Application of data Concurrency Tools**

December 02, 2021

Student: Leo Onyema Sochi Muoghara

Teacher: Karolis Ryselis

Kaunas, 2021

### Table of Contents

[1. Nonlinear Equations 3](#_Toc83817619)

[1.1. The Interval of 𝑓(𝑥) 3](#_Toc83817620)

[1.2. Display the Functions 4](#_Toc83817621)

[1.3. Scanning Method 5](#_Toc83817622)

[1.4. Chord and Newton Method 7](#_Toc83817623)

[1.5. Result Validation 12](#_Toc83817624)

[2. Tasks numbers: 11-15 14](#_Toc83817625)

[Conclusions 15](#_Toc83817626)

[References 15](#_Toc83817627)

[Appendices 16](#_Toc83817628)

[Project Initialization and functions 16](#_Toc83817629)

[Display the Function 16](#_Toc83817630)

[Scanning Method of 𝑓(𝑥) and 𝑔(𝑥) 16](#_Toc83817631)

[Bisection Method 18](#_Toc83817632)

[Chord Method 18](#_Toc83817633)

[Newton Method 19](#_Toc83817634)

[Chord Method of 𝑓(𝑥) and 𝑔(𝑥) 19](#_Toc83817635)

[Newton’s Method of 𝑓(𝑥) and 𝑔(𝑥) 20](#_Toc83817636)

[Tasks numbers: 11-15 21](#_Toc83817637)

**Task:**

1. Solve two nonlinear equations:
   1. Polynomial 𝑓(𝑥) = −0.3𝑥5 − 1.10𝑥4 + 1.14𝑥3 + 3.84𝑥2 + 1.48𝑥 − 0.22 = 0
   2. Transcendental function 𝑔(𝑥) = 2 − ln(𝑥) sin(𝑥2) = 0; 6 ≤ 𝑥 ≤ 9.
2. Body of temperature **𝑇0** is brought to the environment of temperature **𝑇𝐴**. Body temperature does not impact the temperature of the environment. Body temperature **𝑇(𝑡)** depends on environment temperature as **𝑇(𝑡) = (𝑇0 − 𝑇𝐴)𝑒 𝑘𝑡 + 𝑇𝐴** . Here **𝑘** – proportional coefficient, **𝑡** – time. What is the proportionality coefficient k, if it is known that after time period **𝑡1**body temperature is **𝑇1** given that: **T0, K** = 473; **TA, K** = 373; **t1, s** = 20; and **T1, K** = 378.

# Nonlinear Equations

To complete the first task, the following steps were taken:

## The Interval of 𝑓(𝑥)

The first step is to find the interval of the function **𝑓(𝑥)** where the roots exist. Given the function,

𝑓(𝑥) = ,

We have:

𝑓(𝑥) = −0.3𝑥5 − 1.10𝑥4 + 1.14𝑥3 + 3.84𝑥2 + 1.48𝑥 − 0.22 = 0

Multiplying the function by -1, we get:

𝑓(𝑥) = 0.3𝑥5 + 1.10𝑥4 - 1.14𝑥3 - 3.84𝑥2 - 1.48𝑥 + 0.22 = 0

To find the positive value roots:

,

**Eq.1**

Where,

**Eq.2**

and

**Eq.3**

Thus,

= 2 and

Therefore,

To find Rneg, instead of using 𝑓(𝑥), we use -𝑓(-𝑥) because n is odd. Therefore, we get

-𝑓(-𝑥) = 0.3𝑥5 − 1.10𝑥4 - 1.14𝑥3 + 3.84𝑥2 - 1.48𝑥 − 0.22 = 0

To solve for Rneg, we apply **Eq.1**, **Eq.2** and **Eq.3** to get

= 1, and

To find the final estimate, we use this statement:

To find R, we use Eq.4

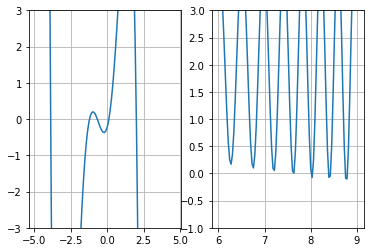
**Eq.4**

Thus,

Therefore, having a final estimate of

## Display the Functions

The second step is to display functions 𝑓(𝑥) and 𝑔(𝑥) so that the roots of the functions can be seen graphically. The code used can be seen **in Appendices.**



**Figure.1 Left Graph is the function 𝑓(𝑥) and Right graph is the function 𝑔(𝑥)**

## Scanning Method

The third step is to find all root isolation intervals of the functions 𝑓(𝑥) and 𝑔(𝑥) using the scanning method with a constant size step (chosen step is **0.1**). For 𝑓(𝑥), then select the interval was obtained in **section 1.1**, for 𝑔(𝑥) the interval provided in the **Task**was used. All the results were displayed graphically and by text seen in **Figure.2, Figure.3, Table.1, and Table.2**.

Chart, line chart

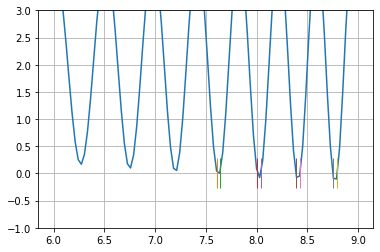
Description automatically generated

**Figure.2 Graph of the function 𝑓(𝑥) with root indicators**

The coloured circles show all the roots.

**Table.1 Results obtained for 𝑓(𝑥) from the Scanning Method**

|  |
| --- |
| Root of f(x) = -0.00000000002832870050  Root = -3.88395555239622192190  xFrom = -3.88395555239840462036; xTo = -3.88395555239403922343  Root of f(x) = 0.00000000003631026035  Root = -1.21661564647220155067  xFrom = -1.21661564655369236476; xTo = -1.21661564639071095861  Root of f(x) = 0.00000000000714270310  Root = -0.68074114703340526944  xFrom = -0.68074114719638667559; xTo = -0.68074114687042375227  Root of f(x)= -0.00000000002789332654  Root = 0.11394900159211759916  xFrom = 0.11394900138257002542; xTo = 0.11394900180166517289  Root of f(x)= -0.00000000007237746513  Root = 2.00069667764764735907  xFrom = 2.00069667763309544384; xTo = 2.00069667766219927429  Total iteration: = 170 |



**Figure.3 Graph of the function 𝑔(𝑥) with root indicators**

The coloured lines show all the roots.

**Table.2 Results obtained for 𝑔(𝑥) from the Scanning Method**

|  |
| --- |
| Root of f(x)= 0.00000000009679101964  Root = 7.61235680685316928873  xFrom= 7.61235680675829140540; xTo= 7.61235680694804806024  Root of f(x)= 0.00000000002802469368  Root = 7.63540009117798046390  xFrom= 7.63540009114072759644; xTo= 7.63540009121523333135  Root of f(x)= -0.00000000007545919445  Root = 8.00772865871196160015  xFrom= 8.00772865866655791933; xTo= 8.00772865875736350461  Root of f(x)= -0.00000000007087175291  Root = 8.04293704922422847403  xFrom= 8.04293704903796502492; xTo= 8.04293704941049369950  Root of f(x)= 0.00000000001057376409  Root = 8.38683266674163263588  xFrom= 8.38683266673115568324; xTo= 8.38683266675210958851  Root of f(x)= 0.00000000001029265562  Root = 8.42846829760930660314  xFrom= 8.42846829755342774604; xTo= 8.42846829766518723659  Root of f(x)= 0.00000000002814681821  Root = 8.75051605630718043471  xFrom= 8.75051605629728612712; xTo= 8.75051605631707651867  Root of f(x)= 0.00000000001843325492  Root = 8.79614506732782786003  xFrom= 8.79614506729988931966; xTo= 8.79614506735576817675  Total iteration: = 248 |

xFrom and xTo are the root isolation interval.

**Scanning Method**

This method works by first selecting the initial guess or isolation interval of the root, i.e., 0.1. Risk of missing roots because the wrong step of discretization was chosen. At the end of each isolation interval, the function value has opposite values. This is known as the ***sign rule***. After, one must narrow down the root value range continuously while the **acceptance tolerance (e.g., 10-10)**of the root value isn’t exceeded. Once the root and its’ intervals are found, one must start from the upper limit of the root isolation interval which was found and use the initial isolation interval and repeat the process. The code used can be found in the **Appendices**.

**Bisection Method**

This method was used concurrently with the scanning method to obtain the roots and its’ interval. When using the bisection method, we must first choose the range of the interval, xn and xn1. By applying the formula below, we obtain the mid of the interval.

Once xmid is obtained, we check if the sign of **𝑓(𝑥mid) is equal to 𝑓(𝑥n).** If they are equal, we assign the value of xmid to xn thereby narrowing the interval; if they are not equal then xmid is assigned to xn1. This process is repeated while the acceptance tolerance isn’t exceeded. The code used can be found in the **Appendices**.

## Chord and Newton Method

Next is to find root values with acceptable tolerance (chosen ) using the Chord and Newton methods.

**Chord Method**

This method has the same condition as the Bisection method in term of the condition which checks the sign of the mid-point and **𝑓(𝑥n).** The Chord method plots a line (Formular for obtaining the line is seen below) from **𝑓(𝑥n)** to **𝑓(𝑥n1)** and assigns the root of that line to xmid, thereby narrowing the isolation interval of the root.

Equations to get the line from **𝑓(𝑥n)** to **𝑓(𝑥n1)**.

This process is repeated while the acceptance tolerance isn’t exceeded. The code used can be found in the **Appendices**.

**Newton’s Method**

Newton’s method first begins selecting an initial guess x0.This will be the starting point to which we will plot the slope of 𝑓(𝑥0) and get the next consecutive point x1, x2, until we find the root by getting the slop closer and closer to the root value (i.e., using the acceptance tolerance as the break statement). The formula of Newton’s Method can be seen below.

All the results were displayed graphically and by text seen below.

Chart, line chart

Description automatically generated

**Figure.4 Graph of the function 𝑓(𝑥) with root indicators using Chord Method**

**Table.1 Results obtained for 𝑓(𝑥) from Chord Method**

|  |
| --- |
| Root of f(x) = -0.00000000009424130920  Root = -3.88395555239512901835  xFrom = -5.90000000000000035527; = -3.88395555239480216869  Root of f(x) = 0.00000000008983677691  Root = -1.21661564644304287519  xFrom = -2.70868554211110046381; = -1.21661564643132846797  Root of f(x) = 0.00000000001146591155  Root = -0.68074114703734500686  xFrom = -0.68074114749345171482; = -0.51661564643132840136  Root of f(x) = -0.00000000003666900117  Root = 0.11394900158845001903  xFrom = 0.11394900153696130296; = 0.28338435356867158754  Root of f(x) = 0.00000000009352849051  Root = 2.00069667764187375525  xFrom = 2.00069667763010361483; = 2.18338435356867233139  Total iteration: = 294 |

Chart, line chart

Description automatically generated

**Figure.5 Graph of the function 𝑔(𝑥) with root indicators using Chord Method**

**Table.5 Results obtained for 𝑔(𝑥) from Chord Method**

|  |
| --- |
| Root of f(x) = -0.00000000004713518464  Root = 7.61235680687980753589  xFrom = 7.60830014519849129329; = 7.61235680693032268351  Root of f(x) = -0.00000000009717782135  Root = 7.63540009115490025948  xFrom = 7.63540009112995488039; = 7.65235680693032183086  Root of f(x) = -0.00000000006760014770  Root = 8.00772865871111605429  xFrom = 7.89782559916319293336; = 8.00772865871504890833  Root of f(x) = -0.00000000004722888747  Root = 8.04293704922675800617  xFrom = 8.04293704921415475440; = 8.05772865871504784252  Root of f(x) = -0.00000000006765077387  Root = 8.38683266674802396778  xFrom = 8.28241342011896719555; = 8.38683266675202254703  Root of f(x) = -0.00000000006266542840  Root = 8.42846829760338422943  xFrom = 8.42846829759090532264; = 8.44683266675202126805  Root of f(x) = -0.00000000008631273474  Root = 8.75051605631491469239  xFrom = 8.66095899070119656926; = 8.75051605632011053615  Root of f(x) = -0.00000000004951905552  Root = 8.79614506732326795202  xFrom = 8.79614506731175183063; = 8.81051605632010925717  Total iteration: = 285 |

Chart, line chart

Description automatically generated

**Figure.6 Graph of the function 𝑓(𝑥) with root indicators using the Newton’s Method**

**Table.6 Results obtained for 𝑓(𝑥) from the Newton’s Method**

|  |
| --- |
| Root Values: -3.88395555239685474902  Root Values of f(x): -3.88395555239685474902  Root Values: -1.21661564649259124060  Root Values of f(x): -1.21661564649259124060  Root Values: -0.68074114702392507503  Root Values of f(x): -0.68074114702392507503  Root Values: 0.11394900160382413767  Root Values of f(x): 0.11394900160382413767  Root Values: 2.00069667764513159369  Root Values of f(x): 2.00069667764513159369  Total iteration: = 38 |

Chart, line chart

Description automatically generated

**Figure.7 Graph of the function 𝑔(𝑥) with root indicators using the Newton’s Method**

**Table.7 Results obtained for 𝑔(𝑥) from the Newton’s Method**

|  |
| --- |
| Root Values: 7.61235680685796101130  Root Value of g(x): 0.00000000007090172893  Root Values: 7.63540009118291074230  Root Value of g(x): 0.00000000005476974430  Root Values: 8.00772865870619554585  Root Value of g(x): -0.00000000002187361403  Root Values: 8.04293704923173535803  Root Value of g(x): -0.00000000000070254913  Root Values: 8.38683266674361682647  Root Value of g(x): -0.00000000001371036618  Root Values: 8.42846829760616778060  Root Value of g(x): -0.00000000002837241553  Root Values: 8.75051605630949325132  Root Value of g(x): -0.00000000000608046946  Root Values: 8.79614506732449186188  Root Value of g(x): -0.00000000003128208803  Total iteration: = 40 |

## Result Validation

By using the website wolframalpha.com, the following results were obtained in order to validate the results gotten above.

Graphical user interface, application

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

**Figure.8 This image shows the calculated results of 𝑓(𝑥) obtained from WoldramAlpha**

Graphical user interface, text, application, email

Description automatically generated

**Figure.9 This image shows the calculated results of 𝑔(𝑥) obtained from WoldramAlpha**

**Table.8 Results obtained for 𝑓(𝑥) from WoldramAlpha against the results I obtained**

|  |  |  |  |
| --- | --- | --- | --- |
| **My Results** | | | **WoldramAlpha** |
| **Scanning Method** | **Chord Method** | **Newton’s Method** |
| -3.88395555239622192190 | -3.88395555239512901835 | -3.88395555239685474902 | -3.88396 |
| -1.21661564647220155067 | -1.21661564644304287519 | -1.21661564649259124060 | -1.21662 |
| -0.68074114703340526944 | -0.68074114703734500686 | -0.68074114702392507503 | -0.680741 |
| 0.11394900159211759916 | 0.11394900158845001903 | 0.11394900160382413767 | 0.113949 |
| 2.00069667764764735907 | 2.00069667764187375525 | 2.00069667764513159369 | 2.0007 |

**Table.9 Results obtained for 𝑔(𝑥) from WoldramAlpha against the results I obtained**

|  |  |  |  |
| --- | --- | --- | --- |
| **My Results** | | | **WoldramAlpha** |
| **Scanning Method** | **Chord Method** | **Newton’s Method** |
| 7.61235680685316928873 | 7.61235680687980753589 | 7.61235680685796101130 | 7.61235680687108 |
| 7.63540009117798046390 | 7.63540009115490025948 | 7.63540009118291074230 | 7.63540009117281 |
| 8.00772865871196160015 | 8.00772865871111605429 | 8.00772865870619554585 | 8.00772865870384 |
| 8.04293704922422847403 | 8.04293704922675800617 | 8.04293704923173535803 | 8.04293704923181 |
| 8.38683266674163263588 | 8.38683266674802396778 | 8.38683266674361682647 | 8.38683266674250 |
| 8.42846829760930660314 | 8.42846829760338422943 | 8.42846829760616778060 | 8.42846829760847 |
| 8.75051605630718043471 | 8.75051605631491469239 | 8.75051605630949325132 | 8.75051605630908 |
| 8.79614506732782786003 | 8.79614506732326795202 | 8.79614506732449186188 | 8.79614506732659 |

As one can see, the results I obtain are approximately accurate to the results gotten from WoldramAlpha. Discrepancies may be due to the acceptance tolerance, or the method used.

# Tasks numbers: 11-15

**Task**

Body of temperature 𝑇0 is brought to the environment of temperature 𝑇𝐴. Body temperature does not impact the temperature of the environment. Body temperature 𝑇(𝑡) depends on environment temperature as

𝑇(𝑡) = (𝑇0 − 𝑇𝐴)𝑒𝑘𝑡 + 𝑇𝐴

Here 𝑘 – proportional coefficient, 𝑡 – time. What is the proportionality coefficient k, if it is known that after time period 𝑡1 body temperature is 𝑇1? Where:

**𝑇0, K** = 473; **𝑇𝐴, K** = 373; **𝑡1, s** = 20; **𝑇1, K** = 378

Inputting the values into the formula, we get:

378 = (473 − 373)𝑒𝑘20 + 373

Equating the equation to zero, we get:

0 = (473 − 373)𝑒𝑘20 + 373 - 378

This task was solved numerically using the Scanning and Bisection method, and the results are shown below.

Chart, line chart

Description automatically generated

**Figure.10 This image shows the calculated results of 𝑔(𝑥) obtained from WoldramAlpha**

**Table.10 Results obtained for 𝑔(𝑥) from the Newton’s Method**

|  |
| --- |
| Root of T(t) = 0.00000000002467004379  k = -0.14978661367745288446  xFrom= -0.14978661368985113356; xTo= -0.14978661366505466312  Total iteration: = 35 |

The code used can be found in the **Appendices**.

# Conclusions

This task was solved successfully and can be backed by the results obtain in comparison to the results gotten from WolframAlpha. Some discrepancies can be seen In the result’s level of accuracy across the different methods used which may be due to the acceptance tolerance, as well as the methodology used to find the results.

We can note that:

* Newton’s Method is the fastest at **33** iterations for f(x) and **35** iterations for g(x),
* followed by Bisection method at **170** iterations for f(x) and **248** iterations for g(x),
* and then Chord Method at **294** iterations for f(x) and **285** iterations for g(x).

# References

* Task of Project I, P170B115 Numerical Methods and Algorithms (2021). *KTU Moodle*. Available at *https://moodle.ktu.edu/* (Accessed: 2 September 2021).
* Solution of a single linear equation, NUMERICAL METHODS AND ALGORITHMS (NMA) P170B115. *KTU Moodle*. Available at *https://moodle.ktu.edu/* (Accessed: 2 September 2021).

# Appendices

## Project Initialization and functions

|  |
| --- |
| import numpy as np;  import matplotlib.pyplot as plt  # given function fx  def fx(x):    return -0.3\*x\*\*5 - 1.10\*x\*\*4 + 1.14\*x\*\*3 + 3.84\*x\*\*2 + 1.48\*x - 0.22  # given function gx  def gx(x):    return 2-np.log(x)\*np.sin(x\*\*2) |

## Display the Function

|  |
| --- |
| # Question 1.2  x1 = np.linspace(-4.9, 4.6, 100)  y1 = fx(x1)  x2 = np.linspace(6, 9, 100)  y2 = gx(x2)  #plot for fx  plt.subplot(1, 2, 1)  plt.plot(x1,y1)  plt.ylim(-3,3)  plt.grid(b=True);  #plot for gx  plt.subplot(1, 2, 2)  plt.plot(x2,y2)  plt.ylim(-1,3)  plt.grid(b=True); |

## Scanning Method of 𝑓(𝑥) and 𝑔(𝑥)

|  |
| --- |
| # Question 1.3a for fx  # Rneg and Rpos are the intial range of the scan  # h is the intial scanning interval  # toX is the local upper limti of the range  # x1 is the root  Rneg, Rpos, h = -4.9, 4.6, 0.1  fromX = Rneg  toX = h+Rneg  itera = 0  x = np.linspace(-4.9, 4.6, 100)  y = fx(x)  plt.plot(x,y)  plt.grid(b=True);  while (toX < Rpos):    if (np.sign(fx(fromX)) != np.sign(fx(toX))):      x1,toX,iter = bisection(fx, fromX, toX + h)      plt.ylim(-3,  3)      plt.plot(x1, fx(x1), markersize=10, marker='o')      fromX=toX      itera = itera + iter    else:      toX = toX + h  print('Total iteration: = {0}'.format(itera), '\n')  print('x = {0:.20f} f(x) = {1:.20f}'.format(x1, fx(x1)))  # Question 1.3b for gx  # Rneg and Rpos are the intial range of the scan  # h is the intial scanning interval  # toX is the local upper limti of the range  # x1 is the root  Rneg, Rpos, h = 6, 9, 0.01  fromX = Rneg  toX = h+Rneg  itera = 0  x = np.linspace(6, 9, 100)  y = gx(x)  plt.plot(x,y)  plt.grid(b=True);  while (toX < Rpos):    if (np.sign(gx(fromX)) != np.sign(gx(toX))):      x1,toX,iter = bisection(gx, fromX, toX + h)      plt.plot(x1, gx(x1), markersize=30, marker='|')      plt.ylim(-1,  3)      fromX=toX      itera = itera + iter    else:      toX = toX + h  print('Total iteration: = {0}'.format(itera), '\n') |

## Bisection Method

|  |
| --- |
| 1. # bisection method 2. # at is the accepted tolerence 3. def bisection(func, xFrom, xTo): 4. xmid = (xFrom + xTo) / 2 5. iter = 0 6. at = 1e-10 7. while (np.abs(func(xmid)) > at): 8. iter += 1 9. if (np.sign(func(xmid)) == np.sign(func(xFrom))): 10. xFrom = xmid 11. else: 12. xTo = xmid 13. xmid = (xFrom + xTo) / 2 15. print("Root of f(x)= {0:.20f}".format(func(xmid))) 16. print("Root = {0:.20f}".format(xmid)) 17. print('xFrom= {0:.20f}; xTo= {1:.20f}'.format(xFrom, xTo),'\n')   24     return xmid,xTo,iter |

## Chord Method

|  |
| --- |
| # Chord method  # at is the accepted tolerence  def chord(func, xFrom, xTo):    k = np.abs((func(xFrom)/func(xTo)))    xmid = ((xFrom+(k\*xTo))/(1+k))    iter = 0    at = 1e-10    while (np.abs(func(xmid)) > at):          iter += 1          if (np.sign(func(xmid)) == np.sign(func(xFrom))):              xFrom = xmid          else:              xTo = xmid          k = np.abs((func(xFrom)/func(xTo)))          xmid = ((xFrom+(k\*xTo))/(1+k))    print("Root of f(x) = {0:.20f}".format(func(xmid)))    print("Root = {0:.20f}".format(xmid))    print('xFrom = {0:.20f};  = {1:.20f}'.format(xFrom, xTo),'\n')    return xmid,xTo,iter |

## Newton Method

|  |
| --- |
| def Dff(f,a,h=0.01):    return (f(a + h) - f(a - h))/(2\*h)  def newton(func, dFunc, x):    iter = 0    while (np.abs(func(x)) > 1e-10):      x = x - (1 /dFunc(func,x)\*func(x))      iter = iter + 1    return x,iter |

## Chord Method of 𝑓(𝑥) and 𝑔(𝑥)

|  |
| --- |
| # Question 1.4a (chord Method) for fx  Rneg, Rpos, h = -5.9, 4.6, 0.1  fromX = Rneg  toX = h+Rneg  itera = 0  x = np.linspace(-5.9, 4.6, 100)  y = fx(x)  plt.plot(x,y)  plt.grid(b=True);  while (toX < Rpos):    if (np.sign(fx(fromX)) != np.sign(fx(toX))):      x1,toX,iter = chord(fx, fromX, toX + h)  plt.ylim(-1,  3)      plt.plot(x1, fx(x1), markersize=10, marker='o')      fromX=toX      itera = itera + iter    else:      toX = toX + h  print('Total iteration: = {0}'.format(itera), '\n')  print('x = {0:.20f} f(x) = {1:.20f}'.format(x1, fx(x1)))  # Question 1.4b (chord Method) for gx  Rneg, Rpos, h = 6, 9, 0.01  fromX = Rneg  toX = h+Rneg  itera = 0  x = np.linspace(6, 9, 100)  y = gx(x)  plt.plot(x,y)  plt.grid(b=True);  while (toX < Rpos):    if (np.sign(gx(fromX)) != np.sign(gx(toX))):      x1,toX,iter = chord(gx, fromX, toX + h)  plt.ylim(-1,  3)      plt.plot(x1, gx(x1), markersize=30, marker='|')      fromX=toX      #toX = toX + h      itera = itera + iter    else:      toX = toX + h  print('Total iteration: = {0}'.format(itera), '\n') |

## Newton’s Method of 𝑓(𝑥) and 𝑔(𝑥)

|  |
| --- |
| #Newton Method for f(x)  xStart,xEnd, h = -5.9, 4.6, 0.01  itera = 0  x = np.linspace(-5.9, 4.6, 100)  y = fx(x)  plt.plot(x,y)  plt.grid(b=True);  while (xStart + h < xEnd):    if (np.sign(fx(xStart)) != np.sign(fx(xStart+h))):      x1,itera = newton(fx,Dff, xStart)      print('Root Values: {0:.20f}'.format(x1), '\n')      plt.ylim(-3,  3)      plt.plot(x1, fx(x1), markersize=10, marker='o')      itera = itera + iter      xStart = xStart + h  print('Total iteration: = {0}'.format(itera), '\n')  #Newton Method for g(x)  xStart,xEnd, h = 6, 9, 0.01  itera = 0  x = np.linspace(6, 9, 100)  y = gx(x)  plt.plot(x,y)  plt.grid(b=True);  while (xStart + h < xEnd):    if (np.sign(gx(xStart)) != np.sign(gx(xStart+h))):      x1,itera = newton(gx,Dff, xStart)      print('Root Values: {0:.20f}'.format(gx(x1)), '\n')      plt.ylim(-1,  3)      plt.plot(x1, gx(x1), markersize=30, marker='|')      itera = itera + iter      xStart = xStart + h  print('Total iteration: = {0}'.format(itera), '\n') |

## Tasks numbers: 11-15

|  |
| --- |
| # given function T(t)  def Tt(k):    x= k\*20    return (473-373)\*np.exp(x)+373-378  Rneg, Rpos, h = -1, 1, 0.001  fromX = Rneg  toX = h+Rneg  itera = 0  x = np.linspace(-0.3, 0.3, 100)  y = Tt(x)  plt.plot(x,y)  plt.grid(b=True);  while (toX < Rpos):    if (np.sign(Tt(fromX)) != np.sign(Tt(toX))):      x1,toX,iter = bisection(Tt,  fromX, toX + h)      plt.ylim(-10,  10)      plt.plot(x1, Tt(x1), markersize=10, marker='o')      fromX=toX      #toX = toX + h      itera = itera + iter    else:      toX = toX + h  print('Total iteration: = {0}'.format(itera), '\n') |