pr1_i_math548_midterm_Lazizbek

March 16, 2024

1 Math 548, Midterm. Problem 1. LS

Problem 1

(20 points) Find the roots and their multiplicities of the following functions in the interval [0,2] using the Newton's method. Comment on your results and discuss any commonalities between the results of (i) and (ii).

i. f(x) = 21.12 - 32.4x + 12x^2

ii. h(x) = 2.7951 - 8.954x + 10.56x^2 - 5.4x^3 + x^4

#In which interval (i) has a solution? Determine all of the solution intervals using the sign comparison of the function at boundaries of the intervals [-b, a] or [c, b] starting symmetrically around both sides of the origin, 0, with a tolerance b as 10^(-3)

[6]:
```
# In which interval it has a solution?

import numpy as np
import math

# f(x) = 21.12 - 32.4*(x) + 12*(x)**2
f0 =  21.12 - 32.4*(0) + 12*(0)**2 # f0 = 21.12
frightb = f0
fleftb = f0
fa = f0
fc = f0
solution_ints = list()
b = 0
a = 0
c = 0
# we'll run through [-b, a] or [c, b]

for b in np.arange(0.0, 2.0, 0.001):
  frightb = 21.12 - 32.4*(b) + 12*(b)**2
  fleftb = 21.12 - 32.4*(-b) + 12*(-b)**2

  if np.sign(fleftb) * np.sign(fa) <= 0: # fleftb <= 0 as fa = f0 = 21.12 > 0:
    print(" = b ", b, ";", "fleftb = ", fleftb)
    solution_ints.append([-b, a])
```

```
    a = -b
    fa = fleftb

  if np.sign(frightb) * np.sign(fc) <= 0: # frightb <= 0 as fc = f0 = 21.12 > 0:
    print("b = ", b, ";", "frightb = ", frightb)
    solution_ints.append([c, b])
    c = b
    fc = frightb

print("solution intervals = ", solution_ints)
```

```
b =  1.101 ; frightb =  -0.005987999999996774
b =  1.6 ; frightb =  3.552713678800501e-15
solution intervals =  [[0, 1.101], [1.101, 1.6]]
```

solution intervals = [] means, I have only multiple roots for my square function because if the given function has any negative values my code would capture it with tolerance of b = 10^(-3), and insert it into solution_ints list as a solution interval.

#Determine the solutions using the Newton's method with a tolerance   as 10^(-6)

[2]:
```
# Determine this solution using the Newton's method with a tolerance   as␣
↪10^(-6)
from scipy.optimize import fsolve


M = 10**6 # in case the program goes into infinite loops
epsilon = 10**(-8)

for i in range(len(solution_ints)):
  solution_int = solution_ints[i]

  print(f"\nRoot {i+1} of f(x) in {solution_int} is as follows: ")
  x0 = solution_int[1] # by the result of (12), x0=2 would give faster␣
↪convergence
  v = 21.12 - 32.4*(x0) + 12*(x0)**2  # starting function value at x0.

  print("steps", "\t ", "x", "\t", "\t", "\t", "f(x)")

  if abs(v) < epsilon:
    print(0, "\t ", x0, "\t", v)
    x1 = x0
    mnumer = (-32.4 + 24*(x1))**2
    mdenom = mnumer - (21.12 - 32.4*(x1) + 12*(x1)**2)*(24)+0.00000001 # in␣
↪case x0 is a desired root
    m = round(mnumer/mdenom)
    print("m = ", m)
    print(f"multipilicity of the root {x1} is {m}")
```

```
else:
  for k in range(1, M):
    x1 = x0 - v/(-32.4 + 24*(x0))
    v = 21.12 - 32.4*(x1) + 12*(x1)**2
    print(k, "\t ", x1, "\t", v)
    if abs(v) < epsilon:
      print(f"\nSo the approximate solution {i+1} after {k} steps is = {x1}")
      # The function fsolve takes in many arguments that you can find in the␣
↪documentation,
      # but the most important two is the function you want to find the root,␣
↪and the initial guess interval.
      f = lambda x: 21.12 - 32.4*(x) + 12*(x)**2
      mnumer = (-32.4 + 24*(x1))**2
      mdenom = mnumer - (21.12 - 32.4*(x1) + 12*(x1)**2)*(24)+0.00000001 # in␣
↪case x0 is a desired root
      m = mnumer/mdenom

      print("Using Python's fsolve function, fsolve(f, [0, 2]) = ", ␣
↪*fsolve(f, solution_int[i]), '\n')
      print("m = ", m)
      print(f"multipilicity of the root {x1} is {round(m)}")
      break
    x0 = x1
```

```
Root 1 of f(x) in [0, 1.101] is as follows:
steps     x                       f(x)
1         1.099997991967872       1.2048241156747963e-05
2         1.099999999991936       4.838796030526282e-11

So the approximate solution 1 after 2 steps is = 1.099999999991936
Using Python's fsolve function, fsolve(f, [0, 2]) =  1.0999999999999992

m =  0.9999999997544808
multipilicity of the root 1.099999999991936 is 1

Root 2 of f(x) in [1.101, 1.6] is as follows:
steps     x                       f(x)
0         1.6     3.552713678800501e-15
m =  1
multipilicity of the root 1.6 is 1
```

#In case I calculated the roots of (i) by hand:

[5]:
```
print(32.4**2-4*21.12*12)
print((32.4+6)/(2*12))
```

```python
print((32.4-6)/(2*12))
```

```
36.0
1.5999999999999999
1.0999999999999999
```

pr1_ii_math548_midterm_Lazizbek

March 17, 2024

1 Math 548, Midterm. Problem 1. LS

Problem 1

(20 points) Find the roots and their multiplicities of the following functions in the interval [0,2] using the Newton's method. Comment on your results and discuss any commonalities between the results of (i) and (ii).

i. f(x) = 21.12 - 32.4x + 12x^2

ii. h(x) = 2.7951 - 8.954x + 10.56x^2 - 5.4x^3 + x^4

#In which interval (ii) has a solution? Determine all of the solution intervals using the sign comparison of the function at boundaries of the intervals [-b, a] or [c, b] starting symmetrically around both sides of the origin, 0, with a tolerance b as 10^(-3)

[3]:
```python
# In which interval it has a solution?

import numpy as np
import math

# h(x) = 2.7951 - 8.954*x + 10.56*x**2 - 5.4*x**3 + x**4
h0 = 2.7951 - 8.954*(0) + 10.56*(0)**2 - 5.4*(0)**3 + (0)**4 # h(0) = 2.7951
hrightb = h0
hleftb = h0
ha = h0
hc = h0
solution_ints = list()
b = 0
a = 0
c = 0
# we'll run through [-b, a] or [c, b]

for b in np.arange(0.0, 2.0, 0.001):
  hrightb = 2.7951 - 8.954*(b) + 10.56*(b)**2 - 5.4*(b)**3 + (b)**4
  hleftb = 2.7951 - 8.954*(-b) + 10.56*(-b)**2 - 5.4*(-b)**3 + (-b)**4

  if np.sign(hleftb) * np.sign(ha) <= 0: # hleftb <= 0 as ha = h0 = 2.7951 > 0:
    print(" = b ", b, ";", "hleftb = ", hleftb)
    solution_ints.append([-b, a])
```

```
    a = -b
    ha = hleftb

  if np.sign(hrightb) * np.sign(hc) <= 0: # hrightb <= 0 as ha = h0 = 2.7951 >␣
↪0:
    print("b = ", b, ";", "hrightb = ", hrightb)
    solution_ints.append([c, b])
    c = b
    hc = hrightb

print("solution intervals = ", solution_ints)
```

```
b =  1.1 ; hrightb =  -6.661338147750939e-16
solution intervals =  [[0, 1.1]]
```

solution intervals = [ ], empty list, means that I have only multiple roots for my function because if the given function has any negative values my code would capture it with tolerance of b = 10^(-3), and insert it into solution_ints list as a solution interval.

#Determine the solutions using the Newton's method with a tolerance   as 10^(-6)

[4]:
```
# Determine this solution using the Newton's method with a tolerance   as␣
↪10^(-6)
from scipy.optimize import fsolve


M = 10**6 # in case the program goes into infinite loops
epsilon = 10**(-8)

for i in range(len(solution_ints)):
  solution_int = solution_ints[i]

  print(f"\nRoot {i+1} of h(x) in {solution_int} is as follows: ")
  x0 = solution_int[1] # by the result of (32), x0=2 would give faster␣
↪convergence even though I am not necessarily starting with x0 = 2
  v = 2.7951 - 8.954*(x0) + 10.56*(x0)**2 - 5.4*(x0)**3 + (x0)**4  # starting␣
↪function value at x0.

  print("steps", "\t ", "x", "\t", "\t", "\t", "h(x)")

  if abs(v) < epsilon:
    print(0, "\t ", x0, "\t", v)
    x1 = x0
    mnumer = ( - 8.954 + 21.12*(x1) - 16.2*(x1)**2 + 4*(x1)**3)**2
    mdenom = mnumer - (2.7951 - 8.954*(x1) + 10.56*(x1)**2 - 5.4*(x1)**3 +␣
↪(x1)**4)*(21.12 - 32.4*(x1) + 12*(x1)**2)+0.000001 # in case x0 is a desired␣
↪root
    print("mnumer = ", mnumer)
```

```
    print("mdenom = ", mdenom)
    m = mnumer/mdenom
    print("m = ", m)
    print(f"multipilicity of the root {x1} is {round(m)}")

    # The function fsolve takes in many arguments that you can find in the␣
↪documentation,
    # but the most important two is the function you want to find the root, and␣
↪the initial guess interval.
    f = lambda x: 2.7951 - 8.954*(x) + 10.56*(x)**2 - 5.4*(x)**3 + (x)**4
    print("To check our work, the solution of h(x) = 0 using Python's fsolve␣
↪function, fsolve(f, [0, 2]) = ",  *fsolve(f, solution_int[i]), '\n')

  else:
    for k in range(1, M):
      x1 = x0 - v/( - 8.954 + 21.12*(x0) - 16.2*(x0)**2 + 4*(x0)**3)
      v = 2.7951 - 8.954*(x1) + 10.56*(x1)**2 - 5.4*(x1)**3 + (x1)**4
      print(k, "\t ", x1, "\t", v)
      if abs(v) < epsilon:
        print(f"\nSo the approximate solution {i+1} after {k} steps is = {x1}")

        mnumer = ( - 8.954 + 21.12*(x1) - 16.2*(x1)**2 + 4*(x1)**3)**2
        mdenom = mnumer - (2.7951 - 8.954*(x1) + 10.56*(x1)**2 - 5.4*(x1)**3 +␣
↪(x1)**4)*(21.12 - 32.4*(x1) + 12*(x1)**2)+0.000001 # in case x1 is the␣
↪actual(real) root
        print("mnumer = ", mnumer)
        print("mdenom = ", mdenom)
        m = mnumer/mdenom
        print("m = ", m)
        print(f"multipilicity of the root {x1} is {round(m)}")

        # The function fsolve takes in many arguments that you can find in the␣
↪documentation,
        # but the most important two is the function you want to find the root,␣
↪and the initial guess interval.
        f = lambda x: 2.7951 - 8.954*(x) + 10.56*(x)**2 - 5.4*(x)**3 + (x)**4
        print("To check our work, the solution of h(x) = 0 using Python's␣
↪fsolve function, fsolve(f, [0, 2]) = ",  *fsolve(f, solution_int[i]), '\n')

        break
      x0 = x1
```

```
Root 1 of h(x) in [0, 1.1] is as follows:
steps      x                      h(x)
0          1.1    -6.661338147750939e-16
mnumer =  0.0
```

```
mdenom =  1e-06
m =  0.0
multipilicity of the root 1.1 is 0
To check our work, the solution of h(x) = 0 using Python's fsolve function,
fsolve(f, [0, 2]) =  1.0999932258418257
```

2 Here how I checked the roots of h(x), h'(x), h''(x), h'''(x) to see what's going on with m:

[5]:
```
# h(x) = 0
f = lambda x: 2.7951 - 8.954*(x) + 10.56*(x)**2 - 5.4*(x)**3 + (x)**4
print("To check our work, the solution of h(x) = 0 using Python's fsolve␣
 ↪function, fsolve(f, [0, 2]) = ",  *fsolve(f, 1000), '\n')
```

```
To check our work, the solution of h(x) = 0 using Python's fsolve function,
fsolve(f, [0, 2]) =  2.100000000000002
```

[16]:
```
# h'(x) = 0
f = lambda x: - 8.954 + 21.12*(x) - 16.2*(x)**2 + 4*(x)**3
print("To check our work, the solution of h'(x) = 0 using Python's fsolve␣
 ↪function, fsolve(f, [0, 2]) = ",  *fsolve(f, solution_int[i]), '\n')
```

```
To check our work, the solution of h'(x) = 0 using Python's fsolve function,
fsolve(f, [0, 2]) =  1.0999999815114856
```

[17]:
```
# h''(x) = 0
f = lambda x: 21.12 - 32.4*(x) + 12*(x)**2
print("To check our work, the solution of h''(x) = 0 using Python's fsolve␣
 ↪function, fsolve(f, [0, 2]) = ",  *fsolve(f, solution_int[i]), '\n')
```

```
To check our work, the solution of h''(x) = 0 using Python's fsolve function,
fsolve(f, [0, 2]) =  1.0999999999999992
```

[18]:
```
# h'''(x) = 0
f = lambda x: - 32.4 + 24*(x)
print("To check our work, the solution of h'''(x) = 0 using Python's fsolve␣
 ↪function, fsolve(f, [0, 2]) = ",  *fsolve(f, solution_int[i]), '\n')
```

```
To check our work, the solution of h'''(x) = 0 using Python's fsolve function,
fsolve(f, [0, 2]) =  1.35
```

[15]:
```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[14]:
```
# !jupyter nbconvert --to pdf /content/pr1_ii_math548_midterm_Lazizbek.ipynb
```

pr2_i_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 2. LS

Problem 2

Solve the following systems of linear equations with the Jacobi iteration method using the initial guess as [0, 0, 0].

- In each case, will the Jacobi iteration converge to a solution? Give the justification for your answer?
- If yes, find the solutions.

i.

```
[ 2  1  6][x1]   [ 9 ]

[ 8  3  2][x2] = [ 13]

[ 1  5  1][x3]   [ 7 ]
```

ii.

```
[ 8  3  2][x1]   [ 13]

[ 1  5  1][x2] = [ 7 ]

[ 2  1  6][x3]   [ 9 ]
```

2 Finding eigenstuff of a matrix

Source:

[ ]:
```
import numpy as np
from numpy.linalg import eig
a = np.array([[0, 2],
              [2, 3]])
w,v=eig(a)
print('E-value:', w)
print('E-vector', v)
```

```
E-value: [-1.  4.]
E-vector [[-0.89442719 -0.4472136 ]
```

```
[ 0.4472136  -0.89442719]]
```

3 Solving Ax=b matrix equation

Source:

#Problem 2. (i)

To check with the actual(real) solution, here, I'm giving the real solution as well:

[ ]:
```
# To solve Ax=b, We start by constructing the arrays for A and b.

A = np.array([[ 2,  1,  6],
              [ 8,  3,  2],
              [ 1,  5,  1]])
b = np.transpose(np.array([ 9,  13,  7]))
# To solve the system we do

x = np.linalg.solve(A,b)
print("Real solution: ")
print(x)
```

```
Real solution:
[1. 1. 1.]
```

Using Jacobi Iteration:

[ ]:
```
A = np.array([[ 2,  1,  6],
              [ 8,  3,  2],
              [ 1,  5,  1]])

L = np.array([[ 0,  0,  0],
              [ 8,  0,  0],
              [ 1,  5,  0]])

D = np.array([[ 2,  0,  0],
              [ 0,  3,  0],
              [ 0,  0,  1]])

U = np.array([[ 0,  1,  6],
              [ 0,  0,  2],
              [ 0,  0,  0]])

D_inverse =  np. linalg. inv(D)
b = np.transpose(np.array([ 9,  13,  7]))
D_inverse_b = np.dot(D_inverse, b)

BJ = np.dot(-D_inverse, L+U)
w,v=eig(BJ)
```

```
print('BJ evalues:', w)

# x0 = np.transpose(np.array([ 0,  0,  0]))
# x = list();
# x.append(x0)

# for i in range(10):
#   x1 = np.dot(BJ, x0)+ D_inverse_b
#   x0 = x1
#   x.append(x1)
# Aproximations = np.array(x)
# print(Aproximations)
# print((2.083**2 + 2.312**2)**(1/2))
```

```
BJ evalues: [-4.16531114+0.j          2.08265557+2.31207612j
2.08265557-2.31207612j]
```

Because the spectral radius of matrix BJ, P(BJ) = 4.16531114 > 1, Jacobi Iteration does NOT converge, so no need to turn on the part of the code for Jacobi iteration carried out.

```
[1]: # !sudo apt-get install texlive-xetex texlive-fonts-recommended
     ↪texlive-plain-generic
```

```
[ ]: # !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

pr2_ii_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 2. LS

Problem 2

Solve the following systems of linear equations with the Jacobi iteration method using the initial guess as [0, 0, 0].

- In each case, will the Jacobi iteration converge to a solution? Give the justification for your answer?
- If yes, find the solutions.

ii.

```
[ 8  3  2][x1]   [ 13]

[ 1  5  1][x2] = [ 7 ]

[ 2  1  6][x3]   [ 9 ]
```

2 Finding eigenstuff of a matrix

Source:

[ ]:
```python
import numpy as np
from numpy.linalg import eig
a = np.array([[0, 2],
              [2, 3]])
w,v=eig(a)
print('E-value:', w)
print('E-vector', v)
```

```
E-value: [-1.  4.]
E-vector [[-0.89442719 -0.4472136 ]
 [ 0.4472136  -0.89442719]]
```

3 Solving Ax=b matrix equation

Source:

#Problem 2. (ii)

To check with the actual(real) solution, here, I'm giving the real solution as well:

[ ]:
```
# To solve Ax=b, We start by constructing the arrays for A and b.

A = np.array([[ 8,  3,  2],
              [ 1,  5,  1],
              [ 2,  1,  6]])
b = np.transpose(np.array([ 13,  7,  9]))
# To solve the system we do

x = np.linalg.solve(A,b)
print("Real solution: ")
print(x)
```

```
Real solution:
[1. 1. 1.]
```

Using Jacobi Iteration:

[ ]:
```
A = np.array([[ 8,  3,  2],
              [ 1,  5,  1],
              [ 2,  1,  6]])

L = np.array([[ 0,  0,  0],
              [ 1,  0,  0],
              [ 1,  1,  0]])

D = np.array([[ 8,  0,  0],
              [ 0,  5,  0],
              [ 0,  0,  6]])

U = np.array([[ 0,  3,  2],
              [ 0,  0,  1],
              [ 0,  0,  0]])

D_inverse =  np. linalg. inv(D)
b = np.transpose(np.array([ 13,  7,  9]))
D_inverse_b = np.dot(D_inverse, b)

BJ = np.dot(-D_inverse, L+U)
w,v=eig(BJ)

print('BJ evalues:', w)

x0 = np.transpose(np.array([ 0,  0,  0]))
x = list();
x.append(x0)
```

```
for i in range(5):
  x1 = np.dot(BJ, x0)+ D_inverse_b
  x0 = x1
  x.append(x1)
Aproximations = np.array(x)
print(Aproximations)
# print((2.083**2 + 2.312**2)**(1/2))
```

```
BJ evalues: [-0.44378452  0.26976158  0.17402294]
[[0.         0.         0.        ]
 [1.625      1.4        1.5       ]
 [0.725      0.775      0.99583333]
 [1.08541667 1.05583333 1.25      ]
 [0.9165625  0.93291667 1.143125  ]
 [0.989375   0.9880625  1.19175347]]
```

Also pay attention that the spectral radius of matrix BJ, P(BJ) = 0.44378452 < 1, Jacobi Iteration does converge.

[ ]:
```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended
↪texlive-plain-generic
```

[ ]:
```
# !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

pr3_i_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 3. LS

Problem 3

You would like to use the fixed-point iteration method to find the roots of f (x) = x - x^2 = 0. Consider the following two formulations.

1.

`x = x + 2(x-x^2).`

2.

`x = x - (x - x^2) / (1 - 2x).`

a) For each formulation carry out the iterations first using the starting value 0.8 and then, using the starting value 0.2.

b) Comment and justify your observations.

#Problem 3. (i)

x = x + 2*(x-x^2).

[1]:
```python
import pandas as pd

initials = list([0.8, 0.2])
steps = list()
approximations = list()
epsilon = 0.000001

for i in range(2):
  x0 = initials[i]
  M = 10
  try:
    for k in range(M):
      steps.append(k)
      approximations.append(x0)
      x1 = x0 + 2*(x0-(x0)**2)
      if abs(x0-x1) < epsilon:
        print(f"\nWhen x0={initials[i]}, |g'({initials[i]})| < 1, so iteration␣
↪converges with tolerance of {epsilon} in {k} steps as follows:")
```

```
            break
        x0 = x1


    d = {'step k = ': steps, 'approximation x = ': approximations}
    df = pd.DataFrame(data=d)
    print(df)
    steps = []
    approximations = []

  except:
    print(f"\nWhen x0={initials[i]}, |g'({initials[i]})|>=1, so iteration␣
↪diverges in {k} steps as follows:")
    steps.pop()
    approximations.pop()
    d = {'step k = ': steps, 'approximation x = ': approximations}
    df = pd.DataFrame(data=d)
    print(df)
    steps = []
    approximations = []
```

```
   step k =   approximation x =
0         0            0.800000
1         1            1.120000
2         2            0.851200
3         3            1.104517
4         4            0.873635
5         5            1.094429
6         6            0.887738
7         7            1.087057
8         8            0.897786
9         9            1.081319
   step k =   approximation x =
0         0            0.200000
1         1            0.520000
2         2            1.019200
3         3            0.980063
4         4            1.019142
5         5            0.980125
6         6            1.019085
7         7            0.980186
8         8            1.019028
9         9            0.980247
```

[ ]:
```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[ ]:
```
# !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

pr3_ii_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 3. LS

Problem 3

You would like to use the fixed-point iteration method to find the roots of f (x) = x - x^2 = 0. Consider the following two formulations.

1.

`x = x + 2(x-x^2).`

2.

`x = x - (x - x^2) / (1 - 2x).`

a) For each formulation carry out the iterations first using the starting value 0.8 and then, using the starting value 0.2.

b) Comment and justify your observations.

2 Problem 3. (ii)

x = x - (x - x^2) / (1 - 2x)

[ ]:
```
import pandas as pd

initials = list([0.8, 0.2])
steps = list()
approximations = list()
epsilon = 0.000000001

for i in range(2):
  x0 = initials[i]
  M = 10
  try:
    for k in range(M):
      steps.append(k)
      approximations.append(x0)
      x1 = x0 - (x0 - (x0)**2) / (1 - 2*x0)
      if abs(x0-x1) < epsilon:
```

```
        print(f"\nWhen x0={initials[i]}, |g'({initials[i]})| < 1, so iteration␣
↪converges with tolerance of {epsilon} in {k} steps as follows:")
        break
      x0 = x1


    d = {'step k = ': steps, 'approximation x = ': approximations}
    df = pd.DataFrame(data=d)
    print(df)
    steps = []
    approximations = []

  except:
    print(f"\nWhen x0={initials[i]}, |g'({initials[i]})|>=1, so iteration␣
↪diverges in {k} steps as follows:")
    steps.pop()
    approximations.pop()
    d = {'step k = ': steps, 'approximation x = ': approximations}
    df = pd.DataFrame(data=d)
    print(df)
    steps = []
    approximations = []
```

```
When x0=0.8, |g'(0.8)| < 1, so iteration converges with tolerance of 1e-09 in 4
steps as follows:
   step k =   approximation x =
0         0            0.800000
1         1            1.066667
2         2            1.003922
3         3            1.000015
4         4            1.000000

When x0=0.2, |g'(0.2)| < 1, so iteration converges with tolerance of 1e-09 in 4
steps as follows:
   step k =   approximation x =
0         0        2.000000e-01
1         1       -6.666667e-02
2         2       -3.921569e-03
3         3       -1.525902e-05
4         4       -2.328306e-10
```

[ ]:
```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[ ]:
```
# !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

pr4_i_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 4. LS

Problem 4

(i)

(a) Use the Newton's method for finding the root of

( ) = ^ − 2 cos( ) = 0

in the interval [0,2] with the starting value x0 = 2.

(b) Carry out 6 iterations and comment on what you observe. If the observation needs any improvement, how will you go about it?

(c) Show that the Newton's method's convergence is of second order.

(ii)

(iii) Now, use the iteration method given below to find the root of

( ) = ^ − 2 cos( ) = 0 in the interval [0,2].

Choose your starting values as x0 = 0.6 and y0 = 0.3388.

yn+1 = yn (2 – f'(xn) yn)

xn+1 = xn – yn+1 f(xn)

(e) Compare your methods and results in (i) and (ii) and discuss any connections between (i) and (ii).

#Problem 4. (i)

[ ]:
```
# In which interval it has a solution?

import numpy as np
import math

# f(x) = math.exp(x) - 2*math.cos(x)

f0 =  math.exp(0) - 2*math.cos(0) # f0 = - 1
frightb = f0
fleftb = f0
fa = f0
```

```
fc = f0
solution_ints = list()
b = 0
a = 0
c = 0
# we'll run through [-b, a] or [c, b]

for b in np.arange(0.0, 2.0, 0.001):
  frightb = math.exp(b) - 2*math.cos(b)
  fleftb = math.exp(-b) - 2*math.cos(-b)

  if np.sign(fleftb) * np.sign(fa) <= 0: # fleftb >= 0 as fa = f0 = -1  < 0:
    # print(" = b ", b, ";", "fleftb = ", fleftb)
    solution_ints.append([-b, a])
    a = -b
    fa = fleftb

  if np.sign(frightb) * np.sign(fc) <= 0: # frightb >= 0 as fc = f0 = -1 < 0:
    # print("b = ", b, ";", "frightb = ", frightb)
    solution_ints.append([c, b])
    c = b
    fc = frightb

print("solution intervals = ", solution_ints)
```

```
solution intervals =  [[0, 0.54], [-1.454, 0]]
```

[ ]:
```
# Determine this solution using the Newton's method with a tolerance  as␣
 ↪10^(-6)
from scipy.optimize import fsolve


M = 6 # in case the program goes into infinite loops
epsilon = 10**(-8)

for i in range(1):
  solution_int = [0, 2]

  print(f"\nRoot {i+1} of f(x) in {solution_int} is as follows: ")
  x0 = solution_int[1] #  x0=2 as required
  v = math.exp(x0) - 2*math.cos(x0)  # starting function value at x0.

  print("steps", "\t ", "x", "\t", "\t", "\t", "f(x)")

  if abs(v) < epsilon:
    print(0, "\t ", x0, "\t", v)
    x1 = x0
```

```
    mnumer = (math.exp(x1) + 2*math.sin(x1))**2
    mdenom = mnumer - (math.exp(x1) - 2*math.cos(x1))*(math.exp(x1) + 2*math.
↪cos(x1))+0.00000001 # in case x0 is a desired root
    m = mnumer/mdenom
    print("m = ", m)
    print(f"multipilicity of the root {x1} is {round(m)}")

  else:
    for k in range(1, M):
      x1 = x0 - v/(math.exp(x0) + 2*math.sin(x0))
      v = math.exp(x1) - 2*math.cos(x1)
      print(k, "\t ", x1, "\t", v)
      if abs(v) < epsilon:
        mnumer = (math.exp(x1) + 2*math.sin(x1))**2
        mdenom = mnumer - (math.exp(x1) - 2*math.cos(x1))*(math.exp(x1) +␣
↪2*math.cos(x1))+0.00000001 # in case x0 is a desired root
        m = mnumer/mdenom
        print("\nm = ", m)
        print(f"multipilicity of the root {x1} is {round(m)}")
        break
      x0 = x1
```

```
Root 1 of f(x) in [0, 2] is as follows:
steps     x                       f(x)
1         1.1071175683826828      2.1311417447814685
2         0.664462389989102       0.3689486075980428
3         0.5483209267201882      0.023543300836922132
4         0.5398302921852223      0.00012382330442339828
5         0.5397851620829291      3.494306399787206e-09

m =  1.0000000002643719
multipilicity of the root 0.5397851620829291 is 1
```

(c) Show that the Newton's method's convergence is of second order.

For this we need to show g(x*) = 0;

g(x*) = 1-1/m = 1-1/1 = 0.

[ ]:
```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[ ]:
```
# !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

pr4_ii_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 4. LS

Problem 4

(ii)

(iii) Now, use the iteration method given below to find the root of

( ) = ^ − 2 cos( ) = 0 in the interval [0,2].

Choose your starting values as x0 = 0.6 and y0 = 0.3388.

yn+1 = yn (2 – f'(xn) yn)

xn+1 = xn – yn+1 f(xn)

(e) Compare your methods and results in (i) and (ii) and discuss any connections between (i) and (ii).

#Problem 4. (ii)

[ ]:
```
# Determine this solution using the Newton's method with a tolerance  as␣
↪10^(-6)
import numpy as np
import math



M = 6 # in case the program goes into infinite loops
epsilon = 10**(-8)

# f(x) = math.exp(x) - 2*math.cos(x)
x0 = 0.6
y0 = 0.3388
for i in range(1, M):
  y1 = y0*(2 - (math.exp(x0) + 2*math.sin(x0))*y0)
  x1 = x0 - y1*(math.exp(x0) - 2*math.cos(x0))

  v = math.exp(x1) - 2*math.cos(x1)
  print(i, "\t ", x1, "\t", v)
  if abs(v) < epsilon:
    break
  y0 = y1
```

```
x0 = x1
```

```
1        0.5419098217166501     0.005836841512263824
2        0.5397977761841749     3.461107957947185e-05
3        0.5397851615702651     2.087791273197581e-09
```

[ ]:
```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended
↪texlive-plain-generic
```

[ ]:
```
# !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```