pr1_a_math548_final_Lazizbek

May 2, 2024

1 Math 548, Final Exam. Problem 1. LS

Problem 1

(a) Use the Newton's method for finding a root of x^3- 3x^2 + x + 3 with the starting value of x0 = 1. Comment on what you observe. If the observation needs any improvement, how will you go about it?

(b) Find (3)^(1/2) using the fixed-point iteration method

#In which interval (a) 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)

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

import numpy as np
import math

# f(x) = (x)**3- 3*(x)**2 + (x) + 3
f0 =  (0)**3- 3*(0)**2 + (0) + 3 # f0 = 3
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 = (b)**3- 3*(b)**2 + (b) + 3
  fleftb = (-b)**3- 3*(-b)**2 + (-b) + 3

  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)
```

```
solution intervals =  [[-0.77, 0]]
```

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)

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


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

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] # gettin the right boundary
  x0 = 1 # Given initial point that satisfied the Tests 1,2,3.
  v = (x0)**3- 3*(x0)**2 + (x0) + 3  # starting function value at x0  = 1.

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

  if abs(v) < epsilon:
    print(0, "\t ", x0, "\t", v)
    x1 = x0
    mnumer = (3*(x1)**2 - 6*(x1) + 1)**2
    mdenom = mnumer - ((x1)**3- 3*(x1)**2 + (x1) + 3)*(6*(x1) - 6) + 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/(3*(x0)**2 - 6*(x0) + 1)
    v = (x1)**3- 3*(x1)**2 + (x1) + 3
    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: (x)**3- 3*(x)**2 + (x) + 3
      mnumer = (3*(x1)**2 - 6*(x1) + 1)**2
      mdenom = mnumer - ((x1)**3- 3*(x1)**2 + (x1) + 3)*(6*(x1) - 6) + 0.
↪00000001 # in case x0 is a desired root
      m = mnumer/mdenom

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

```
Root 1 of f(x) in [-0.77, 0] is as follows:
steps     x      f(x)
1         2.0    1.0
2         1.0    2.0
3         2.0    1.0
4         1.0    2.0
5         2.0    1.0
6         1.0    2.0
7         2.0    1.0
8         1.0    2.0
9         2.0    1.0
```

[ ]:
```
f = lambda x: (x)**3- 3*(x)**2 + (x) + 3
print("Using Python's fsolve function, fsolve(f, [0, 2]) = ",  *fsolve(f, 1),␣
↪'\n')
```

```
Using Python's fsolve function, fsolve(f, [0, 2]) =  1.8165405273476252


/usr/local/lib/python3.10/dist-packages/scipy/optimize/_minpack_py.py:177:
RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
```

[ ]:
```python
(1.816)**3- 3*(1.816)**2 + (1.816) + 3
```

[ ]: 0.9113384959999982

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


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

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] # gettin the right boundary
  # x0 = 1 # Given initial point that satisfied the Tests 1,2,3.
  v = (x0)**3- 3*(x0)**2 + (x0) + 3  # starting function value at x0.

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

  if abs(v) < epsilon:
    print(0, "\t ", x0, "\t", v)
    x1 = x0
    mnumer = (3*(x1)**2 - 6*(x1) + 1)**2
    mdenom = mnumer - ((x1)**3- 3*(x1)**2 + (x1) + 3)*(6*(x1) - 6) + 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/(3*(x0)**2 - 6*(x0) + 1)
      v = (x1)**3- 3*(x1)**2 + (x1) + 3
      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: (x)**3- 3*(x)**2 + (x) + 3
        mnumer = (3*(x1)**2 - 6*(x1) + 1)**2
        mdenom = mnumer - ((x1)**3- 3*(x1)**2 + (x1) + 3)*(6*(x1) - 6) + 0.
↪00000001 # in case x0 is a desired root
```

```
        m = mnumer/mdenom

        print("Using Python's fsolve function, fsolve(f, x0) = ",  *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 [-0.77, 0] is as follows:
steps     x      f(x)
1         -3.0   -54.0
2         -1.826086956521739     -14.919125503410864
3         -1.1467190137392351    -3.5995072936479193
4         -0.8423262771400923    -0.568508861805741
5         -0.7728476364392379    -0.02634488978133831
6         -0.7693013974364497    -6.684039393700658e-05
7         -0.7692923542973596    -4.340714454542649e-10

So the approximate solution 1 after 7 steps is = -0.7692923542973596
Using Python's fsolve function, fsolve(f, x0) =  -0.7692923542386315

m =  0.999999999901299
multipilicity of the root -0.7692923542973596 is 1
```

2 Observations

- I'm not quite sure what happened here (still trying to find out the reason) but the iteration points are jumping only bewteen {1, 2}. I knew that g'(1) = f' '(1) = 0 and thinking about connecting them.
- Regarding the improvements, I think I have now one as you can see from the cell above. If we start teh initial point x0 = 0 as my code found the solution interval and starting point x0 = 0 (whci satisfies the Tests 1,2,3 too), then the approxiamte solution, after quieckly 7 steps, is x ~ -0.77 with a tolerance of 10^(-6) which is very fast!

[ ]:
```
# To download
# !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
# !jupyter nbconvert --to pdf /content/pr1_i_math548_midterm_Lazizbek.ipynb
```