1. Upload the class notebook.

04.16

rootsearch.py

1. ## Find one of the roots of y(x) = -x^2 / 4 +1 with a precision of 1e-4
2. ##
3. ## 1. By choosing sufficiently small dx
4. ## 2. By dividing the interval by 5 and repeating the process starting from [1,3] bracket.
5. ##
6. ## Compare how many iterations are required for each method.
8. from numpy import sign, abs
10. ## f(2) = 0
11. f = lambda x:-x\*x/4+1
13. def rootsearch0(f,a=-100,b=100,dx=0.1):
14. dx = abs(dx)
15. x1 = a; x2 = a+dx;
16. f1 = f(x1); f2 = f(x2)
18. ite = 0
19. while sign(f1) == sign(f2):
20. if (x1 >= b):
21. return False
22. x1 += dx; x2 += dx;
23. f1 = f(x1); f2 = f(x2)
24. ite += 1
25. return ite,x1,x2
27. print(rootsearch0(f,1,3,1.0e-4))
29. def rootsearch1(f,a=-100,b=100,dx=0.1):
30. dx = abs(dx)
31. x1 = a; f1 = f(x1)
32. x2 = b; f2 = f(x2)
34. ite = 0
35. while sign(f1) != sign(f2):
36. if (x1 >= b):
37. return False
38. x1 += dx; f1 = f(x1)
39. ite += 1
41. x1 -= dx; f1 = f(x1)
42. while sign(f1) != sign(f2):
43. if x2 <= x1:
44. x2 += dx
45. return False
46. x2 -= dx; f2 = f(x2)
47. ite += 1
49. x2 += dx
50. return ite,x1,x2
52. print(rootsearch1(f,1,3,1.0e-4))

 $ python rootsearch.py

(10000, 1.9999999999998899, 2.00009999999989)

(20001, 1.99999999999989, 2.00009999999789)

04.17

bisection00.py

1. from numpy import sign, abs
3. def bisection(f,a=-100,b=100):
4. for i in range(50):
5. mp = (a + b)/2 # Midpoint in the bisection formula
6. fa = f(a)
7. fb = f(b)
9. if sign(fa)==sign(fb): return mp,abs(a-b)
11. fm = f(mp)
13. if sign(fa)!=sign(fm):
14. b = mp
15. else:
16. a = mp
17. return mp,abs(a-b)
19. f = lambda x:x\*\*3 - 20.0\*x\*\*2 + 5\*x +3
21. root,error = bisection(f)
22. print('Root:',root,' error:',error)

$ python bisection00.py

Root: -0.28061089748678825 error: 1.7763568394002505e-13

newtonRaphson.py

1. def newtonRaphson(f,x,dx = 0.1,ite = 10):
2. for i in range(ite):
3. ## calulate f'(x)
4. diff = (f(x+dx)-f(x))/dx
5. x -= f(x)/diff
6. return x

projectile\_motion.py

1. ## Projectile motion:
2. ##
3. ## Constraints: x = 300m, y = 61m, vx = - vy at (xsol,ysol)
4. ## Find the initial velocity v, time elapsed t, initial angle theta
5. ##
6. ## x = v\*cos(theta)\*t
7. ## y = v\*sin(theta)\*t - 1/2\*g\*t\*t
8. ## v\_y/v\_x = -1
10. import numpy as np
12. g = 9.8 # m/sec^2
14. def f(xsol):
15. t = xsol[0]; theta = xsol[1]; v = xsol[2]
16. f = np.zeros(3)
18. f[0] = v\*np.cos(theta)\*t - 300 # x axis eq
19. f[1] = v\*np.sin(theta)\*t - 1/2\*g\*t\*t - 61 #y axis eq
20. vx = v\*np.cos(theta)
21. vy = v\*np.sin(theta) - g\*t
22. f[2] = vy/vx + 1
24. return f
26. ## Use newtonRaphson2 module from comphy2019 github
27. from newtonRaphson2 import newtonRaphson2
29. print(newtonRaphson2(f,[5,np.pi/4,60]))

$ python project\_motion.py

[ 8.58332508 0.95279201 60.322577 ]

problem.py

1. '''
2. Roots class exercise
4. 1. Newton Raphson method for roots.
5. Find the cubic root of 70 up to three iterations
6. and give the answers with rational numbers.
8. 2. Find the smallest root of x\*\*3 - 20.0\*x\*\*2 + 5\*x +3 = 0
9. using any method.
10. '''
12. '''
13. 1.
14. import numpy as np
15. f = lambda x:x\*\*3 - 70
16. f' = lambda x:3\*x\*\*2
18. x\_n = x\_(n-1) - f(x\_(n-1))/f'(x\_(n-1))
20. x\_0 = 4
21. x\_1 = 4 + 6/48 = 4.125
22. x\_2 = 4.121288644015917
23. x\_3 = 4.12128529981127
24. f(x\_3) = 1.382716163789155e-10
25. '''
27. ## 2.
28. ## newton-raphson algorithm
30. from bisection00 import bisection
31. from newtonRaphson import newtonRaphson
33. f = lambda x:x\*\*3 - 20.0\*x\*\*2 + 5.0\*x +3
35. root, error = bisection(f)
36. print(root,f(root),error)
37. print(newtonRaphson(f,0,dx=0.001,ite=100),f(newtonRaphson(f,0,dx=0.001,ite=100)))

$ python problem.py

Root: -0.28061089748678825 error: 1.7763568394002505e-13

-0.28061089748678825 -2.0552448631860898e-12 1.7763568394002505e-13

-0.2806108974866634 0.0

newtonRaphson method is more exact

2. determine R, a and b numerically.

1. '''
2. HW6\_2.py
4. (x-a)^2 + (y-b)^2 = R^2
6. xData = [8.21, 0.34, 5.96]
7. yData = [0.00, 6.62, -1.12]
9. Determine R, a and b numerically.
10. '''
12. import numpy as np
14. ## f = (x-a)^2 + (y-b)^2 - R^2 = 0
16. xData = [8.21, 0.34, 5.96]
17. yData = [0.00, 6.62, -1.12]
19. ## Use newtonRaphson2 module from comphy2019 github
20. from newtonRaphson2 import newtonRaphson2
21. ## Make function f for using newtonRaphson2 function
22. ## f return function g
23. ## f :: (xData,yData,xsol) -> (xsol -> list)
24. def f(xData,yData):
25. def g(xsol):
26. R = xsol[0]; a = xsol[1]; b = xsol[2]
27. g = np.zeros(3)
29. for i in range(3):
30. g[i] = (xData[i]-a)\*\*2 + (yData[i]-b)\*\*2 - R\*\*2
32. return g
33. return g
35. x = [5,4,3]
36. root = newtonRaphson2(f(xData,yData),x)
38. print(root)
40. ## Test R, a, b
41. print('')
42. for i in range(3):
43. print(((xData[i]-root[1])\*\*2 + (yData[i]-root[2])\*\*2)\*\*0.5)

$ python HW6\_2.py

[5.21382431 4.83010565 3.96992168]

5.213824307236024

5.213824307236025

5.213824307236025

R = 5.21382431, a = 4.83010565, b = 3.96992168

3. find a suitable interpolating function and calculate numerically the nonzero root of

y(x) = 0.

1. '''
2. find a suitable interpolating function and calculate numerically
3. the nonzero root of y(x) = 0.
4. '''
6. import numpy as np
7. import matplotlib.pyplot as plt
9. xData = np.array([0,0.25,0.50,0.75,1.0,1.25,1.5])
10. yData = np.array([0,-1.2233,-2.2685,-2.8420,-2.2130,2.5478,55.507])
12. ## Make map function for replacing for roop to map function
13. mapli = lambda f,ar:list(map(f,ar))
15. ## Use module newtonRaphson made by me
16. from newtonRaphson import newtonRaphson
18. ## Use module neville from comphy2019 github
19. from neville import \*
21. ## Make function f for using neville function
22. ## f :: (xData, yData) -> (x -> y)
23. ## f(xData,yData) = neville(xData,yData,x)
24. f = lambda xData,yData:lambda x:neville(xData,yData,x)
25. ## g :: (x -> y)
26. g = f(xData,yData)
28. root = newtonRaphson(g,1.1,dx=0.05)
29. print(root,g(root))
31. x = np.linspace(min(xData),max(xData))
32. plt.plot(xData,yData,'o',root,g(root),'o',x,mapli(lambda x:g(x),x),'-')
33. plt.legend(['Input data','Root point','Fitting graph'])
34. plt.grid()
35. plt.savefig("HW6\_3\_1")
37. plt.xlim(root-0.05,root+0.05)
38. plt.ylim(-5,5)
39. plt.savefig("HW6\_3\_2")

$ python HW6\_3.py

1.1924987191344012 1.0784634928799145e-05

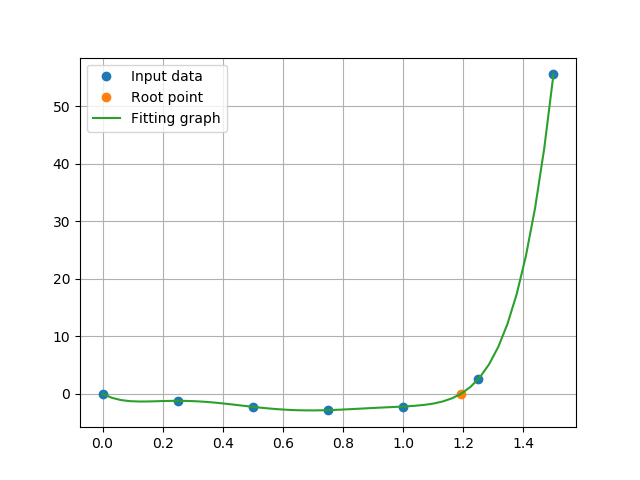


Figure 1 HW6\_3\_1.png

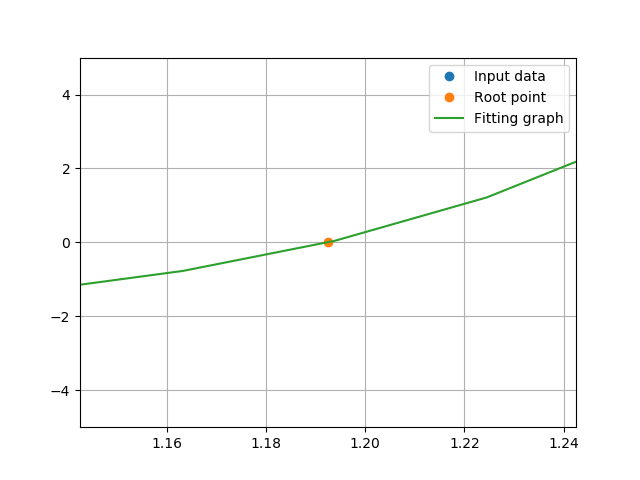


Figure 2 HW6\_3\_2.png