1. Obtain the best fitting functions for the following dataset composed of (xdata, ydata1) and

(xdata, ydata2) and their standard deviation. Plot the data points and the fitting functions.

sineFit.py

1. '''
2. Fitting sine graph form
3. f(x;a,b,omega) = a\*sin(omega\*x) + b\*cos(omega\*x)
4. '''
6. import numpy as np
8. ## sineFit :: coef -> (x -> a\*sin(omega\*x)+b\*cos(omega\*x))
9. sineFit = lambda coef:\
10. lambda x:coef[1]\*np.sin(coef[0]\*x)+coef[2]\*np.cos(coef[0]\*x)
12. ## Using module newtonRaphson2 from comphy2019 github
13. from newtonRaphson2 import newtonRaphson2
15. ## f is function for using newtonRaphson2
16. ## f :: (xData,yData)->((a,b,omega)->[dS/da,dS/db,dS/d(omega)])
17. def f(xData,yData):
18. n = len(xData)
19. ## xsol[0] = omega, xsol[1] = a, xsol[2] = b
20. def aa(xsol):
21. omega = xsol[0]; a = xsol[1]; b = xsol[2]
22. ## Calculate sum of combination of sine, cosine and y values.
23. ys, yc, ss, cc, sc = 0, 0, 0, 0, 0
24. for i in range(n):
25. ys += yData[i]\*np.sin(omega\*xData[i])
26. yc += yData[i]\*np.cos(omega\*xData[i])
27. ss += np.sin(omega\*xData[i])\*np.sin(omega\*xData[i])
28. cc += np.cos(omega\*xData[i])\*np.cos(omega\*xData[i])
29. sc += np.sin(omega\*xData[i])\*np.cos(omega\*xData[i])
31. ## aa[0] = dS/da, aa[1] = dS/db, aa[2] = dS/d(omega)
32. aa = np.zeros(3)
33. aa[0] = ys - a\*ss - b\*sc
34. aa[1] = yc - a\*sc - b\*cc
35. aa[2] = a\*yc - b\*ys - (a\*a-b\*b)\*sc - a\*b\*(cc-ss)
36. return aa
37. return aa
39. ## sineCoef :: (xData, yData) -> ((xsol = initial x value) -> coefficients)
40. sineCoef = lambda xData, yData:\
41. lambda xsol:newtonRaphson2(f(xData,yData),xsol)

main code

q1.py

1. ## Definite the function will be used
2. mapli = lambda f,x:list(map(f,x))
4. def readData( dataName ):
5. x = open( dataName, 'r')
6. temp = []
7. for ii in x.readlines():
8. temp.append(ii)
9. return mapli( float, temp )
11. ## data[0] = xdata, data[1] = ydata1, data[2] = ydata2
12. ## ydata1 is the same as ydata2
13. dataName = ['xdata.dat','ydata1.dat','ydata2.dat']
14. data = mapli( readData, dataName )
16. ## Calculate standard deciation of ydata1, ydata2
17. def stdDev(f,xData,yData):
18. n = len(xData)
19. S = 0
20. for i in range(n):
21. S += (yData[i] - f(xData[i]))\*\*2
22. sigma = (S/(n-3))\*\*0.5
23. return sigma
25. import numpy as np
26. ## Fitting Straight Line using Least-Squares method
27. from polyFit import polyFit
29. def f(coef):
30. def g(x):
31. p = 0
32. for i in range(len(coef)):
33. p += coef[i]\*(x\*\*i)
34. return p
35. return g
37. coef = polyFit(data[0],data[1],3)
38. x = np.linspace(min(data[0]),max(data[0]),1000)
39. poly = mapli(f(coef),x)
41. print('--- Polynomial Fitting ---')
42. print('coefficients ',coef)
43. print('stdDev ',stdDev(f(coef),data[0],data[1]))
44. print('')
46. ## Fitting Sine graph using Least-Squares method
47. from sineFit import sineFit, sineCoef, f
49. xsol = [0.627,2,3]
50. sinCoef = sineCoef(data[0],data[1])(xsol)
51. x = np.linspace(min(data[0]),max(data[0]),1000)
52. sin = mapli(sineFit(sinCoef),x)
54. print('--- Sine Fitting ---')
55. print('a, b, omega ',sinCoef)
56. print('stdDev ',stdDev(sineFit(sinCoef),data[0],data[1]))
58. ## Plot the data points
59. import matplotlib.pyplot as plt
61. plt.plot(data[0],data[1],'-',x,poly,'-',x,sin,'-')
62. plt.legend(['ydata','Poly Fit','Sine Fit'])
63. plt.grid()
64. plt.savefig('q1\_plot')

 $ python q1.py

--- Polynomial Fitting ---

coefficients [-0.19330605 1.18988425 -0.34536691 0.02302446]

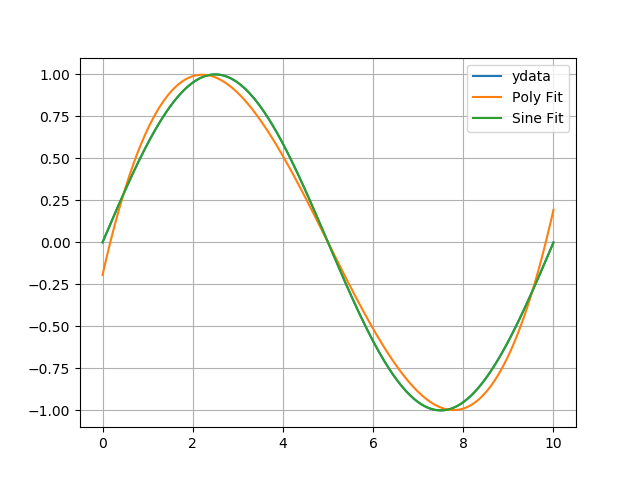
stdDev 0.06814425213272911

--- Sine Fitting ---

a, b, omega [0.62802785 0.99977 0.00145307]

stdDev 0.0006300749566380964

q1.plot.png



Sine Fitting is more detailed than Poly Fitting.

Sine Fitting is almost same as ydata.

2. Given the system below, write the equations and solve for the vertical displacements

linearSolve.py

1. import numpy as np
3. def triangular(A,B):
4. # Initializing parameters
5. nsize = len(A)
6. a = np.array(A)
7. b = np.array(B)
9. for ipiv in range(0,nsize-1):
10. for rnum in range(ipiv+1,nsize):
11. lam = a[rnum,ipiv]/a[ipiv,ipiv]
12. a[rnum] = a[rnum,0:nsize] - lam \* a[ipiv,0:nsize]
13. b[rnum] = b[rnum] - lam \* b[ipiv]
15. return [a,b]
17. def solve(A,B):
18. # Initializing parameters
19. nsize = len(A)
20. a = np.array(A)
21. b = np.array(B)
23. # Triangular
24. for ipiv in range(0,nsize-1):
25. for rnum in range(ipiv+1,nsize):
26. lam = a[rnum,ipiv]/a[ipiv,ipiv]
27. a[rnum] = a[rnum,0:nsize] - lam \* a[ipiv,0:nsize]
28. b[rnum] = b[rnum] - lam \* b[ipiv]
30. x = np.zeros(nsize)
31. x[nsize-1]=b[nsize-1]/a[nsize-1][nsize-1]
32. for n in range(2,nsize+1):
33. sum, i = 0, nsize - n
34. for j in range(i,nsize):
35. sum = sum + a[i][j] \* x[j]
36. x[i] = (b[i] - sum) / a[i][i]
38. return x
40. def solve0(A,B): # Use triangular function
41. # Initializing parameters
42. nsize = len(A)
43. C = triangular(A,B)
44. a = C[0]
45. b = C[1]
47. x = np.zeros(nsize)
48. x[nsize-1]=b[nsize-1]/a[nsize-1][nsize-1]
49. for n in range(2,nsize+1):
50. sum, i = 0, nsize - n
51. for j in range(i,nsize):
52. sum = sum + a[i][j] \* x[j]
53. x[i] = (b[i] - sum) / a[i][i]
55. return x
57. ## Use cholesky decomposition
58. def chole(a,b):
59. L = np.linalg.cholesky(a)
60. U = np.transpose(L)
61. nsize = len(a)
63. y = np.linalg.solve(L,b)
64. x = np.linalg.solve(U,y)
66. return x

q2.py

1. ## q2.py
2. ## A\*x = B
3. ##
4. ## x1 \* k' - (x2 - x1) \* k' - W = 0
5. ## x2 \* (k + k) + (x2 - x1) \* k' - (x3 - x2) \* k - W' = 0
6. ## (x3 - x2) \* k - (x5 - x3) \* (k + k) - (x4 - x3) \* k' - W' = 0
7. ## (x4 - x3) \* k' - (x5 - x4) \* k' - W = 0
8. ## (x5 - x3) \* (k + k) + (x5 - x4) \* k' - W = 0
10. import numpy as np
12. ## p is prime
13. W = 2 #kg
14. Wp = 1.5 #kg
15. k = 0.3 #N/m
16. kp = 1 #N/m
18. A = np.array([[2\*kp, -kp, 0, 0, 0],
19. [ -kp,3\*k+kp, -k, 0, 0],
20. [ 0, -k,3\*k+kp, -kp, -2\*k],
21. [ 0, 0, -kp,2\*kp, -kp],
22. [ 0, 0, -2\*k, -kp,2\*k+kp]])
23. B = np.array([W,Wp,Wp,W,Wp])
25. ## solve function from module linearSolve made myself
26. from linearSolve import solve
28. x = solve(A,B)
29. print('[ x1, x2, x3, x4, x5 ] ',x)

$ python q2.py

[ x1, x2, x3, x4, x5 ] [ 4.40909091 6.81818182 23.48484848 25.62121212 25.75757576]

3. Solve for the three angles, and the three tensions using Newton-Raphson's method and

plot the positions of the balls at equilibrium when (a) M1 = 1 Kg, M2 = 5 Kg, (b) M1 = 1

Kg, M2 = 0 Kg.

q3.py

1. ## q3.py
2. ##
3. ## M1, y: T1\*sin(theta1) - T2\*sin(theta2) - M1\*g = 0
4. ## M1, x: -T1\*cos(theta1) + T2\*cos(theta2) = 0
5. ## M2, y: T2\*sin(theta2) + T3\*sin(theta3) - M2\*g = 0
6. ## M2, x: -T2\*cos(theta2) + T3\*cos(theta3) = 0
7. ## : L1\*cos(theta1) + L2\*cos(theta2) + L3\*cos(theta3) - L = 0
8. ## : L1\*sin(theta1) + L2\*sin(theta2) - L3\*sin(theta3) = 0
9. ##
10. ## Let sin(theta) = sin, cos(theta) = cos
11. ## : sin1^2 + cos1^2 - 1= 0
12. ## : sin2^2 + cos2^2 - 1= 0
13. ## : sin3^2 + cos3^2 - 1= 0
15. import numpy as np
16. ## Use module newtonRaphson2 from comphy2019 github
17. from newtonRaphson2 import newtonRaphson2
19. ## Definite usual function
20. mapli = lambda f,x:list(map(f,x))
21. r = lambda x:mapli(lambda x:round(x,6),x)
23. ## xsol = [ T1, T2, T3, sin1, sin2, sin3, cos1, cos2, cos3 ]
24. ## unpack :: xsol -> (T, sin, cos)
25. unpack = lambda xsol:(xsol[0:3],xsol[3:6],xsol[6:9])
26. ## f is function for using function newtonRaphson2
27. ## f :: (M, L) -> (xsol -> system of equations)
28. ## M = [ M1, M2 ], L = [ L1, L2, L3, L ]
29. def f(M,L):
30. def aa(xsol):
31. g = 9.8 # m/s^2
32. T, sin, cos = unpack(xsol)
34. aa = np.zeros(9)
35. aa[0] = T[0]\*sin[0] - T[1]\*sin[1] - M[0]\*g
36. aa[1] = -T[0]\*cos[0] + T[1]\*cos[1]
37. aa[2] = T[1]\*sin[1] + T[2]\*sin[2] - M[1]\*g
38. aa[3] = -T[1]\*cos[1] + T[2]\*cos[2]
39. aa[4] = L[0]\*cos[0] + L[1]\*cos[1] + L[2]\*cos[2] - L[3]
40. aa[5] = L[0]\*sin[0] + L[1]\*sin[1] - L[2]\*sin[2]
41. aa[6] = sin[0]\*sin[0] + cos[0]\*cos[0] - 1
42. aa[7] = sin[1]\*sin[1] + cos[1]\*cos[1] - 1
43. aa[8] = sin[2]\*sin[2] + cos[2]\*cos[2] - 1
45. return aa
46. return aa
48. L = [0.5, 1.5, 2.5, 3.0]
49. ## (a) M1 = 1 kg, M2 = 5 kg
50. x = np.ones(9)
51. solve = newtonRaphson2(f([1, 5],L),x)
52. T, sin, cos = unpack(solve)
54. thetaSin = mapli(lambda x:np.arcsin(x),sin)
55. thetaCos = mapli(lambda x:np.arccos(x),cos)
57. print('(a) M1 = 1 kg, M2 = 5 kg')
58. print('[ T1, T2, T3 ] ',r(T))
59. print('\ntheta caculated from sin values')
60. print('[ theta1, theta2, theta3 ] ',r(thetaSin))
61. print('\ntheta caculated from cos values')
62. print('[ theta1, theta2, theta3 ] ',r(thetaCos))
64. ## (b) M1 = 1 kg, M2 = 0 kg
65. x = np.ones(9)
66. solve = newtonRaphson2(f([1, 0],L),x)
67. T, sin, cos = unpack(solve)
69. thetaSin = mapli(lambda x:np.arcsin(x),r(sin))
70. thetaCos = mapli(lambda x:np.arccos(x),r(cos))
72. print('')
73. print('(b) M1 = 1 kg, M2 = 0 kg')
74. print('[ T1, T2, T3 ] ',r(T))
75. print('\ntheta caculated from sin values')
76. print('[ theta1, theta2, theta3 ] ',r(thetaSin))
77. print('\ntheta caculated from cos values')
78. print('[ theta1, theta2, theta3 ] ',r(thetaCos))

$ python q3.py

(a) M1 = 1 kg, M2 = 5 kg

[ T1, T2, T3 ] [45.240168, 36.947462, 28.957307]

theta caculated from sin values

[ theta1, theta2, theta3 ] [1.069478, 0.94166, 0.7215]

theta caculated from cos values

[ theta1, theta2, theta3 ] [1.069478, 0.94166, 0.7215]

(b) M1 = 1 kg, M2 = 0 kg

[ T1, T2, T3 ] [9.8, 0.0, 0.0]

theta caculated from sin values

[ theta1, theta2, theta3 ] [1.570796, 0.792559, 0.67807]

theta caculated from cos values

[ theta1, theta2, theta3 ] [1.570796, 0.792559, 0.67807]