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

```
'''
Fitting sine graph form
f(x;a,b,omega) = a*sin(omega*x) + b*cos(omega*x)
'''

import numpy as np

## sineFit :: coef -> (x -> a*sin(omega*x)+b*cos(omega*x))
sineFit = lambda coef:\
    lambda x:coef[1]*np.sin(coef[0]*x)+coef[2]*np.cos(coef[0]*x)

## Using module newtonRaphson2 from comphy2019 github
from newtonRaphson2 import newtonRaphson2

## f is function for using newtonRaphson2
## f :: (xData,yData)->((a,b,omega)->[dS/da,dS/db,dS/d(omega)])
def f(xData,yData):
  n = len(xData)
  ## xsol[0] = omega, xsol[1] = a, xsol[2] = b
  def aa(xsol):
    omega = xsol[0]; a = xsol[1]; b = xsol[2]
    ## Calculate sum of combination of sine, cosine and y values.
    ys, yc, ss, cc, sc = 0, 0, 0, 0, 0
    for i in range(n):
      ys += yData[i]*np.sin(omega*xData[i])
      yc += yData[i]*np.cos(omega*xData[i])
      ss += np.sin(omega*xData[i])*np.sin(omega*xData[i])
      cc += np.cos(omega*xData[i])*np.cos(omega*xData[i])
      sc += np.sin(omega*xData[i])*np.cos(omega*xData[i])

    ## aa[0] = dS/da, aa[1] = dS/db, aa[2] = dS/d(omega)
    aa = np.zeros(3)
    aa[0] = ys - a*ss - b*sc
    aa[1] = yc - a*sc - b*cc
    aa[2] = a*yc - b*ys - (a*a-b*b)*sc - a*b*(cc-ss)
    return aa
  return aa

## sineCoef :: (xData, yData) -> ((xsol = initial x value) -> coefficients)
sineCoef = lambda xData, yData:\
    lambda xsol:newtonRaphson2(f(xData,yData),xsol)
```

main code

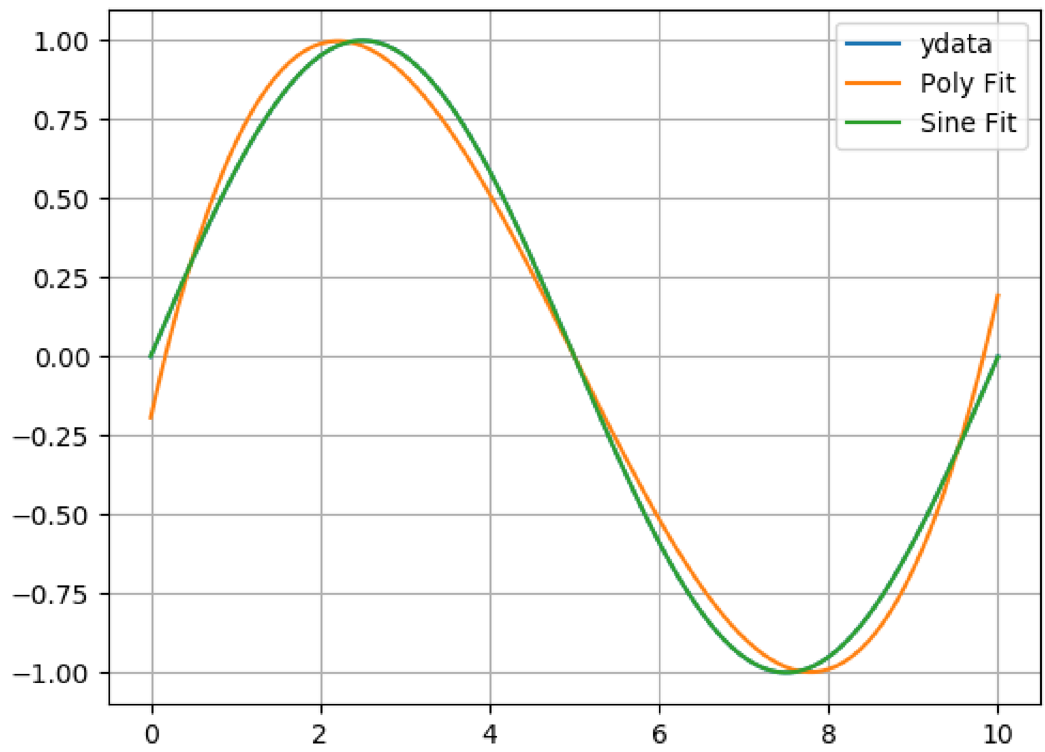
q1.py

```python
## Definite the function will be used
mapli = lambda f,x:list(map(f,x))

def readData( dataName ):
    x = open( dataName, 'r')
    temp = []
    for ii in x.readlines():
        temp.append(ii)
    return mapli( float, temp )

## data[0] = xdata, data[1] = ydata1, data[2] = ydata2
## ydata1 is the same as ydata2
dataName = ['xdata.dat','ydata1.dat','ydata2.dat']
data = mapli( readData, dataName )

## Calculate standard deciation of ydata1, ydata2
def stdDev(f,xData,yData):
    n = len(xData)
    S = 0
    for i in range(n):
        S += (yData[i] - f(xData[i]))**2
    sigma = (S/(n-3))**0.5
    return sigma

import numpy as np
## Fitting Straight Line using Least-Squares method
from polyFit import polyFit

def f(coef):
    def g(x):
        p = 0
        for i in range(len(coef)):
            p += coef[i]*(x**i)
        return p
    return g

coef = polyFit(data[0],data[1],3)
x    = np.linspace(min(data[0]),max(data[0]),1000)
poly = mapli(f(coef),x)

print('--- Polynomial Fitting ---')
print('coefficients ',coef)
print('stdDev ',stdDev(f(coef),data[0],data[1]))
print('')

## Fitting Sine graph using Least-Squares method
from sineFit import sineFit, sineCoef, f

xsol = [0.627,2,3]
sinCoef = sineCoef(data[0],data[1])(xsol)
x = np.linspace(min(data[0]),max(data[0]),1000)
sin = mapli(sineFit(sinCoef),x)

print('--- Sine Fitting ---')
```

```
print('a, b, omega ',sinCoef)
print('stdDev ',stdDev(sineFit(sinCoef),data[0],data[1]))

## Plot the data points
import matplotlib.pyplot as plt

plt.plot(data[0],data[1],'-',x,poly,'-',x,sin,'-')
plt.legend(['ydata','Poly Fit','Sine Fit'])
plt.grid()
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

```
import numpy as np

def triangular(A,B):
    # Initializing parameters
    nsize   = len(A)
    a = np.array(A)
    b = np.array(B)

    for ipiv in range(0,nsize-1):
        for rnum in range(ipiv+1,nsize):
            lam    = a[rnum,ipiv]/a[ipiv,ipiv]
            a[rnum] = a[rnum,0:nsize] - lam * a[ipiv,0:nsize]
            b[rnum] = b[rnum] - lam * b[ipiv]

    return [a,b]

def solve(A,B):
    # Initializing parameters
    nsize = len(A)
    a     = np.array(A)
    b     = np.array(B)

    # Triangular
    for ipiv in range(0,nsize-1):
        for rnum in range(ipiv+1,nsize):
            lam    = a[rnum,ipiv]/a[ipiv,ipiv]
            a[rnum] = a[rnum,0:nsize] - lam * a[ipiv,0:nsize]
            b[rnum] = b[rnum] - lam * b[ipiv]

    x = np.zeros(nsize)
    x[nsize-1]=b[nsize-1]/a[nsize-1][nsize-1]
    for n in range(2,nsize+1):
        sum, i = 0, nsize - n
        for j in range(i,nsize):
            sum = sum + a[i][j] * x[j]
        x[i] = (b[i] - sum) / a[i][i]

    return x

def solve0(A,B): # Use triangular function
    # Initializing parameters
    nsize = len(A)
    C = triangular(A,B)
    a = C[0]
    b = C[1]

    x = np.zeros(nsize)
    x[nsize-1]=b[nsize-1]/a[nsize-1][nsize-1]
    for n in range(2,nsize+1):
        sum, i = 0, nsize - n
        for j in range(i,nsize):
            sum = sum + a[i][j] * x[j]
        x[i] = (b[i] - sum) / a[i][i]
```

```python

    return x

## Use cholesky decomposition
def chole(a,b):
    L = np.linalg.cholesky(a)
    U = np.transpose(L)
    nsize = len(a)

    y = np.linalg.solve(L,b)
    x = np.linalg.solve(U,y)

    return x
```

q2.py

```python
## q2.py
## A*x = B
##
## x1 * k' - (x2 - x1) * k' - W = 0
## x2 * (k + k) + (x2 - x1) * k' - (x3 - x2) * k - W' = 0
## (x3 - x2) * k - (x5 - x3) * (k + k) - (x4 - x3) * k' - W' = 0
## (x4 - x3) * k' - (x5 - x4) * k' - W = 0
## (x5 - x3) * (k + k) + (x5 - x4) * k' - W = 0

import numpy as np

## p is prime
W  = 2 #kg
Wp = 1.5 #kg
k  = 0.3 #N/m
kp = 1 #N/m

A = np.array([[2*kp,   -kp,     0,   0,     0],
            [ -kp,3*k+kp,    -k,   0,     0],
            [   0,    -k,3*k+kp, -kp,  -2*k],
            [   0,     0,   -kp,2*kp,   -kp],
            [   0,     0,  -2*k, -kp,2*k+kp]])
B = np.array([W,Wp,Wp,W,Wp])

## solve function from module linearSolve made myself
from linearSolve import solve

x = solve(A,B)
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

```
## q3.py
##
## M1, y:  T1*sin(theta1) - T2*sin(theta2) - M1*g = 0
## M1, x: -T1*cos(theta1) + T2*cos(theta2)        = 0
## M2, y:  T2*sin(theta2) + T3*sin(theta3) - M2*g = 0
## M2, x: -T2*cos(theta2) + T3*cos(theta3)        = 0
##      :  L1*cos(theta1) + L2*cos(theta2) + L3*cos(theta3) - L = 0
##      :  L1*sin(theta1) + L2*sin(theta2) - L3*sin(theta3)     = 0
##
## Let sin(theta) = sin, cos(theta) = cos
##      :  sin1^2 + cos1^2 - 1= 0
##      :  sin2^2 + cos2^2 - 1= 0
##      :  sin3^2 + cos3^2 - 1= 0

import numpy as np
## Use module newtonRaphson2 from comphy2019 github
from newtonRaphson2 import newtonRaphson2

## Definite usual function
mapli = lambda f,x:list(map(f,x))
r = lambda x:mapli(lambda x:round(x,6),x)

## xsol = [ T1, T2, T3, sin1, sin2, sin3, cos1, cos2, cos3 ]
## unpack :: xsol -> (T, sin, cos)
unpack = lambda xsol:(xsol[0:3],xsol[3:6],xsol[6:9])
## f is function for using function newtonRaphson2
## f :: (M, L) -> (xsol -> system of equations)
## M = [ M1, M2 ], L = [ L1, L2, L3, L ]
def f(M,L):
    def aa(xsol):
        g = 9.8 # m/s^2
        T, sin, cos = unpack(xsol)

        aa = np.zeros(9)
        aa[0] =  T[0]*sin[0] - T[1]*sin[1] - M[0]*g
        aa[1] = -T[0]*cos[0] + T[1]*cos[1]
        aa[2] =  T[1]*sin[1] + T[2]*sin[2] - M[1]*g
        aa[3] = -T[1]*cos[1] + T[2]*cos[2]
        aa[4] =  L[0]*cos[0] + L[1]*cos[1] + L[2]*cos[2] - L[3]
        aa[5] =  L[0]*sin[0] + L[1]*sin[1] - L[2]*sin[2]
        aa[6] =  sin[0]*sin[0] + cos[0]*cos[0] - 1
        aa[7] =  sin[1]*sin[1] + cos[1]*cos[1] - 1
        aa[8] =  sin[2]*sin[2] + cos[2]*cos[2] - 1

        return aa
    return aa

```

```python
L = [0.5, 1.5, 2.5, 3.0]
## (a) M1 = 1 kg, M2 = 5 kg
x = np.ones(9)
solve = newtonRaphson2(f([1, 5],L),x)
T, sin, cos = unpack(solve)

thetaSin = mapli(lambda x:np.arcsin(x),sin)
thetaCos = mapli(lambda x:np.arccos(x),cos)

print('(a) M1 = 1 kg, M2 = 5 kg')
print('[ T1, T2, T3 ] ',r(T))
print('\ntheta caculated from sin values')
print('[ theta1, theta2, theta3 ] ',r(thetaSin))
print('\ntheta caculated from cos values')
print('[ theta1, theta2, theta3 ] ',r(thetaCos))

## (b) M1 = 1 kg, M2 = 0 kg
x = np.ones(9)
solve = newtonRaphson2(f([1, 0],L),x)
T, sin, cos = unpack(solve)

thetaSin = mapli(lambda x:np.arcsin(x),r(sin))
thetaCos = mapli(lambda x:np.arccos(x),r(cos))

print('')
print('(b) M1 = 1 kg, M2 = 0 kg')
print('[ T1, T2, T3 ] ',r(T))
print('\ntheta caculated from sin values')
print('[ theta1, theta2, theta3 ] ',r(thetaSin))
print('\ntheta caculated from cos values')
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]
```