Solutions to Homework 4

Question 1: 20 points.

Problem Statement. Figure 1 is a three-dimensional view of a 2 by 2 km site that is believed to overlay a thick layer of mineral deposits.

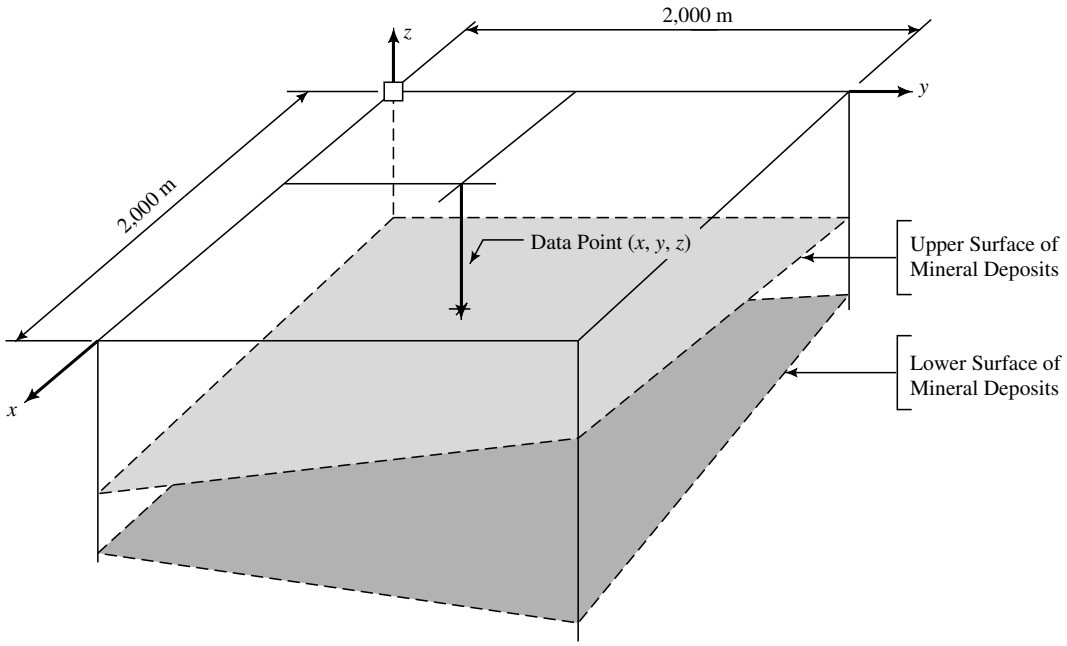

Figure 1: Three-dimensional view of mineral deposits.

To create a model of the mineral deposit profile and establish the economic viability of mining the site, a preliminary subsurface exploration consisting of 16 bore holes is conducted. Each bore hole is drilled to approximately 45 m, with the upper and lower boundaries of mineral deposits being recorded. The bore hole data is as follows:

Borehole	X (m)	Y(m)	Upper Surface (m)	Lower Surface (m)
1,	30.0,	30.0,	-8.5,	-52.5
2,	770.0,	30.0,	-7.0,	-51.8
3,	1230.0,	30.0,	-6.0,	-51.3
4,	1970.0,	30.0,	-4.6,	-50.5

```
 5,   30.0,  770.0,            -22.2,            -53.4
 6,  770.0,  770.0,            -10.8,            -52.6
 7, 1230.0,  770.0,             -9.8,            -52.1
 8, 1970.0,  770.0,             -8.3,            -51.4
 9,   30.0, 1230.0,            -14.7,            -54.0
10,  770.0, 1230.0,            -13.2,            -53.2
11, 1230.0, 1230.0,            -12.2,            -52.7
12, 1970.0, 1230.0,            -10.8,            -52.0
13,   30.0, 1970.0,            -18.4,            -54.8
14,  770.0, 1970.0,            -17.0,            -54.1
15, 1230.0, 1970.0,            -16.0,            -53.6
16, 1970.0, 1970.0,            -14.5,            -53.9
-------------------------------------------------------------
```

With the bore hole data collected, the next step is to create a simplified three-dimensional computer model of the site and subsurface mineral deposits. The mineral deposits will be modeled as a single six-sided object. The four vertical sides are simply defined by the boundaries of the site. The upper and lower sides are to be defined by a three-dimensional plane

$$z(x, y) = a_o + a_1 \cdot x + a_2 \cdot y \tag{1}$$

where coefficients a_o, a_1, and a_2 correspond to minimum values of

$$S\left(a_o,\ a_1,\ a_2\right) = \sum_{i=1}^{N} \left[z_i - z(x_i, y_i)\right]^2 \tag{2}$$

Things to do:

1. Show that minimum value of $S(a_o, a_1, a_2)$ corresponds to the solution of the matrix equations

$$\begin{bmatrix} N & \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i \cdot y_i \\ \sum_{i=1}^{N} y_i & \sum_{i=1}^{N} x_i \cdot y_i & \sum_{i=1}^{N} y_i^2 \end{bmatrix} \cdot \begin{bmatrix} a_o \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} z_i \\ \sum_{i=1}^{N} x_i \cdot z_i \\ \sum_{i=1}^{N} y_i \cdot z_i \end{bmatrix} \tag{3}$$

2. Create a comma-separated datafile (e.g., borehole-data.csv) for the geological surface data.
3. Write a Python program to read the borehole csv datafile, and then create a three-dimensional plot of the geological borehole data at the lower and upper surfaces.
4. Set up and solve the matrix equations derived in part 1 for the upper and lower mineral planes. Compute and print the average depth and volume of mineral deposits enclosed within the site.

Note. The least squares solution corresponds to the minimum value of function $S(a_o, a_1, a_2)$. At the minimum function value, we will have

$$\frac{\partial S}{\partial a_o} = \frac{\partial S}{\partial a_1} = \frac{\partial S}{\partial a_2} = 0 \tag{4}$$

Matrix Equation 3 is simply the three equations 4 written in matrix form. You should find that the equation of the upper surface is close to `z(x,y) = -10.5 + x/500 - y/200` and the lower surface close is to `z(x,y) = -52.5 + x/1000 - y/850`.

Derivation of Least Squares Equations. The least squares solution corresponds to the minimum value of function $S(a_o, a_1, a_2)$. At the minimum function value, we will have:

$$\frac{\partial S}{\partial a_o} = 0 \quad \longrightarrow \quad N a_o + \left[\sum_{i=1}^{N} x_i\right] a_1 + \left[\sum_{i=1}^{N} y_i\right] a_2 = \left[\sum_{i=1}^{N} z_i\right], \tag{5}$$

$$\frac{\partial S}{\partial a_1} = 0 \quad \longrightarrow \quad \left[\sum_{i=1}^{N} x_i\right] a_o + \left[\sum_{i=1}^{N} x_i^2\right] a_1 + \left[\sum_{i=1}^{N} x_i \cdot y_i\right] a_2 = \left[\sum_{i=1}^{N} x_i \cdot z_i\right], \tag{6}$$

$$\frac{\partial S}{\partial a_2} = 0 \quad \longrightarrow \quad \left[\sum_{i=1}^{N} y_i\right] a_o + \left[\sum_{i=1}^{N} x_i \cdot y_i\right] a_1 + \left[\sum_{i=1}^{N} y_i^2\right] a_2 = \left[\sum_{i=1}^{N} y_i \cdot z_i\right]. \tag{7}$$

Matrix equation 3 is simply equations 5 – 7 written in matrix form.

Python Source Code:

```
# ============================================================================
# TestLeastSquaresGeologicalBorings02.py: Least squares analysis of
# geological borings.
#
# Written by: Mark Austin                                        November 2025
# ============================================================================

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
```

```
from pandas import DataFrame
from pandas import read_csv

# ============================================================================
# Functions to print one-dimensional vectors and two-dimensional matrices ...
# ============================================================================

def PrintMatrix(name, a):
    print("Matrix: {:s} ".format(name) );
    for row in a:
        for col in row:
            print("{:16.7e}".format(col), end=" ")
        print("")

def PrintVector(name, a):
    print("");
    print("Vector: {:s} ".format(name) );
    for col in a:
       print("{:16.7e}".format(col), end=" ")
       print("")
    print("")

# ================================================================
# Main function ...
# ================================================================

def main():
    print("--- Enter TestLeastSquaresGeologicalBorings02.main() ... ");
    print("--- ============================================== ... ");
    print("");

    # load dataset

    print("--- ");
    print("--- Part 01: Load geological borehole data file ... ");
    print("--- ");

    df = pd.read_csv('data/geotechnical/borehole-data05.csv')
    print(df)

    # Convert dataframe to numpy array ...

    print("--- Convert dataframe to numpy array ... ");

    data = df.to_numpy()
    print(data)

    x = data[:, 1]
    y = data[:, 2]
    upper = data[:, 3]
    lower = data[:, 4]

    PrintVector("X",x)
    PrintVector("Y",y)
```

```
PrintVector("Upper surface", upper)
PrintVector("Lower surface", lower)

print("--- ");
print("--- Part 02: 3D Scatter plot of geological boring data ... ");
print("--- ");

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

ax.scatter( x, y, upper, c='b', marker='o', label='upper mineral surface')
ax.scatter( x, y, lower, c='g', marker='D', label='lower mineral surface')

ax.set_xlabel('x axis')
ax.set_ylabel('y axis')
ax.set_zlabel('z axis')

plt.title("Scatter Plot of Geological Borings")
plt.legend()
plt.tight_layout()
plt.show()

print("--- ");
print("--- Part 03: Compute terms in least squares matrix ... ");
print("--- ");

# Assemble main least squares matrix ...

print("--- Assemble main least squares matrix ...");

sumx  = 0.0; sumy  = 0.0; sumx2 = 0.0; sumy2 = 0.0; sumxy = 0.0;
sumu = 0.0; sumux = 0.0; sumuy = 0.0;
suml = 0.0; sumlx = 0.0; sumly = 0.0;
for i in range(len(x)):
    sumx = sumx + x[i]
    sumy = sumy + y[i]
    sumx2 = sumx2 + x[i]*x[i]
    sumy2 = sumy2 + y[i]*y[i]
    sumxy = sumxy + x[i]*y[i]

    # terms for upper geological layer ...

    sumu  = sumu  + upper[i]
    sumux = sumux + upper[i]*x[i]
    sumuy = sumuy + upper[i]*y[i]

    # terms for lower geological layer ...

    suml  = suml  + lower[i]
    sumlx = sumlx + lower[i]*x[i]
    sumly = sumly + lower[i]*y[i]

print("--- Assemble A matrix (main least squares analysis) ...");

N = len(x);
```

```
A = np.array( [ [     N,  sumx,  sumy ],
                [  sumx, sumx2, sumxy ],
                [  sumy, sumxy, sumy2 ] ] )

PrintMatrix("A",A)

# B matrix (upper surface) ...

print("--- Assemble B matrix (upper surface) ...");

Bupper = np.array( [ [ sumu ], [ sumux ], [ sumuy ] ] )

PrintMatrix("B (upper surface)",Bupper)

# B matrix (lower surface) ...

print("--- Assemble B matrix (lower surface) ...");

Blower = np.array( [ [ suml ], [ sumlx ], [ sumly ] ] )

PrintMatrix("B (lower surface)",Blower)

print("--- ");
print("--- Part 04: Solve A.x = B for upper surface ... ");
print("--- ");

xupper = np.linalg.solve(A, Bupper)
PrintMatrix("Upper surface equation", xupper);

print("--- ");
print("--- Upper surface equation:");
print("--- z(x,y) = {:6.3f} + {:6.3f}x + {:6.3f}y ...".format( xupper[0][0], xupper[1][0], xupper

print("--- Upper surface mid-point:");
zuppermidpoint = xupper[0][0] + xupper[1][0]*1000 + xupper[2][0]*1000
print("--- z(1000,1000) = {:6.3f} ...".format( zuppermidpoint ) );

print("--- ");
print("--- Part 05: Solve A.x = B for lower surface ... ");
print("--- ");

xlower = np.linalg.solve(A, Blower)
PrintMatrix("Lower surface equation", xlower);

print("--- ");
print("--- Lower surface equation:");
print("--- z(x,y) = {:6.3f} + {:6.3f}x + {:6.3f}y ...".format( xlower[0][0], xlower[1][0], xlower

print("--- Lower surface mid-point:");
zlowermidpoint = xlower[0][0] + xlower[1][0]*1000 + xlower[2][0]*1000
print("--- z(1000,1000) = {:6.3f} ...".format( zlowermidpoint ) );

print("--- ");
print("--- Part 06: Compute volume of minerals ... ");
print("--- ");
```

```
    volume = (zuppermidpoint - zlowermidpoint)*2000*2000
    print("--- Mineral Volume = {:16.3e} (m^3) ...".format( volume ) );

    print("--- ================================================ ... ");
    print("--- Leave TestLeastSquaresGeologicalBorings02.main() ... ");

# call the main method ...

main()
```

Abbreviated Output:

```
--- Enter TestLeastSquaresGeologicalBorings02.main() ...
--- ================================================ ...
---
--- Part 01: Load geological borehole data file ...
---
    Borehole    X (m)     Y(m)   Upper Surface (m)   Lower Surface (m)
0          1     30.0     30.0                -8.5               -52.5
1          2    770.0     30.0                -7.0               -51.8
2          3   1230.0     30.0                -6.0               -51.3
3          4   1970.0     30.0                -4.6               -50.5
4          5     30.0    770.0               -22.2               -53.4
5          6    770.0    770.0               -10.8               -52.6
6          7   1230.0    770.0                -9.8               -52.1
7          8   1970.0    770.0                -8.3               -51.4
8          9     30.0   1230.0               -14.7               -54.0
9         10    770.0   1230.0               -13.2               -53.2
10        11   1230.0   1230.0               -12.2               -52.7
11        12   1970.0   1230.0               -10.8               -52.0
12        13     30.0   1970.0               -18.4               -54.8
13        14    770.0   1970.0               -17.0               -54.1
14        15   1230.0   1970.0               -16.0               -53.6
15        16   1970.0   1970.0               -14.5               -53.9

--- Convert dataframe to numpy array ...

[[ 1.00e+00  3.00e+01  3.00e+01 -8.50e+00 -5.25e+01]
 [ 2.00e+00  7.70e+02  3.00e+01 -7.00e+00 -5.18e+01]
 [ 3.00e+00  1.23e+03  3.00e+01 -6.00e+00 -5.13e+01]
 [ 4.00e+00  1.97e+03  3.00e+01 -4.60e+00 -5.05e+01]
 [ 5.00e+00  3.00e+01  7.70e+02 -2.22e+01 -5.34e+01]
 [ 6.00e+00  7.70e+02  7.70e+02 -1.08e+01 -5.26e+01]
 [ 7.00e+00  1.23e+03  7.70e+02 -9.80e+00 -5.21e+01]
 [ 8.00e+00  1.97e+03  7.70e+02 -8.30e+00 -5.14e+01]
 [ 9.00e+00  3.00e+01  1.23e+03 -1.47e+01 -5.40e+01]
 [ 1.00e+01  7.70e+02  1.23e+03 -1.32e+01 -5.32e+01]
 [ 1.10e+01  1.23e+03  1.23e+03 -1.22e+01 -5.27e+01]
 [ 1.20e+01  1.97e+03  1.23e+03 -1.08e+01 -5.20e+01]
 [ 1.30e+01  3.00e+01  1.97e+03 -1.84e+01 -5.48e+01]
 [ 1.40e+01  7.70e+02  1.97e+03 -1.70e+01 -5.41e+01]
```

```
 [ 1.50e+01  1.23e+03  1.97e+03 -1.60e+01 -5.36e+01]
 [ 1.60e+01  1.97e+03  1.97e+03 -1.45e+01 -5.39e+01]]

Vector: X                       Vector: Y
   3.0000000e+01                   3.0000000e+01
   7.7000000e+02                   3.0000000e+01
   1.2300000e+03                   3.0000000e+01

   ... lines of output removed ...

   7.7000000e+02                   1.9700000e+03
   1.2300000e+03                   1.9700000e+03
   1.9700000e+03                   1.9700000e+03

Vector: Upper surface           Vector: Lower surface
  -8.5000000e+00                  -5.2500000e+01
  -7.0000000e+00                  -5.1800000e+01
  -6.0000000e+00                  -5.1300000e+01

   ... lines of output removed ...

  -1.7000000e+01                  -5.4100000e+01
  -1.6000000e+01                  -5.3600000e+01
  -1.4500000e+01                  -5.3900000e+01

---
--- Part 02: 3D Scatter plot of geological boring data ...
---
--- Part 03: Compute terms in least squares matrix ...
---
--- Assemble main least squares matrix ...
--- Assemble A matrix (main least squares analysis) ...

Matrix: A
   1.6000000e+01    1.6000000e+04    1.6000000e+04
   1.6000000e+04    2.3950400e+07    1.6000000e+07
   1.6000000e+04    1.6000000e+07    2.3950400e+07

--- Assemble B matrix (upper surface) ...

Matrix: B (upper surface)
  -1.9400000e+02
  -1.6824800e+05
  -2.3256000e+05

--- Assemble B matrix (lower surface) ...

Matrix: B (lower surface)
  -8.4390000e+02
  -8.3674700e+05
  -8.5444300e+05

---
--- Part 04: Solve A.x = B for upper surface ...
---
```

```
Matrix: Upper surface equation
  -1.0514012e+01
   3.2390823e-03
  -4.8500704e-03

--- Upper surface equation:
--- z(x,y) = -10.514 +  0.003x + -0.005y ...
--- Upper surface mid-point:
--- z(1000,1000) = -12.125 ...

---
--- Part 05: Solve A.x = B for lower surface ...
---

Matrix: Lower surface equation
  -5.2317356e+01
   8.9970316e-04
  -1.3260968e-03

--- Lower surface equation:
--- z(x,y) = -52.317 +  0.001x + -0.001y ...
--- Lower surface mid-point:
--- z(1000,1000) = -52.744 ...
---
--- Part 06: Compute volume of minerals ...
---
--- Mineral Volume = 1.625e+08 (m^3) ...
--- ================================================ ...
--- Leave TestLeastSquaresGeologicalBorings02.main() ...
```

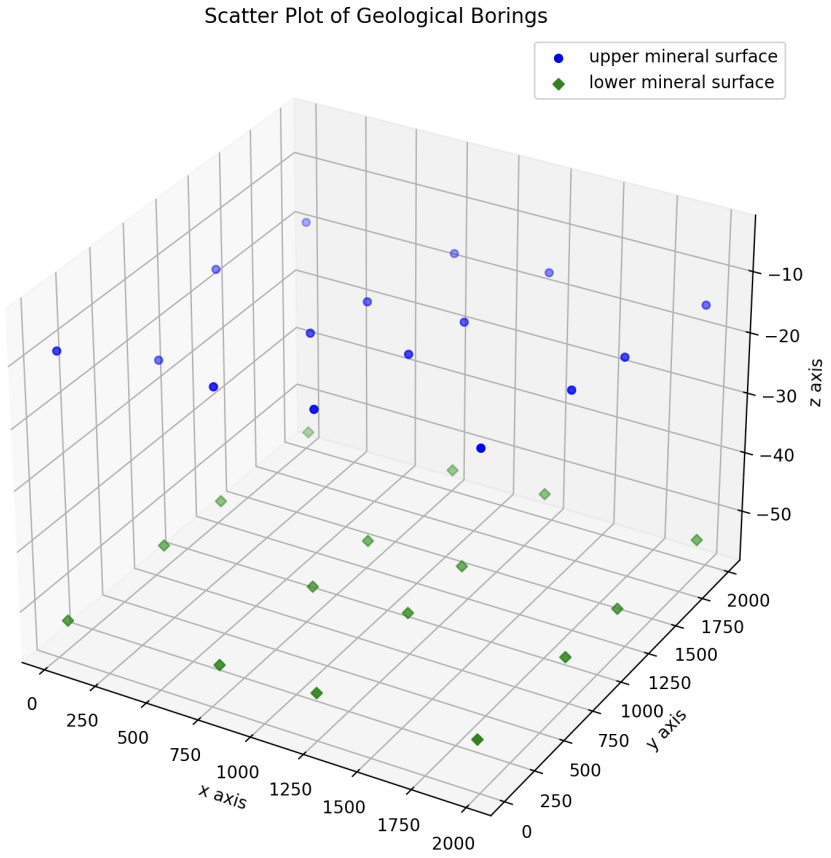

Figure 2: Scatter plot of geological borings data.

Question 2: 10 points.

Problem Statement. It is well known that first derivative of $f(x) = sin(x)$ is $cos(x)$. Given the double angle formulae,

$$sin(a+b) = sin(a)cos(b) + cos(a)sin(b) \tag{8}$$

and

$$sin(a-b) = sin(a)cos(b) - cos(a)sin(b) \tag{9}$$

write a Python program that will estimate forward and central finite difference approximations. For each approximation, plot the error in the derivative estimate versus h about the point x=0. Plots covering the interval h = -$\pi/4$ to h = $\pi/4$ might be reasonable. Repeat for x=$\pi/2$.

Derivation of Difference Approximations. Forward finite difference approximation:

$$\frac{df(x)}{dx} = \left[\frac{\sin(x+h) - \sin(x)}{h}\right] = cos(x)\left[\frac{\sin(h)}{h}\right] + sin(x)\left[\frac{\cos(h)-1}{h}\right]. \tag{10}$$

Central finite difference approximation: Given the double angle formulae,

$$\frac{df(x)}{dx} = \left[\frac{\sin(x+h) - \sin(x-h)}{2h}\right] = cos(x)\left[\frac{\sin(h)}{h}\right]. \tag{11}$$

Preliminary Observations. As expected, equations 10 and 11 both converge toward the analytical solution cos(x) as h approaches zero. The question of interest is: how far do the various finite difference approximations to the derivative of sin(x) deviate from the analytical solution? In general we expect that equations 10 and 11 will provide O(h) and O(h^2) accuracy, respectively. In other words, the central finite difference approximation should be more accurate than the forward finite difference approximation. But this is just an overall expectation and several interesting exceptions exist. First, from equation 10 we expect that when sin(x) = 0, the forward finite difference approximation will be identical to the central difference approximation. And second, from equation 11 we expect that when cos(x) = 0, the central finite difference approximation will provide an exact answer, irrespective of the step length h.

Python Source Code:

```
# ============================================================================
# TestErrorAnalysis02.py Compute and plot accuracy of finite difference approx
# to an analytical derivative, f(x)  = sin(x), df(x)/dx = cos(x),
```

```
# using both the forward and central finite difference approximations.
#
# Two points of interest: 0.0 radians and pi/2 radians. For both cases, we
# evaluate the derivatives x +/- pi/4 radians.
#
# Written By : Mark Austin                                          November 2025
# ============================================================================

import math
import numpy as np
import matplotlib.pyplot as plt

# Main function ...

def main():
    print("--- Enter TestErrorAnalysis02.main()           ... ");
    print("--- ========================================== ... ");

    # Setup working arrays ....

    nopoints = 23;
    h = np.zeros( nopoints );
    error1 = np.zeros( nopoints );
    error2 = np.zeros( nopoints );

    print("--- Part 1: Error analysis when x = 0 ... ");

    x  = 0;
    df = math.cos(x);

    for i in range(len(h)):
        h[i] = (i-11)/10*(math.pi/4.0)
        dx   = h[i]
        error1[i] = (math.sin( x+dx )-math.sin( x ))/dx - df;
        error2[i] = (math.sin( x+dx )-math.sin( x-dx ))/(2*dx) - df;

    # Create plot of errors in derivative estimates ...

    plt.plot( h, error1, '-', label = 'Error: (sin(x+h)-sin(x))/h')
    plt.plot( h, error2, '*', label = 'Error: (sin(x+h)-sin(x-h))/2h')
    plt.title('Finite difference approximation to f(x) = sin(x) about x = 0');
    plt.xlabel('Step length (x+h)-(x)');
    plt.ylabel('Error in derivative estimate');
    plt.legend()
    plt.grid(True)
    plt.show()

    print("--- Part 2: Error analysis when x = math.pi/2 ... ");

    x  = math.pi/2.0;
    df = math.cos(x);

    for i in range(len(h)):
        h[i] = (i-11)/10*(math.pi/4.0)
        dx   = h[i]
```

```
        error1[i] = (math.sin( x+dx )-math.sin( x ))/dx - df;
        error2[i] = (math.sin( x+dx )-math.sin( x-dx ))/(2*dx) - df;

    # Create plot of errors in derivative estimates ...

    plt.plot( h, error1, '-', label = 'Error: (sin(x+h)-sin(x))/h')
    plt.plot( h, error2, '*', label = 'Error: (sin(x+h)-sin(x-h))/2h')
    plt.title('Finite difference approximation to f(x) = sin(x) about x = pi/2');
    plt.xlabel('Step length (x+h)-(x)');
    plt.ylabel('Error in derivative estimate');
    plt.legend()
    plt.grid(True)
    plt.show()

    print("--- ======================================== ... ");
    print("--- Leave TestErrorAnalysis02.main()          ... ");

# call the main method ...

main()
```

Abbreviated Output:

```
--- Enter TestErrorAnalysis02.main()          ...
--- ======================================== ...

--- Part 1: Error analysis when x = 0 ...

--- Part 2: Error analysis when x = math.pi/2 ...

--- ======================================== ...
--- Leave TestErrorAnalysis02.main()          ...
```

Figures 3 and 4 validate the preliminary predictions.

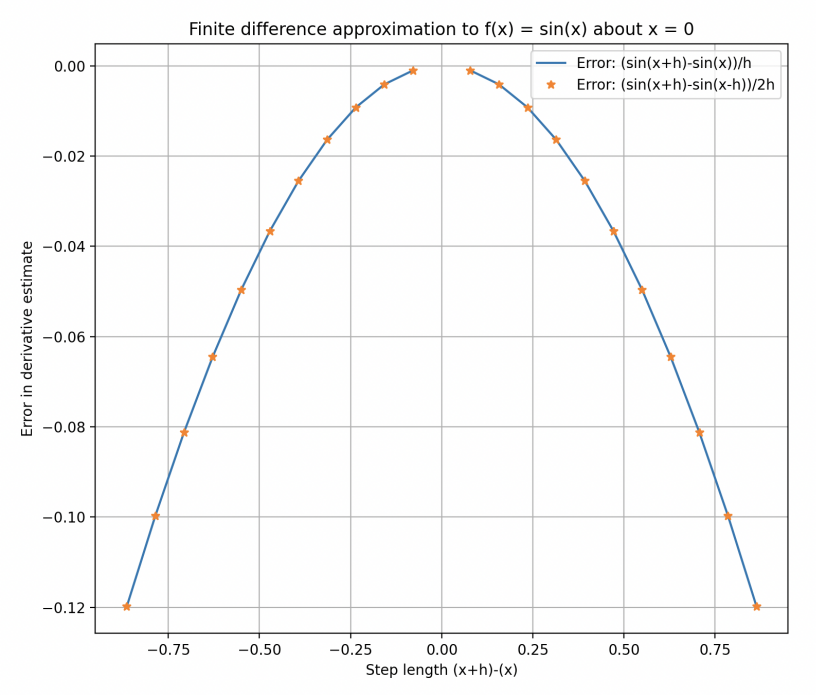

Figure 3: Finite difference approximations to f(x) = sin(x), x = 0 radians.

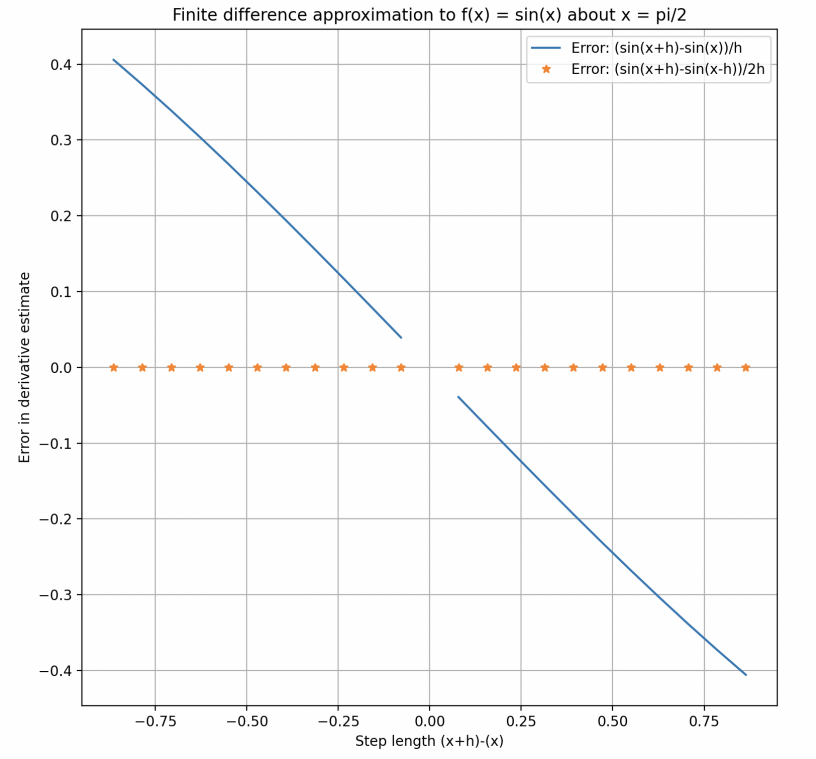

Figure 4: Finite difference approximations to f(x) = sin(x), x = pi/2 radians.

Question 3: 10 points.

Problem Statement. This question covers function interpolation with the methods of divided differences and lagrange interpolation, and curve fitting with the method of least squares. The whole question is motivated by the small dataset:

```
   x |  0.0 |   2.0 |   3.0 |   5.0
-------------------------------------
f(x) | 36.0 |  32.0 |  27.0 |  11.0
```

Part [3a] (5 pts). Use the method of **divided differences** to find a polynomial of lowest order that will fit the dataset. Be sure to show all of your working.

Solution: We seek a third order polynomial p(x):

$$p(x) = f[x_0] + f[x_o, x_1](x - x_o) + f[x_o, x_1, x_2](x - x_o)(x - x_1) + f[x_o, x_1, x_2, x_3](x - x_o)(x - x_1)(x - x_2). \tag{12}$$

where

x_i	$f[x_i] = f(x_i)$	$f[,]$	$f[,,]$	$f[,,,]$
0.0	36	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$
2.0	32	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	
3.0	27	$f[x_2, x_3]$		
5.0	11			

Elements in the divided difference table are as follows:

$$f[x_0, x_1] = \left[\frac{f(x_1) - f(x_0)}{x_1 - x_0}\right] = \left[\frac{32 - 36}{2 - 0}\right] = -2. \tag{13}$$

$$f[x_1, x_2] = \left[\frac{f(x_2) - f(x_1)}{x_2 - x_1}\right] = \left[\frac{27 - 32}{3 - 2}\right] = -5. \tag{14}$$

$$f[x_2, x_3] = \left[\frac{f(x_2) - f(x_1)}{x_2 - x_1}\right] = \left[\frac{11 - 27}{5 - 3}\right] = -8. \tag{15}$$

The second column of computations:

$$f[x_0, x_1, x_2] = \left[\frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}\right] = \left[\frac{-5 + 2}{3 - 0}\right] = -1.0. \tag{16}$$

$$f[x_1, x_2, x_3] = \left[\frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1}\right] = \left[\frac{-8+5}{5-2}\right] = -1.0. \tag{17}$$

The third column computations:

$$f[x_0, x_1, x_2, x_3] = \left[\frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_3]}{x_3 - x_0}\right] = \left[\frac{0-0}{5-0}\right] = 0.0 \tag{18}$$

Thus, the interpolated polynomial is:

$$\begin{aligned} f(x) &= f[x_0] + f[x_o, x_1](x - x_o) + f[x_0, x_1, x_2](x - x_o)(x - x_1) \\ &= 36 + -2(x-0) - 1(x-0)(x-2) + 0(x-0)(x-2)(x-3) \\ &= 36 - x^2 \end{aligned}$$

Part [3b] (5 pts). Check your answer in Part 3a by computing the functional form via the method of **Lagrange Interpolation**. Be sure to show all of your working.

Solution: For the given dataset,

$$f(x) = f(x_o)p_o(x) + f(x_1)p_1(x) + f(x_2)p_2(x) + f(x_3)p_3(x) \tag{19}$$

where

$$p_o(x) = \frac{(x-2)(x-3)(x-5)}{(0-2)(0-3)(0-5)} = \left[\frac{-x^3 + 10x^2 - 31x + 30}{30}\right]. \tag{20}$$

$$p_1(x) = \frac{(x-0)(x-3)(x-5)}{(2-0)(2-3)(2-5)} = \left[\frac{x^3 - 8x^2 + 15x}{6}\right]. \tag{21}$$

$$p_2(x) = \frac{x(x-2)(x-5)}{3(3-2)(3-5)} = \left[\frac{-x^3 + 7x^2 - 10x}{6}\right]. \tag{22}$$

$$p_3(x) = \frac{x(x-2)(x-3)}{5(5-2)(5-3)} = \left[\frac{x^3 - 5x^2 + 6x}{30}\right]. \tag{23}$$

Also note: f(x_o) = 36, f(x_1) = 32, f(x_2) = 27 and f(x_3) = 11.

Substitute equations 20 through 23 into equation 19. Notice that the constant term in equation 20 is one, and the cubic and linear terms cancel out completely. This leaves:

$$f(x) = 36 - x^2. \tag{24}$$