Solutions to Homework 5

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.

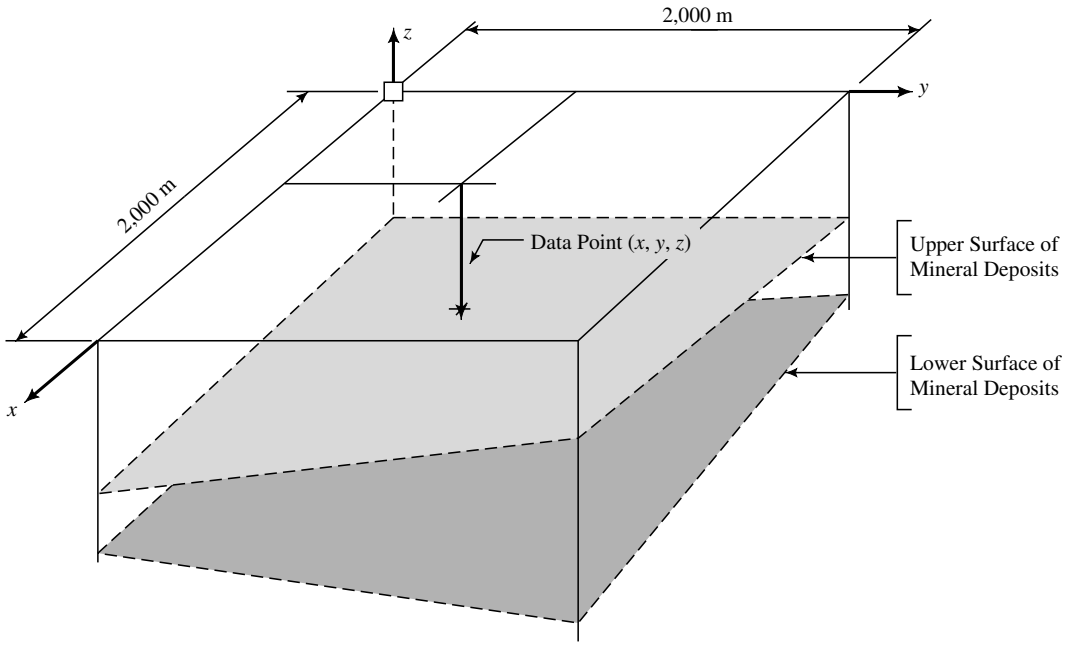

Figure 1: Three-dimensional view of mineral deposits.

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,	-18.5,	-42.5
2,	770.0,	30.0,	-17.0,	-41.8
3,	1230.0,	30.0,	-16.0,	-41.3
4,	1970.0,	30.0,	-14.6,	-40.5

```
 5,   30.0,  770.0,          -32.2,          -43.4
 6,  770.0,  770.0,          -20.8,          -42.6
 7, 1230.0,  770.0,          -19.8,          -42.1
 8, 1970.0,  770.0,          -18.3,          -41.4
 9,   30.0, 1230.0,          -24.7,          -44.0
10,  770.0, 1230.0,          -23.2,          -43.2
11, 1230.0, 1230.0,          -22.2,          -42.7
12, 1970.0, 1230.0,          -20.8,          -42.0
13,   30.0, 1970.0,          -28.4,          -44.8
14,  770.0, 1970.0,          -27.0,          -44.1
15, 1230.0, 1970.0,          -26.0,          -43.6
16, 1970.0, 1970.0,          -24.5,          -43.9
-------------------------------------------------------------
```

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.

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:

$$\frac{\partial S}{\partial a_o} = 0 \quad \longrightarrow \quad N a_o + \left[\sum_{i=1}^{N} x_i\right] a_1 + \left[\sum_{i=1}^{N} y_i\right] a_2 = \left[\sum_{i=1}^{N} z_i\right], \tag{4}$$

$$\frac{\partial S}{\partial a_1} = 0 \quad \longrightarrow \quad \left[\sum_{i=1}^{N} x_i\right] a_o + \left[\sum_{i=1}^{N} x_i^2\right] a_1 + \left[\sum_{i=1}^{N} x_i \cdot y_i\right] a_2 = \left[\sum_{i=1}^{N} x_i \cdot z_i\right], \tag{5}$$

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

Matrix equation 3 is simply equations 4 – 6 written in matrix form.

Python Source Code:

```
# ============================================================================
# TestLeastSquaresGeologicalBorings02.py: Least squares analysis of
# geological borings.
#
# Written by: Mark Austin                                          May 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-data04.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               -18.5               -42.5
1          2    770.0     30.0               -17.0               -41.8
2          3   1230.0     30.0               -16.0               -41.3
3          4   1970.0     30.0               -14.6               -40.5
4          5     30.0    770.0               -32.2               -43.4
5          6    770.0    770.0               -20.8               -42.6
6          7   1230.0    770.0               -19.8               -42.1
7          8   1970.0    770.0               -18.3               -41.4
8          9     30.0   1230.0               -24.7               -44.0
9         10    770.0   1230.0               -23.2               -43.2
10        11   1230.0   1230.0               -22.2               -42.7
11        12   1970.0   1230.0               -20.8               -42.0
12        13     30.0   1970.0               -28.4               -44.8
13        14    770.0   1970.0               -27.0               -44.1
14        15   1230.0   1970.0               -26.0               -43.6
15        16   1970.0   1970.0               -24.5               -43.9

--- Convert dataframe to numpy array ...

Vector: X
   3.0000000e+01
   7.7000000e+02
   1.2300000e+03
   1.9700000e+03
   3.0000000e+01
   7.7000000e+02
   1.2300000e+03
   1.9700000e+03
   3.0000000e+01
   7.7000000e+02
   1.2300000e+03
   1.9700000e+03
   3.0000000e+01
   7.7000000e+02
   1.2300000e+03
   1.9700000e+03

Vector: Y
   3.0000000e+01
   3.0000000e+01
   3.0000000e+01
   3.0000000e+01
```

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

Vector: Upper surface
  -1.8500000e+01
  -1.7000000e+01
  -1.6000000e+01
  -1.4600000e+01
  -3.2200000e+01
  -2.0800000e+01
  -1.9800000e+01
  -1.8300000e+01
  -2.4700000e+01
  -2.3200000e+01
  -2.2200000e+01
  -2.0800000e+01
  -2.8400000e+01
  -2.7000000e+01
  -2.6000000e+01
  -2.4500000e+01

Vector: Lower surface
  -4.2500000e+01
  -4.1800000e+01
  -4.1300000e+01
  -4.0500000e+01
  -4.3400000e+01
  -4.2600000e+01
  -4.2100000e+01
  -4.1400000e+01
  -4.4000000e+01
  -4.3200000e+01
  -4.2700000e+01
  -4.2000000e+01
  -4.4800000e+01
  -4.4100000e+01
  -4.3600000e+01
  -4.3900000e+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)
  -3.5400000e+02
  -3.2824800e+05
  -3.9256000e+05

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

Matrix: B (lower surface)
  -6.8390000e+02
  -6.7674700e+05
  -6.9444300e+05
---
--- Part 04: Solve A.x = B for upper surface ...
---
Matrix: Upper surface equation
  -2.0514012e+01
   3.2390823e-03
  -4.8500704e-03
---
--- Upper surface equation:
--- z(x,y) = -20.514 +  0.003x + -0.005y ...
---

--- Upper surface mid-point:
--- z(1000,1000) = -22.125 ...
---
--- Part 05: Solve A.x = B for lower surface ...
---
Matrix: Lower surface equation
  -4.2317356e+01
   8.9970316e-04
  -1.3260968e-03

---
--- Lower surface equation:
--- z(x,y) = -42.317 +  0.001x + -0.001y ...
---

--- Lower surface mid-point:
--- z(1000,1000) = -42.744 ...
---
--- Part 06: Compute volume of minerals ...
---
--- Mineral Volume =        8.248e+07 (m^3) ...
--- ============================================== ...
--- Leave TestLeastSquaresGeologicalBorings02.main() ...
```

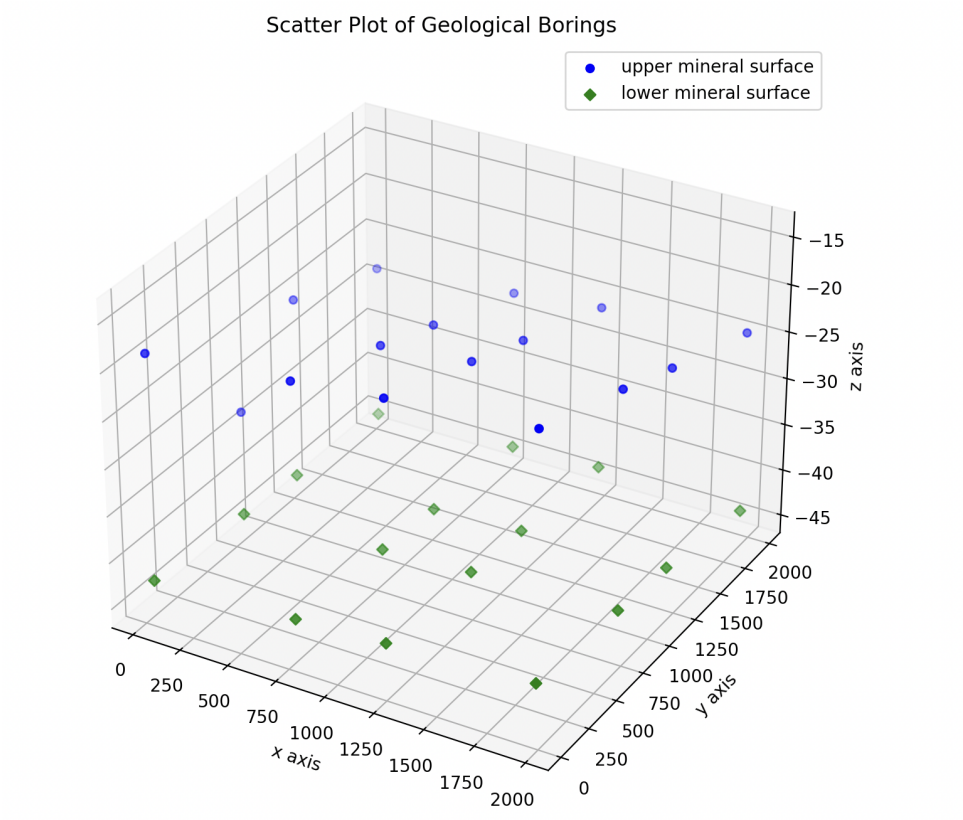

Figure 2: Scatter plot of geological borings data.

Question 2: 20 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 [2a] (10 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):

$$p(x) = f[x_0]+f[x_o,x_1](x-x_o)+f[x_o,x_1,x_2](x-x_o)(x-x_1)+f[x_o,x_1,x_2,x_3](x-x_o)(x-x_1)(x-x_2). \tag{7}$$

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:

$$f[x_0,x_1] = \left[\frac{f(x_1)-f(x_0)}{x_1-x_0}\right] = \left[\frac{32-36}{2-0}\right] = -2. \tag{8}$$

$$f[x_1,x_2] = \left[\frac{f(x_2)-f(x_1)}{x_2-x_1}\right] = \left[\frac{27-32}{3-2}\right] = -5. \tag{9}$$

$$f[x_2,x_3] = \left[\frac{f(x_2)-f(x_1)}{x_2-x_1}\right] = \left[\frac{11-27}{5-3}\right] = -8. \tag{10}$$

The second column of computations:

$$f[x_0, x_1, x_2] = \left[\frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}\right] = \left[\frac{-5+2}{3-0}\right] = -1.0. \tag{11}$$

$$f[x_1, x_2, x_3] = \left[\frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1}\right] = \left[\frac{-8+5}{5-2}\right] = -1.0. \tag{12}$$

The third column computations:

$$f[x_0, x_1, x_2, x_3] = \left[\frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_3]}{x_3 - x_0}\right] = \left[\frac{0-0}{5-0}\right] = 0.0 \tag{13}$$

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 [2b] (10 pts). Check your answer in part 2a by computing the functional form via the method of **Lagrange Interpolation**. Be sure to show all of your working.

Solution: For the given dataset,

$$f(x) = f(x_o)p_o(x) + f(x_1)p_1(x) + f(x_2)p_2(x) + f(x_3)p_3(x) \tag{14}$$

where

$$p_o(x) = \frac{(x-2)(x-3)(x-5)}{(0-2)(0-3)(0-5)} = \left[\frac{-x^3 + 10x^2 - 31x + 30}{30}\right]. \tag{15}$$

$$p_1(x) = \frac{(x-0)(x-3)(x-5)}{(2-0)(2-3)(2-5)} = \left[\frac{x^3 - 8x^2 + 15x}{6}\right]. \tag{16}$$

$$p_2(x) = \frac{x(x-2)(x-5)}{3(3-2)(3-5)} = \left[\frac{-x^3 + 7x^2 - 10x}{6}\right]. \tag{17}$$

$$p_3(x) = \frac{x(x-2)(x-3)}{5(5-2)(5-3)} = \left[\frac{x^3 - 5x^2 + 6x}{30}\right]. \tag{18}$$

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

Substitute equations 15 through 18 into equation 14. Notice that the constant term in equation 15 is one, and the cubic and linear terms cancel out completely. This leaves:

$$f(x) = 36 - x^2. \tag{19}$$

Question 3: 20 points.

Problem Statement. Figure 3 shows an enclosed region A-B-C-D-E-F-G-H whose boundary is defined by two curves:

$$f(x) = \big[x^2 - 16\big]. \tag{20}$$

and

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

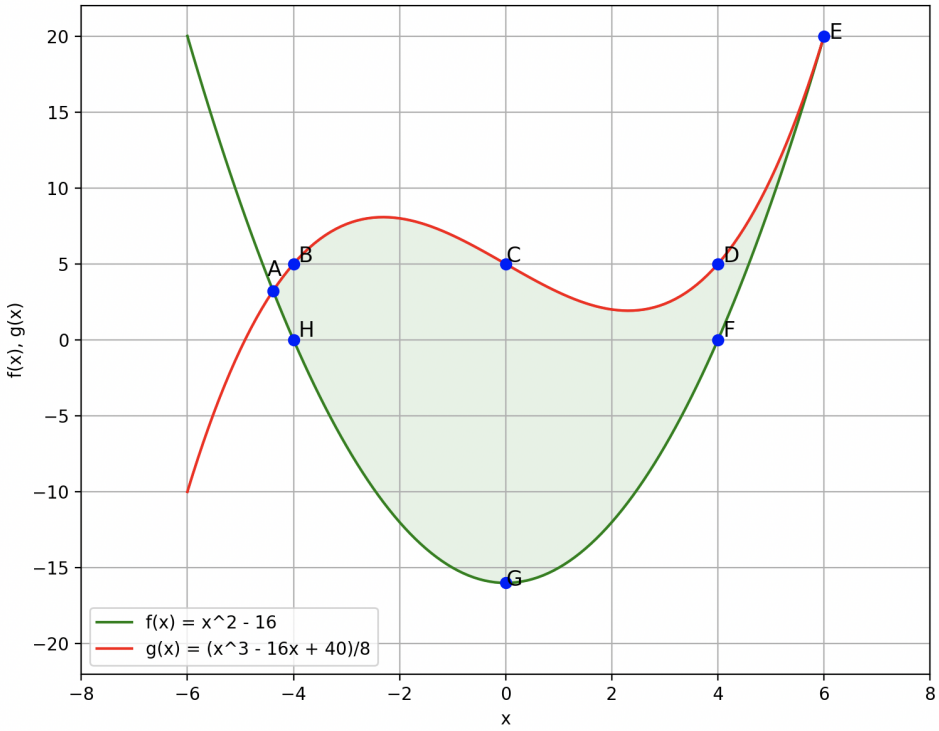

Figure 3: Filled region A-B-C-D-E-F-G-H.

From the graphic we see that curves f(x) and g(x) intersect at points A and E (i.e., in the interval $[-4, -5]$ and again $[5, 7]$).

Part [3a] (6 pts). Show that coordinate points A and E are defined by solutions to the cubic equation:

$$x^3 - 8x^2 - 16x + 168 = 0. \tag{22}$$

This is a hand calculation, so show all of your working.

Solution. At the point of intersection, $f(x) = g(x)$, i.e.,

$$\left[x^2 - 16\right] = \left[\frac{x^3}{8} - 2x + 5\right] \tag{23}$$

Rearranging equation 23 gives the required result. Equation 22 can be factorized:

$$(x - 6)\left(x^2 - 2x - 28\right) = 0. \tag{24}$$

Thus, Point A has coordinates: (x,y) = (1 - $\sqrt{29}$, 14-2$\sqrt{29}$). Point E has coordinates: (x,y) = (6,20).

Part [3b] (4 pts). Using calculus, or otherwise, show that the area of the shaded region is $\left[781 + 145\sqrt{29}\right]/12$ $\approx$ 130.154.

Solution. Notice that within the interval [1 - $\sqrt{29}$, 6], g(x) > f(x). If we let s(x) = g(x) - f(x), then the area of the shaded region is:

$$\text{Shaded Area} = \int_{1-\sqrt{29}}^{6} s(x)dx$$

$$= \left[\frac{x^4}{32} - \frac{x^3}{3} - x^2 + 21x\right]_{1-\sqrt{29}}^{6}$$

$$= \frac{1}{12}\left[781 + 145\sqrt{29}\right] \approx 130.154.$$

Part [3c] (6 pts). Demonstrate how you can use **one step** of Simpson's Rule, to obtain a high-accuracy estimate the area of region A-B-C-D-E-F-G-H.

Solution. Using one step of Simpson's Rule:

$$I = \int_a^b s(x)dx = \frac{h}{3}\left[s(a) + 4s(m) + s(b)\right] \tag{25}$$

where point a = (1 - $\sqrt{29}$), point b = 6, h = (b-a)/2, and m is the midpoint m = (a+b)/2 = (7 - $\sqrt{29}$)/2. Also, notice that by design, that s(a) = s(b) = 0.0. Hence, equation 25 simplfies to:

$$I = \int_a^b s(x)dx = \frac{4h}{3}s(m) = \frac{1}{12}\left[781 + 145\sqrt{29}\right] \approx 130.154. \tag{26}$$

Part [3d] (4 pts). What is the expected error with Simpson's Rule? Briefly compare and discuss the analytical and numerical results.

Solution. The error estimate for Simpson's Rule is as follows:

$$I = \int_a^b s(x)dx = S_1 - \frac{|s^{''''}(\xi)|}{180}h^4(b-a). \tag{27}$$

Given that s(x) is a cubic function, the fourth derivative will be zero. Hence, we expect one step of Simpson's Rule to provide an exact answer.

Question 4: 10 points.

Problem Statement: Consider the integral

$$I = \int_0^2 3x^2 + 4x^3 + 5x^4 dx = 56. \tag{28}$$

Write a Python program to compute numerical approximations to equation 28 using:

1. The Trapezoid rule,
2. Simpson's rule, and
3. Two-point Gauss Quadrature.

For cases 1 and 2, use only three data ordinates. Compute and print the absolute and relative errors for each numerical procedure.

Python Source Code:

```
# ================================================================================
# TestIntegration03.py: Use trapezod algorithm to integrate functions.
#
# Written By: Mark Austin                                                 May 2025
# ================================================================================

import math;
import Integration;

# Define mathematical functions ...

def f1(x):
    return 3*x**2 + 4*x**3 + 5*x**4

# main method ...

def main():
    print("--- Enter TestIntegration03.main()   ... ");
    print("--- ======================================== ... ");

    analytic_i = 56;

    print("--- ");
    print("--- Part 1: Integrate f1(x) with Trapezoid ... ");
    print("--- ");
```

```
# Initialize problem setup ...

a =  0.0
b =  2.0
nointervals = 2

print("--- Inputs:")
print("---   a = {:9.4f} ...".format(a) )
print("---   b = {:9.4f} ...".format(b) )
print("---   no intervals = {:d} ...".format(nointervals) )

# Compute numerical solution to integral ..

print("--- Execution:")
xi = Integration.trapezoid( f1, a, b, nointervals )

# Summary of computations ...

print("--- Output:")
print("---   Numerical Integral = {:14.4e} ...".format( xi ) )

# Error Analysis ...

print("--- Error Analysis:")
print("---   Absolute error = {:14.4e} ...".format( abs( xi-analytic_i )) )
print("---   Relative error = {:14.4e} ...".format( abs( xi-analytic_i )/analytic_i ) )

print("--- ");
print("--- Part 2: Integrate f1(x) with Simpson ... ");
print("--- ");

# Initialize problem setup ...

a = 0.0;
b = 2.0
nointervals = 2

print("--- Inputs:")
print("---   a = {:9.4f} ...".format(a) )
print("---   b = {:9.4f} ...".format(b) )
print("---   no intervals = {:d} ...".format(nointervals) )

# Compute numerical solution to integral ..

print("--- Execution:")
xi = Integration.simpson( f1, a, b, nointervals )

# Summary of computations ...

print("--- Output:")
print("---   Numerical Integral = {:14.4e} ...".format( xi ) )

# Error Analysis ...

print("--- Error Analysis:")
```

```
    print("---   Absolute error = {:14.4e} ...".format( abs( xi-analytic_i )) )
    print("---   Relative error = {:14.4e} ...".format( abs( xi-analytic_i )/analytic_i ) )

    print("--- ");
    print("--- Part 3: Integrate f1(x) with Two-Point Gauss Quadrature ... ");
    print("--- ");

    a = 0.0;
    b = 2.0

    w0 = 1.0
    u0 = - 1.0/math.sqrt(3.0)
    x0 = ((b-a)/2.0)*(1 + u0)
    w1 = 1.0
    u1 =   1.0/math.sqrt(3.0)
    x1 = ((b-a)/2.0)*(1 + u1)

    xi = ((b-a)/2.0)*( w0*f1(x0) + w1*f1(x1) )

    print("---   w0 = {:14.6e}, u0 = {:14.6e}, x0 = {:14.6e} ...".format( w0, u0, x0 ) )
    print("---   w1 = {:14.6e}, u1 = {:14.6e}, x1 = {:14.6e} ...".format( w1, u1, x1 ) )
    print("---   Numerical Integral f1(x) dx = {:14.4f} ...".format( xi ) )

    # Error Analysis ...

    print("--- Error Analysis:")
    print("---   Absolute error = {:14.4e} ...".format( abs( xi-analytic_i )) )
    print("---   Relative error = {:14.4e} ...".format( abs( xi-analytic_i )/analytic_i ) )

    print("--- ======================================== ... ");
    print("--- Leave TestIntegration03.main()   ... ");

# call the main method ...

main()
```

Abbreviated Output:

```
--- Enter TestIntegration03.main()   ...
--- ======================================== ...
---
--- Part 1: Integrate f1(x) with Trapezoid ...
---
--- Inputs:
---   a =    0.0000 ...
---   b =    2.0000 ...
---   no intervals = 2 ...
--- Execution:
--- Output:
---   Numerical Integral =     7.4000e+01 ...
--- Error Analysis:
---   Absolute error =     1.8000e+01 ...
```

```
---   Relative error =     3.2143e-01 ...
---
--- Part 2: Integrate f1(x) with Simpson ...
---
--- Inputs:
---   a =    0.0000 ...
---   b =    2.0000 ...
---   no intervals = 2 ...
--- Execution:
--- Output:
---   Numerical Integral =     5.7333e+01 ...
--- Error Analysis:
---   Absolute error =     1.3333e+00 ...
---   Relative error =     2.3810e-02 ...
---
--- Part 3: Integrate f1(x) with Two-Point Gauss Quadrature ...
---
---   w0 =   1.000000e+00, u0 =  -5.773503e-01, x0 =   4.226497e-01 ...
---   w1 =   1.000000e+00, u1 =   5.773503e-01, x1 =   1.577350e+00 ...
---   Numerical Integral f1(x) dx =        55.1111 ...
--- Error Analysis:
---   Absolute error =     8.8889e-01 ...
---   Relative error =     1.5873e-02 ...
---
--- ======================================== ...
--- Leave TestIntegration03.main()   ...
```

Question 5: 10 points.

Problem Statement: Write a Python program that uses Romberg Integration to show:

$$I = \int_0^1 \left[\frac{4}{1+x^2} \right] dx = \pi. \tag{29}$$

Start off by evaluating the function at 0, $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$, and 1. Compute and print the absolute and relative errors.

Python Source Code:

```
# ======================================================================
# TestIntegration04.py: Use rhomberg algorithm to integrate functions.
#
# Written By: Mark Austin                                    May 2025
# ======================================================================

import math;
import Integration;

# Define mathematical functions ...

def f1(x):
    numerator   = 4
    denominator = 1 + x**2
    return numerator/denominator

# main method ...

def main():
    print("--- Enter TestIntegration04.main()   ... ");
    print("--- ======================================== ... ");

    # Analytic solution ...

    analytic_i = math.pi
    print("--- Analytic solution = {:16.10f} ...".format(analytic_i) )

    print("--- ");
    print("--- Part 1: Integrate f1(x) with Romberg Integration ... ");
    print("--- ");

    # Initialize problem setup ...

    a = 0.0;
    b = 1.0
    nointervals = 4

    print("--- Inputs:")
```

```
    print("---   a = {:9.4f} ...".format(a) )
    print("---   b = {:9.4f} ...".format(b) )
    print("---   no intervals = {:d} ...".format(nointervals) )

    # Compute numerical solution to integral ..

    print("--- Execution:")
    xi = Integration.romberg( f1, a, b, nointervals )

    # Summary of computations ...

    print("--- Output:")
    print("---   Numerical Integral = {:16.10f} ...".format( xi ) )

    # Error Analysis ...

    print("--- Error Analysis:")
    print("---   Absolute error = {:14.4e} ...".format( abs( xi-analytic_i )) )
    print("---   Relative error = {:14.4e} ...".format( abs( xi-analytic_i )/analytic_i ) )

    print("--- ======================================== ... ");
    print("--- Leave TestIntegration04.main()   ... ");

# call the main method ...

main()
```

Abbreviated Output:

```
--- Enter TestIntegration04.main()   ...
--- ======================================== ...
---
--- Analytic solution =     3.1415926536 ...
---
--- Part 1: Integrate f1(x) with Romberg Integration ...
---
--- Inputs:
---   a =    0.0000 ...
---   b =    1.0000 ...
---   no intervals = 4 ...

--- Execution:
---   Initialize Romberg Integration Table ...

Matrix: Romberg Integration Table (empty)

  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00

---   Compute trapezoid rule for first column ...
```

```
---   Iterate over levels of refinement ...
---     Extrapolation for column  2 ...
---     Extrapolation for column  3 ...
---     Extrapolation for column  4 ...

Matrix: Romberg Integration Table (instantiated)
  3.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
  3.10000000e+00   3.13333333e+00   0.00000000e+00   0.00000000e+00
  3.13117647e+00   3.14156863e+00   3.14211765e+00   0.00000000e+00
  3.13898849e+00   3.14159250e+00   3.14159409e+00   3.14158578e+00

---
--- Output:
---   Numerical Integral =     3.1415857838 ...
---
--- Error Analysis:
---   Absolute error =     6.8698e-06 ...
---   Relative error =     2.1867e-06 ...
---
--- ======================================== ...
--- Leave TestIntegration04.main()   ...
```

Question 6: 10 points.

Problem Statement: Theoretical considerations indicate that:

$$\int_0^4 x^3 \left[16 - x^2\right] dx = \frac{1024}{3} \approx 341.33. \tag{30}$$

Part [6a] (5 pts). Use the method of Romberg integration to obtain an $O(h^6)$ accurate estimate of equation 30. Be sure to show all steps in your working.

Solution: To get an $O(h^6)$ accurate estimate we need three levels of refinement with Trapezoid (i.e., h = 4, h = 2 and h = 1).

```
--- Inputs:
---   a =    0.0000 ...
---   b =    4.0000 ...
---   no intervals = 3 ...
---
--- Execution:
---   Compute trapezoid rule for first column ...
---   Extrapolation for column  2 ...
---   Extrapolation for column  3 ...

Matrix: Romberg Integration Table (instantiated)
  0.00000000e+00   0.00000000e+00   0.00000000e+00
  1.92000000e+02   2.56000000e+02   0.00000000e+00
  3.00000000e+02   3.36000000e+02   3.41333333e+02

---  Integral I = 341.3333333333 ...  <--- Numerical integration is exact ...
```

Part [6b] (5 pts). Evaluate equation 30 using 3-pt Gauss Quadrature. Be sure to show all steps in your working.

Solution: Integrating f1(x) with 3-point quadrature ...

```
---   w0 =   5.555556e-01, u0 =  -7.745967e-01, x0 =   4.508067e-01 ...
---   w1 =   8.888889e-01, u1 =   0.000000e+00, x1 =   2.000000e+00 ...
---   w2 =   5.555556e-01, u2 =   7.745967e-01, x2 =   3.549193e+00 ...
---
---   f1(x0) =   1.447236e+00 ...
---   f1(x1) =   9.600000e+01 ...
---   f1(x2) =   1.521528e+02 ...
---
---   Integral I = 3.41333333e+02 ... <--- Numerical integration is exact ...
```