ENCE 201 Engineering Information Processing, Spring Semester, 2024

Solutions to Homework 4

Question 1: 20 points. Motor vehicle accidents in New York City are the leading cause of death for the city residents. Statistics indicate that close to one in four accidents results in someone being injured or losing their life. The problem is particularly acute for accidents involving high-speed, and/or when accidents involve pedestrians, bicyclists, or motor cycles.

The purpose of this question is to take a first step toward formally analyzing data on motor vehicle accidents and, specifically, understand where and when vehicle accidents occur? We will not investigate the particulars of who has been injured and associated details on the vehicles involved.

Accident Data: The folder `python-code.d/data/cities/nyc/` contains data files that can be used in the analysis of motor vehicle accidents in NYC. The main file, Motor-Vehicle-Collisions.csv, comprises 2.06 million motor vehicle accidents recorded across the five boroughs of NYC and for about a decade.

The data is organized into 29 columns:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2062758 entries, 0 to 2062757
Data columns (total 29 columns):
 #   Column                         Dtype
---  ------                         -----
 0   CRASH DATE                     object
 1   CRASH TIME                     object
 2   BOROUGH                        object
 3   ZIP CODE                       object
 4   LATITUDE                       float64
 5   LONGITUDE                      float64
 6   LOCATION                       object
 7   ON STREET NAME                 object
 8   CROSS STREET NAME              object
 9   OFF STREET NAME                object
 10  NUMBER OF PERSONS INJURED      float64
 11  NUMBER OF PERSONS KILLED       float64
 12  NUMBER OF PEDESTRIANS INJURED  int64
 13  NUMBER OF PEDESTRIANS KILLED   int64
 14  NUMBER OF CYCLIST INJURED      int64
 15  NUMBER OF CYCLIST KILLED       int64
 16  NUMBER OF MOTORIST INJURED     int64
 17  NUMBER OF MOTORIST KILLED      int64
 18  CONTRIBUTING FACTOR VEHICLE 1  object
 19  CONTRIBUTING FACTOR VEHICLE 2  object
 20  CONTRIBUTING FACTOR VEHICLE 3  object
```

```
 21  CONTRIBUTING FACTOR VEHICLE 4  object
 22  CONTRIBUTING FACTOR VEHICLE 5  object
 23  COLLISION_ID                   int64
 24  VEHICLE TYPE CODE 1            object
 25  VEHICLE TYPE CODE 2            object
 26  VEHICLE TYPE CODE 3            object
 27  VEHICLE TYPE CODE 4            object
 28  VEHICLE TYPE CODE 5            object
dtypes: float64(4), int64(7), object(18)
memory usage: 456.4+ MB
None
(2062758, 29)
```

Geospatial Data: The data files `dcm-nyc-major-street.csv` and `nyc-shoreline.csv` contain geospatial data on the main streets and shoreline in the NYC area.

Things to do: Write a Python program that will read the motor vehicle accidents and geospatial data files, and systematically filter and transform the data into spatial and temporal views of accident events in Lower Manhattan. Figure **??** shows a spatial view of streets in Lower Manhattan. Figure **??** shows a heatmap/temporal view of accidents, organized along the dimensions of time-of-the-day and day of the week. Thus, for this question, columns 0 – 5 of the accident data are most relevant.

For the spatial view:

1. Filter the accident data to only keep accidents occuring in Manhattan – this operation will reduce the number of accidents from 2 million to approximately 318,000. Then, remove from consideration accidents that do not have a (lat,long) coordinates – there are about 10,000 of them.

1. Add locations of remaining accidents to Figure **??**. Small blue dots might work, but there are over 300,000 of them ...

For the temporal view:

2. For each accident date, extract the day of the week and time of day. Map this data to rows and columns in the heatmap/temporal view, then display. Again, there are over 300,000 data points, so a good strategy might be to display the data as percentages?

Part 1: Python Source Code: (Spatial View of Data)

```
# ==========================================================================================
# TestDataProcessingMVAinNYC02.py: Two purposes:
#
# 1. Read, process and visualize data from data/cities/nyc/Motor-Vehicle-Collisions.csv
#    2,062,760 motor vehicle collisions in NYC.
```

```
#
# 2. Use geopandas to display motor vehicle collisions in NYC ...
#
# Written by: Mark Austin                                                   April, 2024
# ==========================================================================================

import numpy as np
import pandas as pd
import geopandas as gpd

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

from shapely import wkt

from pandas import DataFrame
from pandas import Series
from pandas import read_csv

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

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

    # Load and print dataset

    print("--- ");
    print("--- Part 01: Load motor vehicle collisions data file ... ");
    print("--- ");

    df = pd.read_csv('data/cities/nyc/Motor-Vehicle-Collisions.csv')

    # Dataframe description ...

    print("--- Motor vehicle collisions dataframe description ... ");

    print( df.describe() )

    # Dataframe info and shape ...

    print("--- Motor vehicle collisions dataframe info and shape ... ");

    print( df.info() )
    print( df.shape )

    # Filter dataframe to keep only airports located in Maryland ...

    print("--- Part 02: Extract motor vehicle accidents in Manhattan ... ");

    options = ['MANHATTAN']
    dfManhattan = df [ df['BOROUGH'].isin(options) ].copy()
```

```
print(dfManhattan)

# Dataframe info and shape ...

print( dfManhattan.info() )
print( dfManhattan.shape )

print("--- Part 03: Print individual rows of dfMD ... ");

print("--- Item  Zip Code Latitude/Longitude      ...");
print("--- ================================================================================ ...

# Traverse rows of dataframe ...

i = 1
for index, row in dfManhattan.iterrows():
    zipcode = str( row["ZIP CODE"] );
    lat     = row["LATITUDE"];
    long    = row["LONGITUDE"];
    print("--- {:4d}:  {:3s}, ({:f}, {:f}) ... ".format(i, zipcode, lat, long ));
    i = i + 1;

print("--- ================================================================================ ...
print("--- ");

print("--- Part 04: Convert dfMD dataframe to list, then print ... ");

# Convert dfMD dataframe to list ...

manhattanlist = dfManhattan.values.tolist();

# load dataset

print("--- Part 05: Load nyc data files ... ");

dfnycstreets   = pd.read_csv("data/cities/nyc/dcm-nyc-major-streets.csv")
dfnycshoreline = pd.read_csv("data/cities/nyc/nyc-shoreline.csv")

# Dataframe description ...

print("--- Major streets in NYC ... ");

print( dfnycstreets.describe() )

# Dataframe info and shape ...

print("--- NYC Dataframe info and shape ... ");

print( dfnycstreets.info() )
print( dfnycstreets.shape )

# Filter dataframe to keep only the "major streets in manhattan" entries ...

print("--- Part 06: Filter dataframe to keep \"Major streets in Manhattan\" ... ")
```

```
options = ['Manhattan']
dfmanhattan = dfnycstreets [ dfnycstreets['Borough'].isin(options) ].copy()

print('--- Manhattan dataframe :\n', dfmanhattan )

# Convert dataframe to a list ...

manhattanlist = dfmanhattan.values.tolist();

print("--- Part 07: Assemble Geodataframes for NYC main streets and shorline ... ");

# Geodataframe layer for nyc main streets ...

dfmanhattan['the_geom'] = dfmanhattan['the_geom'].apply(wkt.loads)
dfmanhattan.rename(columns={'the_geom': 'geometry'}, inplace=True)

gdf01 = gpd.GeoDataFrame(dfmanhattan, crs='epsg:4326')
gdf01.geometry

# Geodataframe layer for nyc shoreline ...

dfnycshoreline['the_geom'] = dfnycshoreline['the_geom'].apply(wkt.loads)
dfnycshoreline.rename(columns={'the_geom': 'geometry'}, inplace=True)

gdf02 = gpd.GeoDataFrame(dfnycshoreline, crs='epsg:4326')
gdf02.geometry

# Geodataframe layer for nyc motor vehicle accidents ...

gdf03 = gpd.GeoDataFrame( dfManhattan,
        geometry=gpd.points_from_xy(dfManhattan.LONGITUDE, dfManhattan.LATITUDE ))

print("--- ");
print("--- Part 08: Plot geodataframes for shoreline and street geometry ...");
print("--- ");

# Plot geodataframes for shoreline and street geometry ...

ax = gdf01.plot( color='red', edgecolor='black')
# ax.set_xlim( -74.05, -73.90)
ax.set_xlim( -74.02, -73.94)
ax.set_ylim(  40.70,  40.80)
ax.set_aspect('equal')
ax.set_title("318k Motor Vehicle Accidents in Manhattan")

gdf01.plot(ax=ax, color =   'red', linewidth=1 )
gdf02.plot(ax=ax, color = 'green', linewidth=2 )
gdf03.plot(ax=ax, color =  'blue', markersize = 1, label= 'Accidents')

plt.xlabel('longitude')
plt.ylabel('latitude')
plt.grid(True)
plt.show()
```

```
    print("--- ================================================== ... ");
    print("--- Leave TestDataProcessingNYC.main()                ... ");

# call the main method ...

if __name__ == "__main__":
    main()
```

Abbreviated Output: (Spatial View of Data)

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

--- Part 01: Load motor vehicle collisions data file ...
--- Motor vehicle collisions dataframe info and shape ...

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2062758 entries, 0 to 2062757
Data columns (total 29 columns):
 #   Column                        Dtype
---  ------                        -----
 0   CRASH DATE                    object
 1   CRASH TIME                    object
 2   BOROUGH                       object
 3   ZIP CODE                      object
 4   LATITUDE                      float64
 5   LONGITUDE                     float64

 ... lines of output removed ...

 28  VEHICLE TYPE CODE 5           object
dtypes: float64(4), int64(7), object(18)
memory usage: 456.4+ MB
None
(2062758, 29)

--- Part 02: Extract motor vehicle accidents in Manhattan ...
--- Part 03: Print individual rows of dfMD ...

--- Item  Zip Code Latitude/Longitude      ...
--- ============================================================================ ...
---    1:  10017.0, (40.751440, -73.973970) ...

 ... lines of output removed ...

--- 318799:  10024.0, (40.784355, -73.981170) ...
--- 318800:  10034.0, (40.872314, -73.912740) ...
--- ============================================================================ ...
---
--- Part 04: Convert dfMD dataframe to list, then print ...
---
--- Part 05: Load nyc data files ...
---
```

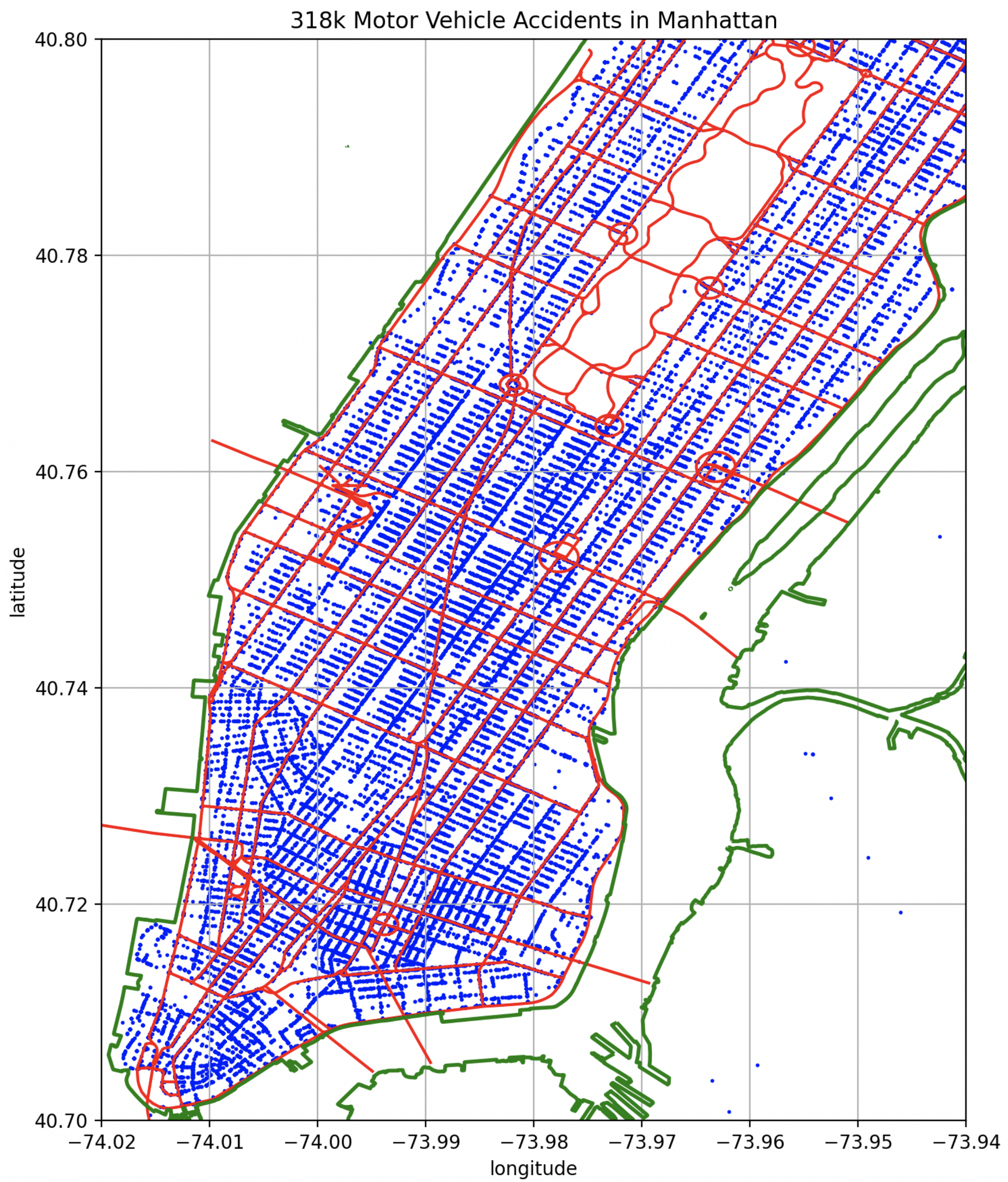

Figure 1: Spatial distribution of motor vehicle accidents in Manhattan.

```
--- Part 06: Filter dataframe to keep "Major streets in Manhattan" ...
---
--- Manhattan dataframe :
                                               the_geom  ... Route_Stat
0    MULTILINESTRING ((-73.9910241043406 40.7236494...  ...   Existing
1    MULTILINESTRING ((-74.01012109296306 40.738654...  ...   Existing
2    MULTILINESTRING ((-73.98761512117859 40.770415...  ...   Existing
3    MULTILINESTRING ((-74.01167376211146 40.709687...  ...   Existing

....

712  MULTILINESTRING ((-73.97564786198876 40.797105...  ...   Existing
715  MULTILINESTRING ((-74.01559321133946 40.707737...  ...   Existing
723  MULTILINESTRING ((-73.92450384092336 40.876242...  ...   Existing
730  MULTILINESTRING ((-73.9645029299501 40.7589745...  ...   Existing

[171 rows x 7 columns]
---
--- Part 07: Assemble Geodataframes for NYC main streets and shorline ...
--- Part 08: Plot geodataframes for shoreline and street geometry ...
--- ================================================ ...
--- Leave TestDataProcessingNYC.main()                ...
```

Part 2: Python Source Code: (Temporal View of Data)

```
# ==========================================================================================
# TestDataProcessingAccidentsNYC01.py: Two purposes:
#
# 1. Read, process and visualize data from data/cities/nyc/Motor-Vehicle-Collisions.csv
#    2,062,760 motor vehicle collisions in NYC.
#
# 2. Create plot of temporal dimensions of motor vehicle accidents ...
#
# Written by: Mark Austin                                                       April, 2024
# ==========================================================================================

import math

import numpy as np
import pandas as pd
import seaborn as sns

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

from pandas import DataFrame
from pandas import Series
from pandas import read_csv

from datetime import datetime

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

```
# =================================================================

def main():
    print("--- Enter TestDataProcessingAccidentsNYC01.main()         ... ");
    print("--- ===================================================== ... ");
    print("");

    # Load and print dataset

    print("--- ");
    print("--- Part 01: Load motor vehicle collisions data file ... ");
    print("--- ");

    df = pd.read_csv('data/cities/nyc/Motor-Vehicle-Collisions.csv')

    # Dataframe description ...

    print("--- Motor vehicle collisions dataframe description ... ");

    print( df.describe() )

    # Dataframe info and shape ...

    print("--- Motor vehicle collisions dataframe info and shape ... ");

    print( df.info() )
    print( df.shape )

    # Filter dataframe to keep only airports located in Maryland ...

    print("--- Part 02: Extract and print airports located in Maryland ... ");

    options = ['MANHATTAN']
    dfManhattan = df [ df['BOROUGH'].isin(options) ].copy()

    # Dataframe info and shape ...

    print( dfManhattan.info() )
    print( dfManhattan.shape )

    print("--- Part 03: Convert dfMD dataframe to list, then print ... ");

    # Convert dfMD dataframe to list ...

    manhattanlist = dfManhattan.values.tolist();

    # Assemble heatmap ...

    print("--- ");
    print("--- Part 04: Assemble heatmap ...");
    print("--- ");

    heatmapdata = np.zeros( [ 12, 7] )

    i = 1; welldefinedlatlong = 0;
```

```
for row in manhattanlist:
    crashdate = str( row[0] );
    crashtime = str( row[1] );
    zipcode = str( row[3] );
    lat     = row[4];
    long    = row[5];

    # Traverse list, print details of individual motor vehicle accidens ...

    if  math.isnan(lat) == False and math.isnan(long) == False:
        welldefinedlatlong += 1;

        # Example with the standard date and time format

        date_format = '%m/%d/%Y %H:%M'
        date_obj = datetime.strptime( crashdate + " " + crashtime, date_format)

        days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

        # Increment data in heatmap cells ...

        colno = date_obj.weekday();
        hour  = date_obj.hour;

        if hour <= 2:
           rowno = 0;
        elif 2 < hour <=  4:
           rowno = 1;
        elif 4 < hour <=  6:
           rowno = 2;
        elif 6 < hour <=  8:
           rowno = 3;
        elif 8 < hour <= 10:
           rowno = 4;
        elif 10 < hour <= 12:
           rowno = 5;
        elif 12 < hour <= 14:
           rowno = 6;
        elif 14 < hour <= 16:
           rowno = 7;
        elif 16 < hour <= 18:
           rowno = 8;
        elif 18 < hour <= 20:
           rowno = 9;
        elif 20 < hour <= 22:
           rowno = 10;
        else:
           rowno = 11;

        heatmapdata[rowno][colno] += 1;

    i = i + 1;

print("--- Days count ... ")
```

```
    print( heatmapdata )

    print("--- Motor vehicle accidents with lat/long = {:d} ... ".format( welldefinedlatlong ) );
    print("--- Total no motor vehicle accidents      = {:d} ... ".format( i ) );

    df = pd.DataFrame( heatmapdata )

    # Change the column names and row indexes ...

    df.columns = [   "Mon",  "Tue",  "Wed", "Thu", "Fri", "Sat", "Sun"]
    df.index   = ["12-2am","2-4am","4-6am","6-8am","8-10am","10-12pm",
                  "12-2pm","2-4pm","4-6pm","6-8pm","8-10pm","10-12am" ]

    # Default heatmap with active annotations ...

    p1 = sns.heatmap(df, annot=True, fmt=".1f")

    # This sets the yticks "upright" with 0, as opposed to sideways with 90.

    plt.yticks(rotation=0)
    plt.title("318k Vehicle Accidents in Manhattan (2013-2024)")
    plt.show()

    print("--- ================================================= ... ");
    print("--- Leave TestDataProcessingAccidentsNYC01.main()     ... ");

# call the main method ...

if __name__ == "__main__":
    main()
```

Abbreviated Output: (Temporal View of Data)

```
--- Enter TestDataProcessingAccidentsNYC01.main()         ...
--- ================================================= ...

--- Part 01: Load motor vehicle collisions data file ...
---
--- Motor vehicle collisions dataframe description ...
--- Motor vehicle collisions dataframe info and shape ...
---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2062758 entries, 0 to 2062757
Data columns (total 29 columns):
 #   Column                         Dtype
---  ------                         -----
 0   CRASH DATE                     object
 1   CRASH TIME                     object
 2   BOROUGH                        object
 3   ZIP CODE                       object
 4   LATITUDE                       float64
 5   LONGITUDE                      float64
```

```
 ... lines of output removed ...

 28  VEHICLE TYPE CODE 5            object
dtypes: float64(4), int64(7), object(18)
memory usage: 456.4+ MB
None
(2062758, 29)

--- Part 03: Convert dfMD dataframe to list, then print ...
--- Part 04: Assemble heatmap ...
--- Days count ...

[[2048. 1923. 2179. 2517. 3188. 4883. 4814.]
 [ 691.  530.  594.  716.  984. 2010. 2298.]
 [1357. 1374. 1228. 1325. 1352. 1402. 1186.]
 [3544. 3819. 3567. 3589. 3595. 1834. 1188.]
 [5378. 5902. 5440. 5600. 5658. 3089. 2076.]
 [5423. 6083. 5644. 5947. 6175. 4149. 3203.]
 [5559. 6191. 6428. 6335. 6417. 4819. 4318.]
 [5322. 5963. 6216. 6061. 6332. 4811. 4372.]
 [5049. 5704. 6066. 6015. 6523. 4798. 4285.]
 [3486. 4279. 4425. 4679. 4922. 4280. 3649.]
 [2452. 2909. 3206. 3633. 3865. 3831. 2774.]
 [ 907. 1186. 1257. 1405. 1783. 1852. 1063.]]

--- Motor vehicle accidents with lat/long = 308929 ...
--- Total no motor vehicle accidents      = 318801 ...

--- ================================================== ...
--- Leave TestDataProcessingAccidentsNYC01.main()     ...
```

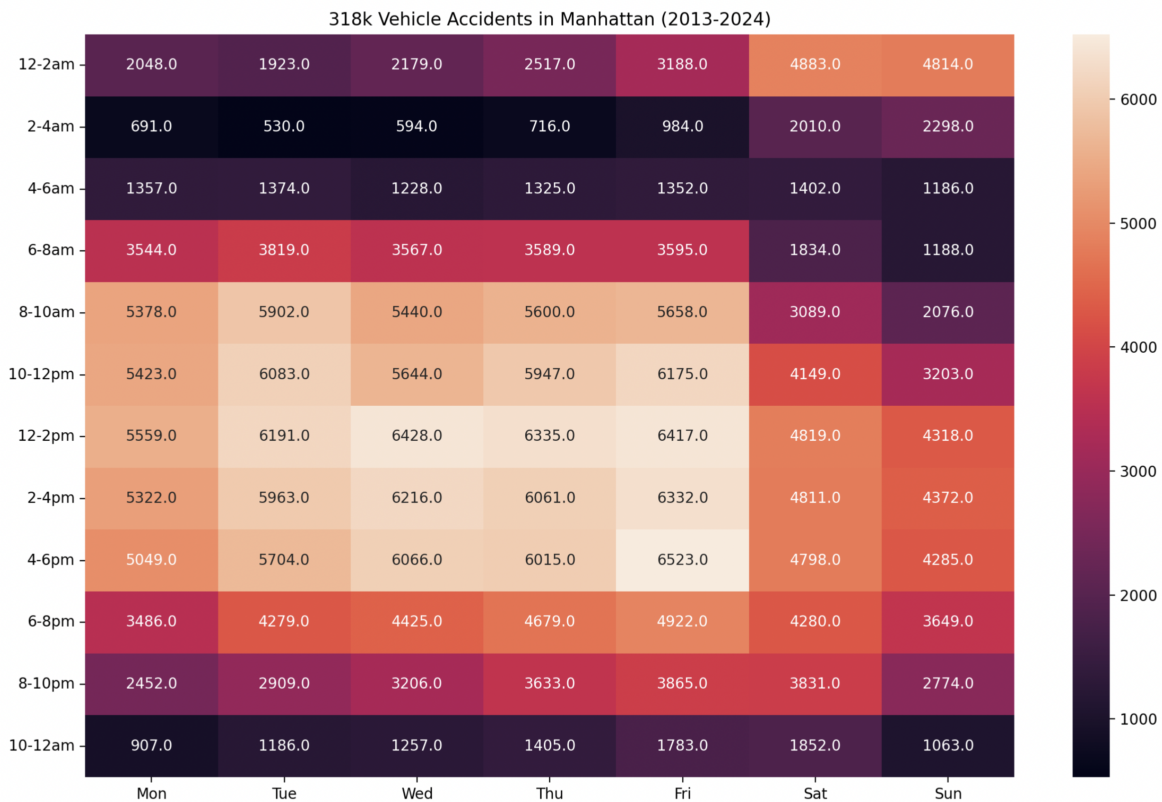

Figure 2: 318k motor vehicle accidents in Manhattan (2013-2024).

Question 2: 20 points. Figure 3 is a three-dimensional view of a 2 by 2 km site that is believed to overlay a thick layer of mineral deposits.

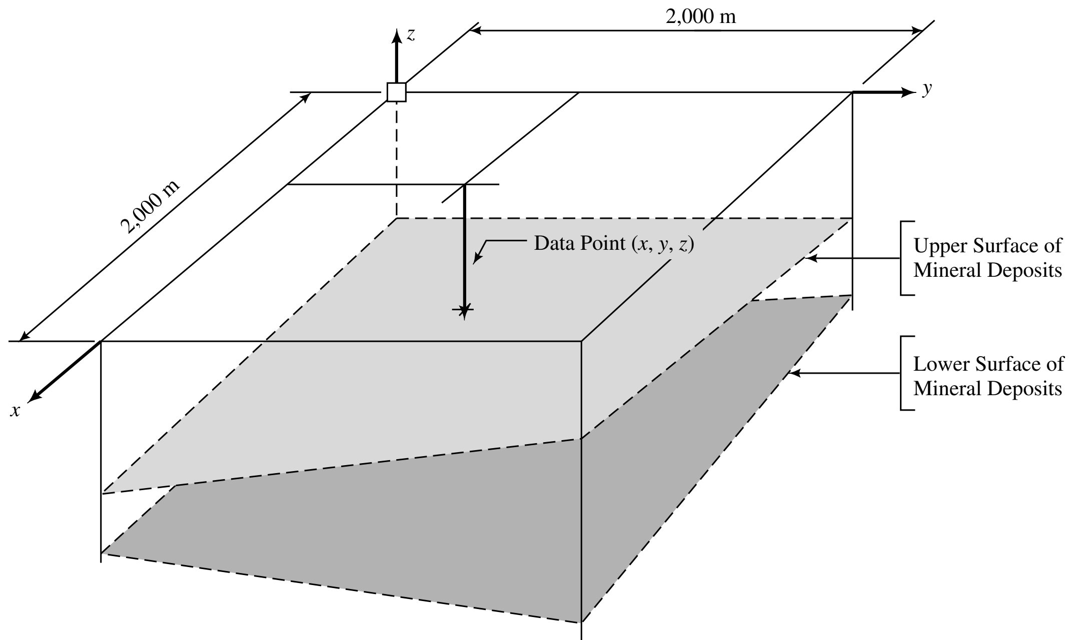

Figure 3: Three-dimensional view of mineral deposits.

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	10.0	10.0	-28.5	-42.5
2	750.0	10.0	-27.0	-41.8
3	1250.0	10.0	-26.0	-41.3
4	1990.0	10.0	-24.6	-40.5
5	10.0	750.0	-32.2	-43.4
6	750.0	750.0	-30.8	-42.6
7	1250.0	750.0	-29.8	-42.1
8	1990.0	750.0	-28.3	-41.4
9	10.0	1250.0	-34.7	-44.0
10	750.0	1250.0	-33.2	-43.2
11	1250.0	1250.0	-32.2	-42.7
12	1990.0	1250.0	-30.8	-42.0
13	10.0	1990.0	-38.4	-44.8
14	750.0	1990.0	-37.0	-44.1
15	1250.0	1990.0	-36.0	-43.6
16	1990.0	1990.0	-34.5	-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.

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) = -28.5 + x/500 - y/200` and the lower surface close is to `z(x,y) = -42.5 + x/1000 - y/850`.

Python Source Code:

```
# =====================================================================================
# TestLeastSquaresGeologicalBorings.py: Least squares analysis of geological borings.
#
# Written by: Mark Austin                                                 April, 2024
# =====================================================================================

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 TestLeastSquaresGeologicalBorings.main() ... ");
    print("--- =============================================== ... ");
    print("");

    # load dataset

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

    df = pd.read_csv('data/borehole-data02.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 TestLeastSquaresGeologicalBorings.main() ... ");

# call the main method ...

main()
```

Abbreviated Output:

```
--- Enter TestLeastSquaresGeologicalBorings.main() ...
--- =============================================== ...
---
--- Part 01: Load geological borehole data file ...
---
    Borehole    X (m)     Y(m)   Upper Surface (m)   Lower Surface (m)
0          1     10.0     10.0               -28.5               -42.5
1          2    750.0     10.0               -27.0               -41.8
2          3   1250.0     10.0               -26.0               -41.3

   ... lines of output removed ...             ... lines of output removed ...

13        14    750.0   1990.0               -37.0               -44.1
14        15   1250.0   1990.0               -36.0               -43.6
15        16   1990.0   1990.0               -34.5               -43.9

--- Convert dataframe to numpy array ...

Vector: X                                    Vector: Y
   1.0000000e+01                                1.0000000e+01
   7.5000000e+02                                1.0000000e+01


   1.2500000e+03                                1.9900000e+03
   1.9900000e+03                                1.9900000e+03
```

```
Vector: Upper surface                        Vector: Lower surface
  -2.8500000e+01                                -4.2500000e+01
  -2.7000000e+01                                -4.1800000e+01

   ... lines of output removed ...              ... lines of output removed ...

  -3.6000000e+01                                -4.3600000e+01
  -3.4500000e+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    2.4340800e+07    1.6000000e+07
   1.6000000e+04    1.6000000e+07    2.4340800e+07

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

Matrix: B (upper surface)
  -5.0400000e+02
  -4.8755600e+05
  -5.4585200e+05
--- Assemble B matrix (lower surface) ...

Matrix: B (lower surface)
  -6.8390000e+02
  -6.7656900e+05
  -6.9469700e+05

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

--- Upper surface equation:
--- z(x,y) = -28.454 +  0.002x + -0.005y ...
--- Upper surface mid-point:
--- z(1000,1000) = -31.500 ...

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

--- Lower surface equation:
--- z(x,y) = -42.328 +  0.001x + -0.001y ...
--- Lower surface mid-point:
--- z(1000,1000) = -42.744 ...

--- Part 06: Compute volume of minerals ...

--- Mineral Volume =        4.498e+07 (m^3) ...
--- ============================================= ...
--- Leave TestLeastSquaresGeologicalBorings.main() ...
```

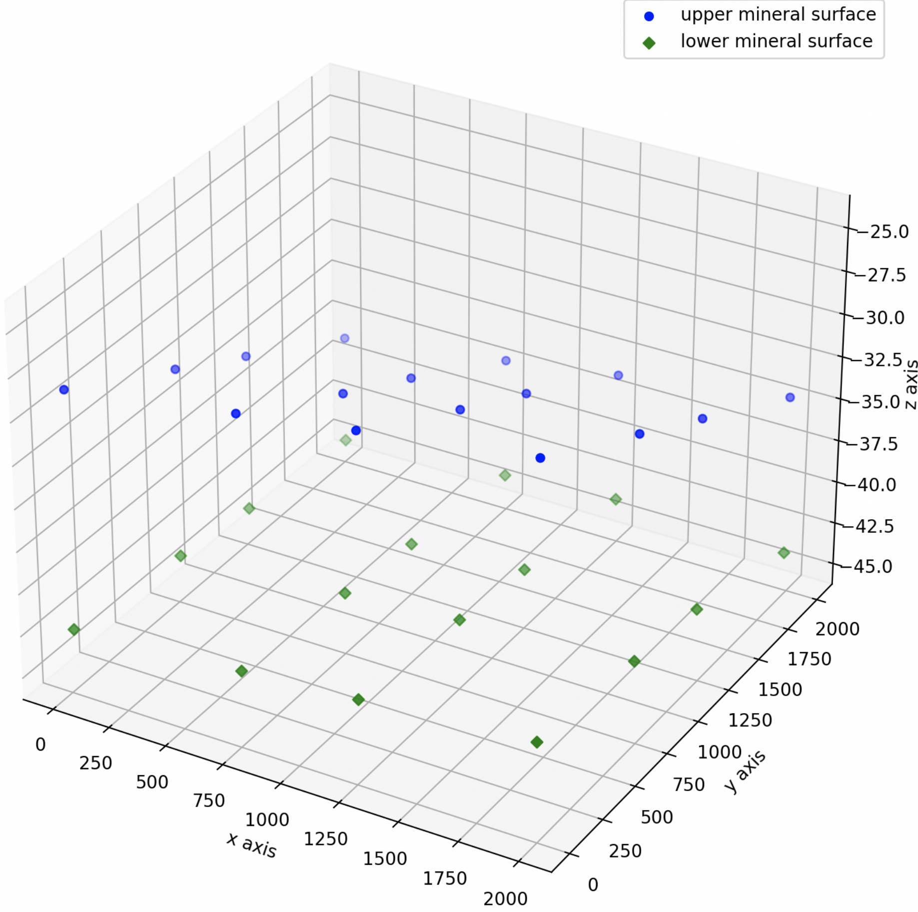

Figure 4: Scatter plot of geological borings data.

Question 3: 10 points. It is well known that first derivative of $f(x) = sin(x)$ is $cos(x)$. Given the double angle formulae,

$$sin(a + b) = sin(a)cos(b) + cos(a)sin(b) \tag{5}$$

and

$$sin(a - b) = sin(a)cos(b) - cos(a)sin(b) \tag{6}$$

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 Appropriate Equations:

Forward finite difference approximation:

$$\frac{df(x)}{dx} = \left[\frac{\sin(x+h) - \sin(x)}{h} \right] = cos(x) \left[\frac{\sin(h)}{h} \right] + sin(x) \left[\frac{\cos(h) - 1}{h} \right]. \tag{7}$$

Central finite difference approximation: Given the double angle formulae,

$$\frac{df(x)}{dx} = \left[\frac{\sin(x+h) - \sin(x-h)}{2h} \right] = cos(x) \left[\frac{\sin(h)}{h} \right]. \tag{8}$$

Preliminary Observations. As expected, equations 7 and 8 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 7 and 8 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 7 we expect that when sin(x) = 0, the forward finite difference approximation will be identical to the central difference approximation. And second, from equation 8 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                                          April 2024
# ==============================================================================

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 5 and 7 validate the preliminary predictions.

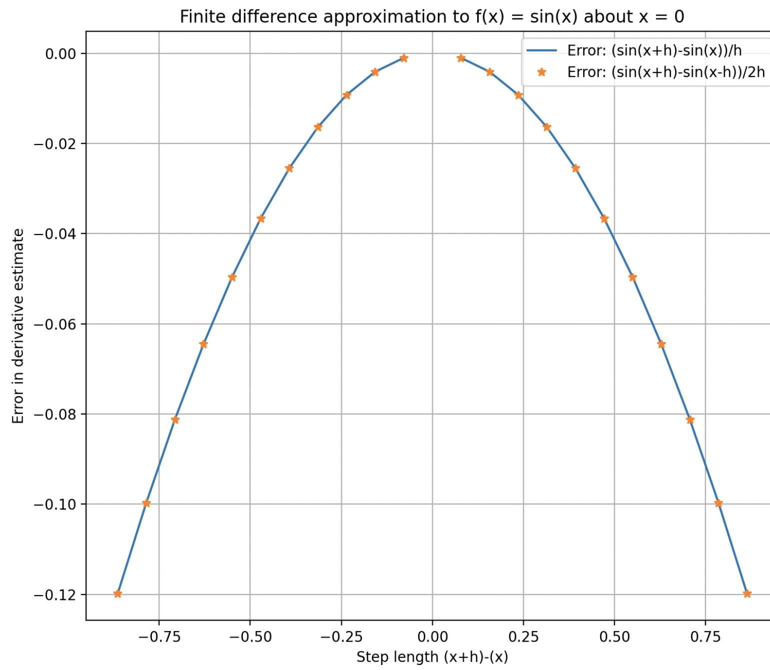

Figure 5: Finite difference approximations to f(x) = sin(x), x = 0 radians.

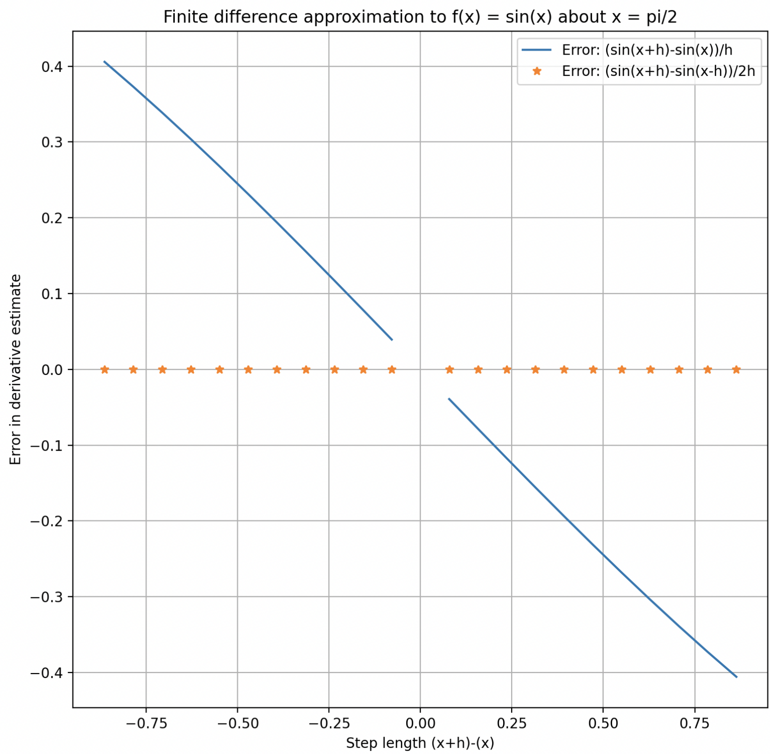

Figure 6: Finite difference approximations to f(x) = sin(x), x = pi/2 radians.

Question 4: 10 points. For the set of data,

```
      x  |    -1    2    4    5     6
   ------*--------------------------
    f(x) |     2    7   10    3     4
```

write a Python program that uses the methods of divided differences and Lagrange interpolation formula to approximate the functional value at x = 3.0. Create plots of the dataset coordinates and interpolated polynomial curves.

Python Source Code:

```
# ===============================================================================
# TestInterpolationLagrange01.py: Use method of divided differences and Lagrange
# interpolation on the dataset:
#
#     x  |    -1    2    4    5     6
#  ------*--------------------------
#   f(x) |     2    7   10    3     4
#
# Written By: Mark Austin                                          November 2023
# ===============================================================================

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

import Interpolation;

# ============================================================================
# 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 method ...

def main():
```

```
    print("--- Enter TestInterpolationLagrange01.main()  ... ");
    print("--- ========================================= ... ");

    print("--- Part 01: Create dataset ... ");

    # (x,y) data points ...

    x = np.array( [ -1, 2,  4, 5, 6 ] )
    y = np.array( [  2, 7, 10, 3, 4 ] )

    PrintVector("Data Points x",x);
    PrintVector("Data Points y",y);

    # test data point ...

    t = 3.0

    print("--- Part 02: Method of divided differences ... ");
    print("---   Compute divided difference table ... ");

    diffTable = Interpolation.divideddifference(x, y)[0, :]

    # evaluate on new data points

    print("---   Evaluate on new data points ... ");

    x_new = np.arange( -1.0, 7.0, .25 )
    y_new = Interpolation.newtonpolynomial(diffTable, x, x_new)

    print("---   Plot divided difference polynomial ... ");

    plt.figure(figsize = (12, 8))
    plt.plot(x_new, y_new, 'b', x, y, 'ro')
    plt.title('Fourth-order polynomial fit')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.grid()
    plt.show()

    print("--- Part 03: Method of lagrange interpolation ... ");

    p = Interpolation.lagrange(x, y, t)
    print("--- p(3.0) --> {:f} ...".format(p) )

# call the main method ...

main()
```

Abbreviated Output:

```
--- Part 01: Create dataset ...

Vector: Data Points x              Vector: Data Points y
```

```
  -1.0000000e+00                      2.0000000e+00
   2.0000000e+00                      7.0000000e+00
   4.0000000e+00                      1.0000000e+01
   5.0000000e+00                      3.0000000e+00
   6.0000000e+00                      4.0000000e+00

--- Part 02: Method of divided differences ...

Matrix: divided difference table

  2.0000   1.6667  -0.0333  -0.4667   0.3107
  7.0000   1.5000  -2.8333   1.7083   0.0000
 10.0000  -7.0000   4.0000   0.0000   0.0000
  3.0000   1.0000   0.0000   0.0000   0.0000
  4.0000   0.0000   0.0000   0.0000   0.0000

--- Part 03: Method of lagrange interpolation ...

--- p(3.0) --> 12.885714 ...
```

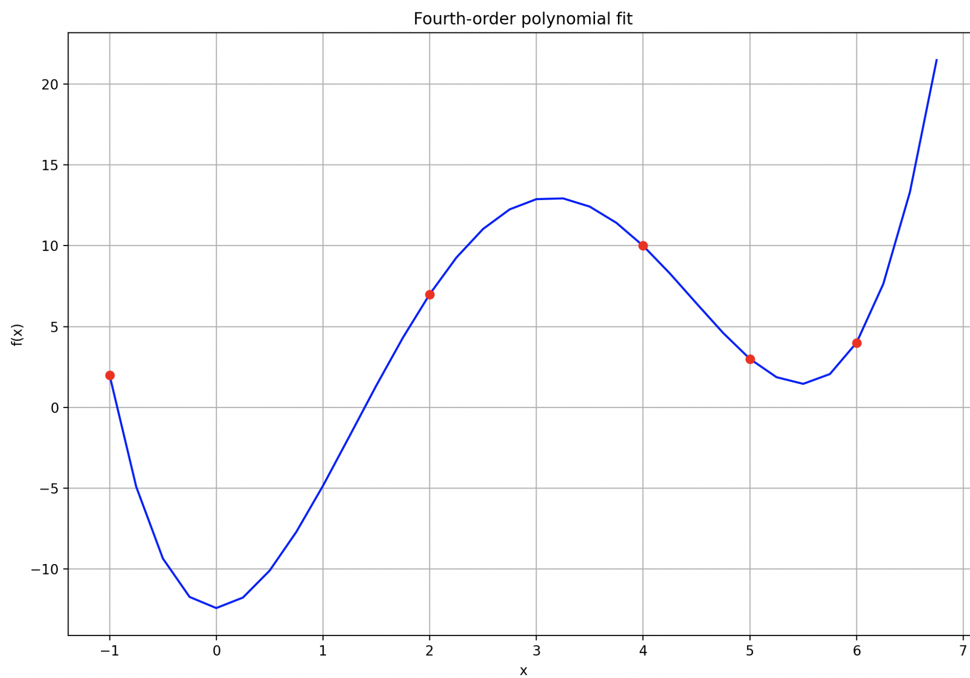

Figure 7: Fourth-order polynomial interpolation.

Question 5: 10 points. Consider the integral

$$I = \int_0^{10} 3x^2 + 4x^3 + 5x^4 dx = 111,000. \quad (9)$$

Write a Python program to compute numerical approximations to equation 9 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:

```
# ===============================================================================
# TestIntegrationTrapezoid01.py: Use trapezod algorithm to integrate functions.
#
# Written By: Mark Austin                                              July 2023
# ===============================================================================

import math;
import Integration;

# Define mathematical functions ...

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

# main method ...

def main():
    analytic_i = 111000;

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

    # Initialize problem setup ...

    a =  0.0; b =  10.0
    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("--- Part 2: Integrate f1(x) with Simpson ... ");

# Initialize problem setup ...

a = 0.0; b = 10.0
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("--- Part 3: Integrate f1(x) with Two-Point Gauss Quadrature ... ");

a = 0.0; b = 10.0

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 ) )

# call the main method ...

main()
```

Abbreviated Output:

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

--- Inputs:
---   a =    0.0000 ...
---   b =   10.0000 ...
---   no intervals = 2 ...
--- Execution:
--- Output:
---   Numerical Integral =     1.5425e+05 ...
--- Error Analysis:
---   Absolute error =     4.3250e+04 ...
---   Relative error =     3.8964e-01 ...

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

--- Inputs:
---   a =    0.0000 ...
---   b =   10.0000 ...
---   no intervals = 2 ...
--- Execution:
--- Output:
---   Numerical Integral =     1.1517e+05 ...
--- Error Analysis:
---   Absolute error =     4.1667e+03 ...
---   Relative error =     3.7538e-02 ...

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

---   w0 =   1.000000e+00, u0 =  -5.773503e-01, x0 =   2.113249e+00 ...
---   w1 =   1.000000e+00, u1 =   5.773503e-01, x1 =   7.886751e+00 ...
---   Numerical Integral f1(x) dx =    108222.2222 ...
--- Error Analysis:
---   Absolute error =     2.7778e+03 ...
---   Relative error =     2.5025e-02 ...
```

Question 6: 10 points. Write a Python program that uses Romberg Integration to show:

$$\int_0^1 \left[\frac{1+x^2}{1+x^4} \right] dx = \frac{\pi \cdot \sqrt{2}}{4} \tag{10}$$

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:

```
# ================================================================================
# TestIntegrationTrapezoid02.py: Demonstrate Romberg integration ....
#
# Written By: Mark Austin                                          November 2023
# ================================================================================

import math;
import Integration;

# Define mathematical function ...

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

# main method ...

def main():
    # Analytic solution ...

    analytic_i = math.pi * math.sqrt(2)/4.0
    print("--- Analytic solution = {:16.10f} ...".format(analytic_i) )

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

    # 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 ) )

# call the main method ...

main()
```

Abbreviated Output:

```
--- Analytic solution =     1.1107207345 ...

--- 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)
  1.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
  1.08823529e+00   1.11764706e+00   0.00000000e+00   0.00000000e+00
  1.10544499e+00   1.11118156e+00   1.11075052e+00   0.00000000e+00
  1.10941454e+00   1.11073773e+00   1.11070814e+00   1.11070747e+00

--- Output:
---   Numerical Integral =     1.1107074672 ...

--- Error Analysis:
---   Absolute error =     1.3267e-05 ...
---   Relative error =     1.1945e-05 ...
```