Indian Institute of Technology
(Indian School of Mines)
Dhanbad -826004

Department of Applied Geophysics

Winter 2023-24

April 15, 2024

Course code : GPC522

MAGNETIC METHODS LAB ASSIGNMENTS

Name: Riya Singh Rathore
Admission No.: 20JE0801
Integrated Master of Technology

Github: https://github.com/RiyaSinghRathore/Magnetic-Methods

<!DOCTYPE html>

Objective:

1. Develop a function to determine, whether a given number is even or odd.

2. Develop a function to search the Maximum and Minimum from an Array of Numbers.

3. Plot the given topography data with proper labeling.

Code:

```
## 1.
def function1(n):
    if n%2==0: return "EVEN"
    else: return "ODD"

# Example 1
function1(235)
```

```
'ODD'
```

```
def function2(arr):
    max = min = arr[0]
    for n in arr:
        if n > max : max = n
        elif n < min: min = n
    return max, min

# Example 2
Array = [5, 3, 8, 1, 9, 2, 7]
max, min = function2(Array)
print("Maximum :", max)
print("Minimum :", min)
```

```
Maximum : 9
Minimum : 1
```

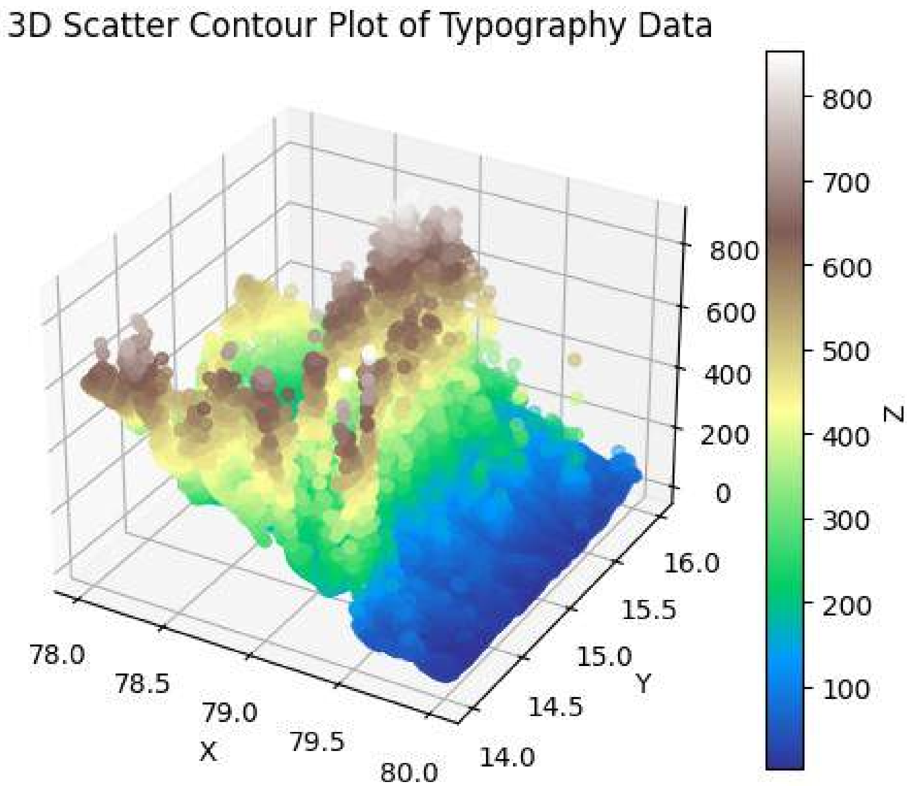
```
# Importing neccesary libraries

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation

import pandas as pd
```

```
/var/folders/f0/k3cxr5sj5gb40qhbcd5dzq6m0000gn/T/
ipykernel_9804/3227113166.py:7: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major
release of pandas (pandas 3.0),
```

```
(to allow more performant data types, such as the Arrow string type,
and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at
https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

Table:

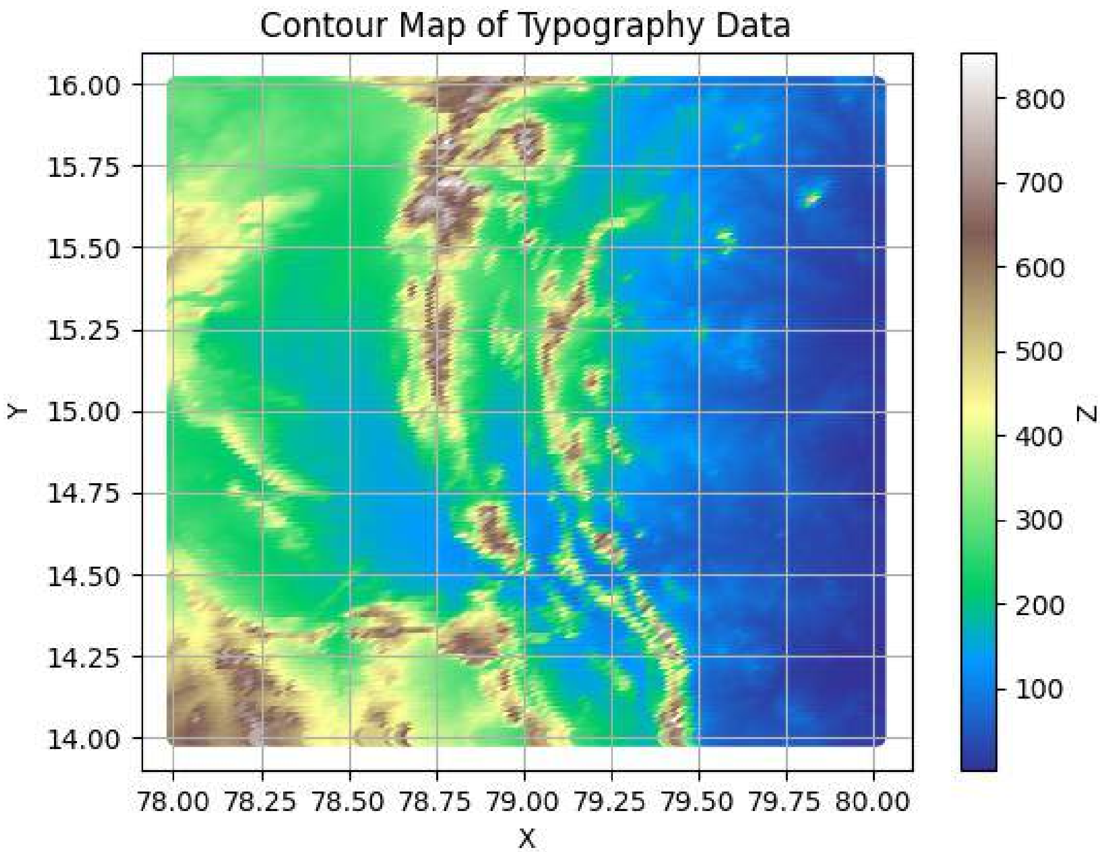
```
data = pd.read_table("Data/Topography_practical_1.txt", sep='\\s+')
data

             X        Y      Z
0      78.0083  15.9965  291.0
1      78.0250  15.9965  285.0
2      78.0417  15.9965  281.0
3      78.0583  15.9965  273.0
4      78.0750  15.9965  267.0
...        ...      ...    ...
15120  79.9417  14.0003   21.0
15121  79.9583  14.0003   19.0
15122  79.9750  14.0003   17.0
15123  79.9917  14.0003   17.0
15124  80.0083  14.0003   17.0

[15125 rows x 3 columns]

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

scatter = ax.scatter(data['X'], data['Y'], data['Z'], c=data['Z'],
cmap='terrain')
plt.colorbar(scatter, label='Z')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('3D Scatter Contour Plot of Typography Data')
plt.grid()
```




```
# Create and save the GIF
def update(angle): ax.view_init(elev=10, azim=angle)
ani = FuncAnimation(fig, update, frames=range(0, 360, 5))
ani.save('Plots/P1_3DContour.gif', writer='imagemagick', fps=30)

MovieWriter imagemagick unavailable; using Pillow instead.
```

Graph:

```
plt.scatter(data['X'], data['Y'], c=data['Z'], cmap='terrain')
plt.colorbar(label='Z')
plt.title('Contour Map of Typography Data')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid()
```



Result & Conclusion:

The analysis of the topography data obtained from the magnetic methods practical revealed significant insights into the surface features of the study area. By plotting the XYZ topography data in both 3D scatter contour and contour map formats, we gained a comprehensive understanding of the terrain's elevation variations.

<!DOCTYPE html>

```
# Importing Libraries
import numpy as np

from scipy.interpolate import griddata, Rbf, NearestNDInterpolator
from scipy.spatial import Delaunay

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.ticker as ticker

import pandas as pd
```

For the given data, use the various interpolation techniques for gridding and write your observations.

- Kriging Interpolation
- Nearest-neighbor Interpolation
- Radial Average Interpolation
- Triangulation with Linear Interpolation

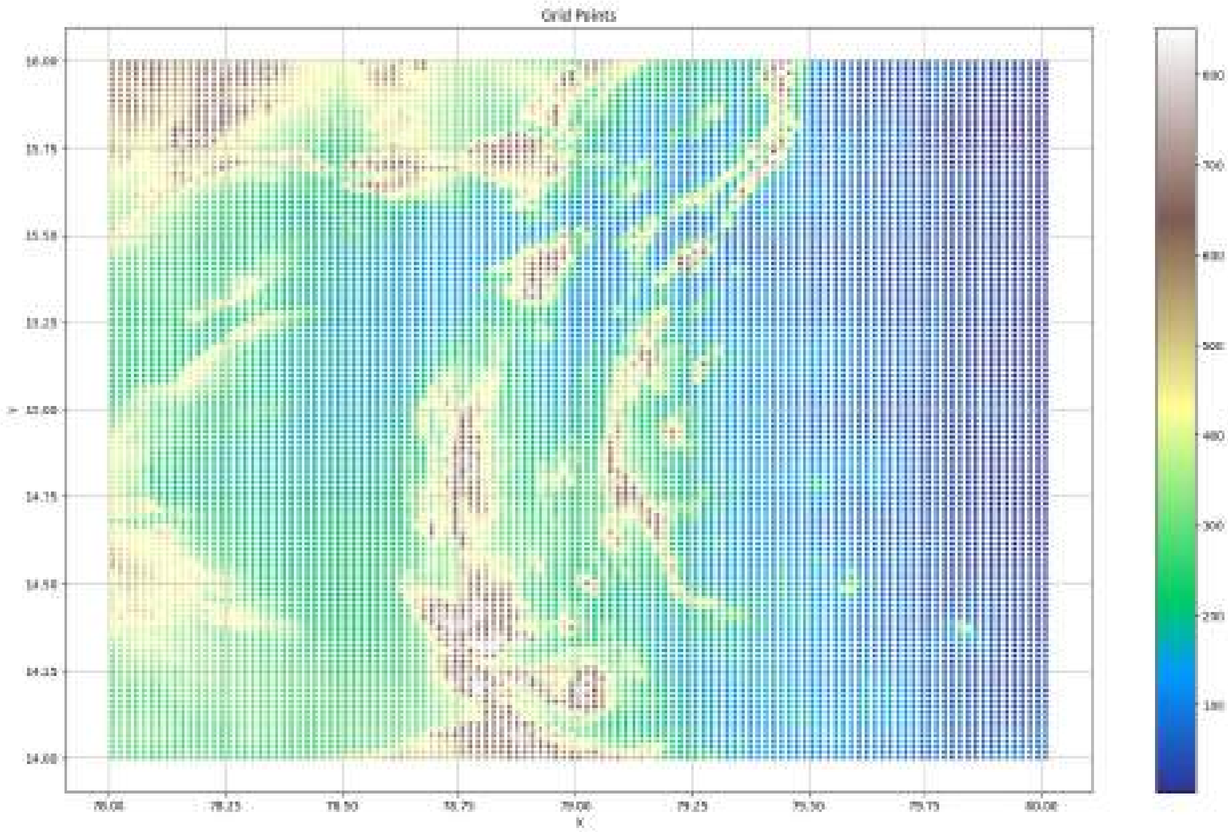
```
# Load data
data = np.loadtxt("Data/Topography_practical_1.txt", skiprows=1)
X = data[:, 0]
Y = data[:, 1]
Z = data[:, 2]

# DataFrame
df=pd.DataFrame({"X": X, "Y": Y, "Z":Z})
df
```

	X	Y	Z
0	78.0083	15.9965	291.0
1	78.0250	15.9965	285.0
2	78.0417	15.9965	281.0
3	78.0583	15.9965	273.0
4	78.0750	15.9965	267.0
...	...	...	...
15120	79.9417	14.0003	21.0
15121	79.9583	14.0003	19.0
15122	79.9750	14.0003	17.0
15123	79.9917	14.0003	17.0
15124	80.0083	14.0003	17.0

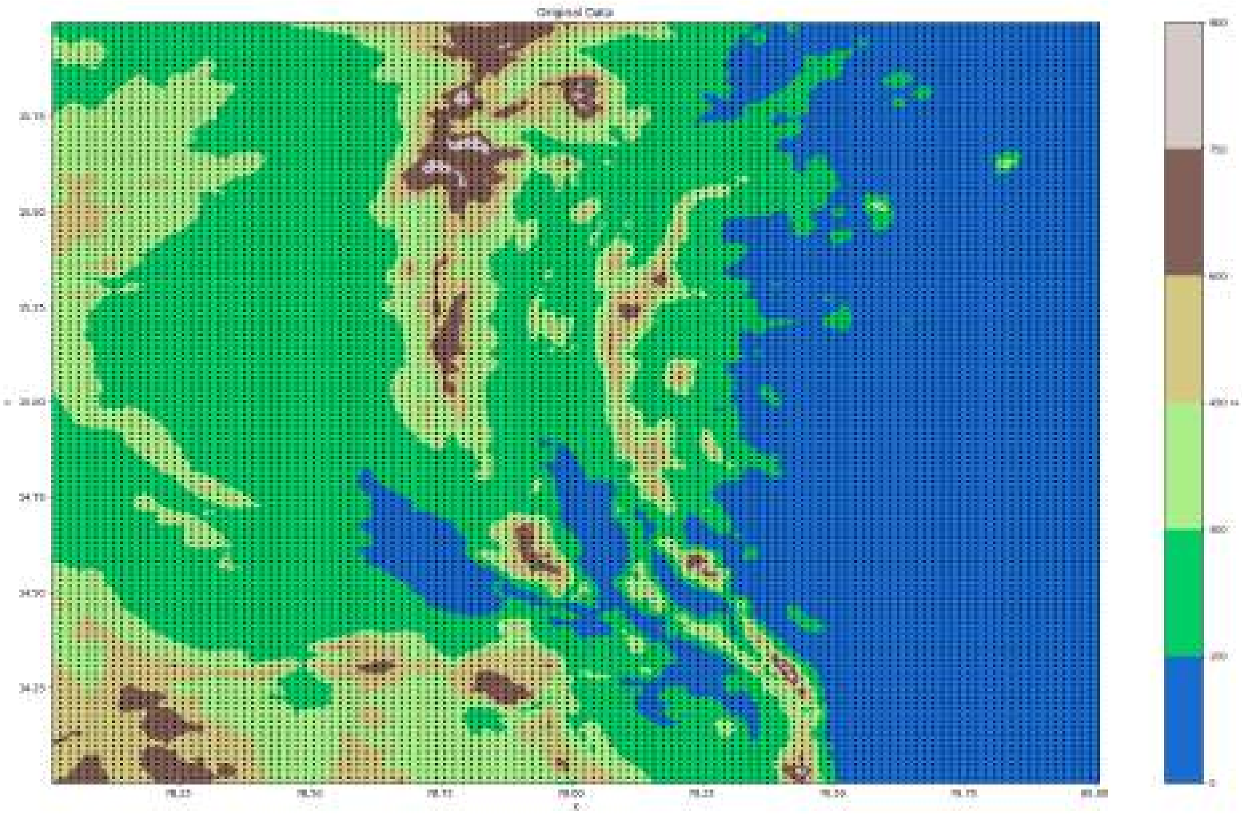
[15125 rows x 3 columns]

```
grid_X, grid_Y = np.meshgrid(np.unique(X), np.unique(Y))
fig = plt.figure(figsize=(20, 12))
plt.scatter(grid_X.flatten(), grid_Y.flatten(), c=Z.flatten(),
```

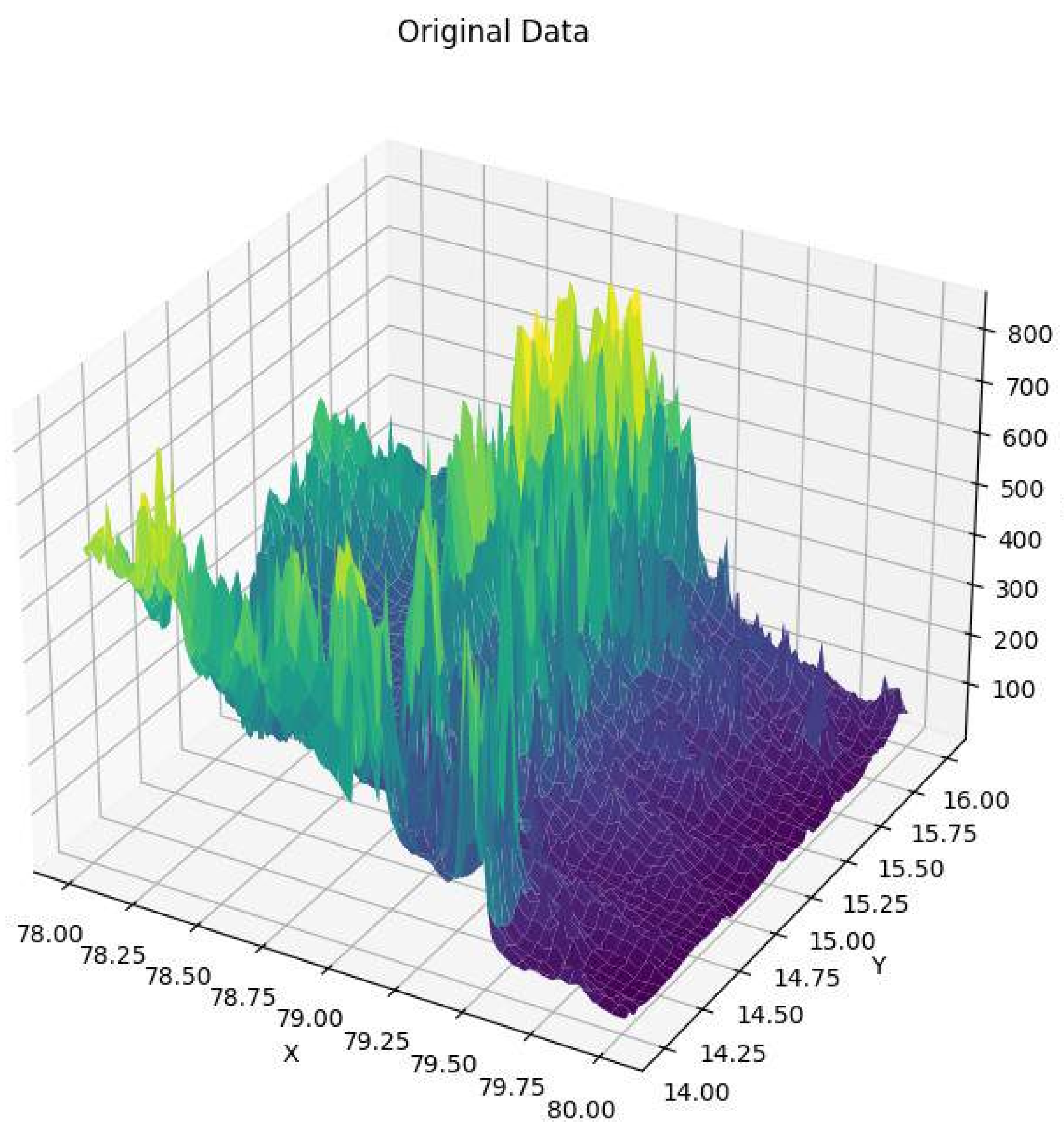
```
cmap="terrain", s=10)
plt.colorbar(label='Z')
plt.title('Grid Points')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid()
```




```
plt.figure(figsize=(20, 12))
plt.tricontourf(X, Y, Z, cmap='terrain')
plt.colorbar(label='Z')
plt.scatter(X, Y, color='k', s=4)
plt.title('Original Data')
plt.xlabel('X')
plt.ylabel('Y')
plt.tight_layout()
plt.show()
```




```
def plot_3d_contour(X, Y, grid_X, grid_Y, grid_Z, title):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(grid_X, grid_Y, grid_Z, cmap='viridis')
    ax.set_title(title)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    plt.show()

plot_3d_contour(X, Y, grid_X, grid_Y, griddata((X, Y), Z, (grid_X,
grid_Y), method='linear'), 'Original Data')
plt.tight_layout()
plt.show()
```

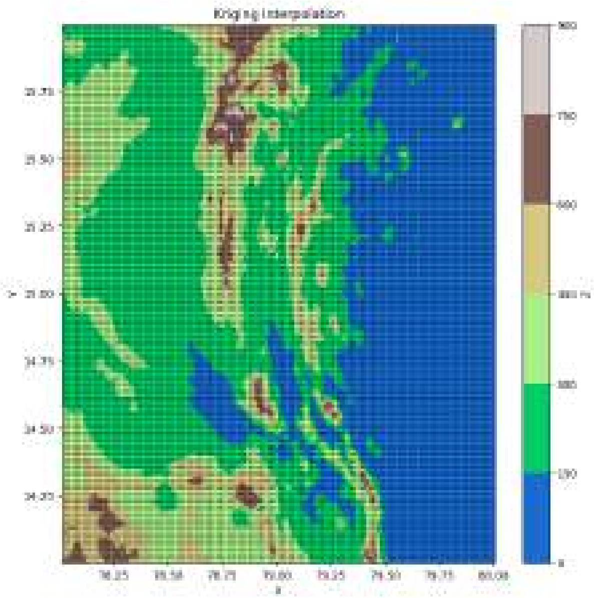


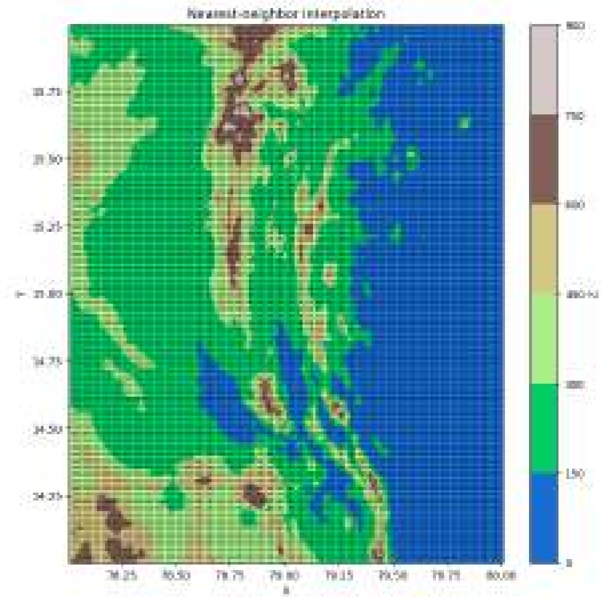

```
<Figure size 640x480 with 0 Axes>
```

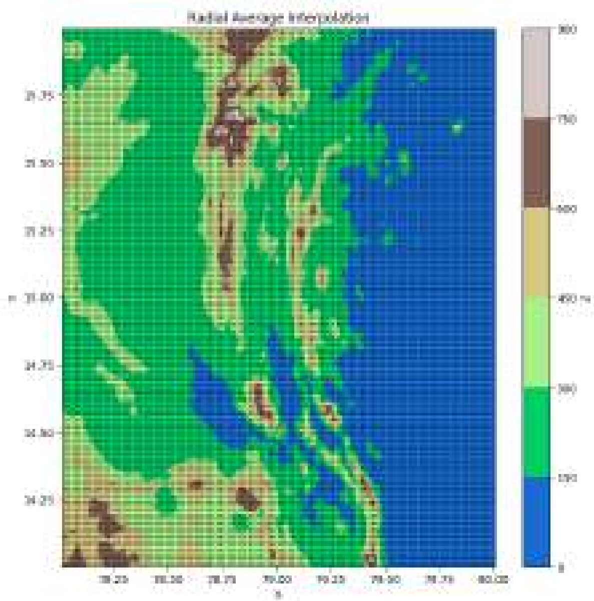
```python
fig = plt.figure(figsize=(20, 20))

# Subplot 1: Kriging Interpolation
plt.subplot(2, 2, 1)
rbf_kriging = Rbf(X, Y, Z, function='gaussian')
grid_Z_kriging = rbf_kriging(grid_X, grid_Y)
plt.contourf(grid_X, grid_Y, grid_Z_kriging, cmap='terrain')
plt.colorbar(label='Z')
plt.scatter(X, Y, color='k', s=0.5)
plt.title('Kriging Interpolation')
```

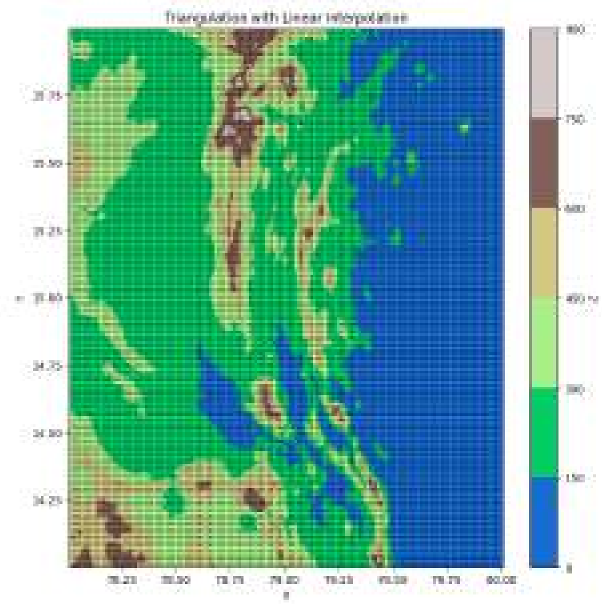
```
plt.xlabel('X')
plt.ylabel('Y')

# Subplot 2: Nearest-neighbor Interpolation
plt.subplot(2, 2, 2)
grid_Z_nearest = griddata((X, Y), Z, (grid_X, grid_Y),
method='nearest')
plt.contourf(grid_X, grid_Y, grid_Z_nearest, cmap='terrain')
plt.colorbar(label='Z')
plt.scatter(X, Y, color='k', s=0.5)
plt.title('Nearest-neighbor Interpolation')
plt.xlabel('X')
plt.ylabel('Y')

# Subplot 3: Radial Average Interpolation
plt.subplot(2, 2, 3)
triangulation = Delaunay(np.column_stack((X, Y)))
interp_rbf = Rbf(X, Y, Z, function='linear')
grid_Z_radial = interp_rbf(grid_X, grid_Y)
plt.contourf(grid_X, grid_Y, grid_Z_radial, cmap='terrain')
plt.colorbar(label='Z')
plt.scatter(X, Y, color='k', s=0.5)
plt.title('Radial Average Interpolation')
plt.xlabel('X')
plt.ylabel('Y')

# Subplot 4: Triangulation with Linear Interpolation
plt.subplot(2, 2, 4)
interp_triang = NearestNDInterpolator(triangulation, Z)
grid_Z_triang = interp_triang(np.column_stack((grid_X.flatten(),
grid_Y.flatten())))
grid_Z_triang = grid_Z_triang.reshape(grid_X.shape)
plt.contourf(grid_X, grid_Y, grid_Z_triang, cmap='terrain')
plt.colorbar(label='Z')
plt.scatter(X, Y, color='k', s=0.5)
plt.title('Triangulation with Linear Interpolation')
plt.xlabel('X')
plt.ylabel('Y')
```

```
Text(0, 0.5, 'Y')
```



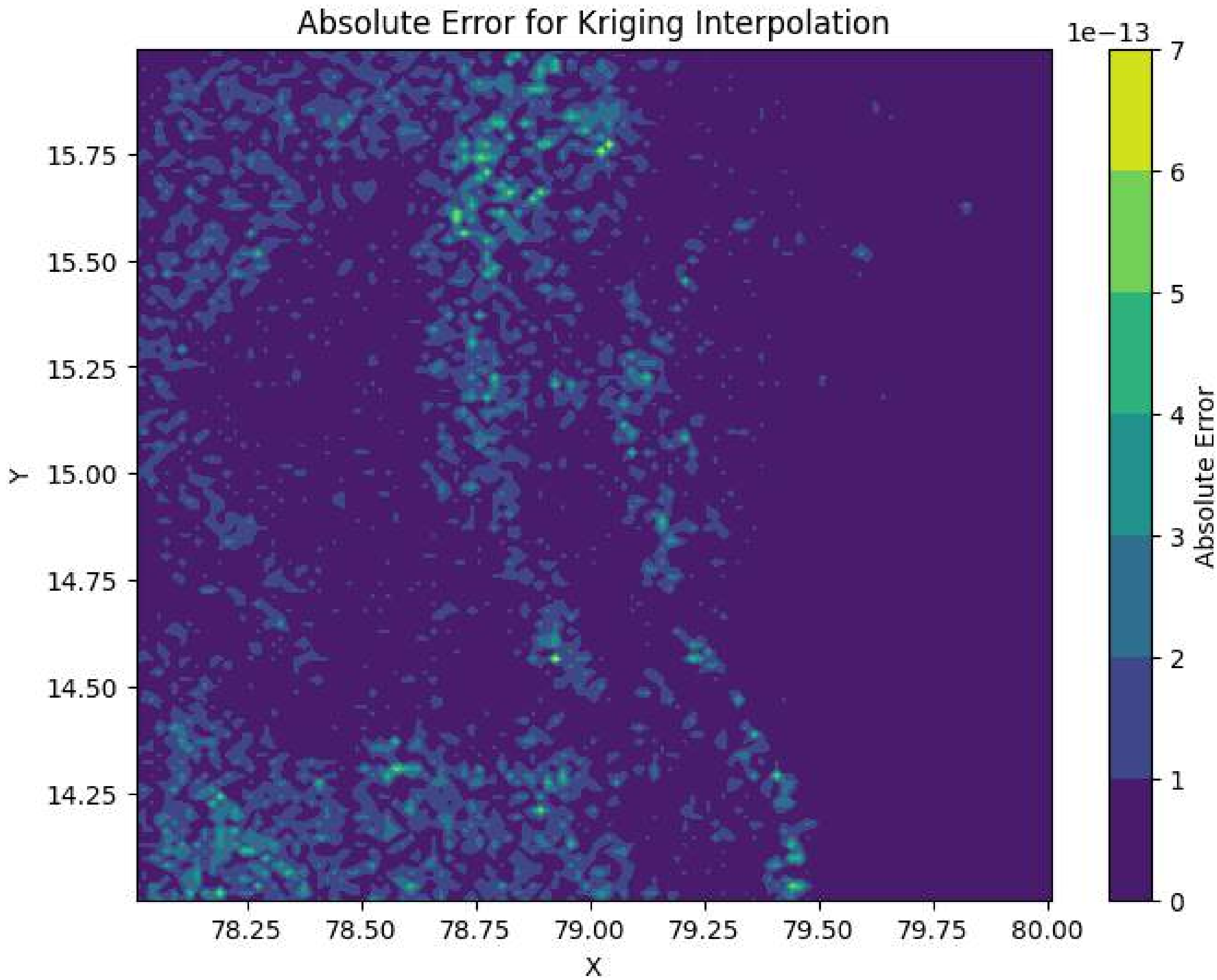




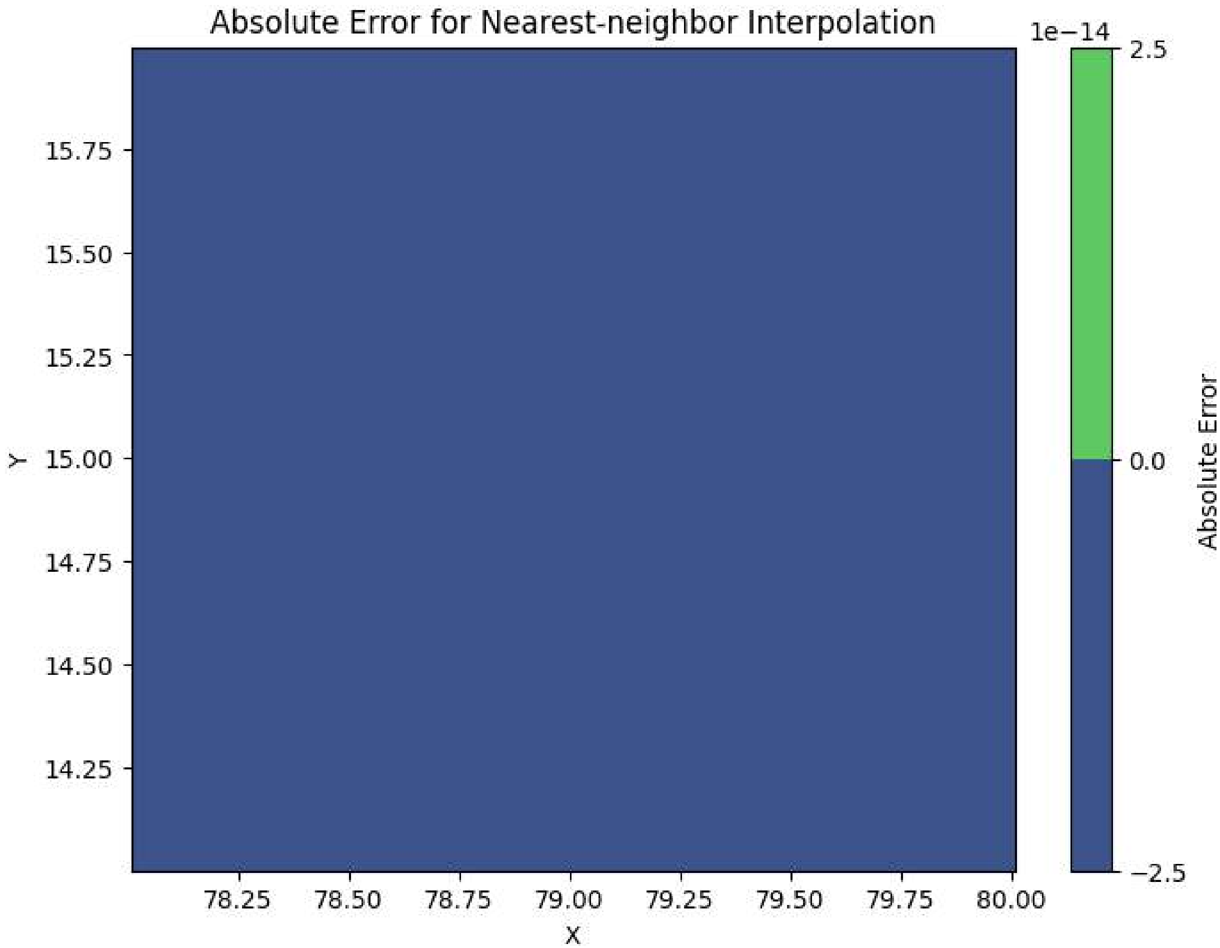



```
def plot_error(X, Y, grid_X, grid_Y, grid_Z, method):
    original_Z = griddata((X, Y), Z, (grid_X, grid_Y),
method='linear')
    error = np.abs(original_Z - grid_Z)
    plt.figure(figsize=(8, 6))
    plt.contourf(grid_X, grid_Y, error, cmap='viridis')
    plt.colorbar(label='Absolute Error')
    plt.title(f'Absolute Error for {method}')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()
```

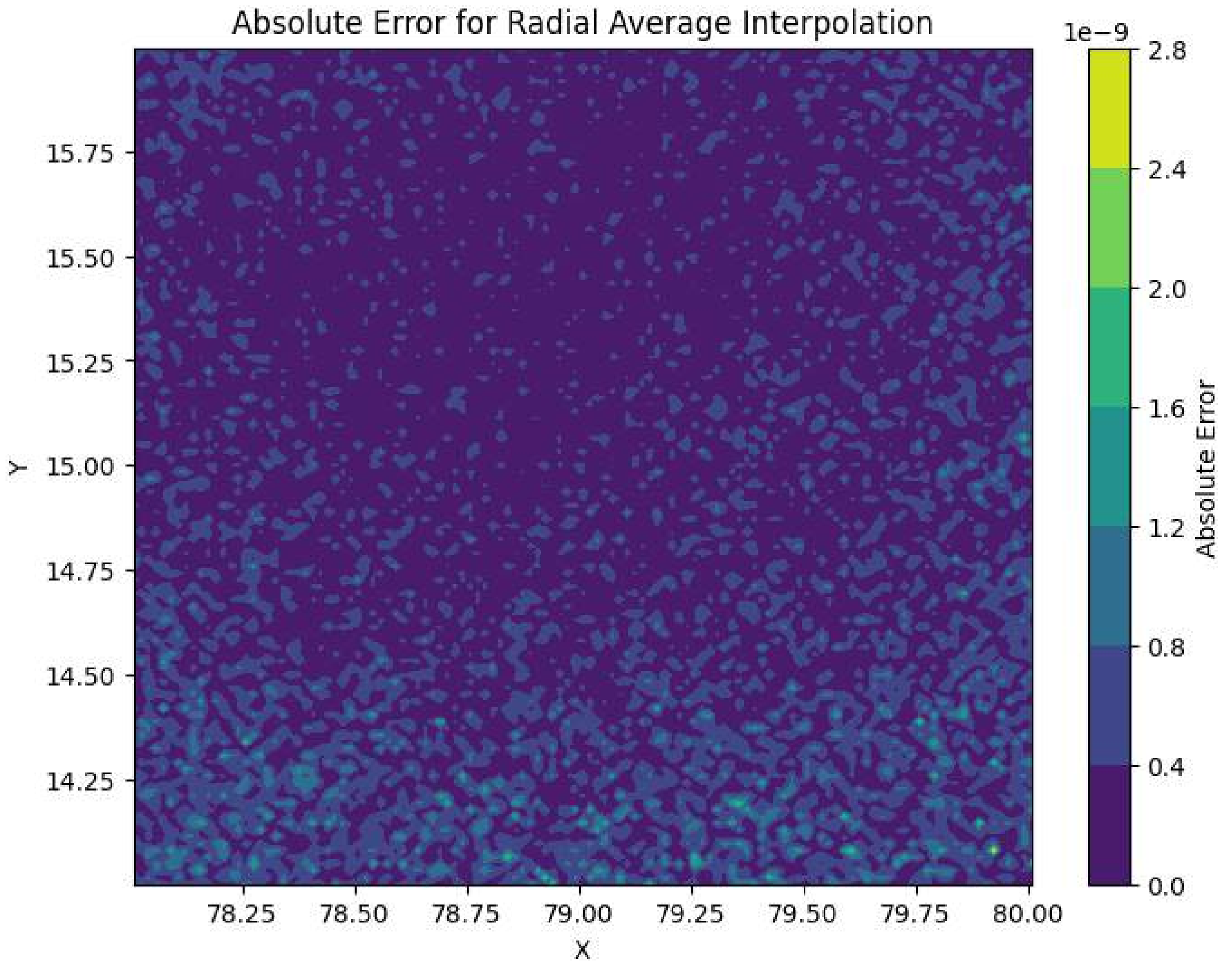
```python
rbf_kriging = Rbf(X, Y, Z, function='gaussian')
grid_Z_kriging = rbf_kriging(grid_X, grid_Y)
plot_error(X, Y, grid_X, grid_Y, grid_Z_kriging, 'Kriging
Interpolation')
```



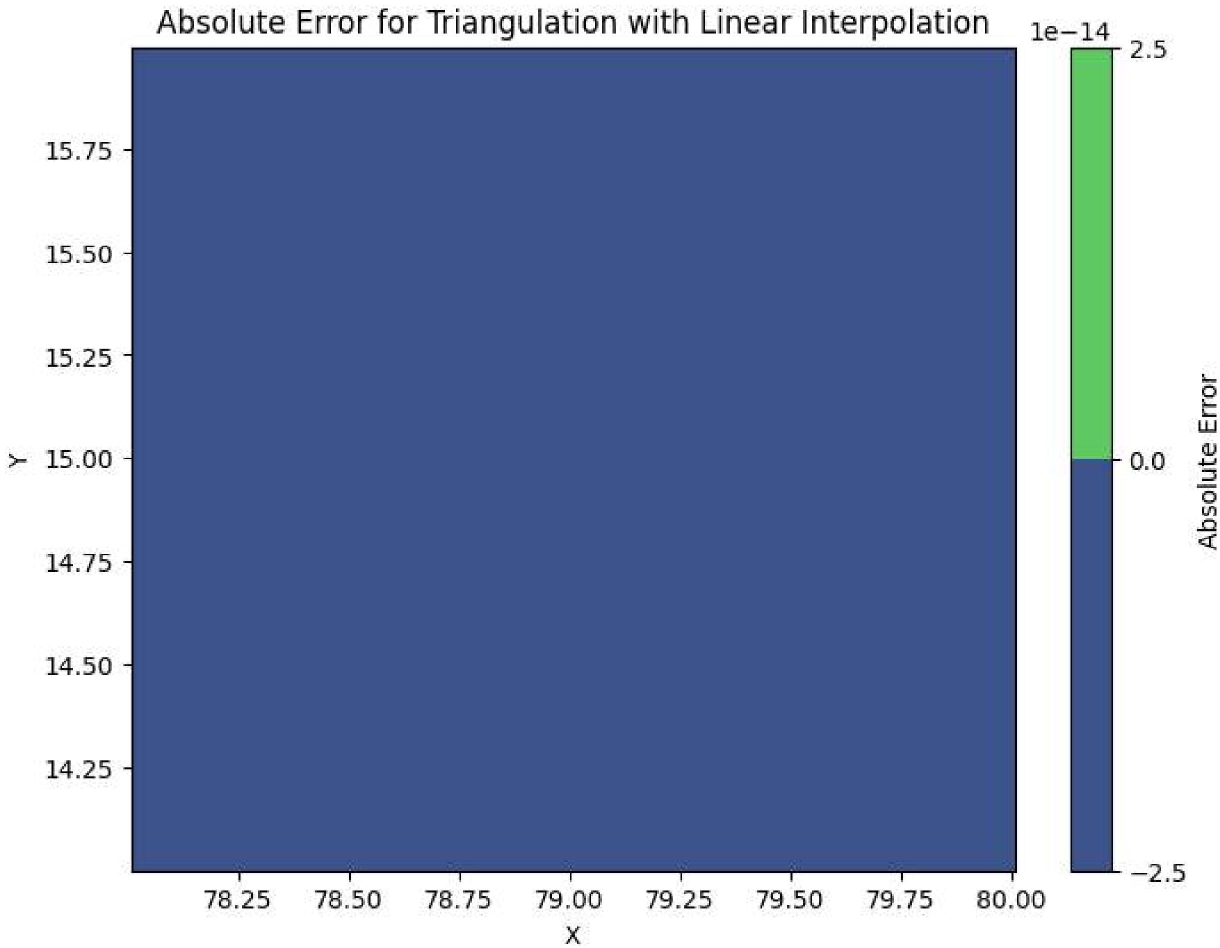

```python
grid_Z_nearest = griddata((X, Y), Z, (grid_X, grid_Y),
method='nearest')
plot_error(X, Y, grid_X, grid_Y, grid_Z_nearest, 'Nearest-neighbor
Interpolation')
```




```
interp_rbf = Rbf(X, Y, Z, function='linear')
grid_Z_radial = interp_rbf(grid_X, grid_Y)
plot_error(X, Y, grid_X, grid_Y, grid_Z_radial, 'Radial Average
Interpolation')
```




```
interp_triang = NearestNDInterpolator(triangulation, Z)
grid_Z_triang = interp_triang(np.column_stack((grid_X.flatten(),
grid_Y.flatten())))
grid_Z_triang = grid_Z_triang.reshape(grid_X.shape)
plot_error(X, Y, grid_X, grid_Y, grid_Z_triang, 'Triangulation with
Linear Interpolation')
```


Absolute Error for Triangulation with Linear Interpolation
Y
15.75
15.50
15.25
15.00
14.75
14.50
14.25
78.25
78.50
78.75
79.00
79.25
79.50
79.75
80.00
X
1e−14
2.5
0.0
−2.5
Absolute Error

<!DOCTYPE html>

Objective:

(a). Calculate the root mean square intensity of the dipole and quadrople components of the geomagnetic field at the Earth’s surface for IGRF Models. (b). Plot the root mean square intensity of both dipole and quadrupole components of the geomagnetic field as a function of time. (c). Estimate average rate change of both dipole and quadrupole components of the geomagnetic field.

Formula Used:

$$F=\sqrt{(n+1)\cdot\sum_{n=1}\sum_{m=0}\left[\left(g_n^m\right)^2+\left(h_n^m\right)^2\right)}$$

where, n = Order m = Degree g,h = Gauss Coefficients

Theory:

The objective of this experiment is to compute the root mean square (RMS) intensity of the dipole and quadrupole components of the geomagnetic field at the Earth's surface using the International Geomagnetic Reference Field (IGRF) models. The RMS intensity provides a measure of the overall strength of these components, which are crucial for understanding Earth's magnetic field variations.

The formula utilized for calculating the RMS intensity involves summing the squares of the Gauss coefficients for each degree and order up to a certain limit, and then taking the square root of the sum. The Gauss coefficients, denoted as (g_n^m) and (h_n^m), represent the coefficients associated with the spherical harmonic expansion of the geomagnetic field. The terms "order" ((n)) and "degree" ((m)) refer to the parameters of the expansion, capturing the complexity and spatial variability of the geomagnetic field.

The expression ((n+1)) in the formula accounts for the number of coefficients contributing to each order, ensuring that higher orders contribute proportionally to the overall intensity. The summation over both (n) and (m) encompasses all relevant coefficients needed to compute the RMS intensity accurately.

Furthermore, to fulfill the objective (b), the RMS intensity of both dipole and quadrupole components is plotted as a function of time. This visualization facilitates the examination of temporal variations in the geomagnetic field's dipole and quadrupole strengths, providing insights into the Earth's dynamic magnetic behavior over time.

Lastly, to address objective (c), the average rate of change of both dipole and quadrupole components of the geomagnetic field is estimated. This analysis offers valuable information regarding the long-term trends and fluctuations in Earth's magnetic field, contributing to our understanding of geophysical processes influencing magnetic field dynamics.

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

```
import pandas as pd
!pip install python-docx
from docx import Document
```

```
Collecting python-docx
  Using cached python_docx-1.1.0-py3-none-any.whl.metadata (2.0 kB)
Requirement already satisfied: lxml>=3.1.0 in
/Users/riyarathore/miniconda3/lib/python3.11/site-packages (from
python-docx) (5.1.0)
Requirement already satisfied: typing-extensions in
/Users/riyarathore/miniconda3/lib/python3.11/site-packages (from
python-docx) (4.5.0)
Using cached python_docx-1.1.0-py3-none-any.whl (239 kB)
Installing collected packages: python-docx
Successfully installed python-docx-1.1.0
```

Data:

The Gauss coefficients for the dipole and quadrupole components of the geomagnetic field from the various IGRF models are provided below.

```
doc = Document('Assignments/Practical 3_WS_23-24.docx')

# Convert docx file to pandas dataframe
tables = []
for table in doc.tables:
    data = []
    for row in table.rows:
        row_data = []
        for cell in row.cells:
            row_data.append(cell.text)
        data.append(row_data)
    df = pd.DataFrame(data[1:], columns=data[0])
    tables.append(df)

df
```

```
   gnm IGRF-1985 IGRF-1990 IGRF-1995 IGRF-2000 IGRF-2005 IGRF-2010
IGRF-2015  \
0  g10    -29873    -29775    -29692  -29619.4  -29554.6  -29496.6  -
29441.5
1  g11     -1905     -1848     -1784   -1728.2  -1669.05  -1586.42  -
1501.77
2  h11      5500      5406      5306    5186.1   5077.99   4944.26
4795.99
3  g20     -2072     -2131     -2200   -2267.7  -2337.24  -2396.06  -
2445.88
4  g21      3044      3059      3070    3068.4   3047.69   3026.34
3012.2
5  h21     -2197     -2279     -2366   -2481.6   -2594.5  -2708.54  -
```

```
2845.41
6  g22       1687      1686      1681    1670.9   1657.76   1668.17   
1676.35
7  h22       -306      -373      -413      -458   -515.43   -575.73   -
642.17

  IGRF-2020
0  -29404.8
1   -1450.9
2    4652.5
3   -2499.6
4      2982
5   -2991.6
6      1677
7    -734.6
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   gnm        8 non-null      object
 1   IGRF-1985  8 non-null      object
 2   IGRF-1990  8 non-null      object
 3   IGRF-1995  8 non-null      object
 4   IGRF-2000  8 non-null      object
 5   IGRF-2005  8 non-null      object
 6   IGRF-2010  8 non-null      object
 7   IGRF-2015  8 non-null      object
 8   IGRF-2020  8 non-null      object
dtypes: object(9)
memory usage: 708.0+ bytes
```

```
# Convert columns to numeric (float) type
df.iloc[:, 1:] = df.iloc[:, 1:].apply(pd.to_numeric)
for col in df.columns[1:]:  df[col] = df[col].astype(int)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   gnm        8 non-null      object
 1   IGRF-1985  8 non-null      int64 
 2   IGRF-1990  8 non-null      int64 
 3   IGRF-1995  8 non-null      int64 
```

```
 4   IGRF-2000  8 non-null      int64
 5   IGRF-2005  8 non-null      int64
 6   IGRF-2010  8 non-null      int64
 7   IGRF-2015  8 non-null      int64
 8   IGRF-2020  8 non-null      int64
dtypes: int64(8), object(1)
memory usage: 708.0+ bytes
```

Code:

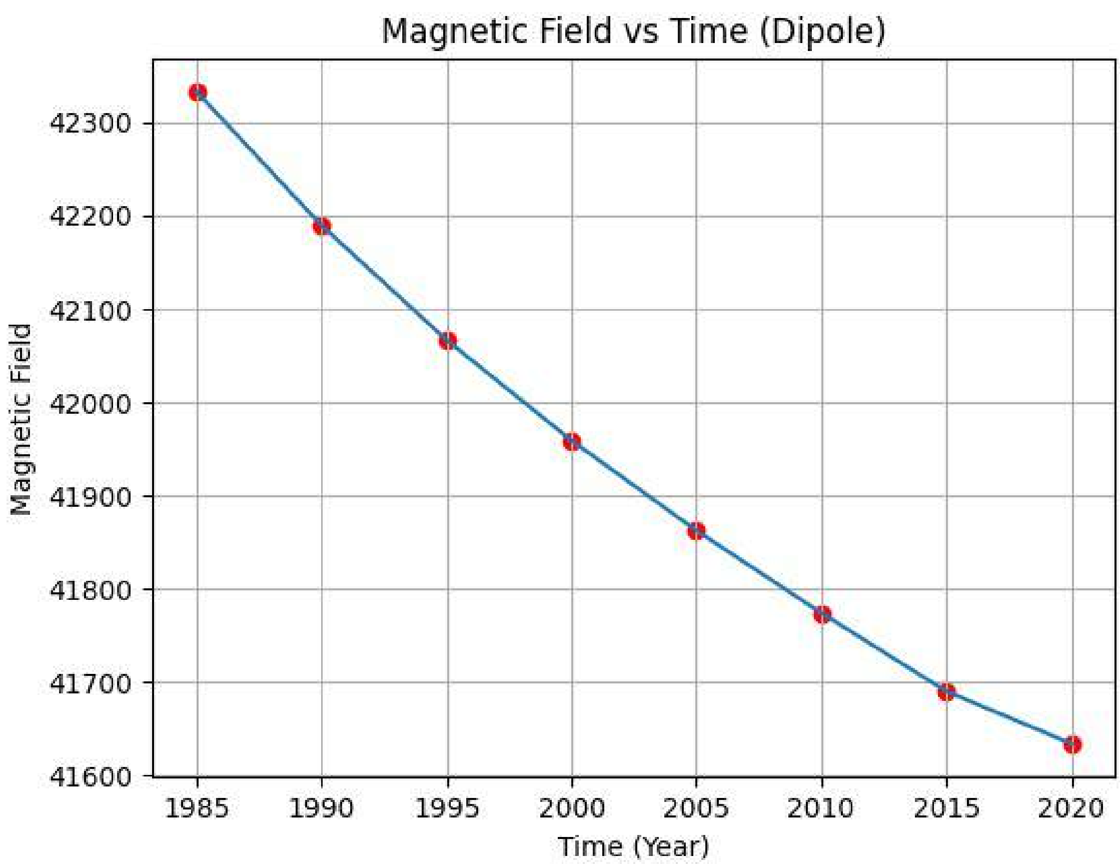
(a). Calculate the root mean square intensity of the dipole and quadrople components of the geomagnetic field at the Earth's surface for IGRF Models.

```
values2 = []
for col in df.iloc[0:3, 1:].columns:
    n = 1
    columns = df.iloc[0:2, 1:][col]
    d = np.sqrt(np.sum((n + 1) * columns ** 2))
    values2.append(d)

print("Root mean square intensity of dipole is:\n",
np.array(values2).reshape(8, 1))

Root mean square intensity of dipole is:
 [[42332.61518026]
 [42189.23391103]
 [42066.55488628]
 [41958.81659437]
 [41862.26169236]
 [41773.90123031]
 [41689.93840245]
 [41634.06576351]]

values3 = []
for col in df.iloc[3:7, 1:].columns:
    n = 2
    columns = df.iloc[3:7, 1:][col]
    d = np.sqrt(np.sum((n + 1) * columns ** 2))
    values3.append(d)

print("Root mean square intensity of quadrupole is:\n",
np.array(values3).reshape(8, 1))

Root mean square intensity of quadrupole is:
 [[7980.95821315]
 [8112.04271438]
 [8250.20308841]
 [8395.7442791 ]
 [8524.12980896]
 [8662.50194805]
```

```
 [8823.80360162]
 [8982.605691  ]]
```

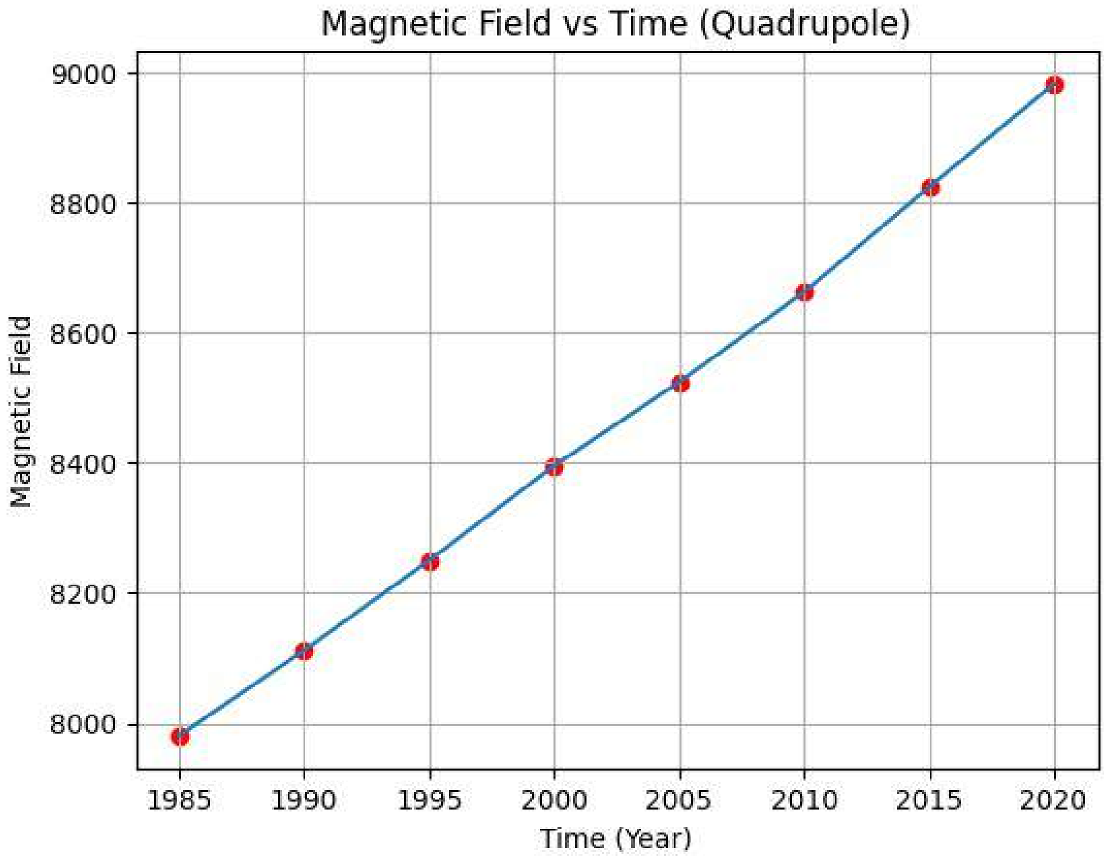
(b). Plot the root mean square intensity of both dipole and quadrupole components of the geomagnetic field as a function of time.

```
l = list(df.iloc[:, 1:].columns)
l1 = [int(i[5:]) for i in l]

plt.scatter(l1, values2, color='r')
plt.plot(l1, values2)
plt.xlabel("Time (Year)")
plt.ylabel("Magnetic Field")
plt.title("Magnetic Field vs Time (Dipole)")
plt.grid()
```




```
plt.scatter(l1, values3, color='r')
plt.plot(l1, values3)
plt.xlabel("Time (Year)")
plt.ylabel("Magnetic Field")
plt.title("Magnetic Field vs Time (Quadrupole)")
```

```
plt.grid()
plt.show()
```



(c). Estimate average rate change of both dipole and quadrupole components of the geomagnetic field.

$$F = \sqrt{(n+1) \cdot \sum_{n=1} \sum_{m=0} \left[\left(g_n^m \right)^2 + \left(h_n^m \right)^2 \right)}$$

where, n = Order m = Degree g,h = Gauss Coefficients

```
# Calculate the average rate of change per year for dipole and
quadrupole
avg_dipole = np.mean(np.diff(values2) / np.diff(l1))
avg_quadpole = np.mean(np.diff(values3) / np.diff(l1))

print("Average rate of change per year for dipole:", avg_dipole)
print("Average rate of change per year for quadrupole:", avg_quadpole)

Average rate of change per year for dipole: -19.958554764387166
Average rate of change per year for quadrupole: 28.618499367248923
```

Conclusion:

The experiment successfully calculated the root mean square (RMS) intensity of the dipole and quadrupole components of the geomagnetic field using the International Geomagnetic Reference Field (IGRF) models. The RMS intensity provides a measure of the overall strength of these components at the Earth's surface

<!DOCTYPE html>

Objective:

a) Process the raw data by applying necessary corrections.

b) Plot the Diurnal curve for the entire period of the survey.

c) Plot the raw magnetic data and processed magnetic data. Discuss the likely geologic sources of the fluctuations in the total field magnetic anomaly.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pyIGRF

# Base Magnetometer Readings
data1 = {
    'Time': ['08:56:59 AM', 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55,
60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130,
135, 140, 145, 150, 155],
    'Reading': [48512.75, 48512.58, 48512.49, 48512.65, 48516.97,
48513.29, 48515.77, 48516.68, 48517.12, 48520.73, 48515.25, 48518.35,
48512.31, 48512.21, 48512.35, 48512.49, 48512.09, 48512.88, 48513.57,
48513.97, 48513.49, 48513.37, 48513.64, 48512.88, 48514.61, 48515.17,
48514.24, 48513.54, 48511.55, 48513.46, 48512.56, 48512.7],
}

# Magnetometer Readings
df1 = pd.DataFrame(data1)
df1
```

	Time	Reading
0	08:56:59 AM	48512.75
1	5	48512.58
2	10	48512.49
3	15	48512.65
4	20	48516.97
5	25	48513.29
6	30	48515.77
7	35	48516.68
8	40	48517.12
9	45	48520.73
10	50	48515.25
11	55	48518.35
12	60	48512.31
13	65	48512.21
14	70	48512.35
15	75	48512.49
16	80	48512.09

17	85	48512.88
18	90	48513.57
19	95	48513.97
20	100	48513.49
21	105	48513.37
22	110	48513.64
23	115	48512.88
24	120	48514.61
25	125	48515.17
26	130	48514.24
27	135	48513.54
28	140	48511.55
29	145	48513.46
30	150	48512.56
31	155	48512.70

```
# Magnetometer Reading
data2 = {
    'Station': ['Base', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 'Base'],
    'Time': ['8:00', '8:10', '8:20', '8:30', '8:40', '8:50', '9:00',
'9:10', '9:20', '9:30', '9:40', '9:50', '9:60', '10:10', '10:20',
'10:30', '10:40', '10:50', '11:00', '11:10', '11:20', '11:30',
'11:40', '11:50', '12:00', '12:30'],
    'Longitude': [None, 76.2751, 76.276, 76.277, 76.27775, 76.27875,
76.2797, 76.28075, 76.2817, 76.2837, 76.28475, 76.2858, 76.2867,
76.2877, 76.2887, 76.2897, 76.29075, 76.2917, 76.2927, 76.2937,
76.29475, 76.29575, 76.2967, 76.29775, 76.29875, None],
    'Latitude': [None, 27.36647, 27.3662, 27.36611, 27.36602,
27.36586, 27.3656, 27.3655, 27.365305, 27.364972, 27.364805,
27.364638, 27.36447, 27.36427, 27.36411, 27.36394, 27.36377, 27.36361,
27.36338, 27.36327, 27.36308, 27.36291, 27.36277, 27.36258, 27.36241,
None],
    'Reading': [47217.73, 47289.25, 47311.38, 47328.11, 47333.13,
47327.16, 47290.79, 47286.93, 47278.15, 47302.61, 47311.65, 47309.25,
47383.11, 47287.73, 47272.13, 47276.19, 47270.64, 47275.67, 47284.21,
47288.37, 47311.97, 47281.7, 47309.33, 47288.94, 47296.3, 47229.37]
}

df2 = pd.DataFrame(data2)
df2
```

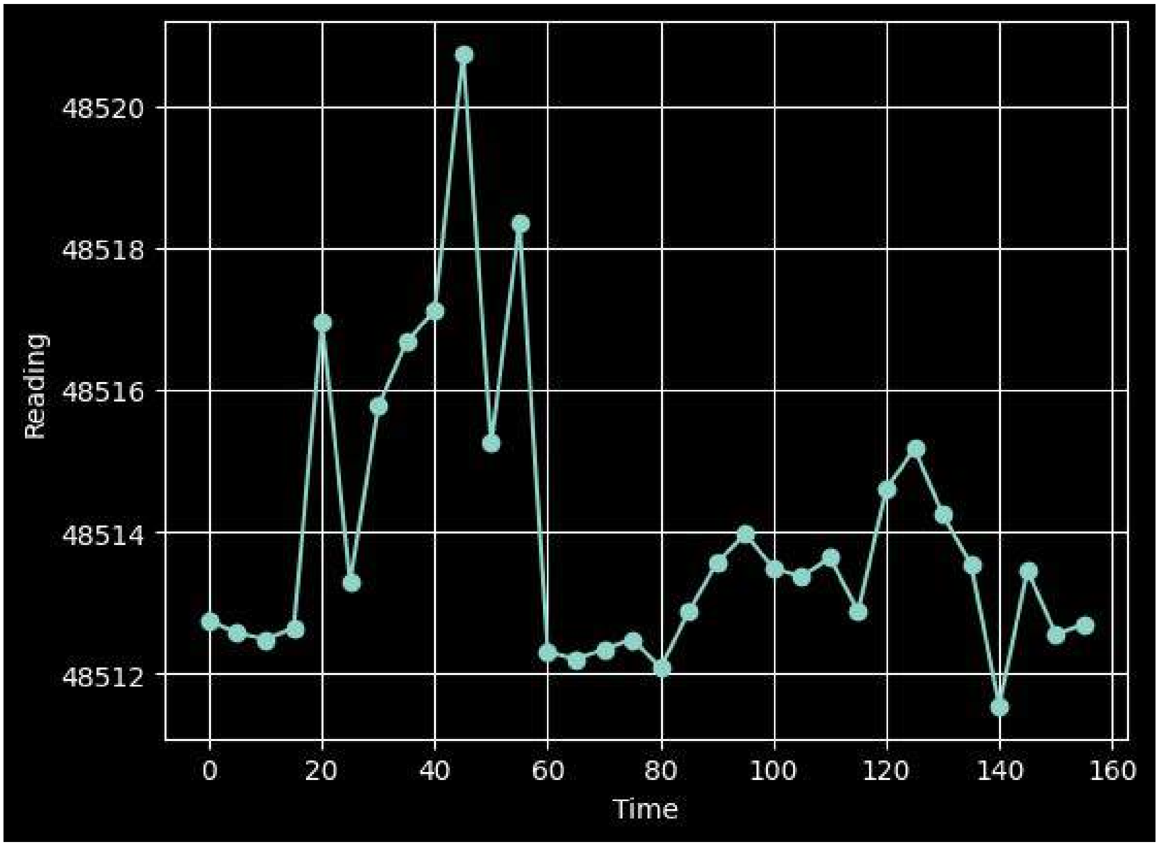
	Station	Time	Longitude	Latitude	Reading
0	Base	8:00	NaN	NaN	47217.73
1	1	8:10	76.27510	27.366470	47289.25
2	2	8:20	76.27600	27.366200	47311.38
3	3	8:30	76.27700	27.366110	47328.11
4	4	8:40	76.27775	27.366020	47333.13
5	5	8:50	76.27875	27.365860	47327.16
6	6	9:00	76.27970	27.365600	47290.79

```
7        7   9:10   76.28075  27.365500  47286.93
8        8   9:20   76.28170  27.365305  47278.15
9        9   9:30   76.28370  27.364972  47302.61
10      10   9:40   76.28475  27.364805  47311.65
11      11   9:50   76.28580  27.364638  47309.25
12      12   9:60   76.28670  27.364470  47383.11
13      13  10:10   76.28770  27.364270  47287.73
14      14  10:20   76.28870  27.364110  47272.13
15      15  10:30   76.28970  27.363940  47276.19
16      16  10:40   76.29075  27.363770  47270.64
17      17  10:50   76.29170  27.363610  47275.67
18      18  11:00   76.29270  27.363380  47284.21
19      19  11:10   76.29370  27.363270  47288.37
20      20  11:20   76.29475  27.363080  47311.97
21      21  11:30   76.29575  27.362910  47281.70
22      22  11:40   76.29670  27.362770  47309.33
23      23  11:50   76.29775  27.362580  47288.94
24      24  12:00   76.29875  27.362410  47296.30
25    Base  12:30        NaN        NaN  47229.37
```

```
df1.loc[0, 'Time'] = 0
df1['Time'] = df1['Time'].astype(int)
print(df1.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Time     32 non-null     int64  
 1   Reading  32 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 644.0 bytes
None
```

```
plt.plot(df1['Time'], df1['Reading'], marker='o')
plt.xlabel('Time')
plt.ylabel('Reading')
plt.grid()
```




```
df1['Reading_diff'] = df1['Reading'].diff()
df1['Diurnal_rate'] = df1['Reading_diff'] / 5
rate = df1.iloc[::2]['Diurnal_rate']
rate

0        NaN
2     -0.018
4      0.864
6      0.496
8      0.088
10    -1.096
12    -1.208
14     0.028
16    -0.080
18     0.138
20    -0.096
22     0.054
24     0.346
26    -0.186
28    -0.398
30    -0.180
Name: Diurnal_rate, dtype: float64
```

```
lat=np.array(df2.Latitude)
lon=np.array(df2.Longitude)
igrf=[ ]
for i in np.arange(1,25):
    k=pyIGRF.igrf_value(float(lat[i]),float(lon[i]),0, 2019)
    igrf.append(k)

igrf1=[0]*24
for j in range(24):
    igrf1[j]=igrf[j][-1]

igrf1.insert(0,0)
igrf1.insert(25,0)

df2["IGRF"]=igrf1

rt=(df2.Reading[0]-df2.Reading[25])/270
k=[rt,rt,rt,rt,rt]
rate=np.insert(rate,0,k);
print( 'diurnal rate:\n','\n',rate)
```

```
diurnal rate:

 [-0.04311111 -0.04311111 -0.04311111 -0.04311111 -0.04311111
nan
 -0.018       0.864       0.496       0.088      -1.096      -1.208
  0.028      -0.08        0.138      -0.096       0.054       0.346
 -0.186      -0.398      -0.18       ]
```

```
# Calculate anomaly
df2['Anomaly'] = df2.iloc[:, 5] - df2.iloc[:, 6]
df2.iloc[:, -1] = 0
df2.iloc[25, -1] = 0
df2
```

	Station	Time	Longitude	Latitude	Reading	IGRF	Anomaly
0	Base	8:00	NaN	NaN	47217.73	0.000000	0.0
1	1	8:10	76.27510	27.366470	47289.25	47592.082435	0.0
2	2	8:20	76.27600	27.366200	47311.38	47592.038861	0.0
3	3	8:30	76.27700	27.366110	47328.11	47592.100451	0.0
4	4	8:40	76.27775	27.366020	47333.13	47592.134855	0.0
5	5	8:50	76.27875	27.365860	47327.16	47592.159772	0.0
6	6	9:00	76.27970	27.365600	47290.79	47592.126864	0.0

```
7        7   9:10   76.28075  27.365500  47286.93  47592.188643
0.0
8        8   9:20   76.28170  27.365305  47278.15  47592.189780
0.0
9        9   9:30   76.28370  27.364972  47302.61  47592.232781
0.0
10      10   9:40   76.28475  27.364805  47311.65  47592.259451
0.0
11      11   9:50   76.28580  27.364638  47309.25  47592.286118
0.0
12      12   9:60   76.28670  27.364470  47383.11  47592.295952
0.0
13      13  10:10   76.28770  27.364270  47287.73  47592.299892
0.0
14      14  10:20   76.28870  27.364110  47272.13  47592.324783
0.0
15      15  10:30   76.28970  27.363940  47276.19  47592.344433
0.0
16      16  10:40   76.29075  27.363770  47270.64  47592.369516
0.0
17      17  10:50   76.29170  27.363610  47275.67  47592.388965
0.0
18      18  11:00   76.29270  27.363380  47284.21  47592.377176
0.0
19      19  11:10   76.29370  27.363270  47288.37  47592.428248
0.0
20      20  11:20   76.29475  27.363080  47311.97  47592.442843
0.0
21      21  11:30   76.29575  27.362910  47281.70  47592.462478
0.0
22      22  11:40   76.29670  27.362770  47309.33  47592.492393
0.0
23      23  11:50   76.29775  27.362580  47288.94  47592.506980
0.0
24      24  12:00   76.29875  27.362410  47296.30  47592.526607
0.0
25    Base  12:30        NaN        NaN  47229.37      0.000000
0.0
```

Conclusions:

There is one anomalous zone observed clearly between 0 to 0.6 km and a spike kind of zone at around 1.2-1.3 km. But the clear anomaly can be seen between 0 to 0.6 km, after that the plot is ambiguous to interpret.

<!DOCTYPE html>

Objective:

a) Plot the raw magnetic data.

b) Process the raw magnetic data by applying necessary corrections (Diurnal and IGRF corrections).

c) Plot the Diurnal curve for the entire period of the survey.

Code:

```
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
/var/folders/f0/k3cxr5sj5gb40qhbcd5dzq6m0000gn/T/
ipykernel_21730/2881088594.py:4: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major
release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type,
and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at
https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

```
# Read Base Magnetometer Readings data
path1 = "Data/BASE-MAGNETIC-READINGS-13FEB2023.txt"
df1 = pd.read_csv(path1, sep='\t', parse_dates={'TIME': ['TIME-H',
'TIME-M', 'TIME-S']})
df1['TIME'] = pd.to_datetime(df1['TIME'], format='%H %M %S')
df1
```

```
/var/folders/f0/k3cxr5sj5gb40qhbcd5dzq6m0000gn/T/
ipykernel_21730/3612014625.py:3: FutureWarning: Support for nested
sequences for 'parse_dates' in pd.read_csv is deprecated. Combine the
desired columns with pd.to_datetime after parsing instead.
  df1 = pd.read_csv(path1, sep='\t', parse_dates={'TIME': ['TIME-H',
'TIME-M', 'TIME-S']})
/var/folders/f0/k3cxr5sj5gb40qhbcd5dzq6m0000gn/T/ipykernel_21730/36120
14625.py:3: UserWarning: Could not infer format, so each element will
be parsed individually, falling back to `dateutil`. To ensure parsing
is consistent and as-expected, please specify a format.
  df1 = pd.read_csv(path1, sep='\t', parse_dates={'TIME': ['TIME-H',
'TIME-M', 'TIME-S']})
```

```
                    TIME  BASE-MAG-READINGS
0   1900-01-01 08:30:02          46653.128
1   1900-01-01 08:31:02          46652.628
2   1900-01-01 08:32:02          46652.028
3   1900-01-01 08:33:02          46652.028
4   1900-01-01 08:34:02          46651.628
..                  ...                ...
446 1900-01-01 15:56:02          46634.628
447 1900-01-01 15:57:02          46634.828
448 1900-01-01 15:58:02          46634.928
449 1900-01-01 15:59:02          46635.428
450 1900-01-01 16:00:02          46635.528

[451 rows x 2 columns]
```

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 451 entries, 0 to 450
Data columns (total 2 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   TIME               451 non-null    datetime64[ns]
 1   BASE-MAG-READINGS  451 non-null    float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 7.2 KB
```

```
path2 = "Data/RAW-MAG-DATA-DIST-13FEB2023-100M-100M-TIME-15-30.txt"
df2 = pd.read_csv(path2, sep='\t')
df2['TIME'] = pd.to_datetime(df2['TIME AM/PM'], format='%I:%M %p')
df2.drop(columns=['TIME AM/PM'], inplace=True)
df2
```

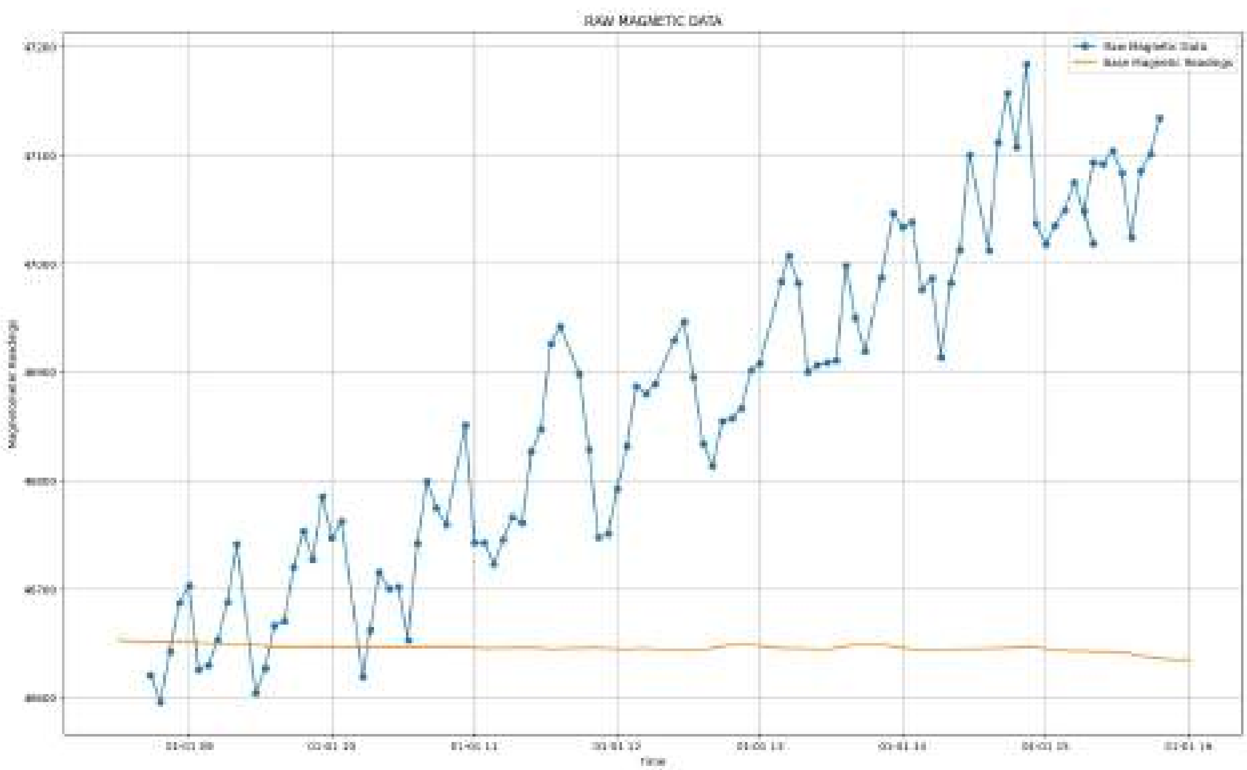
```
    X(m)  Y(m)   RAW-MAG     IGRF                TIME
0      0     0  46621.14  46381.2 1900-01-01 08:44:00
1    100     0  46595.78  46388.8 1900-01-01 08:48:00
2    200     0  46642.70  46396.4 1900-01-01 08:52:00
3    300     0  46687.13  46404.0 1900-01-01 08:56:00
4    400     0  46703.25  46411.5 1900-01-01 09:00:00
..   ...   ...       ...      ...                 ...
95   500   900  47083.70  46881.6 1900-01-01 15:32:00
96   600   900  47023.86  46889.2 1900-01-01 15:36:00
97   700   900  47085.55  46896.8 1900-01-01 15:40:00
98   800   900  47100.99  46904.3 1900-01-01 15:44:00
99   900   900  47133.94  46911.8 1900-01-01 15:48:00

[100 rows x 5 columns]
```

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   X(m)     100 non-null    int64
 1   Y(m)     100 non-null    int64
 2   RAW-MAG  100 non-null    float64
 3   IGRF     100 non-null    float64
 4   TIME     100 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(2), int64(2)
memory usage: 4.0 KB
```

a) Plot the raw magnetic data.

```
plt.figure(figsize=(20, 12))
plt.plot(df2['TIME'], df2['RAW-MAG'], marker='o', label='Raw Magnetic
Data')
plt.plot(df1['TIME'], df1['BASE-MAG-READINGS'], label='Base Magnetic
Readings')
plt.xlabel('Time')
plt.ylabel('Magnetometer Readings')
plt.title("RAW MAGNETIC DATA")
plt.legend()
plt.grid()
```



b) Process the raw magnetic data by applying necessary corrections (Diurnal and IGRF corrections).

Diurnal Rate Formula

The diurnal rate, representing the change in magnetic field intensity over time, can be calculated using the following formula:

Diurnal Rate = (Base reading at the end of the survey - Base reading at the starting of the survey) / (Time difference between starting and end of the survey (in minute))

Diurnal Correction Formula

The diurnal correction for magnetic field readings can be calculated using the following formula:

Diurnal correction = Diurnal rate × Time

```
Diurnal_Rate = df1["BASE-MAG-READINGS"].iloc[0] - df1["BASE-MAG-
READINGS"].iloc[-1]

for i in range(len(df1) - 1):
    df1["DIURNAL_CORRECTION"] = Diurnal_Rate * (df1["TIME"].iloc[i+1]
- df1["TIME"].iloc[i])

df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 451 entries, 0 to 450
Data columns (total 3 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   TIME                451 non-null    datetime64[ns]
 1   BASE-MAG-READINGS   451 non-null    float64
 2   DIURNAL_CORRECTION  451 non-null    timedelta64[ns]
dtypes: datetime64[ns](1), float64(1), timedelta64[ns](1)
memory usage: 10.7 KB
```

```
# Convert DIURNAL_CORRECTION from timedelta to seconds, then to
integer
df1["DIURNAL_CORRECTION"] =
df1["DIURNAL_CORRECTION"].dt.total_seconds().astype(int)
df1
```

```
                   TIME  BASE-MAG-READINGS  DIURNAL_CORRECTION
0   1900-01-01 08:30:02          46653.128                1055
1   1900-01-01 08:31:02          46652.628                1055
2   1900-01-01 08:32:02          46652.028                1055
3   1900-01-01 08:33:02          46652.028                1055
4   1900-01-01 08:34:02          46651.628                1055
..                  ...                ...                 ...
```

```
446 1900-01-01 15:56:02          46634.628                1055
447 1900-01-01 15:57:02          46634.828                1055
448 1900-01-01 15:58:02          46634.928                1055
449 1900-01-01 15:59:02          46635.428                1055
450 1900-01-01 16:00:02          46635.528                1055

[451 rows x 3 columns]
```

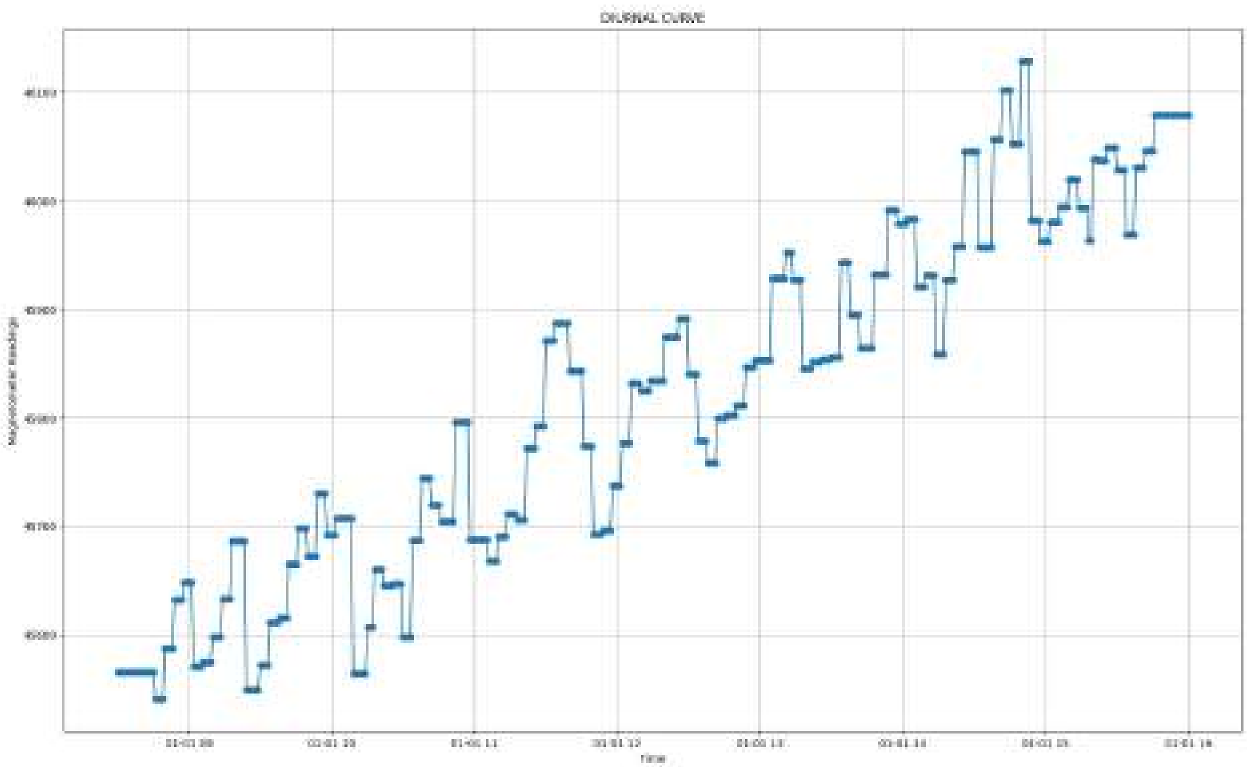
```
df1.sort_values(by='TIME', inplace=True)
df2.sort_values(by='TIME', inplace=True)
data = pd.merge_asof(df1, df2, on='TIME', direction='nearest')

data['DIURNAL_CORRECTED'] = data['RAW-MAG'] -
data["DIURNAL_CORRECTION"]
data
```

```
                   TIME  BASE-MAG-READINGS  DIURNAL_CORRECTION  X(m)
Y(m)  \
0   1900-01-01 08:30:02          46653.128                1055     0
0
1   1900-01-01 08:31:02          46652.628                1055     0
0
2   1900-01-01 08:32:02          46652.028                1055     0
0
3   1900-01-01 08:33:02          46652.028                1055     0
0
4   1900-01-01 08:34:02          46651.628                1055     0
0
..                  ...                ...                 ...   ...
...
446 1900-01-01 15:56:02          46634.628                1055   900
900
447 1900-01-01 15:57:02          46634.828                1055   900
900
448 1900-01-01 15:58:02          46634.928                1055   900
900
449 1900-01-01 15:59:02          46635.428                1055   900
900
450 1900-01-01 16:00:02          46635.528                1055   900
900

      RAW-MAG     IGRF  DIURNAL_CORRECTED
0    46621.14  46381.2           45566.14
1    46621.14  46381.2           45566.14
2    46621.14  46381.2           45566.14
3    46621.14  46381.2           45566.14
4    46621.14  46381.2           45566.14
..        ...      ...                ...
446  47133.94  46911.8           46078.94
447  47133.94  46911.8           46078.94
```

```
448  47133.94  46911.8           46078.94
449  47133.94  46911.8           46078.94
450  47133.94  46911.8           46078.94

[451 rows x 8 columns]
```

c) Plot the Diurnal curve for the entire period of the survey.

```
plt.figure(figsize=(20, 12))
plt.plot(data['TIME'], data['DIURNAL_CORRECTED'], marker='o')
plt.xlabel('Time')
plt.ylabel('Magnetometer Readings')
plt.title("DIURNAL CURVE")
plt.grid()
```



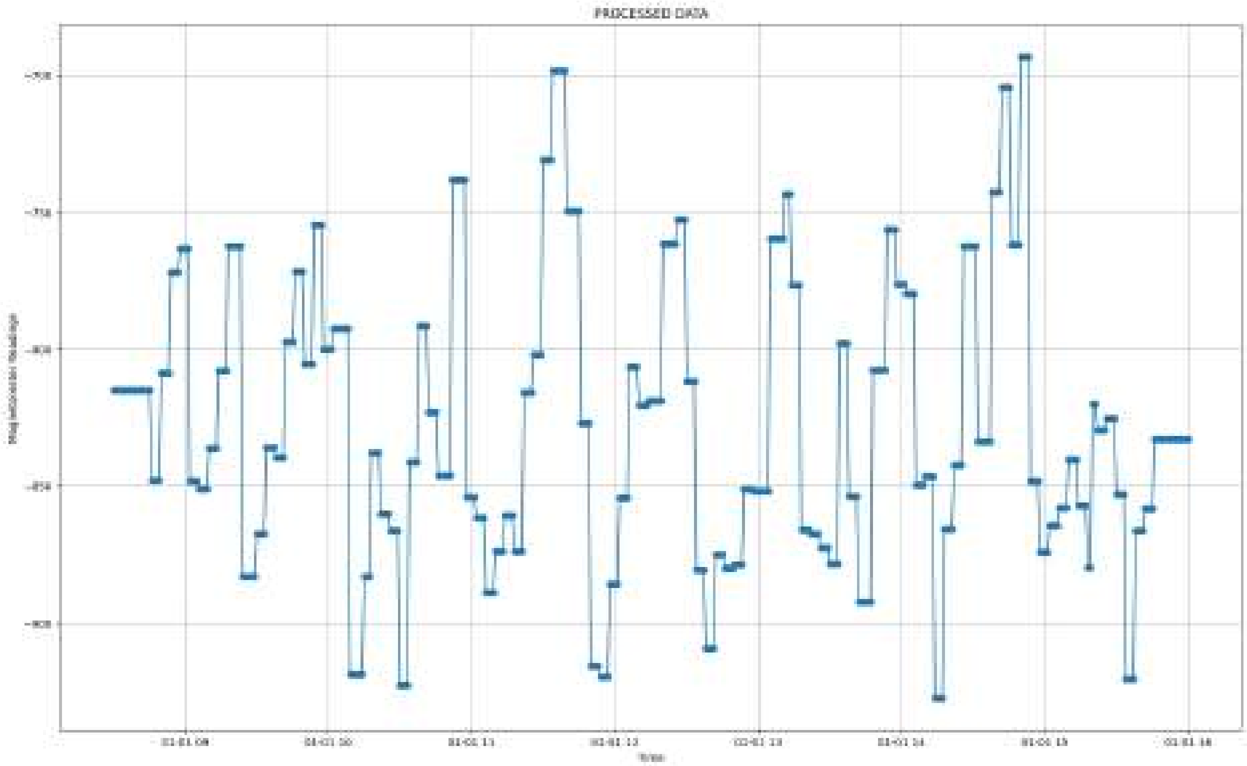
IGRF Correction: The International Geomagnetic Reference Field (IGRF) is a standard mathematical description of the large-scale structure of the Earth's main magnetic field and its secular variation. Subtracting the IGRF field from the Earth's field will in principle result in anomalies caused by sources within the Earth's crust where the temperatures are less than the Curie temperatures of important magnetic minerals.

```
data['PROCESSED'] = data['DIURNAL_CORRECTED'] - data["IGRF"]
data

                   TIME  BASE-MAG-READINGS  DIURNAL_CORRECTION  X(m)
Y(m)  \
```

```
0    1900-01-01 08:30:02          46653.128                1055     0
0
1    1900-01-01 08:31:02          46652.628                1055     0
0
2    1900-01-01 08:32:02          46652.028                1055     0
0
3    1900-01-01 08:33:02          46652.028                1055     0
0
4    1900-01-01 08:34:02          46651.628                1055     0
0
..                  ...                ...                 ...   ...
...
446  1900-01-01 15:56:02          46634.628                1055   900
900
447  1900-01-01 15:57:02          46634.828                1055   900
900
448  1900-01-01 15:58:02          46634.928                1055   900
900
449  1900-01-01 15:59:02          46635.428                1055   900
900
450  1900-01-01 16:00:02          46635.528                1055   900
900

      RAW-MAG     IGRF  DIURNAL_CORRECTED  PROCESSED
0    46621.14  46381.2           45566.14    -815.06
1    46621.14  46381.2           45566.14    -815.06
2    46621.14  46381.2           45566.14    -815.06
3    46621.14  46381.2           45566.14    -815.06
4    46621.14  46381.2           45566.14    -815.06
..        ...      ...                ...        ...
446  47133.94  46911.8           46078.94    -832.86
447  47133.94  46911.8           46078.94    -832.86
448  47133.94  46911.8           46078.94    -832.86
449  47133.94  46911.8           46078.94    -832.86
450  47133.94  46911.8           46078.94    -832.86

[451 rows x 9 columns]
```

d) Plot the processed magnetic data.

```
plt.figure(figsize=(20, 12))
plt.plot(data['TIME'], data['PROCESSED'], marker='o')
plt.xlabel('Time')
plt.ylabel('Magnetometer Readings')
plt.title("PROCESSED DATA")
plt.grid()
```



Conclusion:

This concludes the usage of diurnal correction in processing raw magnetic data: it is necessary to have continuous base station readings to correct for the time varying magnetic field.

<!DOCTYPE html>

Objective:

To perform regional-residual separation on magnetic data.

Plots:

1. Regional:

2. Residual:

Residual Interpretation:

1. Almost spherical body implied by closed contour lines. 2. Strike direction gives the presence of other point sources.

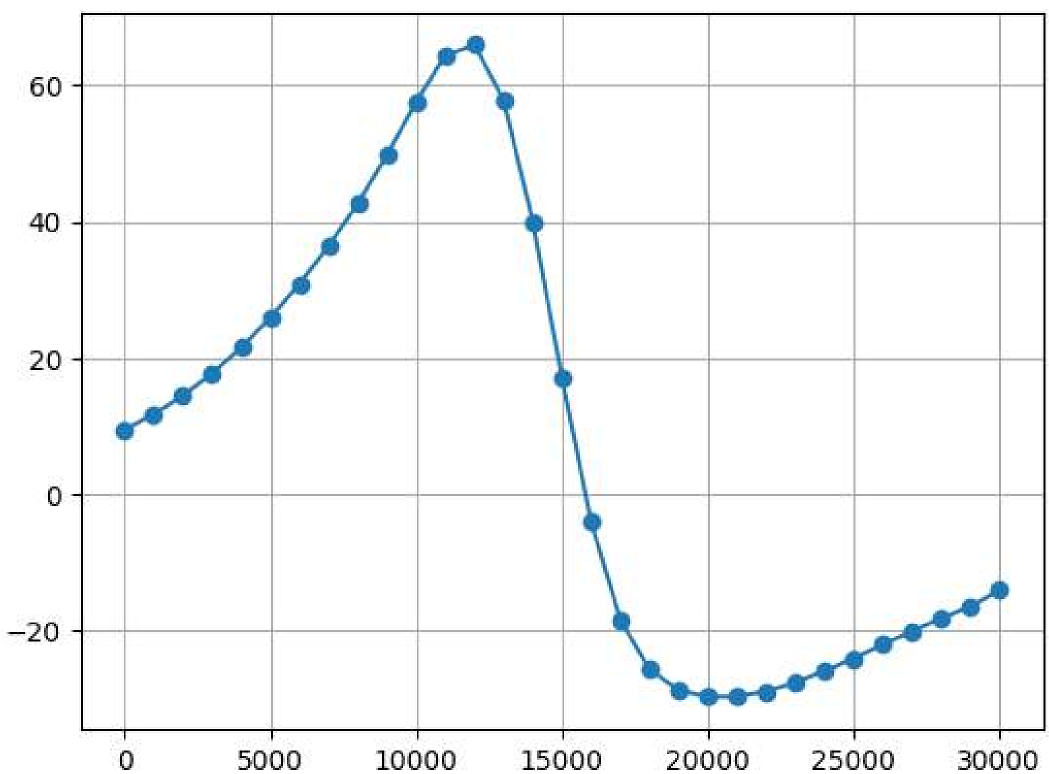
<!DOCTYPE html>

Objective:

Compute and plot the residual magnetic anomaly along the given magnetic profile data using the polynomial regression (LSM) technique. Also, write your comments on residual and polynomial regression anomaly plots.

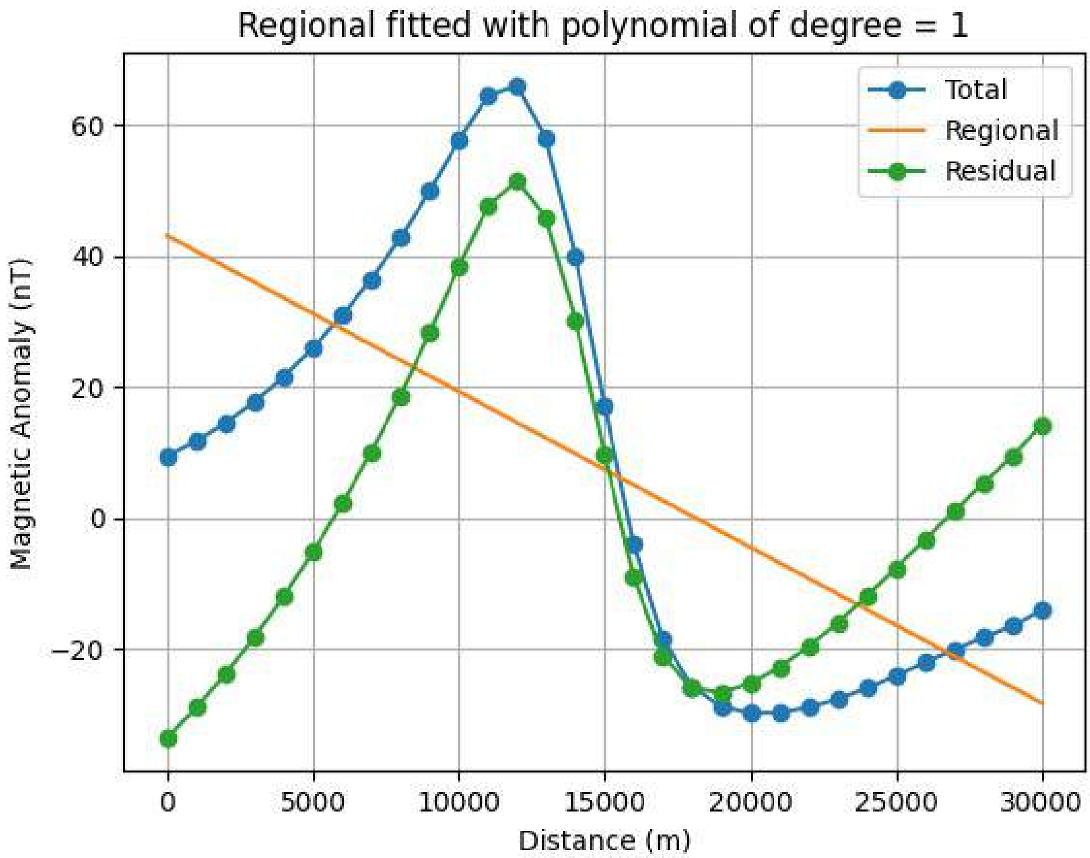
Code:

```
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt

distance = np.arange(0,31000,1000)
magnetic_anomaly =
np.array([9.45,11.7,14.43,17.68,21.5,25.9,30.86,36.43,42.69,49.8,57.6,
64.4,
                             65.96,57.8,39.97,17.09,-4.12,-18.47,-
25.79,-28.84,-29.83,-29.76,
                             -29,-27.71,-26.04,-24.14,-22.13,-20.18,-
18.3,-16.55,-14.15])

plt.plot(distance,magnetic_anomaly, marker="o")
plt.grid()
```



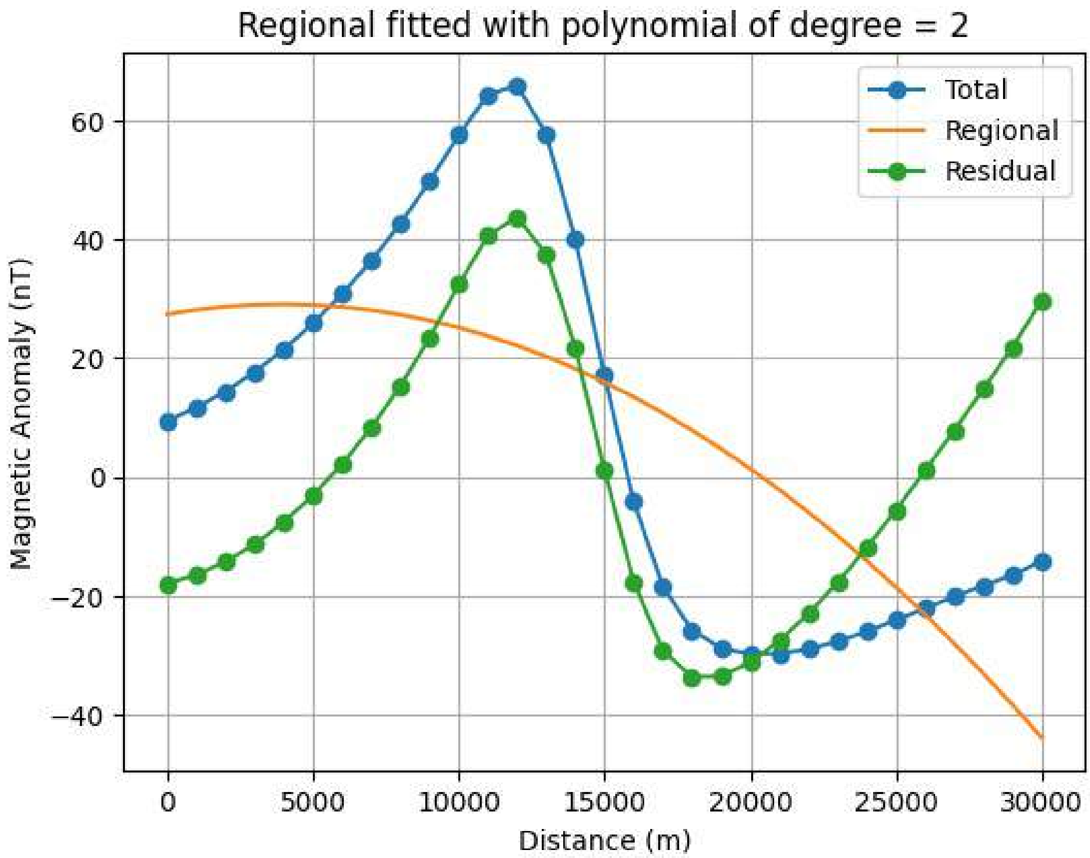
```python
class PolynomialFit:
    def __init__(self, x, y, deg):
        self.k = np.polyfit(x, y, deg)
        self.z = np.poly1d(self.k)

    def calculation(self, x, y):
        # mse
        err = np.sqrt(np.mean((self.z(x) - y)**2))
        print(f'Mean Squared Error: {err}')

        # r-squared
        g = np.sum((y-self.z(x))**2)
        m = np.sum((y-np.mean(y))**2)
        if m == 0: r2 = 0
        else: r2 = 1 - (g/m)
        print(f'R_squared: {r2}')

        # calculations
        regional = self.z(x)
        residual = y - regional
        print(f'Sum of residual: {np.sum(residual)}')
        print(f'Sum of Moment of residual about x (distance): 
{np.sum(residual*x)}')
        print(f'Sum of Moment of residual about y (magnetic anomaly): 
{np.sum(residual*y)}')
        return regional, residual

    def plot(self, x, y, deg):
        regional, residual = self.calculation(x, y)
        plt.plot(x, y, 'o-', label='Total')
        plt.plot(x, regional, label='Regional')
        plt.plot(x, residual, 'o-', label='Residual')
        plt.xlabel('Distance (m)')
        plt.ylabel('Magnetic Anomaly (nT)')
        plt.title(f'Regional fitted with polynomial of degree = 
{deg}')
        plt.legend()
        plt.grid()

Deg = 1
pfit = PolynomialFit(distance, magnetic_anomaly, Deg)
pfit.plot(distance, magnetic_anomaly, Deg)
```

```
Mean Squared Error: 24.107591314360484
R_squared: 0.4375817674437179
Sum of residual: 2.575717417130363e-13
Sum of Moment of residual about x (distance): 6.810296326875687e-09
Sum of Moment of residual about y (magnetic anomaly): 
18016.45472838709
```



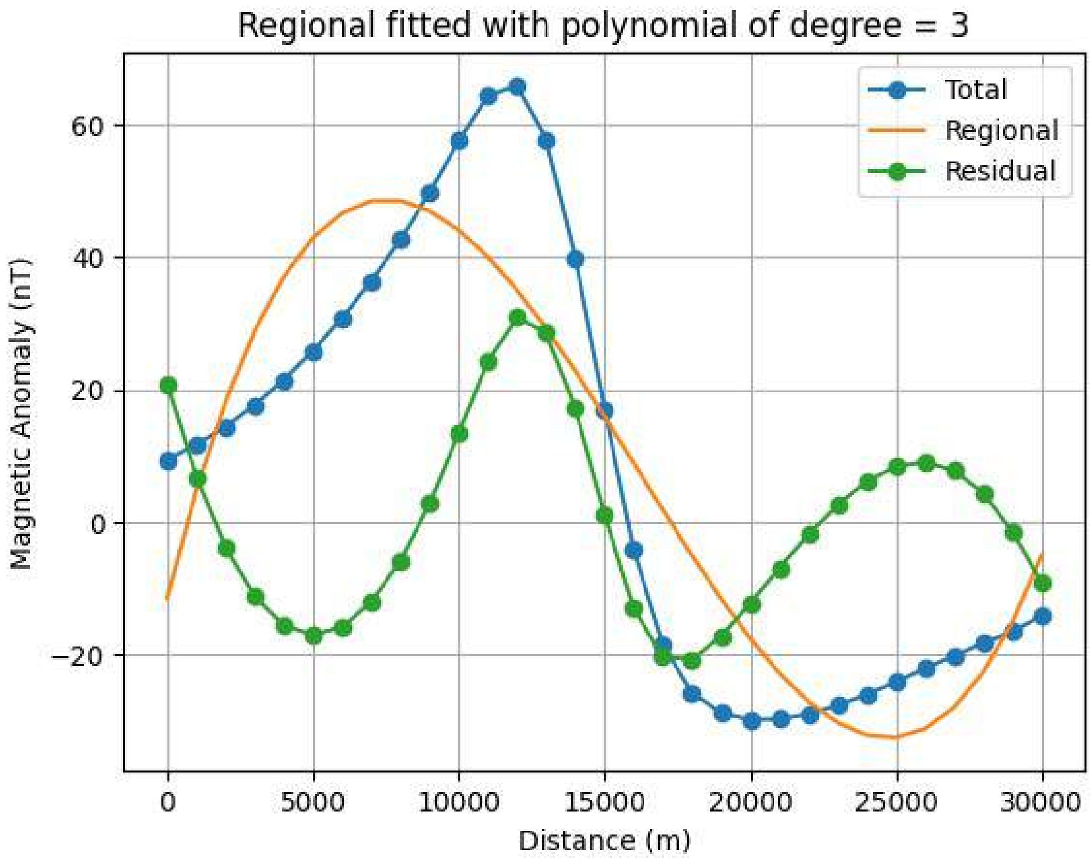

```
Deg = 2
pfit = PolynomialFit(distance, magnetic_anomaly, Deg)
pfit.plot(distance, magnetic_anomaly, Deg)
```

```
Mean Squared Error: 22.849319872344076
R_squared: 0.49475932656749255
Sum of residual: 3.304023721284466e-13
Sum of Moment of residual about x (distance): 6.6356733441352844e-09
Sum of Moment of residual about y (magnetic anomaly):
16184.833977489634
```



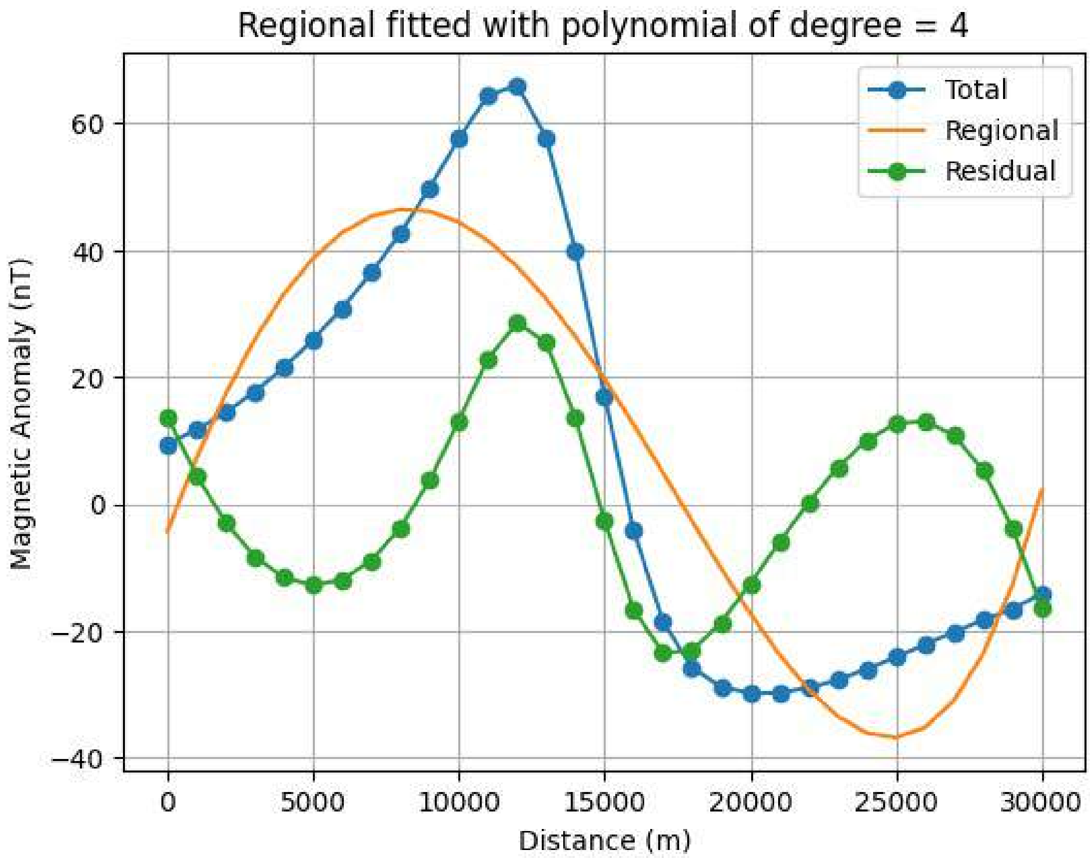

```
Deg = 3
pfit = PolynomialFit(distance, magnetic_anomaly, Deg)
pfit.plot(distance, magnetic_anomaly, Deg)
```

```
Mean Squared Error: 14.274793293087205
R_squared: 0.8028070397571232
Sum of residual: 1.9966250874858815e-12
Sum of Moment of residual about x (distance): -9.138602763414383e-09
Sum of Moment of residual about y (magnetic anomaly):
6316.861430371542
```



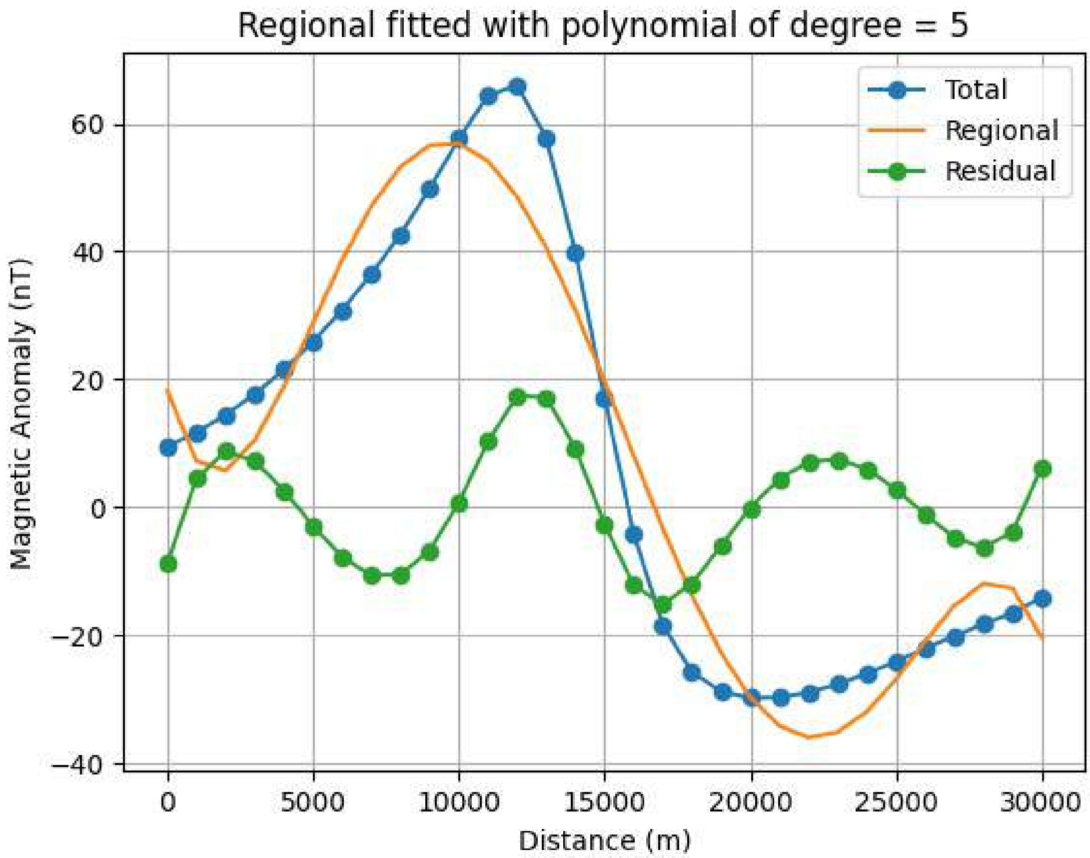

```
Deg = 4
pfit = PolynomialFit(distance, magnetic_anomaly, Deg)
pfit.plot(distance, magnetic_anomaly, Deg)
```

```
Mean Squared Error: 13.888989444314632
R_squared: 0.8133220401102047
Sum of residual: 7.531752999057062e-13
Sum of Moment of residual about x (distance): -6.2282197177410126e-09
Sum of Moment of residual about y (magnetic anomaly):
5980.024861312842
```



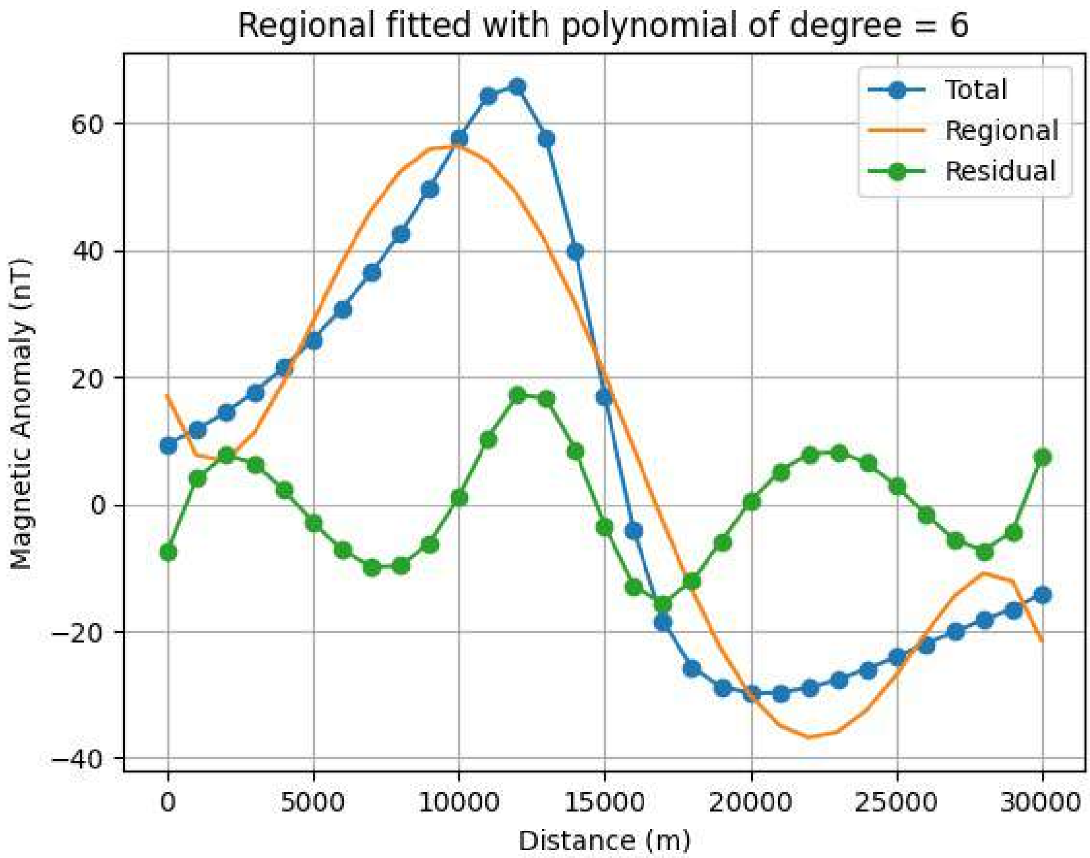

```
Deg = 5
pfit = PolynomialFit(distance, magnetic_anomaly, Deg)
pfit.plot(distance, magnetic_anomaly, Deg)
```

```
Mean Squared Error: 8.45005697217573
R_squared: 0.9309011173906645
Sum of residual: 1.4475531884272641e-11
Sum of Moment of residual about x (distance): -7.9016899690032e-08
Sum of Moment of residual about y (magnetic anomaly):
2213.5073478247004
```




```
Deg = 6
pfit = PolynomialFit(distance, magnetic_anomaly, Deg)
pfit.plot(distance, magnetic_anomaly, Deg)
```

```
Mean Squared Error: 8.423954262145356
R_squared: 0.9313273588432536
Sum of residual: -7.643663479939278e-12
Sum of Moment of residual about x (distance): 6.606569513678551e-08
Sum of Moment of residual about y (magnetic anomaly):
2199.8531677317046
```



Interpretation:

From the above observations of regional curves, we can interpret that the curves with degree 3 and 4 are giving better estimate of residual anomaly. At regions having high and low in total magnetic anomaly curve, there are two anomaly highs in residual anomaly curve. Some of the conditions we need to keep in mind that the algebraic sum of the residual anomalies must be zero and their sum of moments taken in two mutually perpendicular directions about the arbitrary origin are separately zero. The moment of anomalies need not to be zero.

<!DOCTYPE html>

Objective:

Discuss the effect of magnetic inclination (i) and depth (z) of the sphere body on the total magnetic anomaly profiles. Assume that magnetization only due to induction.

Formula used:

Theory:

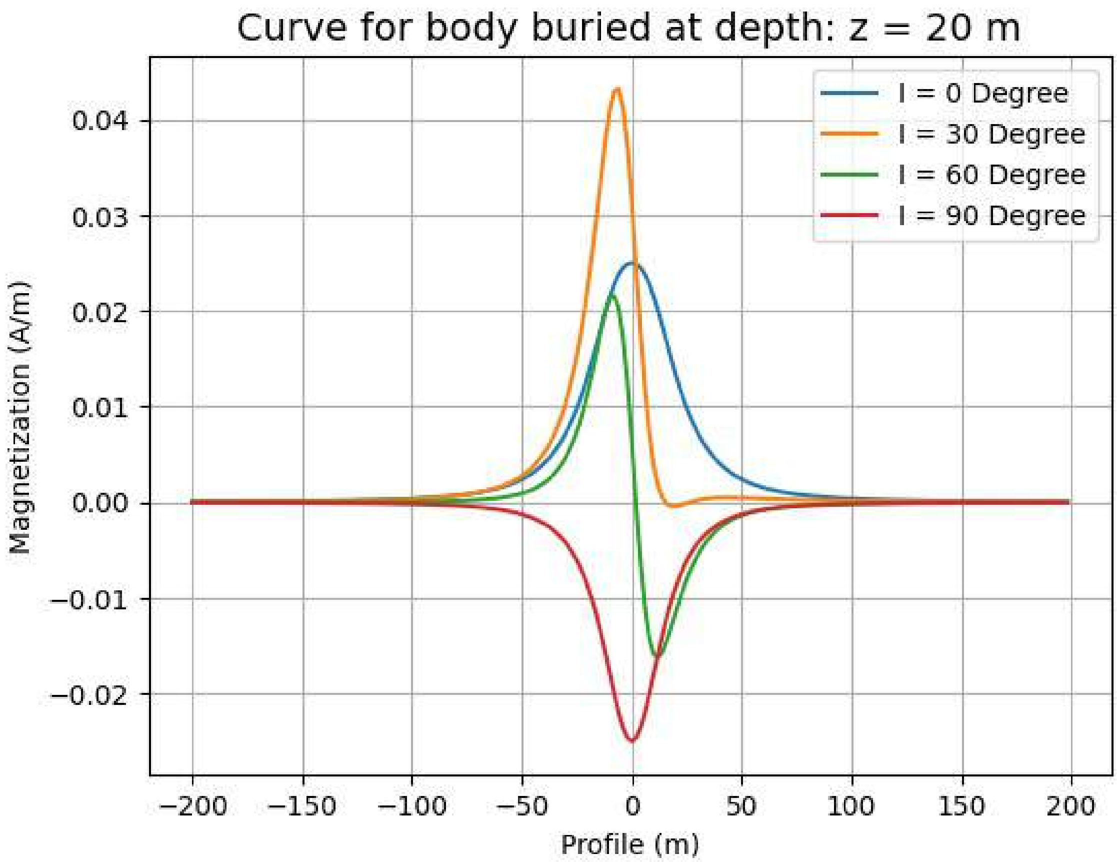
Magnetic inclination refers to the angle between the Earth's magnetic field lines and the horizontal plane. The magnetic field of a magnetic sphere body depends on the angle between the magnetic moment of the body and the Earth's magnetic field lines.

Depth (z) of the magnetic sphere body can also have a significant effect on the total magnetic anomaly profiles. As the sphere body is moved deeper into the Earth, the strength of the magnetic field that is measured at the surface will decrease. This is because the magnetic field must travel through more of the Earth's magnetic field, which causes it to become weaker.

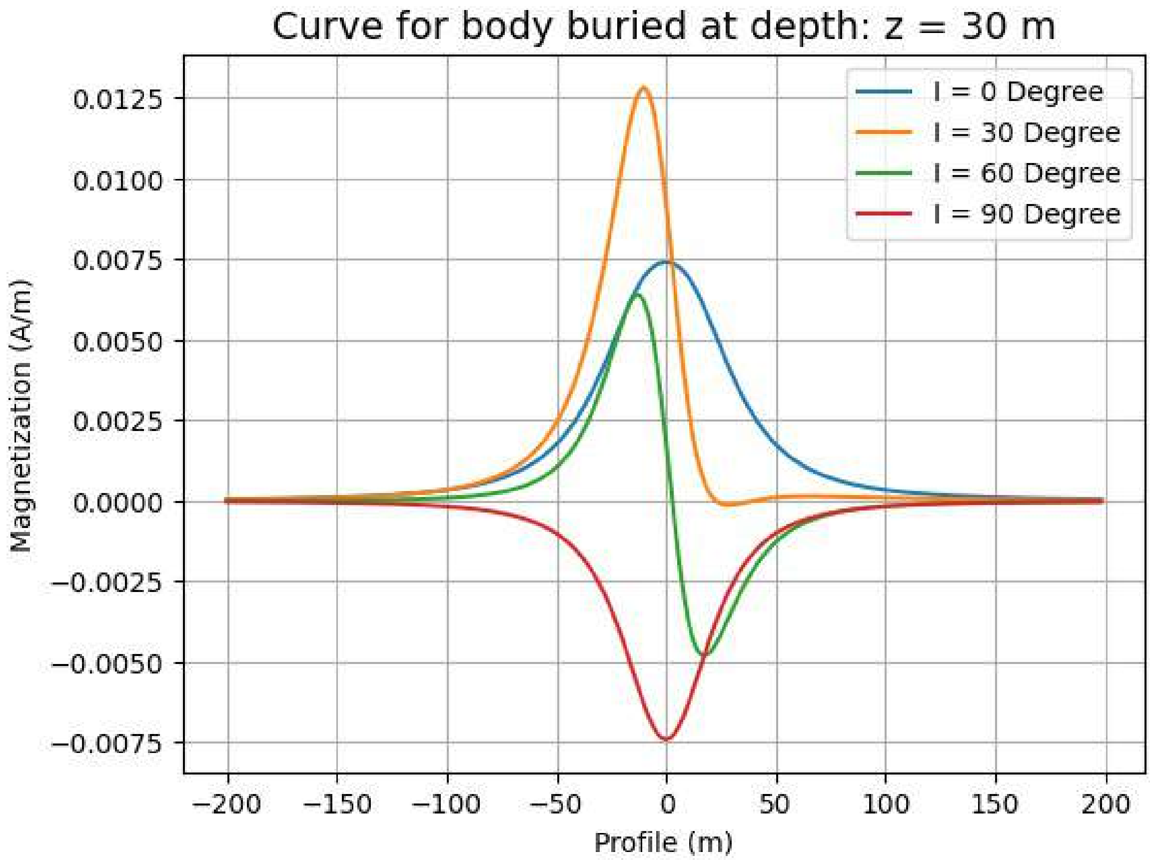
Code:

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

class SphericalBody:
    def __init__(self):
        self.x = np.arange(-200, 200, 2)
        self.i = [0, 30, 60, 90]
        self.z = [20, 30, 50]
        self.M = 200

    def calculate(self):
        jj = []
        for k in range(len(self.z)):
            for j in range(len(self.i)):

a=(self.x**2+self.z[k]**2)*np.cos(2*np.radians(self.i[j]))
                b=(self.x**2)*(np.cos(np.radians(self.i[j])))**2
                c=-3*self.x*self.z[k]*np.sin(2*np.radians(self.i[j]))
                d=(self.z[k]**2)*(np.sin(2*np.radians(self.i[j])))**2
                e=(self.x**2+self.z[k]**2)**(5/2)
                f=self.M*(a+b+c+d)/e
                jj.append(f)
        return jj

    def plot(self, n):
        result = self.calculate()
```

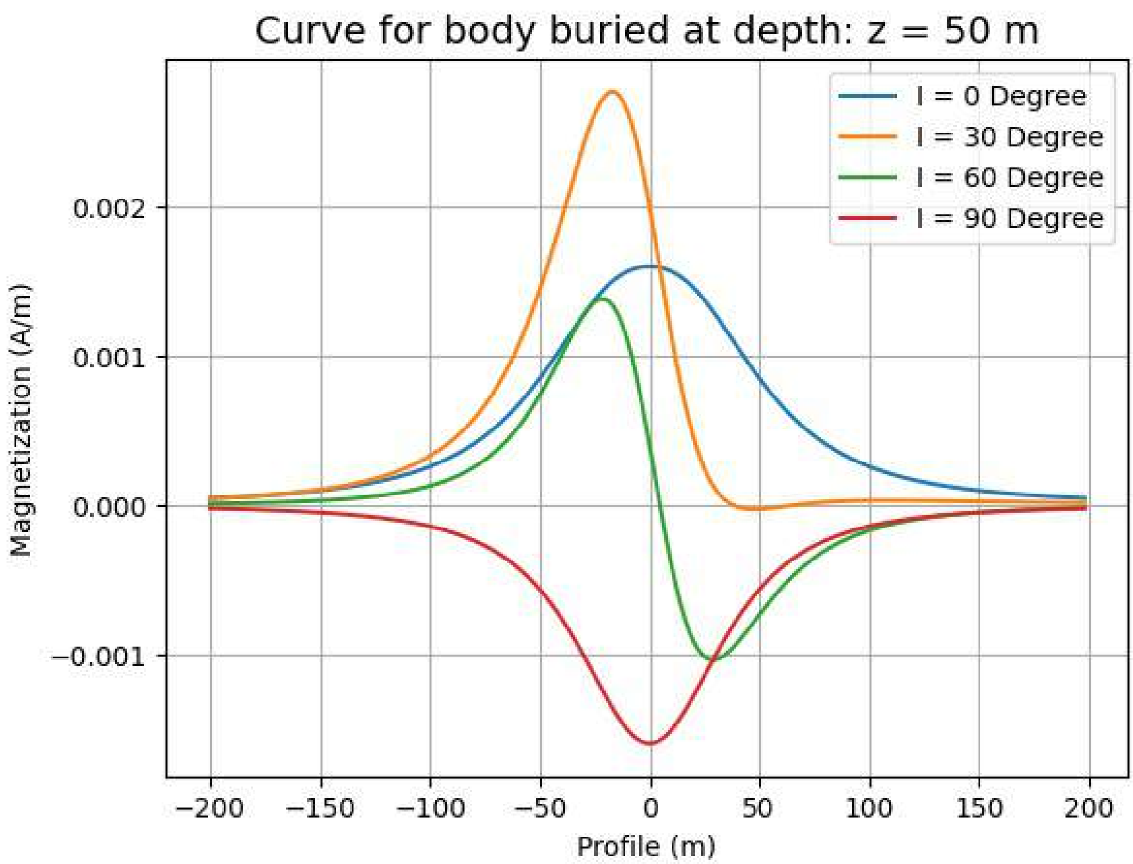
```
        for i in range(4):
            plt.plot(self.x, result[i+n*4], label=f'I = {self.i[i]}
Degree')
        plt.legend()
        plt.xlabel('Profile (m)')
        plt.ylabel('Magnetization (A/m)')
        plt.title(f'Curve for body buried at depth: z = {self.z[n]}
m', fontsize=14)
        plt.grid()

spherical_body = SphericalBody()
spherical_body.plot(0)
```




```
spherical_body.plot(1)
```




```
spherical_body.plot(2)
```



Interpretation:

From the above curves observation we can infer that, the effect of depth as : • As the depth increases the peak value of magnetization decreases. • Curve is narrow for the shallow objects while it is broader for the deeper objects.

Also the effect of inclination is : • 0 degree corresponds to equator, we get the global maxima • 90 degrees corresponds to Magnetic Pole, we get a curve with global minima.

When we are moving from equator towards the pole, magnetic inclination increases, thus peak of curve changes from positive to negative.

<!DOCTYPE html>

Objective:

Discuss the effect of magnetic inclination (i) and depth (z) of the sphere body on the total magnetic anomaly profiles. Assume that magnetization only due to induction.

Formula Used:

Theory:

Effect of Magnetic Inclination (i):

- Magnetic inclination refers to the angle between the magnetic field lines and the horizontal plane. As the inclination angle changes, it affects the distribution of magnetization within the Earth's subsurface.
- When the inclination angle is zero (i = 0°), indicating the magnetic field lines are parallel to the Earth's surface, the magnetic anomaly profile exhibits certain characteristics. As the inclination angle increases, the anomaly profile changes accordingly.
- Higher inclination angles may result in more pronounced anomalies, especially when the inclination angle is close to 90°, indicating nearly vertical magnetic field lines.

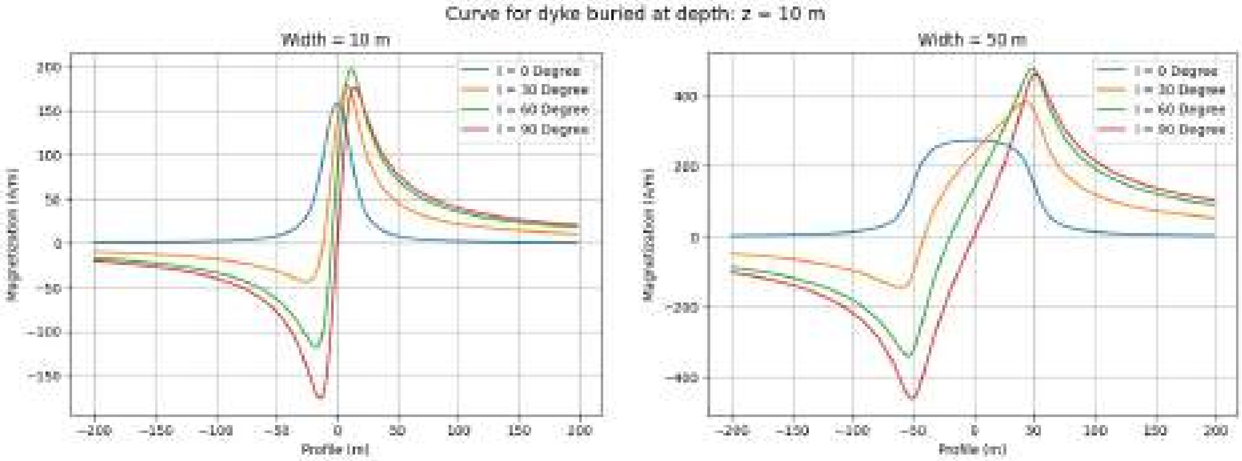
Effect of Depth (z):

- The depth of the spherical body influences the magnetic anomaly profile by altering the distribution of magnetization within the subsurface.
- When the body is closer to the surface (lower depth), the magnetic anomaly tends to have a sharper and more localized shape. This is because the magnetic effect of the body is more concentrated in the vicinity of the surface.
- On the other hand, as the depth of the body increases, the magnetic anomaly becomes broader and more subdued. This is due to the attenuation of the magnetic signal with depth, resulting in a more diffused anomaly profile.

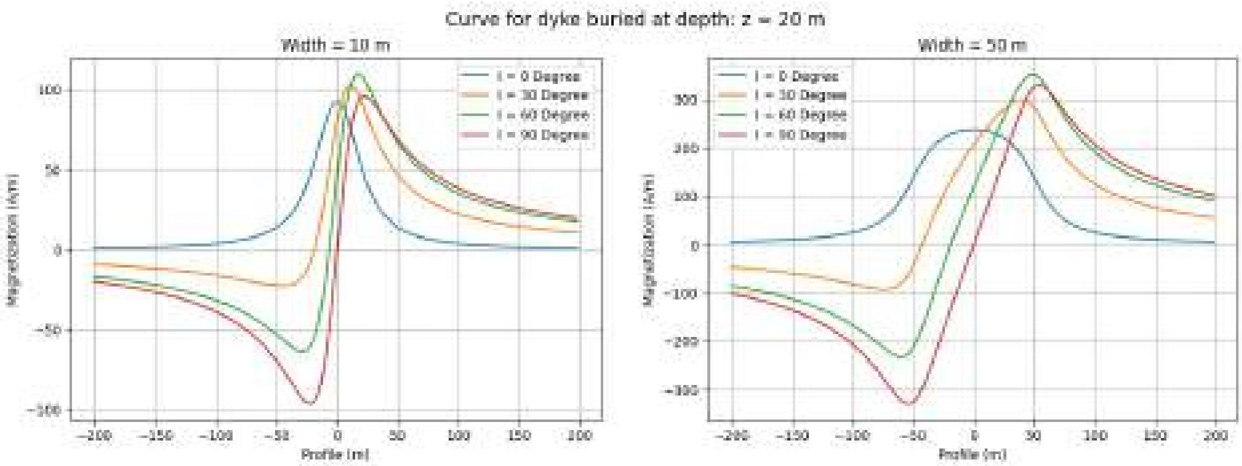
Code:

```
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt

class DykeBody:
    def __init__(self):
        self.X = np.arange(-200, 200, 2)
        self.I = np.array([0, 30, 60, 90])
        self.Q = np.radians(self.I)
        self.Z = [10, 20, 30]
        self.T = [10, 50]
        self.CF = 100
```

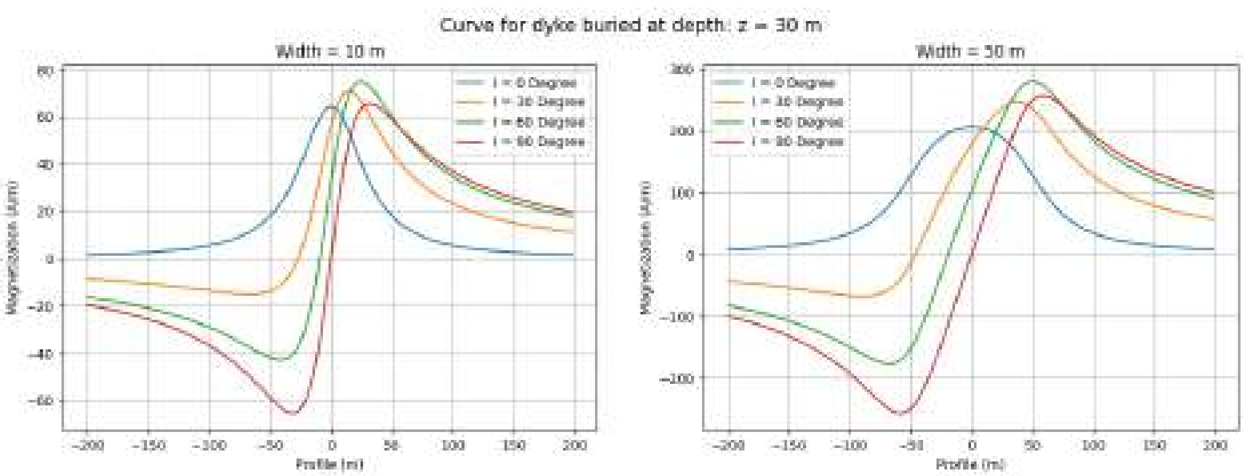
```
    def calculate(self, n, m):
        jj = []
        for j in range(len(self.Q)):
            A = np.arctan((self.X + self.T[m]) / self.Z[n]) -
np.arctan((self.X - self.T[m]) / self.Z[n])
            B = np.log(((self.X + self.T[m])**2 + self.Z[n]**2) /
((self.X - self.T[m])**2 + self.Z[n]**2))
            F = self.CF * (A * np.cos(self.Q[j]) + B *
np.sin(self.Q[j]))
            jj.append(F)
        return jj

    def plot(self, n):
        result1 = self.calculate(n , m=0)
        result2 = self.calculate(n , m=1)
        fig, axs = plt.subplots(1,2, figsize=(16, 5))
        fig.suptitle(f'Curve for dyke buried at depth: z = {self.Z[n]}
m', fontsize=14)

        for i in range(4):
            axs[0].plot(self.X, result1[i], label=f'I = {self.I[i]}
Degree')
            axs[1].plot(self.X, result2[i], label=f'I = {self.I[i]}
Degree')

        axs[0].set_title(f'Width = {self.T[0]} m')
        axs[1].set_title(f'Width = {self.T[1]} m')

        for ax in axs.flat:
            ax.set(xlabel='Profile (m)', ylabel='Magnetization (A/m)')
            ax.legend()
            ax.grid()


Dyke = DykeBody()
Dyke.plot(0)
```




```
Dyke.plot(1)
```




```
Dyke.plot(2)
```



Interpretation:

Effect of Depth:

- As the depth of the dyke increases, the peak value of magnetization tends to decrease. This reduction in peak value occurs due to the attenuation of the magnetic signal with depth.
- Dykes closer to the surface exhibit narrower anomaly curves, reflecting their more localized magnetic effect. In contrast, deeper dykes produce broader anomaly curves, as the magnetic signal is dispersed over a wider area with increasing depth.

Effect of Inclination:

- At 0 degrees inclination, corresponding to the equator, the magnetic anomaly tends to reach its global maximum. This occurs due to the alignment of magnetic field lines parallel to the surface.
- Conversely, at 90 degrees inclination, corresponding to the Magnetic Pole, the magnetic anomaly typically exhibits a global minimum. This phenomenon arises because magnetic field lines become nearly vertical, resulting in a reduced magnetic anomaly signal.
- Transitioning from the equator towards the pole, the magnetic inclination increases, leading to a change in the peak of the curve from positive to negative. This shift in peak polarity is a characteristic feature observed in dyke magnetic anomaly profiles.

<!DOCTYPE html>

Objective:

Discuss the effect of depth (z2), and the direction of arbitrary magnetization angle (θ) of vertical fault on the total magnetic anomaly profiles. Assume that magnetization is only due to induction. The general expression for the total field anomaly over a vertical fault is given below:

THEORY:

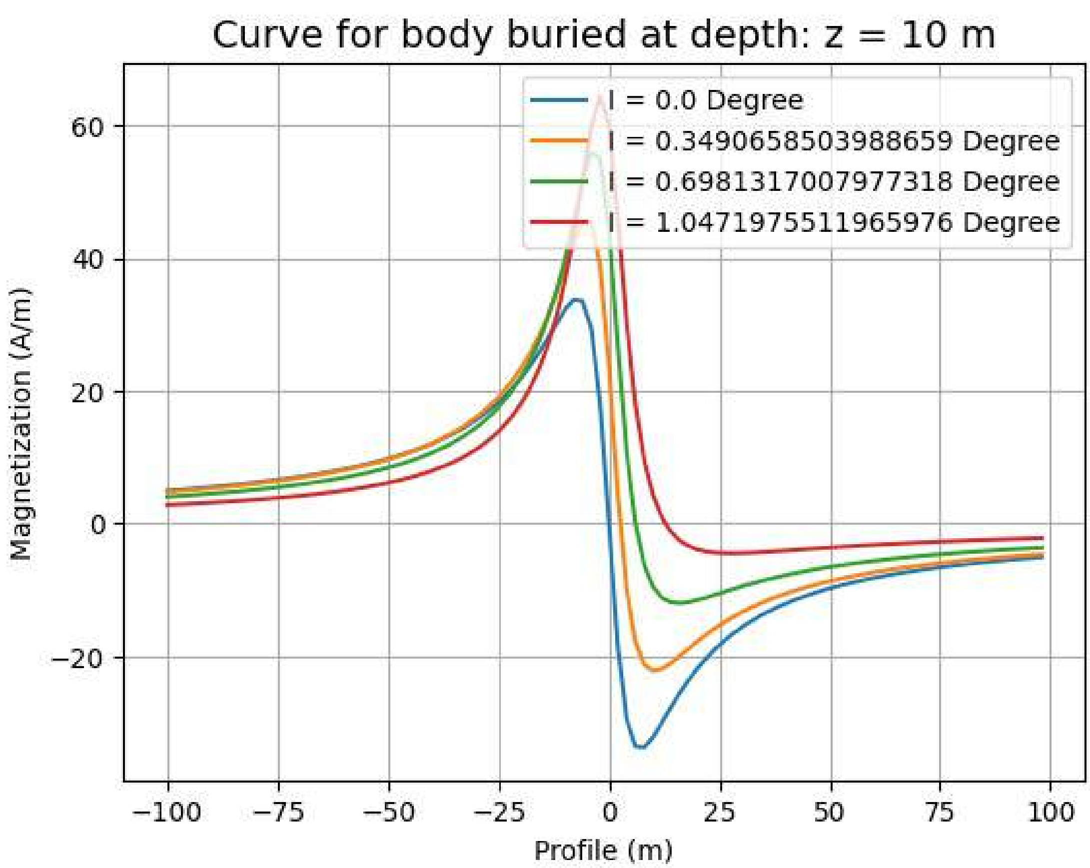
The gravity anomaly of a body is caused by the density contrast ($\Delta\rho$) between the body and its surroundings. The shape of the anomaly is determined by the shape of the body and its depth of burial. Similarly, a magnetic anomaly originates in the magnetization contrast (ΔM) between rocks with different magnetic properties. The shape of the anomaly depends not only on the shape and depth of the source object but also on its orientation to the profile and to the inducing magnetic field, which itself varies in intensity and direction with geographical location. In oceanic magnetic surveying the magnetization contrast results from differences in the remanent magnetizations of crustal rocks, for which the Königsberger ratio is much greater than unity (i.e., Qn >>1). Commercial geophysical prospecting is carried out largely in continental crustal rocks, for which the Königsberger ratio is much less than unity (i.e., Qn >>1) and the magnetization may be assumed to be induced by the present geomagnetic field. The magnetization contrast is then due to susceptibility contrast in the crustal rocks. If k represents the susceptibility of an orebody, k0 the susceptibility of the host rocks and F the strength of the inducing magnetic field, Eq. 1 allows us to write the magnetization contrast as

$$\Delta M = (k - k_0) * F$$

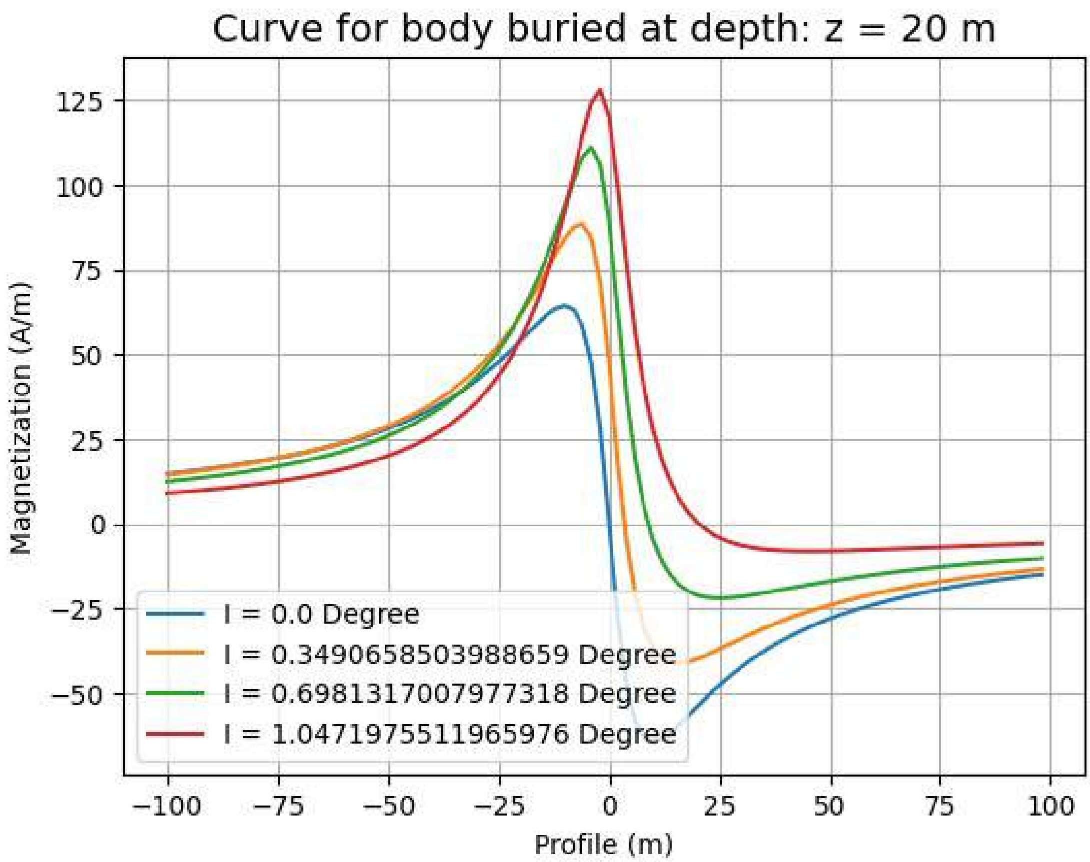
Formula Used:

```
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt

class VerticalFault:
    def __init__(self):
        self.x = np.arange(-100, 100, 2)
        self.I = np.array([0, 20, 40, 60])
        self.theta = np.radians(self.I)
        self.Z1 = [10, 20, 30]
        self.Z2 = 5
        self.CF = 100

    def calculate(self, n):
        jj = []
        for j in range(len(self.theta)):
            A = np.arctan((self.x ) / self.Z1[n]) - np.arctan((self.x)
/ self.Z2)
```

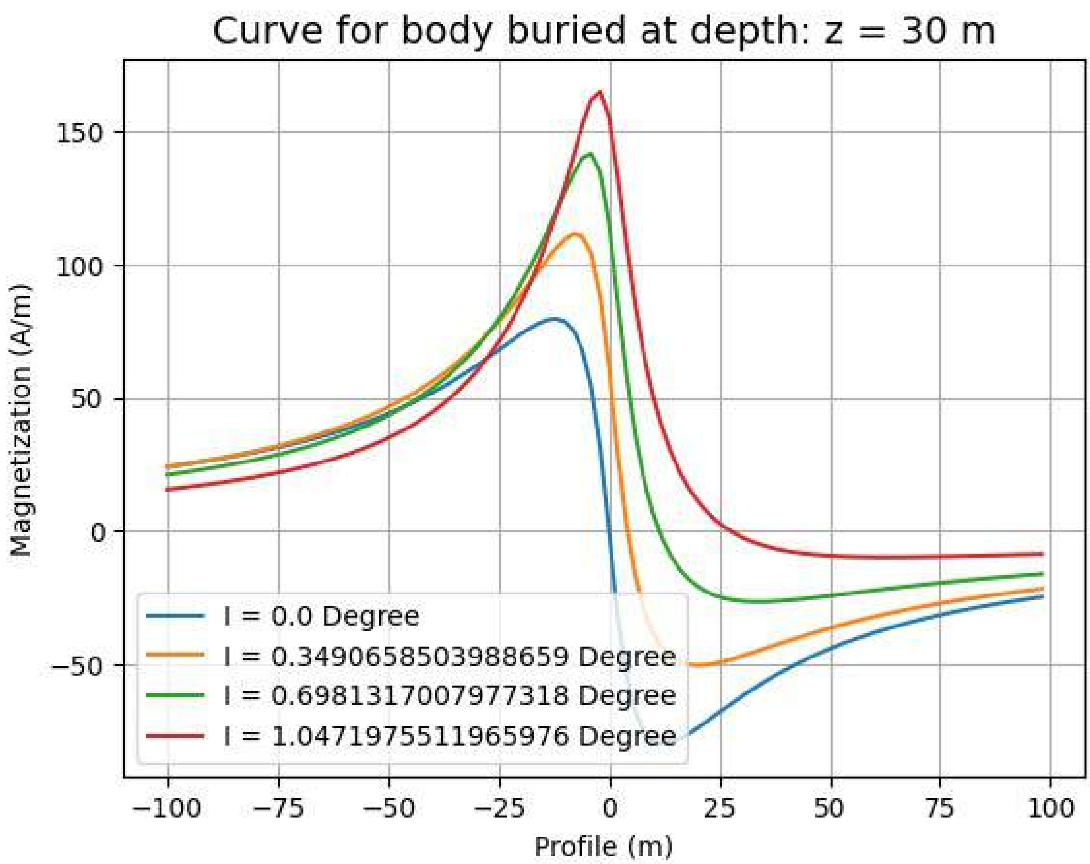
```
            B = 0.5*np.log(((self.x)**2 + self.Z1[n]**2) / ((self.x)**2 + self.Z2**2))
            F = self.CF * (A * np.cos(self.theta[j]) + B * np.sin(self.theta[j]))
            jj.append(F)
        return jj

    def plot(self, n):
        result = self.calculate(n)
        for i in range(4):
            plt.plot(self.x, result[i], label=f'I = {self.theta[i]} Degree')
        plt.legend()
        plt.xlabel('Profile (m)')
        plt.ylabel('Magnetization (A/m)')
        plt.title(f'Curve for body buried at depth: z = {self.Z1[n]} m', fontsize=14)
        plt.grid()

fault=VerticalFault()
fault.plot(0)
```



```python
fault.plot(1)
```




```python
fault.plot(2)
```



INTERPRETATION:

From the plots, it can be observed that as the depth of the bottom of the inclined fault is increased the magnitudes of the positive peaks and negative peaks of the curves also increases. But the increment of the values for the negative peaks are more than the positive peaks. As the direction of arbitrary magnetization angle increases, the positive peaks of the curves for the same plot decrease and the negative peaks increase. Also the width of the curve increases with increase of the depth of the bottom of the inclined fault.