B132584

Interpretation of Electromagnetic Survey over Potential Ore Body

Abstract

This investigation used an electromagnetic survey to acquire data about an ore body in central Sweden. The strong negative anomaly indicates the presence of such a body. Two cross-sections of the real and imaginary data are used to estimate the dip of the body, while the maximum imaginary and real values are used to determine the conductivity of and depth to the body. The dip, conductivity and depth to the body are 70 ± 15 degrees, 1.9 ± 0.4 mholms and 10 meters respectively.

Introduction

Electromagnetic surveys are frequently used to find ore bodies as they are cheap and easy to conduct. While the data tends to be extremely imprecise, it provides enough information to indicate the presence or absence of an ore body and its rough location. This survey will provide estimates of the dip, conductivity, and depth to the ore body, allowing miners to begin to dig down to the body.

Method

This type of electromagnetic survey requires two investigators to walk a fixed distance apart wearing magnetic coils, taking readings at the frequency required by the survey. The magnetic field generated by one coil induces a magnetic field in the ore body and this field is measured by the second coil.

The data is then analysed to estimate the dip, depth, and conductivity of the ore body. The dip can be estimated by comparing the positive anomalies in the data on either side of the ore body. Once the dip is roughly known, the maximum of the real and imaginary data can be plotted to determine the depth and conductivity of the ore body.

Results

A heatmap and contour maps show the imaginary data below:

The first hand-drawn contour map shows the central minimum and the maxima on either side, while the second shows contours of equal value. The hand-drawn and computer generated contour show the same thing; the only difference is that the computer generated contour is neater and probably more accurate.

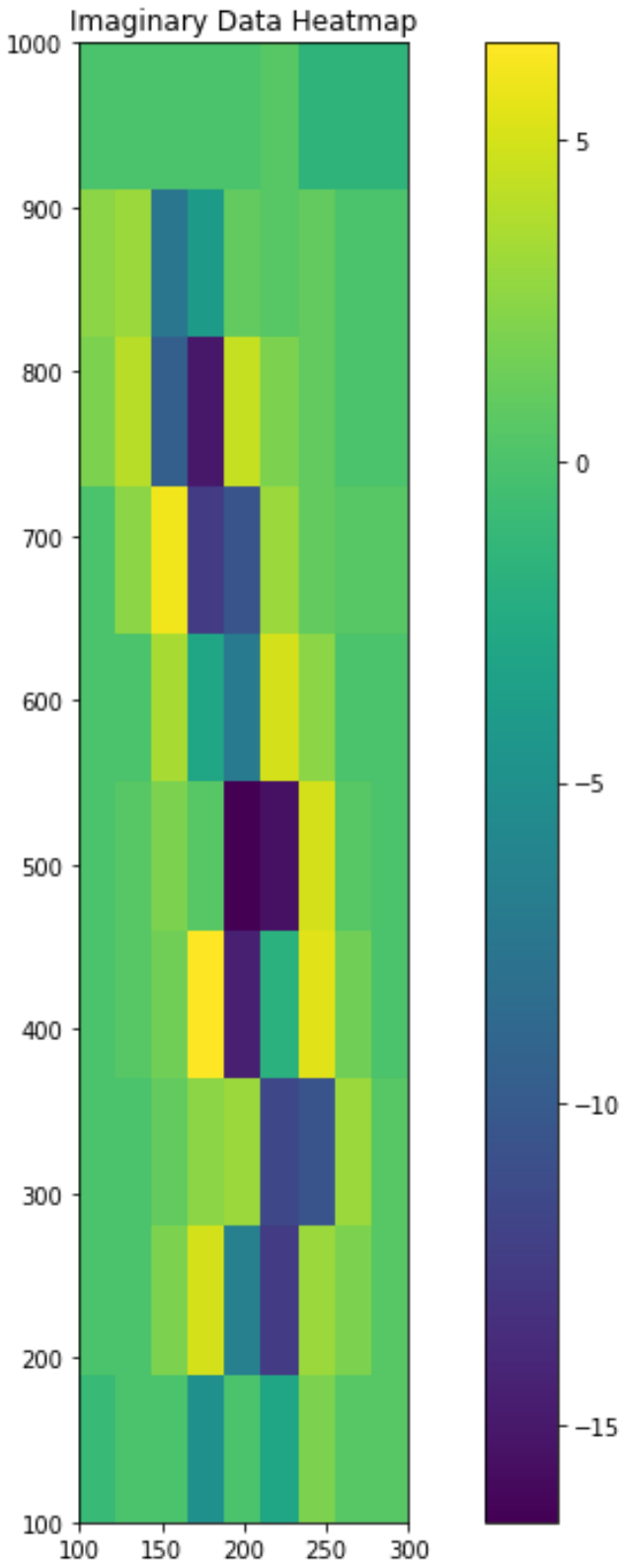
Imaginary Data Heatmap
1000
900
800
700
600
500
400
300
200
100
100
150
200
250
300
5
0
−5
−10
−15

Fig. 1

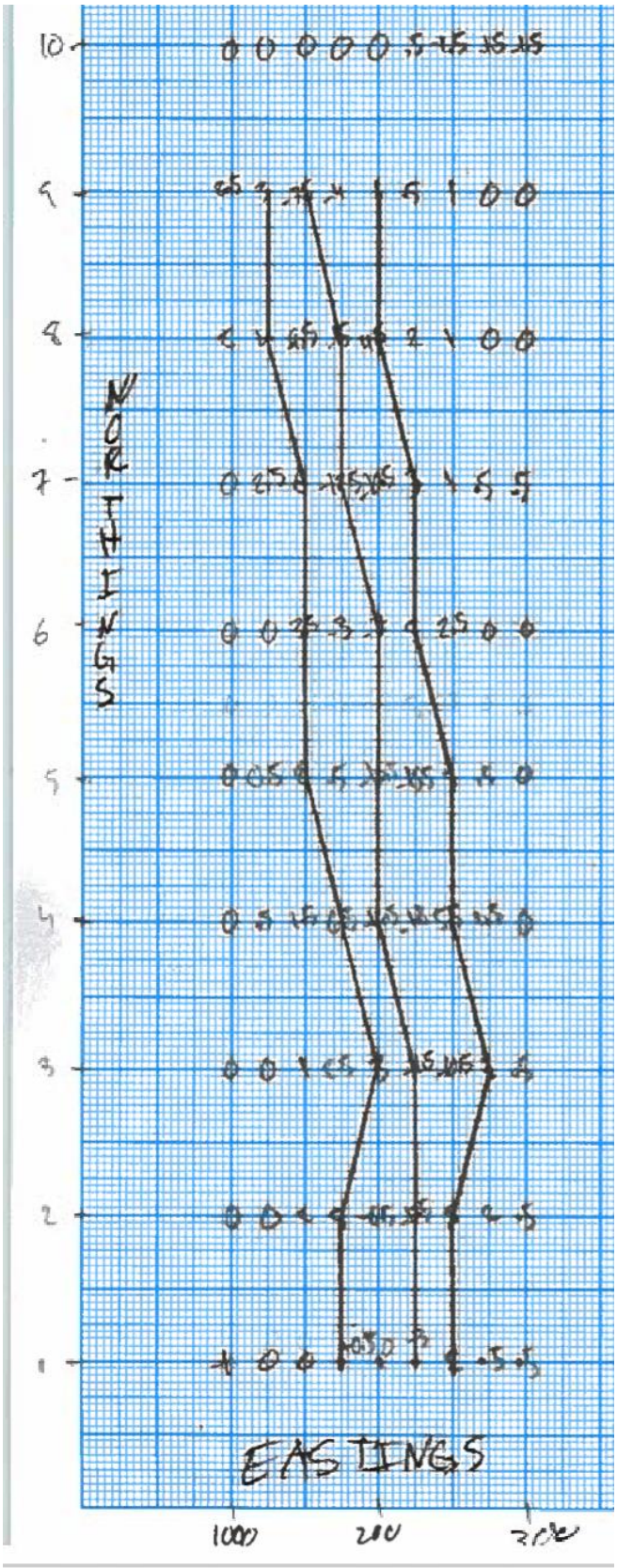

Fig. 2

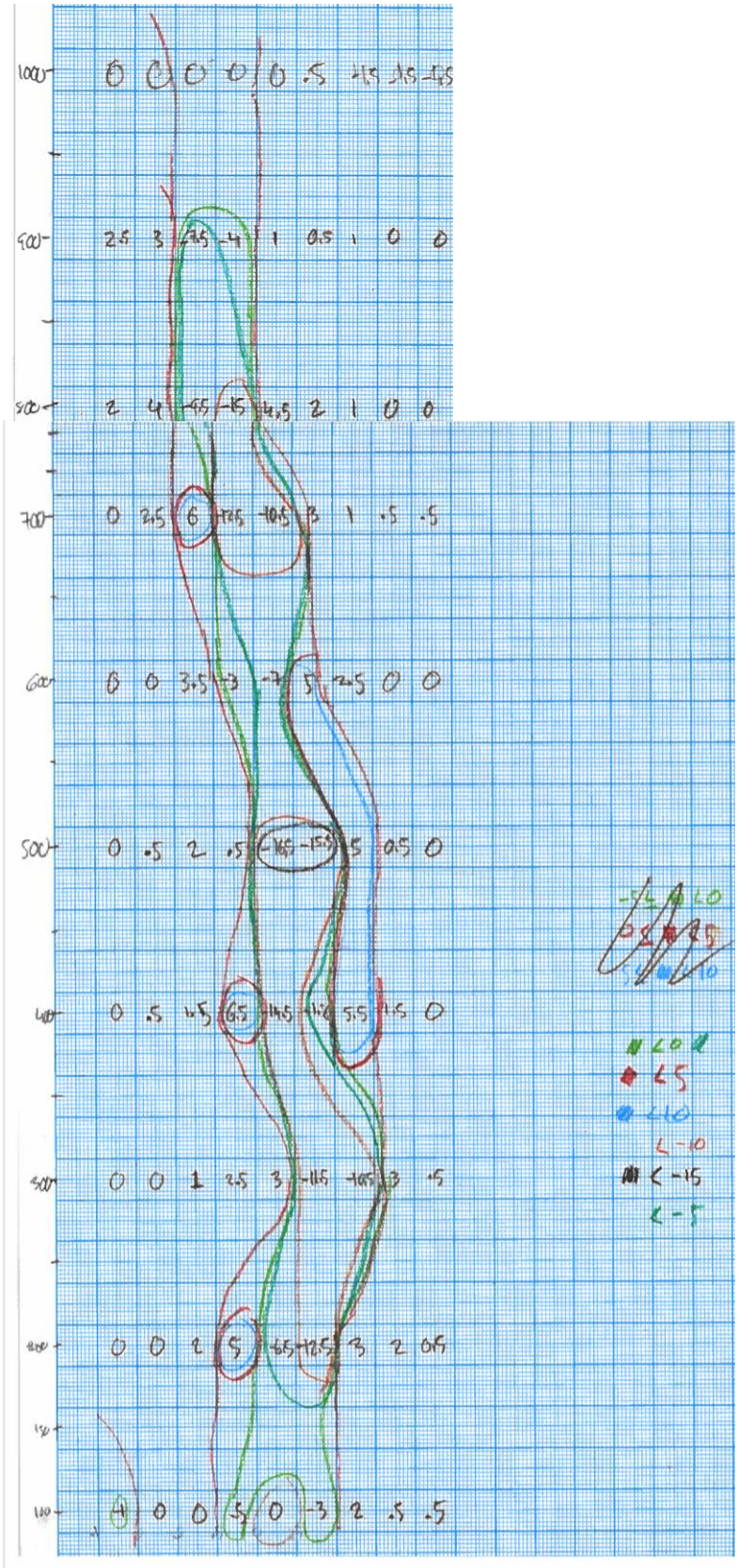

Fig. 3

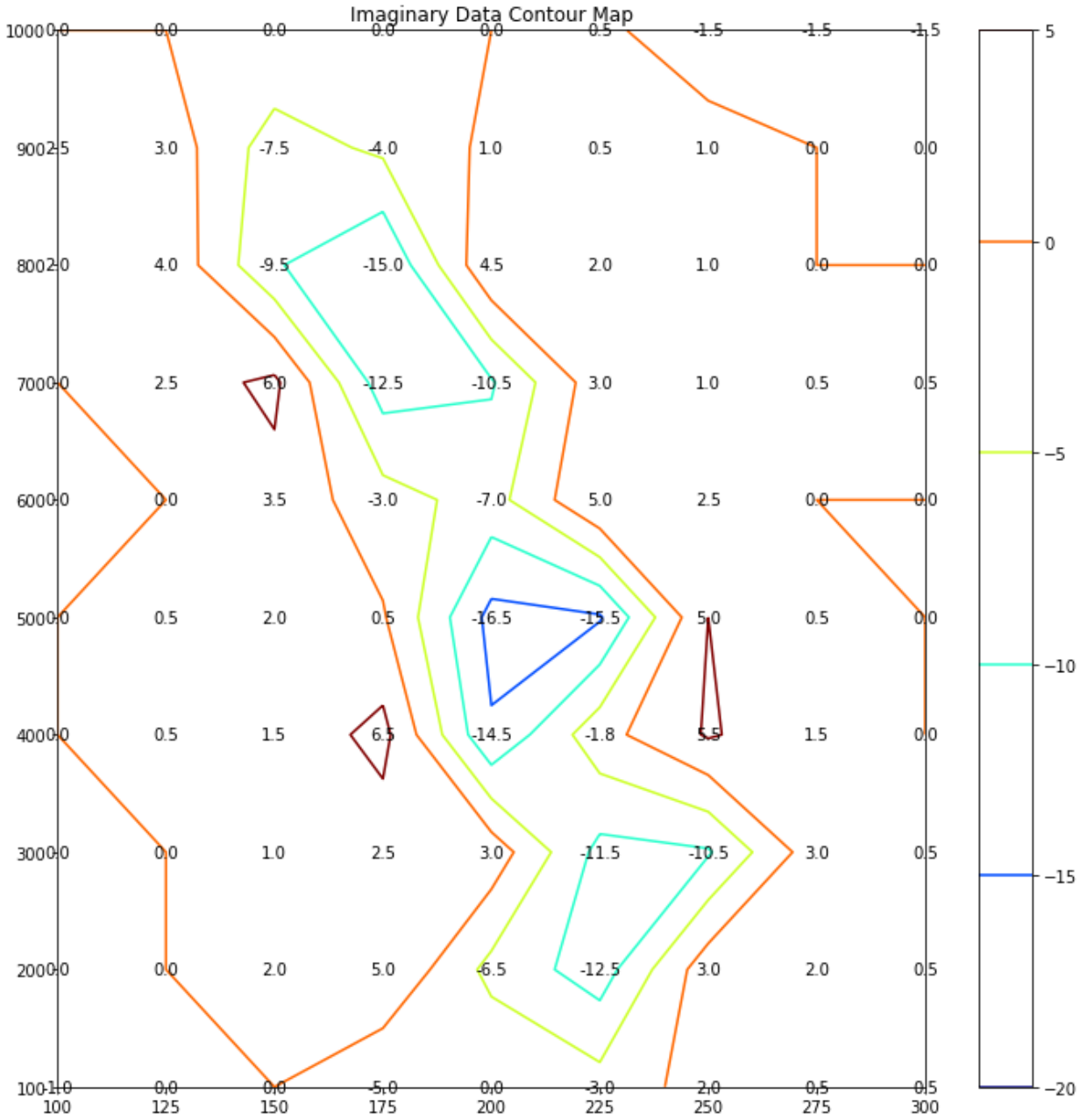

Fig. 4

Cross-sections of the data show the positive anomalies on either side of the ore body that are used to estimate the dip of the body. Both raw and symmetrized data are presented for comparison. The symmetrized data was created by taking the average between each real and imaginary point, and plotting positive and negative versions of the resulting curve. Symmetrical data is used because it makes calculations easier, and the data is so imprecise that altering it does not have a huge impact on the final results.

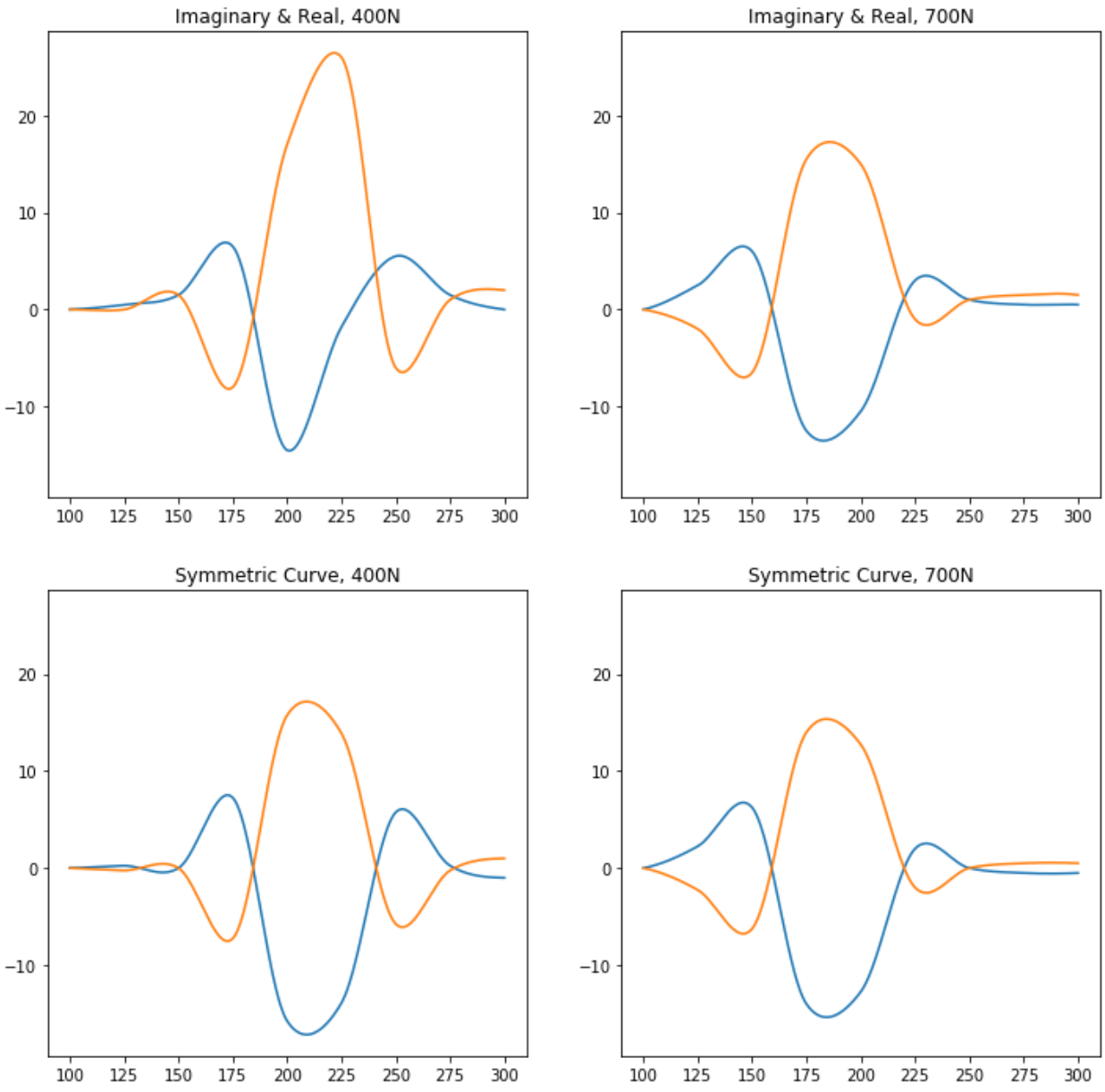

Fig. 5

Estimating errors with any accuracy is difficult given a limited knowledge of this type of survey or the accuracy of the equipment used. Here, the quoted errors are estimated based simply on the order of magnitude of the data- none of the data exceeds a value of one hundred, so the error for all data points is estimated to have a value of ten, and this is propagated throughout all calculations.

While this is an imprecise method, the data is already understood to be so imprecise that accurately estimating errors is not crucial.

The area between the curves for the raw and symmetric 400N and 700N data were as follows:

400N, raw data, left curve: 300 $\pm$ 10

400N, raw data, right curve: 250 $\pm$ 10

700N, raw data, left curve: 370 $\pm$ 10

700N, raw data, right curve: 90 $\pm$ 10

400N, symmetric data, left curve: 300 $\pm$ 10

400N, symmetric data, right curve: 250 $\pm$ 10

700N, symmetric data, left curve: 370 $\pm$ 10

700N, symmetric data, right curve: 90 $\pm$ 10

The ratios between the right and left curve are as follows:

400N, raw data: 0.8 $\pm$ 0.1

700N, raw data: 0.2 $\pm$ 0.1

400N, symmetric data: 0.8 $\pm$ 0.1

700N, symmetric data: 0.2 $\pm$ 0.1

Here, proper error propagation would yield a result of 1 $\pm$ 1 for 400N and 0 $\pm$ 1 for 700N; however, this suggests that dip of the ore body ranges from 0 – 90 degrees. Since this is useless information, I fudged the data a bit to make it more helpful. The numbers above result from the calculations made, they are just quoted to more accuracy than they should be.

The Python code used to perform the above analysis is included in an appendix.

Consulting the graph of ratio vs dip, the dip is 70 $\pm$ 15°.

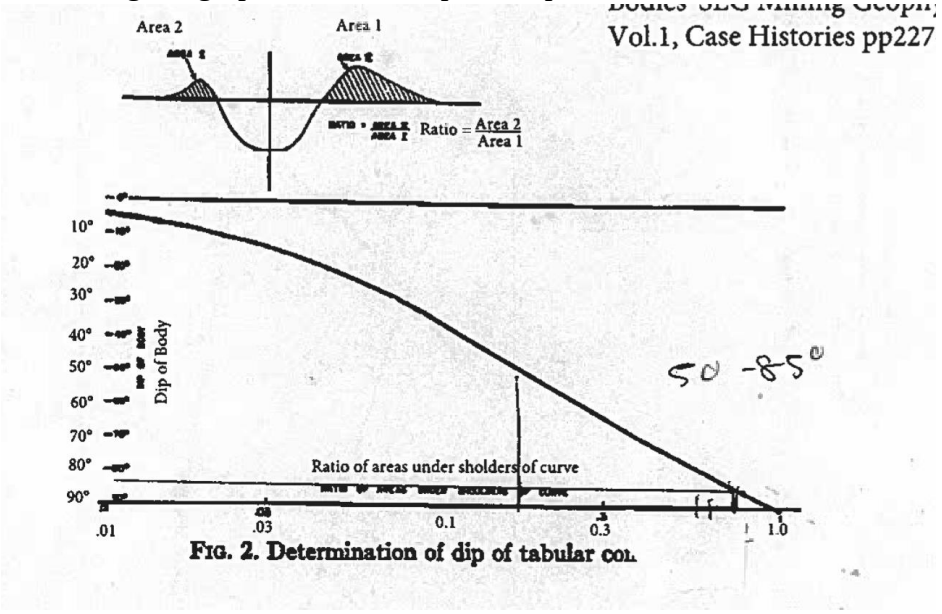

FIG. 2. Determination of dip of tabular COL

Fig. 6

Plotting the real maximum (-26) and the imaginary maximum (-16.5) on the graph for 60 degree dip, the depth is determined to be

$0.2L=0.2*50m=9 \pm 1m$,

and $\alpha=20 \pm 5$.

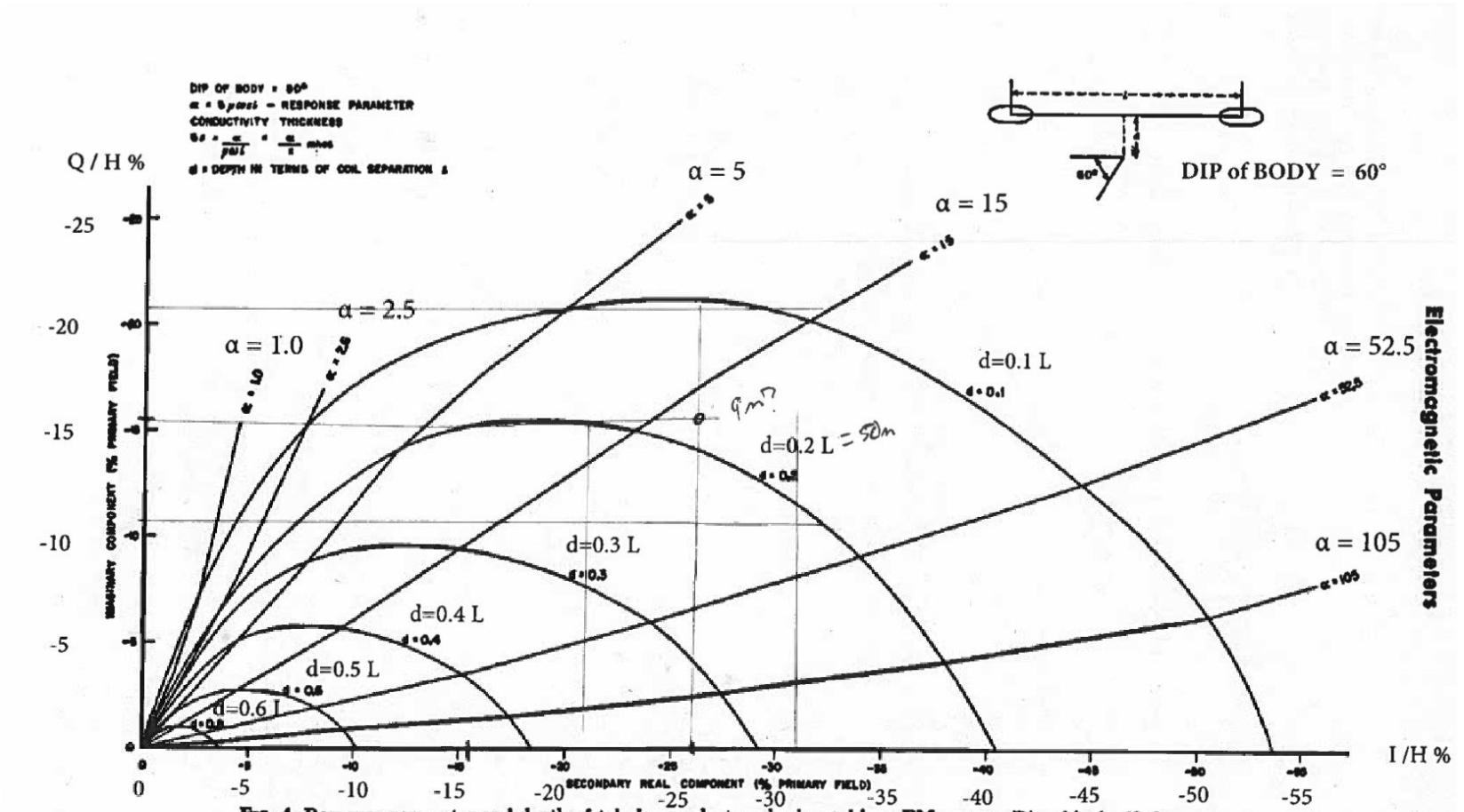

FIG. 4. Response parameter and depth of tabular conductors, horizontal loop EM system. Dip of body 60 degrees.

Fig. 7

Given

Permeability of free space $\mu = 4\pi \times 10^{-7}$ Henry/m

Angular frequency $\omega = 3520\pi$ radians/second

Conductor thickness $s = 55 \pm 5$ m

Coil separation l=50m

And the equation $\alpha = \sigma\mu\omega sl$,

Conductivity $\sigma = \omega\mu sl/\alpha$

$= (4\pi \times 10^{-7}$ Henry/m)(3520π radians/second)(55m)(50m)/20 $\pm$ 5

$=1.9 \pm 0.4$ mholm

Conclusion

In conclusion, the depth to the ore body is 9 ± 1m, the dip is $70 \pm 15^{\circ}$, and the conductivity is 1.9 ± 0.4 mholm.

Discussion

This investigation raised the issue of how to pragmatically analyse low-quality data. In this analysis, the real and imaginary data were altered to be symmetrical, but presented alongside the raw data so that their differences could be analysed and commented upon. Making the data symmetrical in this way makes calculations easier. However, in this case, there was virtually no difference in the results produced by each data set, especially with this investigation's low levels of accuracy, meaning that altering the data in this way is not sacrilegious, but a practical solution, so long as it is acknowledged.

This investigation also raised the issue how to estimate errors with very little information about the survey or equipment used. Errors are expected to be quoted in any scientific report, but when the basis for their estimation is nothing more than a wild guess, one wonders how useful they really are. Basing their estimation on the order of magnitude of the data and the degree of accuracy to which it is quoted seems as good an estimation method as any, but it is a guess nonetheless. Just as errors indicate the degree of confidence in the result, quoting errors indicates a level of confidence in the accuracy of the error. When there is no basis for the quoted error, perhaps an error should not be quoted at all.

This type of electromagnetic survey is often the first used in any geophysical investigation because it is cheap and easy, not because it is remarkably accurate. It is therefore needless to say that more work could be done in this area, such as a relative gravity survey, which could detect the higher density of the ore body, or a seismic survey, which could detect any changes in seismic wave velocity associated with the body. These surveys would more

precisely locate the body, and require any potential miners to dig fewer boreholes in order to find it.

References

"Centripetal Catmull-Rom spline." Wikipedia, available at: https://en.wikipedia.org/wiki/Centripetal_Catmull–Rom_spline

Williams, W (2020) "Electromagnetic Survey Interpretation." University of Edinburgh.

Appendix

This is a copy of the Python code used to create the graphs and perform some of the calculations in this report.

```
#Import relevant libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from shapely.geometry import LineString, Polygon

import bisect

import itertools

#Load in data from Excel spreadsheet

data=pd.read_excel('Documents/Geomag Data.xlsx',header=None)

data.columns=np.arange(100,301,25);
data.index=[1000,900,800,700,600,500,400,300,200,100]


#Create heatmap

plt.figure(figsize=(12,12))

plt.imshow(data,extent=[100,300,100,1000],cmap='viridis')

plt.colorbar()

plt.title('Imaginary Data Heatmap')

#Create contour map

plt.figure(figsize=(12,12))

plt.contour(np.arange(100,301,25),[1000,900,800,700,600,500,400,300,200,100],data,levels=[-20,-15,-10,-5,0,5],cmap='jet')

#Display data points

for i in data.index:

    for j in data.columns:


            plt.text(j,i,str(data.loc[i,j]),fontsize=10,horizontalalignment='center',verticalalignment='center')
```

```
plt.colorbar()
plt.title('Imaginary Data Contour Map')
#The following code was stolen from the Wikipedia entry on Catmull-Rom splines- this is the spline used by Excel to create
#smooth curves
import numpy
import pylab as plt

def CatmullRomSpline(P0, P1, P2, P3, nPoints=100):
    """
    P0, P1, P2, and P3 should be (x,y) point pairs that define the Catmull-Rom spline.
    nPoints is the number of points to include in this curve segment.
    """
    # Convert the points to numpy so that we can do array multiplication
    P0, P1, P2, P3 = map(numpy.array, [P0, P1, P2, P3])

    # Calculate t0 to t4
    alpha = 0.5
    def tj(ti, Pi, Pj):
        xi, yi = Pi
        xj, yj = Pj
        return (((xj-xi)**2 + (yj-yi)**2)**0.5)**alpha + ti

    t0 = 0
    t1 = tj(t0, P0, P1)
    t2 = tj(t1, P1, P2)
    t3 = tj(t2, P2, P3)

    # Only calculate points between P1 and P2
```

```
    t = numpy.linspace(t1, t2, nPoints)

    # Reshape so that we can multiply by the points P0 to P3
    # and get a point for each value of t.
    t = t.reshape(len(t), 1)
    #print(t)
    A1 = (t1-t)/(t1-t0)*P0 + (t-t0)/(t1-t0)*P1
    A2 = (t2-t)/(t2-t1)*P1 + (t-t1)/(t2-t1)*P2
    A3 = (t3-t)/(t3-t2)*P2 + (t-t2)/(t3-t2)*P3
    #print(A1)
    #print(A2)
    #print(A3)
    B1 = (t2-t)/(t2-t0)*A1 + (t-t0)/(t2-t0)*A2
    B2 = (t3-t)/(t3-t1)*A2 + (t-t1)/(t3-t1)*A3

    C = (t2-t)/(t2-t1)*B1 + (t-t1)/(t2-t1)*B2
    return C

def CatmullRomChain(P):
    """
    Calculate Catmull–Rom for a chain of points and return the combined curve.
    """
    sz = len(P)

    # The curve C will contain an array of (x, y) points.
    C = []
    for i in range(sz-3):
        c = CatmullRomSpline(P[i], P[i+1], P[i+2], P[i+3])
```

```
        C.extend(c)


    return C
#Store the data for each cross-section in variables, adding zeros on either end so that all data is included in the spline
im_400=np.transpose(np.array([np.arange(75,326,25),[0,0,.5,1.5,6.5,-14.5,-1.8,5.5,1.5,0,0]]))
im_700=np.transpose(np.array([np.arange(75,326,25),[0,0,2.5,6,-12.5,-10.5,3,1,.5,.5,0]]))
re_400=np.transpose(np.array([np.arange(75,326,25),[0,0,0,1.5,-8,17,26,-6,1,2,0]]))
re_700=np.transpose(np.array([np.arange(75,326,25),[0,0,-2,-6.5,15.5,15,-1,1,1.5,1.5,0]]))
#Calculate the average of the imaginary and real data for 400N and 700N
med_400=np.mean([[0,0,.5,1.5,6.5,-14.5,-1.8,5.5,1.5,0,0],-1*np.array([0,0,0,1.5,-8,17,26,-6,1,2,0])],axis=0)
med_700=np.mean([[0,0,2.5,6,-12.5,-10.5,3,1,.5,.5,0],-1*np.array([0,0,0,1.5,-8,17,26,-6,1,2,0])],axis=0)
#Store these data coupled with their eastings
med_400=np.transpose(np.array([np.arange(75,326,25),med_400]))
med_700=np.transpose(np.array([np.arange(75,326,25),med_700]))
#Create subplots of the raw and symmetric data for 400N and 700N
plt.figure(figsize=(12,12))


ax0=plt.subplot(221)
x0,y0=zip(*CatmullRomChain(im_400))
plt.plot(x0,y0)
x1,y1=zip(*CatmullRomChain(re_400))
plt.plot(x1,y1)
plt.title('Imaginary & Real, 400N')


ax1=plt.subplot(222,sharey=ax0)
x2,y2=zip(*CatmullRomChain(im_700))
```

```
plt.plot(x2,y2)
x3,y3=zip(*CatmullRomChain(re_700))
plt.plot(x3,y3)
plt.title('Imaginary & Real, 700N')

ax2=plt.subplot(223,sharey=ax0)
x4,y4=zip(*CatmullRomChain(med_400))
plt.plot(x4,y4)
x5,y5=x4,-1*np.array(y4)
plt.plot(x5,y5)
plt.title('Symmetric Curve, 400N')

ax3=plt.subplot(224,sharey=ax0)
x6,y6=zip(*CatmullRomChain(med_700))
plt.plot(x6,y6)
x7,y7=x6,-1*np.array(y6)
plt.plot(x7,y7)
plt.title('Symmetric Curve, 700N')
#Create LineString objects out of each of the plotted curves
line0=LineString(np.transpose(np.array([x0,y0])))
line1=LineString(np.transpose(np.array([x1,y1])))
line2=LineString(np.transpose(np.array([x2,y2])))
line3=LineString(np.transpose(np.array([x3,y3])))
line4=LineString(np.transpose(np.array([x4,y4])))
line5=LineString(np.transpose(np.array([x5,y5])))
line6=LineString(np.transpose(np.array([x6,y6])))
line7=LineString(np.transpose(np.array([x7,y7])))
#Print the coordinates of the intersection points for each pair of curves
```

```
print('400 Raw: '+line0.intersection(line1).wkt)

print('700 Raw: '+line2.intersection(line3).wkt)

print('400 Symmetric: '+line4.intersection(line5).wkt)

print('700 Symmetric: '+line6.intersection(line7).wkt)

#Create Polygon objects out of the lobes on either side of the center and print their areas

#Left, 400N

area0=[[tuple(a) for a in
np.transpose(np.array([x0,y0]))[bisect.bisect(x0,150):bisect.bisect(x0,184.4563151813285)
]],

[tuple(a) for a in
np.transpose(np.array([x1,y1]))[bisect.bisect(x1,150):bisect.bisect(x1,184.4563151813285)
]][::-1]]

print(Polygon(list(itertools.chain.from_iterable(area0))).area)


#Right

area1=[[tuple(a) for a in
np.transpose(np.array([x0,y0]))[bisect.bisect(x0,240.9030252990246):bisect.bisect(x0,277.
0160857720188)]],

[tuple(a) for a in
np.transpose(np.array([x1,y1]))[bisect.bisect(x1,240.9030252990246):bisect.bisect(x1,277.
0160857720188)]][::-1]]

print(Polygon(list(itertools.chain.from_iterable(area1))).area)

#Left, 700N

area0=[[tuple(a) for a in
np.transpose(np.array([x2,y2]))[bisect.bisect(x2,100):bisect.bisect(x2,159.0456135311209
)]],

[tuple(a) for a in
np.transpose(np.array([x3,y3]))[bisect.bisect(x3,100):bisect.bisect(x3,159.0456135311209
)]][::-1]]

print(Polygon(list(itertools.chain.from_iterable(area0))).area)


#Right
```

```
area2=[[tuple(a) for a in
np.transpose(np.array([x2,y2]))[bisect.bisect(x2,219.9970696911267):bisect.bisect(x2,250)
]],

[tuple(a) for a in
np.transpose(np.array([x3,y3]))[bisect.bisect(x3,219.9970696911267):bisect.bisect(x3,250)
]][::-1]]

print(Polygon(list(itertools.chain.from_iterable(area2))).area)

#Left, symmetric 400

area0=[[tuple(a) for a in
np.transpose(np.array([x4,y4]))[bisect.bisect(x4,150):bisect.bisect(x4,184.3572642578207)
]],

[tuple(a) for a in
np.transpose(np.array([x5,y5]))[bisect.bisect(x5,150):bisect.bisect(x5,184.3572642578207)
]][::-1]]

print(Polygon(list(itertools.chain.from_iterable(area0))).area)


#Right, symmetric 400

area2=[[tuple(a) for a in
np.transpose(np.array([x4,y4]))[bisect.bisect(x4,241.0123773960759):bisect.bisect(x4,277.
0110880200208)]],

[tuple(a) for a in
np.transpose(np.array([x5,y5]))[bisect.bisect(x5,241.0123773960759):bisect.bisect(x5,277.
0110880200208)]][::-1]]

print(Polygon(list(itertools.chain.from_iterable(area2))).area)

#Left, symmetric 700

area0=[[tuple(a) for a in
np.transpose(np.array([x6,y6]))[bisect.bisect(x6,100):bisect.bisect(x6,159.0416762498833)
]],

[tuple(a) for a in
np.transpose(np.array([x7,y7]))[bisect.bisect(x7,100):bisect.bisect(x7,159.0416762498833)
]][::-1]]

print(Polygon(list(itertools.chain.from_iterable(area0))).area)


#Right, symmetric 700
```

```
area2=[[tuple(a) for a in
np.transpose(np.array([x6,y6]))[bisect.bisect(x6,219.9995205523607):bisect.bisect(x6,250)
]],

[tuple(a) for a in
np.transpose(np.array([x7,y7]))[bisect.bisect(x7,219.9995205523607):bisect.bisect(x7,250)
]][::-1]]

print(Polygon(list(itertools.chain.from_iterable(area2))).area)

#Calculate the ratio of the areas for each pair

print('Raw 400N: '+str(245.1942909907392/298.8464544583992))

print('Raw 700N: '+str(89.38146424598777/365.6164637264152))

print('Symmetric 400N: '+str(247.4153169491708/298.2723486788354))

print('Symmetric 700N: '+str(89.41925434963633/365.65498925546376))
```