Sobhit Soni
21411033

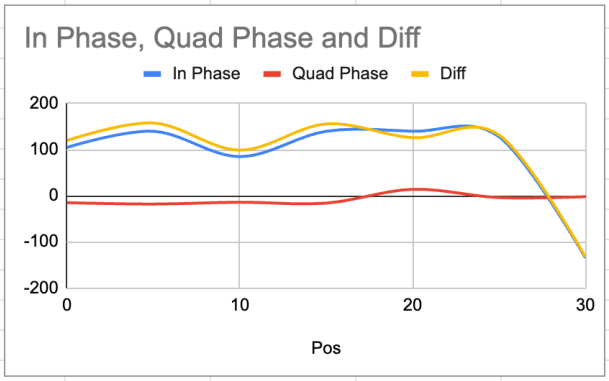
VLF (Very Low Frequency) Method

Aim: To detect conductive body beneath the subsurface using VLF method

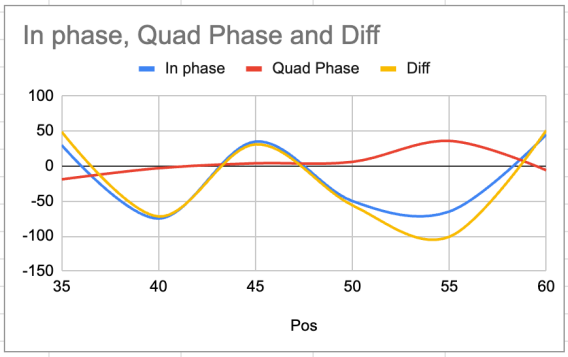
Data Acquisition:

1) Data was acquired in the lawn of Earth Science Department, IIT Roorkee

Pos	In Phase	Quad Phase	Diff
0	105	-15	120
5	140	-18	158
10	85	-14	99
15	140	-16	156
20	140	14	126
25	128	-4	132
30	-135	-2	-133



Pos	In phase	Quad Phase	Diff
35	30	-19	49
40	-75	-3	-72
45	35	4	31
50	-50	6	-56
55	-65	36	-101
60	45	-6	51



2) Amount of data is very less and the Initial Plot of In-phase and Quad Phase Component also looks noisy at first sight.

Methods/Procedure:

Here are the corrected versions of the sentences:

1) QuadPhase and Inphase data were multiplied by the primary magnetic field to obtain the secondary magnetic field.

2) This secondary magnetic field was processed using the Phasor Filter function, and the resultant data were linearly interpolated to obtain more valid points of InPhase and QuadPhase data.

$$F(n) = (H_{n-2} + H_{n-1}) - (H_{n+1} + H_{n+2}),$$

where H_n is the measured data (real or imaginary part of the magnetic transfer function) at nth station, $F(n)$ is Fraser filtered data obtained at the center of H_{n-2} and H_{n+2}.

3) The resultant InPhase and QuadPhase data were inserted into the matrix according to the survey lines

4) In an area of 30x30m we interpolate using an algorithm where a value at a point is predicted using the average of its neighboring non-null value neighbors.

5) We started with the null value having the highest number of non-null neighbors and predicted the value for it, and then updated the neighbor count of all the neighbors of the newly updated value and so on.

6) After obtaining all the values in the grid, a 3-D plot was created.

7) 3 cross-sections of this 3-D plot were studied to draw out the following inferences represented.

8) Color gradient map was plotted for both QuadPhase and InPhase.

Code:

```
from numpy import linspace as np_linspace
def interpolate_1D(data_1d_matrix, res_mult):

   result = []

   for i in range(len(data_1d_matrix)-1):
       result += list(np_linspace(data_1d_matrix[i], data_1d_matrix[i+1],
res_mult, False))

   result += [data_1d_matrix[-1]]

   return result


def interpolate_2D(data_2d_matrix, missingPointValue = None):

   neighbours = [[0 for x in y] for y in data_2d_matrix]
   interpolate_order = [[] for x in range(8)]

   restart = True

   for y in range(len(data_2d_matrix)):

       for x in range(len(data_2d_matrix[y])):

           if data_2d_matrix[y][x] is not None:

               for i in range(max(0, y-1), min(y+2, len(data_2d_matrix))):

                   for j in range(max(x-1, 0), min(x+2,
len(data_2d_matrix[i]))):

                       if data_2d_matrix[i][j] is None:

                           # update count of neighbour's neighbours
                           neighbours[i][j] += 1
```

```
    for y in range(len(data_2d_matrix)):

        for x in range(len(data_2d_matrix[y])):

            if neighbours[y][x] > 0:

                interpolate_order[neighbours[y][x]-1] += [(y, x)]

    while(restart):

        restart = False

        for y in range(7, 0, -1):

            for x in range(len(interpolate_order[y])):

                if
data_2d_matrix[interpolate_order[y][x][0]][interpolate_order[y][x][1]] is
None:
                    # print(y, interpolate_order[y][x])
                    sum = 0
                    c = 0
                    for i in range(max(0, interpolate_order[y][x][0]-1),
min(interpolate_order[y][x][0]+2, len(data_2d_matrix))):

                        for j in range(max(0,
interpolate_order[y][x][1]-1), min(interpolate_order[y][x][1]+2,
len(data_2d_matrix[i]))):

                            if data_2d_matrix[i][j] is not None:

                                # update its value
                                sum += data_2d_matrix[i][j]
                                c += 1

                            elif i != interpolate_order[y][x][0] or j !=
interpolate_order[y][x][1]:

                                # update count of neighbour's neighbours
                                neighbours[i][j] += 1
```

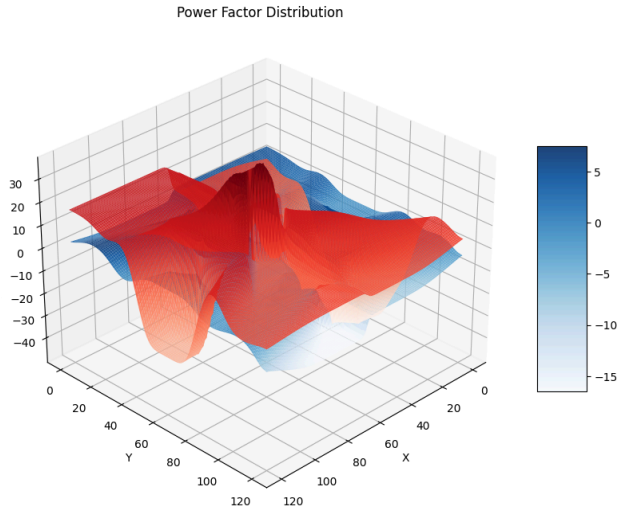
```
                                interpolate_order[neighbours[i][j]-1] +=
[(i, j)]

                                if (neighbours[i][j]-1 > y):
                                    restart = True


data_2d_matrix[interpolate_order[y][x][0]][interpolate_order[y][x][1]] =
sum/c


                    if restart:
                        break

            interpolate_order[y] = interpolate_order[y][x+1:]

            if restart:
                break

    return data_2d_matrix
# data_2d_matrix = [
    # [None, 10, None,    5, None],
    # [None,  5, None,  2.5, None],
    # [  50, 15,    1,  -15,  -50],
    # [None,  5, None,  2.5, None],
    # [None, 20, None,   10, None]
# ]

# matrix_print(interpolate_2D(data_2d_matrix, None))
from numpy import array as np_array
from numpy import meshgrid as np_meshgrid
from numpy import arange as np_arange

import matplotlib.pyplot as plotter
from mpl_toolkits.mplot3d import Axes3D




def matrix_print(matrix):
```

```
   for x in matrix:
       for y in x:
           # print(f'{y:8g}', end = ' ')
           print(y, end = ' ')
       print()

   print()


# grid = [
   # [None, None, None, None, None, [-5, +5], None, [101, 15], None, None,
None, None, None],
   # [None, None, None, None, None, None, None, None, None, None, None,
None, None],
   # [None, None, None, None, None, [-15, -13], None, [45, 13], None,
None, None, None, None],
   # [None, None, None, None, None, None, None, None, None, None, None,
None, None],
   # [None, None, None, None, None, [10, -28], None, [-12, -12], None,
None, None, None, None],
   # [None, None, None, None, None, None, None, None, None, None, None,
None, None],
   # [[-150, 28], None, [15, 32], None, [103, 14], [-25, -20], [120, 12],
[-13, -26], [-5, 12], None, [-65, 22], None, [20, 8]],
   # [None, None, None, None, None, None, None, None, None, None, None,
None, None],
   # [None, None, None, None, None, [-75, 8], None, [-10, -20], None,
None, None, None, None],
   # [None, None, None, None, None, None, None, None, None, None, None,
None, None],
   # [None, None, None, None, None, [-105, -2], None, [-6, -23], None,
None, None, None, None],
   # [None, None, None, None, None, None, None, None, None, None, None,
None, None],
   # [None, None, None, None, None, [-105, 25], None, [-16, 32], None,
None, None, None, None]
# ]

survey_data = [[], [], []]
```

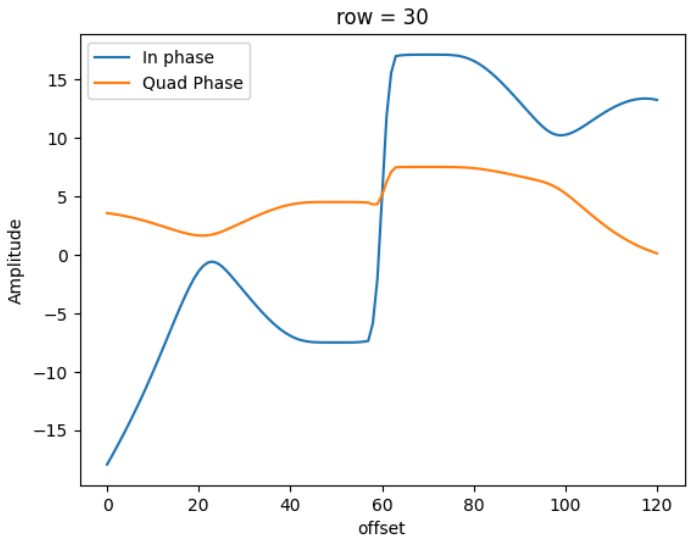
```
survey_data[0] = [
   (-5, 5),
   (-15, -13),
   (10, -28),
   (-25, -20),
   (-75, 8),
   (-105, -2),
   (-105, 25)
]

survey_data[1] = [
   (101, 15),
   (45, 13),
   (-12, -12),
   (-13, -26),
   (-10, -20),
   (-6, -23),
   (-16, 32)
]

survey_data[2] = [
   (-150, 28),
   (15, 32),
   (103, 14),
   (120, 12),
   (-5, 12),
   (-65, 22),
   (20, 8)
]


Res_mult = 20 # must be even
Hp = 1


Re_Hs = [[Hp * x[0] for x in y] for y in survey_data]
Im_Hs = [[Hp * x[1] for x in y] for y in survey_data]

for i in range(len(Re_Hs)):
```

```
    Re_Hs[i] = interpolate_1D(Re_Hs[i], Res_mult)

for i in range(len(Im_Hs)):

    Im_Hs[i] = interpolate_1D(Im_Hs[i], Res_mult)


PF_In = [
    [
        (y[x-2] + y[x-1]) - (y[x+1] + y[x+2])
        for x in range(2, len(y)-2)
    ]
    for y in Re_Hs
]

PF_Qd = [
    [
        (y[x-2] + y[x-1]) - (y[x+1] + y[x+2])
        for x in range(2, len(y)-2)
    ]
    for y in Im_Hs
]


PF_In_grid = [[ None for x in range((len(survey_data[2])-1)*Res_mult+1)]
for y in range((len(survey_data[0])-1)*Res_mult+1)]
PF_Qd_grid = [[ None for x in range((len(survey_data[2])-1)*Res_mult+1)]
for y in range((len(survey_data[0])-1)*Res_mult+1)]


for i in range(2):
    for j in range(2, len(PF_In_grid)-2):
        PF_In_grid[j][int(2.5*Res_mult)+i*Res_mult] = PF_In[i][j-2]

for j in range(2, len(PF_In_grid[0])-2):
    PF_In_grid[3*Res_mult][j] = PF_In[2][j-2]

for i in range(2):
    for j in range(2, len(PF_Qd_grid)-2):
```

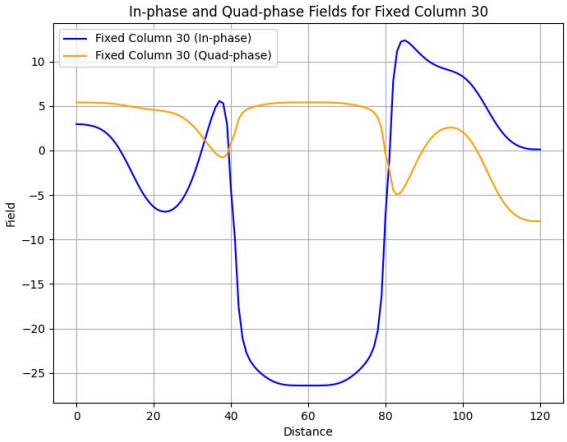
```
       PF_Qd_grid[j][int(2.5*Res_mult)+i*Res_mult] = PF_Qd[i][j-2]

for j in range(2, len(PF_Qd_grid[0])-2):
   PF_Qd_grid[3*Res_mult][j] = PF_Qd[2][j-2]


PF_In_grid = interpolate_2D(PF_In_grid, None)
PF_Qd_grid = interpolate_2D(PF_Qd_grid, None)


data_Qd = np_array(PF_Qd_grid)
data_In = np_array(PF_In_grid)

length = data_Qd.shape[0]
width = data_Qd.shape[1]
x, y = np_meshgrid(np_arange(length), np_arange(width))

fig = plotter.figure(figsize=(10, 8))
ax = fig.add_subplot(1,1,1, projection='3d')

surf_Qd = ax.plot_surface(x, y, data_Qd, cmap='Blues', alpha=0.9,
rstride=1, cstride=1)

surf_In = ax.plot_surface(x, y, data_In, cmap='Reds', alpha=0.9,
rstride=1, cstride=1)

fig.colorbar(surf_Qd, ax=ax, shrink=0.5, aspect=5)

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Power Factor')

ax.set_title('Power Factor Distribution')

ax.view_init(elev=30, azim=45)

plotter.show()
```



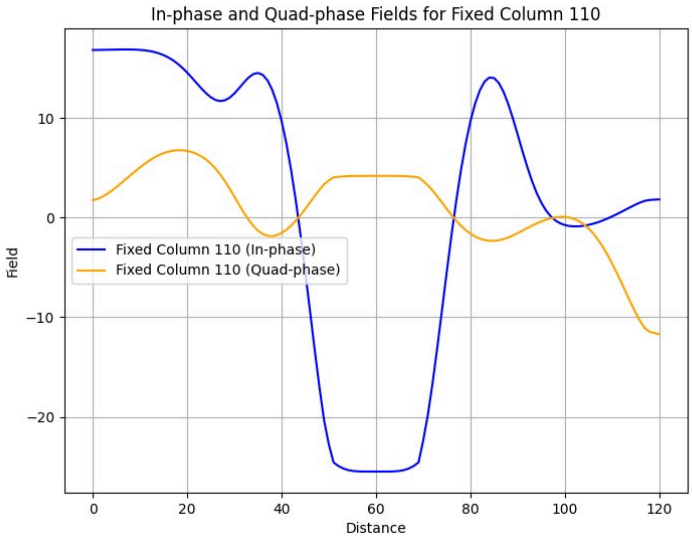

```
 row = 30

#for row in range(0,10,121):
 MagIn = []
 offsetIn = []
 MagQuad = []
 offsetQuad = []
 val = 0
 for j in range(len(PF_In_grid[row])):
   MagIn.append(PF_In_grid[row][j])
   offsetIn.append(val)
   val+=1
 val = 0
 for j in range(len(PF_Qd_grid[row])):
   MagQuad.append(PF_Qd_grid[row][j])
   offsetQuad.append(val)
   val+=1

 plotter.plot(offsetIn,MagIn,label = "In phase")
 plotter.plot(offsetQuad,MagQuad,label = "Quad Phase")
 plotter.xlabel("offset")
 plotter.ylabel("Amplitude")
 plotter.title(f"row = {row}")
 plotter.legend()
```



(Fig 1)



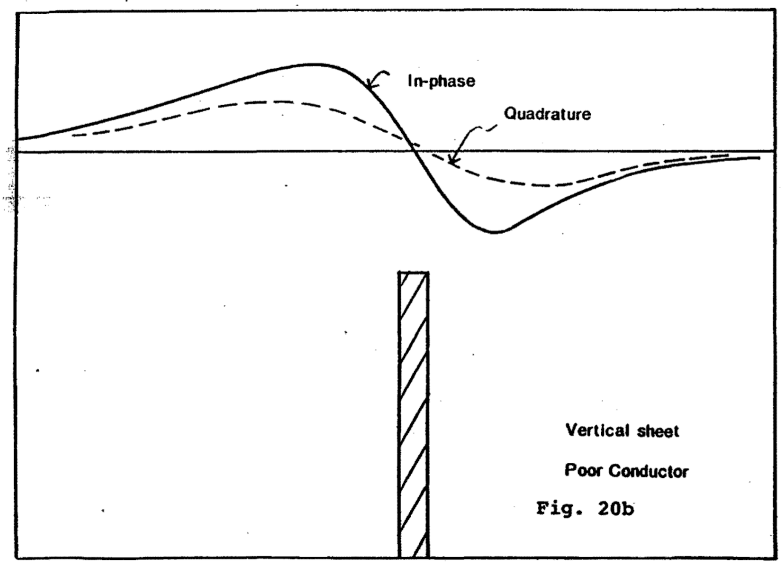
(Fig 2)



(Fig 3)

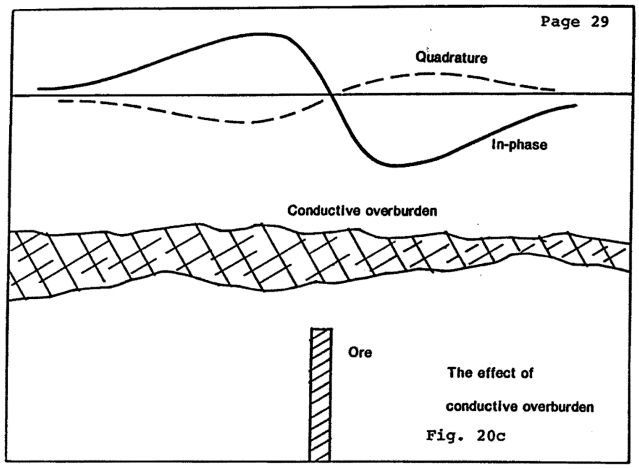
Interpretations:

1) We can see that the InPhase and QuadPhase components intersect at a point. (Fig 1).
2) Approximately they are in-phase with lower amplitude of QuadPhase component for a majority of offset



(Fig 1.1)

3) In fig 2 we can see a graph that can be interpreted as:



(Fig 2.1)

(**Result**)

1) From the above diagram fig -1.1 (Page 29 research paper over VLF) we can infer that there might be a vertical sheet of Poor conductor present. (horizontal survey profile in y-axis on the surface only).
2) From the fig 2.1 ((Page 29 research paper over VLF) we can conclude that there might be an ore present beneath the surface with a conductive overburden.(for fig 2 and 3).In some of the horizontal structures and it can be confirmed as the graph pattern is repeated for two horizontal survey profile (x-axis)

Note:- VLF data is very noisy and interpolation was done to approximate the subsurface conductivities they may not be accurate and VLF provides relative data and being the starting point of field survey it needs to be followed up by other survey methods to get the better results.

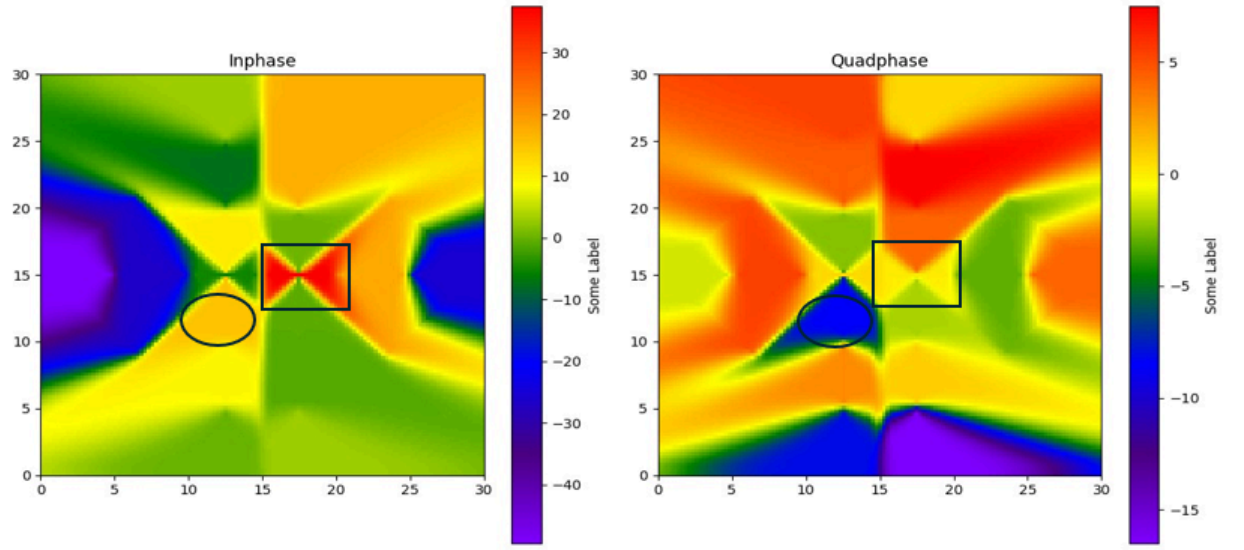
The interpretations made and confirmed by the Fraser-Filtered 2D color map: -

Fraser Filter Profiles

The data processing technique commonly referred to as the Fraser Filter was applied to the raw data. This filter transforms In-Phase cross overs and inflections into positive peaks, while Quadrature responses are negative to positive giving a negative peak anomaly when the Fraser Filter is applied.

Fraser-Filtered Plan Maps In Phase and Quadrature Phase

We have used the Fraser-filtered profiled data and produces contoured results on a plan map. Positive peaks in the In-Phase Component are shown as orange and negative peaks in the Quadrature component are shown on the plan maps as blue to dark blue. The intensity of the response is measured on the scale bar to the right of the plan maps.



Box Zone: - The High peaks (or Redshift) in In phase map and Low peaks (or Greenshift) in the quad phase mark the inflection or crossover of the conductive anomalous zone surrounded by a relatively High resistive body.

Ellipse zone:- The High peaks (or Redshift) in In phase map and Low peaks (or Blueshift) in the quad phase mark the inflection or crossover of the conductive anomalous zone surrounded by a relatively High resistive body.

Contributions:

- **Sobhit Soni** - Fraser Filter equations, methodology to get to the final In-phase and Quad Phase points, Data Interpretation of color graded In Phase and Out Phase map.
- **Arkanil Paul** - Data Interpolation using Algorithm (averaging out the neighboring values to get the value of unknown point). Writing code to plot color graded In Phase and Quad Phase map, and 3-D plot.
- **Swayam Pal** - Data Interpretation of In-Phase and Quad Phase components using models given in research paper, Write code to find 2-D cross section of 3-D Plot.