MA203 : Numerical Methods Project Report

Mathematically modeling the charge distribution on the surface of a spacecraft in outer space orbits.

The Mathonauts

Jiya Desai (22110107)
Kavya Gotecha (22110115)
Kishan Ved (22110122)
Shrinivas Kulkarni (22110126)
Nishi Shah (22110171)

Table of Contents

1 Acknowledgement

We extend our heartfelt gratitude to Professor Soumyabrata Chakrabarty for providing the problem statement and invaluable guidance that shaped our group project. His expertise and continuous support were instrumental in defining the project's direction and scope.

We are equally grateful to Professor Dilip Srinivas Sundaram and Professor Akshaa Vatwani, our course instructors, for their constant mentorship during this project. Their constructive feedback and guidance ensured that our project met the course's academic standards.

Their combined expertise, encouragement, and dedication significantly influenced our ability to navigate project challenges effectively. We deeply appreciate their investment in our academic growth.

2 Problem statement

Understanding the interaction of a spacecraft with the outer space environment forms a crucial part of its functioning. Various phenomena including solar radiation, plasma interaction, dielectric charging, etc, cause a spacecraft to acquire a net charge, which is non-uniformly distributed. This charge distribution poses a risk to the functioning of the spacecraft. Unevenly distributed charge on a spacecraft's surface can potentially be harmful for several reasons including:

- **Electrostatic forces**: The uneven charge can cause electrostatic forces to act which can attract or repel nearby charged particles in space. This can affect the spacecraft's orientation and trajectory, making it unstable.
- **Surface charging**: Charged particles in outer space can lead to localized regions of surface charging. These can create electrical discharges such as Corona discharge or arcing, which can damage sensitive spacecraft components.
- **Radio frequency interference**: Uneven charge distribution can also result in radio frequency interference. This interference can harm spacecraft communication systems.
- **Thermal effects**: Unevenly charged surfaces may heat or cool down unevenly, affecting the temperature distribution. This can impact efficiency and lead to overheating or freezing of critical components.

It is therefore necessary to study and quantify the charge distribution on the surface of the spacecraft in a mathematical way.

The goal of this project is to mathematically model the charge distribution on the surface of the spacecraft in outer space orbits. For the sake of simplicity as well as relevance to current applications, we have chosen a spacecraft modeled on CUBESAT to study the charge distribution profile of its body. We are considering the body of the spacecraft to be a symmetric metallic cube, with a specified edge length. This helps us to focus more on the methodology of arriving at our result by simplifying the object under consideration.

The electric potential on the spacecraft body has only small local variations, so it is generally considered constant. This potential is considered as 1V (this value does not matter as we want to find the relative charge density at different points). Since the body of the spacecraft is equipotential at all times, and we need to devise equations for finding the potential at every point in terms of the unknown charge density ρ. Equations for finding the charge density are made using the method of moments. Following this, we will get 'N' equations in 'N' distinct variables, which can then be solved using numerical methods like Gaussian elimination, Gauss-Jordan algorithm, TDMA or others of this kind, depending on the equations obtained after performing the above-mentioned steps.

3 Method proposed

3.1 Method of Moments - Discretization

The method of moments is a numerical method of estimating an unknown function, f, given an equation of the form:

$$L(f) = g \tag{i}$$

- L is an linear operator (like summation, integration etc.).
- g is a known function, called the "source" or the "excitation" function.
- f is the function with the unknown parameter.

This method involves expanding the function f into a series of functions in the domain of L, f_1, f_2, f_3,... as:

$$f = \sum_{i=1}^{n} (a_i \cdot f_i) \tag{ii}$$

where a_1, a_2, ... are unknown constants, and f_1, f_2, ... are known basis functions.

For an exact solution, the equation is summed to infinity, but for approximate solutions (i.e., numerical approximations), it is summed up to a finite number of functions. It is evident that the greater the number of basis functions, the finer will be the approximation.

Now, using the linear property of the operator L, we can substitute equation (ii) in (i) as:

$$\sum_{i=1}^{n} (a_i \cdot L(f_i)) = g \tag{iii}$$

Depending on the problem the method is being applied to, we can assume a suitable inner product to exist between a and $L(f)$, $< a, L(f) >$, which results in g. Now, we define a set of weighting functions, or testing functions, $w1$, $w2$, $w3$,... such that we can take the inner product of equation (iii) with these functions, correspondingly, as:

$$\sum_{n} \alpha_n < w_m, Lf_n > = < w_m, g > \tag{iv}$$

m=1, 2, 3,....This set of equations can be written in the matrix form as:

$$[l_{mn}] = \begin{bmatrix} < w_1, Lf_1 > & < w_1, Lf_2 > & ... \\ < w_2, Lf_1 > & < w_2, Lf_2 > & ... \\ ... & ... & ... \end{bmatrix} \tag{v}$$

$$[\alpha_n] = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ . \\ . \\ . \end{bmatrix} ; [g_m] = \begin{bmatrix} < w_1, g > \\ < w_2, g > \\ < w_3, g > \\ . \\ . \\ . \end{bmatrix} \tag{vi}$$

If the matrix $[l]$ is non-singular, its inverse, $[l]^{-1}$ exists. The α_n are then given by

$$[\alpha_n] = [l_{nm}]^{-1}[g_m] \tag{vii}$$

and the solution for f is given by equation (ii). For the concise expression of this result, we define the matrix of functions

$$[\bar{f}_n] = \begin{bmatrix} f_1 & f_2 & f_3 & ... \end{bmatrix} \tag{viii}$$

$$(vii), (viii) \implies f = [\bar{f}_n][\alpha_n] = [\bar{f}_n][l^{-1}_{nm}][g_m] \tag{ix}$$

3.2 Gauss-Jordan method

Gauss Jordan method is an algorithm used to solve a set of linear equations. It is quite similar to Gaussian elimination except we convert the matrix into a reduced row echelon form instead of row echelon form as in the Gaussian method.

1. The process begins by first expressing the system as a matrix, and then reducing it to an equivalent system by simple row operations.
2. The matrix that represents the system is called the augmented matrix.
3. The arithmetic manipulation that is used to move from a system to a reduced equivalent system is called a row operation. These elementary row operations include:
   - Row Swapping : We can exchange any two rows.
   - Scalar Multiplication: Multiply any row with a constant.
   - Row Sum: Add a multiple of one row to another row.
4. Once a system is expressed as an augmented matrix, the Gauss-Jordan method reduces the system into a series of equivalent systems by using the row operations.
5. This row reduction continues until the system is expressed in what is called the reduced row echelon form. The reduced row echelon form of the coefficient matrix has 1's along the main diagonal and zeros elsewhere.
6. Now we can very clearly see our solutions in the augmented matrix.

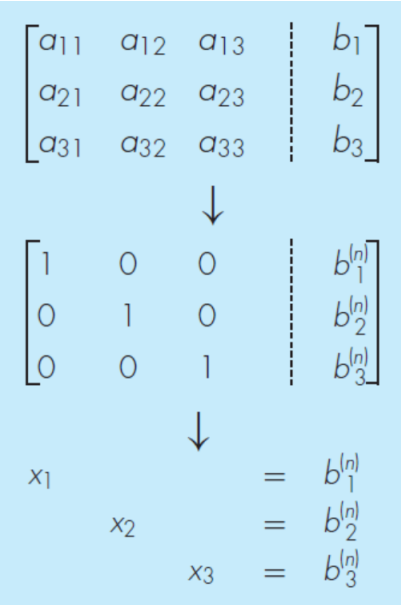

Figure 1: *Gauss-Jordan elimination*

4 Approach and solution (numerical)

4.1 Implementation of Discretisation

In our analysis of a CubeSat model, we aim to calculate and visualise the surface charge density on each of its surfaces. To achieve this, we break down the problem into several steps.

First, we focus on a single surface, dividing it into smaller segments organized in an m*n grid. Each of these segments will have its own electric potential, resulting from both the contributions of other segments and its self-potential. We calculate the potential at each segment by considering the influence of all other segments and then sum this with its self-potential. This process is repeated for every segment on the surface, allowing us to form an equation where the surface charge density of the segment is unknown while its potential is known.

Now, we have the equation for potential of each segment. Using that we need to find the charge density of each segment.

Here, we use the method of moments to solve for the surface charge density. The general form of equation is given as

$$g = L(f)$$

The equation that we use here is given as:

$$\phi = \frac{1}{4\pi\epsilon} \int \frac{\sigma \rho ds}{|r - r'|} \tag{x}$$

where, the operator L corresponds to integration, the known function g is potential, while the unknown function f is a function of ρ, which is the charge density.

The above equation is for expressing the electric potential generated on one segment because of the charge on another segment. But for calculating the self-electric potential of a segment, we use the following expression:

$$V(self) = \alpha \frac{\sqrt{cd}}{\pi\epsilon_0} \ln(1 + \sqrt{2}) \tag{xi}$$

where α is the charge density of the segment (assumed to be uniform for the segment), and c and d are the side lengths of the segment.
Now, for expressing the total electric potential of a single surface only due to its own charge distribution, we use the following equations, which involve the basis functions and the weighting functions as explained above.

$$V[m] = \sum_{n=1}^{M} N(\alpha[n] \cdot \frac{\Delta S}{\sqrt{(x[m] - x[n])^2 + (y[m] - y[n])^2}} \tag{xii}$$

$$L[m][n] = \frac{\Delta S}{\sqrt{(x[m] - x[n])^2 + (y[m] - y[n])^2}} \tag{xiii}$$

$$(xii), (xiii) \implies V[m] = \sum_{n=1}^{M} N(a[n] \cdot L[m][n]) \tag{xiv}$$

After solving these equations using a Python program, we ultimately get a system of m*n equations in n variables (where m*n is the number of segments we divided the plane surface into), where the variables correspond to the unknown charge density of each of the segments and the equations represent the electric potential for each segment (which is known).

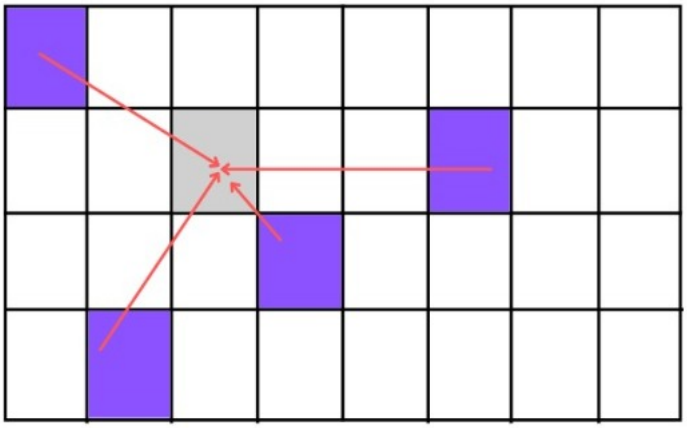

Figure 2: *How the potential on each segment is affected by its own charge distribution and the charge distribution on all other segments*

Expanding our analysis to two surfaces, we compute the potential on one surface in relation to the opposite surface (parallel plates). This will give us a matrix (termed as L12 in the code) consisting of the values of charge densities on each segment of the two parallel plates. It is to be noted that L12 and L21 will be the same, since we are considering two symmetrically parallel identical plates. Also, L11 is the notation for the charge density matrix of plate 1 due to itself.

Similarly, we then calculate the potential on each surface with respect to the surfaces perpendicular to it (perpendicular plates). This comprehensive approach ensures that we account for all interactions between surfaces.

Finally, by repeating these calculations for all six surfaces and considering their interactions, we arrive at the overall electric potential for the entire CubeSat model. We can then visualise the obtained charge distribution on each surface in both two and three dimensions.

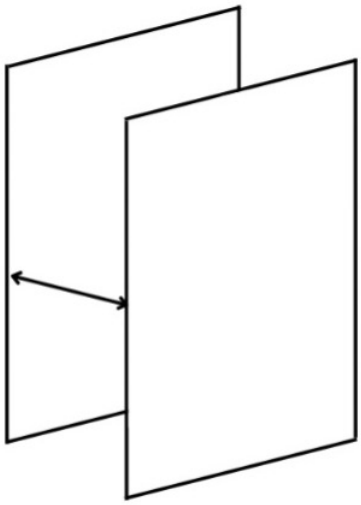

Figure 3: *Electrostatic interaction between two parallel, identical surfaces*

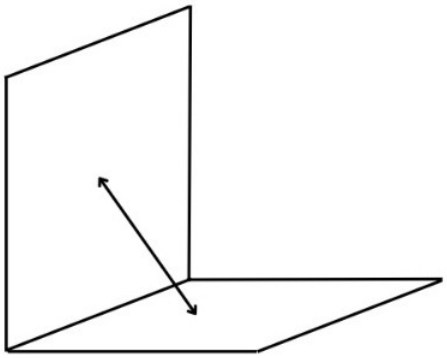

Figure 4: *Electrostatic interaction between two perpendicular surfaces with a common edge*

4.2 Implementation of Gauss-Jordan method

The i^{th} row of the L matrix has elements corresponding to functions for finding potential of segments in the i^{th} row of the plate. For 2 parallel plates or 2 perpendicular plates, this L matrix contains 4 parts in it:

$$\mathrm{L} = \begin{bmatrix} [l^{11}] & [l^{12}] \\ [l^{21}] & [l^{22}] \end{bmatrix}$$

- $[l^{11}]$ represents the charge densities on all segments of plate 1 under the influence of plate 1 itself.
- $[l^{22}]$ represents the charge densities on all segments of plate 2 under the influence of plate 2 itself.
- $[l^{12}] = [l^{21}]$ where $[l^{12}]$ represents the charge densities on all segments of plate 2 under

the influence of plate 1, and $[l^{21}]$ represents the charge densities on all segments of plate 1 under the influence of plate 2.

For a single isolated plate, the L matrix becomes $[l^{11}]$ itself.

This L matrix is multiplied with a vector α containing unknown elements that correspond to the charge density of every segment in a single plate (or in both the plates, depending upon the case). This gives the potential vector g(known) that contains values for potential of every segment.

$$L.\alpha = g$$

These system of equations are solved using Gauss-Jordan Elimination. This method is used as the obtained matrix was not a tridiagonal matrix, and even though this is $O(n^3)$, this works reasonably well to compute the unknown vector α.

5 Code with explanation

The most relevant code blocks and their explanation is given in this report. Code for plotting and basic imports are eliminated. The complete code is available here.

Defining the plate dimensions and number of segments.
Defining the vector g (named as potential_vector).

```
n = 50  # horizontal segments
m = 50  # vertical segments
length = 15  # length of plate
breadth = 15  # breadth of plate
area_of_segment = (length / n) * (breadth / m)
potential_vector = np.ones(n * m)  # vector g
```

5.1 Consider a single isolated equipotential surface.

We find the charge distribution on a segment, by considering the effect of all other charged segments and also its self-potential.

```
L = [] # The final L matrix
k = 9000000000
# size of segments
c = length/n
d = length/m

for i in range(m*n):
    l=[]
    for j in range(n*m):
        xi=(i//n)+1
        yi=i%n
        xj=(j//m)+1
        yj=j%m
        if xi!=xj or yi!=yj:
            distance=((xi-xj)**2 + (yi-yj)**2)**(1/2)
        l.append(k*area_of_segment/distance)
        if xi==xj and yi==yj: # self potential
            l.append(k*np.log(1+np.sqrt(2))*4*np.sqrt(c*d))
    L.append(l)
solutions = np.linalg.solve(L, potential_vector)
```

5.2 Gauss-Jordan Elimination

This is a recursive process used to solve the equations obtained.

```
def gauss_jordan_elimination(matrix, n):
    for i in range(n):
        if(matrix[i][i]==0):
            for j in range(i+1,n):
                if(matrix[j][i]!=0):
                    matrix[i][i],matrix[j][i] =
                       matrix[j][i],matrix[i][i]
                    break
            else:
                print("Infinite solutions")
                return "Infinite solutions"
    return gauss_elimination(matrix,n,0)


def gauss_elimination(matrix, n, r):
    if(r==n):
        return backward_elimination(matrix, n, n-1)

    p = matrix[r][r]
    for i in range(r,n+1):
        matrix[r][i]/=p

    for i in range(r+1,n):
        p = matrix[i][r]
        for j in range(r,n+1):
            matrix[i][j] -= matrix[r][j]*p

    return gauss_elimination(matrix, n, r+1)

def backward_elimination(matrix, n, r):
    if(r==-1):
        return [row[-1] for row in matrix]

    for i in range(r-1,-1,-1):
        p = matrix[i][r]
        for j in range(r,n+1):
            matrix[i][j] -= matrix[r][j]*p
    return backward_elimination(matrix, n, r-1)
```

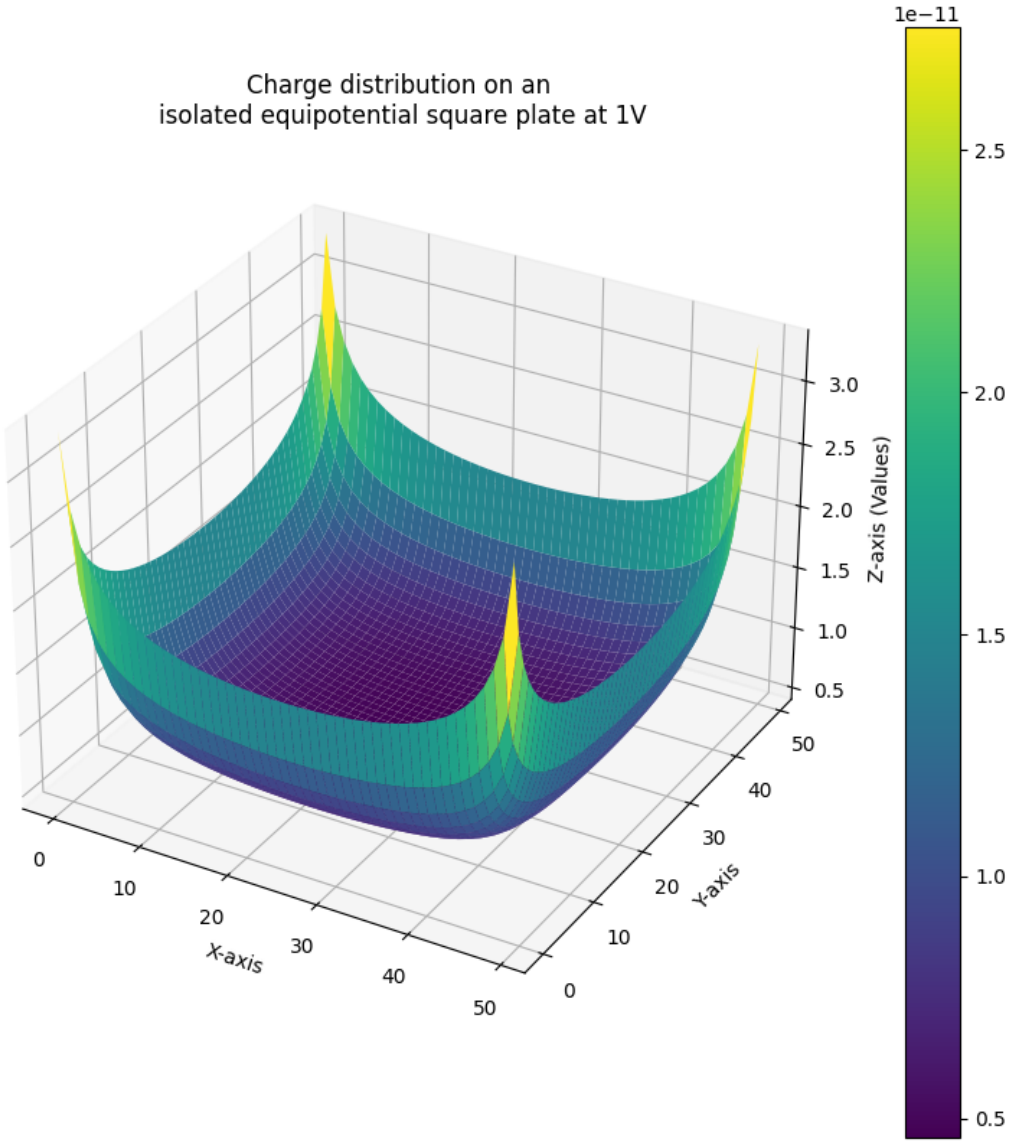

Figure 5: *Output of the above code blocks*

5.3 Considering 2 parallel equipotential surfaces.

Calculating the L matrix:

Finding L11 and L12:

```
import numpy as np
L = []
k = 9000000000
c = length/n
d = length/m
z2 = 15
z1 = 0
eps = 8.854187817*(10**-12)

for i in range(m*n):
  l=[]
  for j in range(n*m): # Finding L11
        xi=(i//n)+1
        yi=i%n
        xj=(j//m)+1
        yj=j%m
        if xi!=xj or yi!=yj:
          distance=((xi-xj)**2 + (yi-yj)**2)**(1/2)
          l.append(k*area_of_segment/distance)
        if xi==xj and yi==yj:
          l.append(k*np.log(1+np.sqrt(2))*4*np.sqrt(c*d))
  for j in range(n*m): # Finding L12
        xi=(i//n)+1
        yi=i%n
        xj=(j//m)+1
        yj=j%m
        if xi!=xj or yi!=yj:
          distance=((xi-xj)**2 + (yi-yj)**2 +
             (z2-z1)**2)**(1/2)
          l.append(k*area_of_segment/distance)
        if xi==xj and yi==yj:
          l.append((0.282*c/eps)*(np.sqrt(1+(np.pi*(z2-z1)**2))/(c*d)
             - np.pi*(z2-z1)/c))
  L.append(l)
```

Finding L21 and L22:

```
for i in range(m*n):
  l=[]
  for j in range(n*m): # Finding L21
        xi=(i//n)+1
        yi=i%n
        xj=(j//m)+1
        yj=j%m
        if xi!=xj or yi!=yj:
          distance=((xi-xj)**2 + (yi-yj)**2 +
             (z2-z1)**2)**(1/2)
          l.append(k*area_of_segment/distance)
        if xi==xj and yi==yj:
          l.append((0.282*c/eps)*(np.sqrt(1+(np.pi*(z2-z1)**2))/(c*d)
             - np.pi*(z2-z1)/c))

  for j in range(n*m): # Finding L22
        xi=(i//n)+1
        yi=i%n
        xj=(j//m)+1
        yj=j%m
        if xi!=xj or yi!=yj:
          distance=((xi-xj)**2 + (yi-yj)**2)**(1/2)
          l.append(k*area_of_segment/distance)
        if xi==xj and yi==yj:
          l.append(k*np.log(1+np.sqrt(2))*4*np.sqrt(c*d))
  L.append(l)
```

Defining the vector g for 2 parallel surfaces:

```
p_v = []
for i in range(5000):
  p_v.append(1)
```

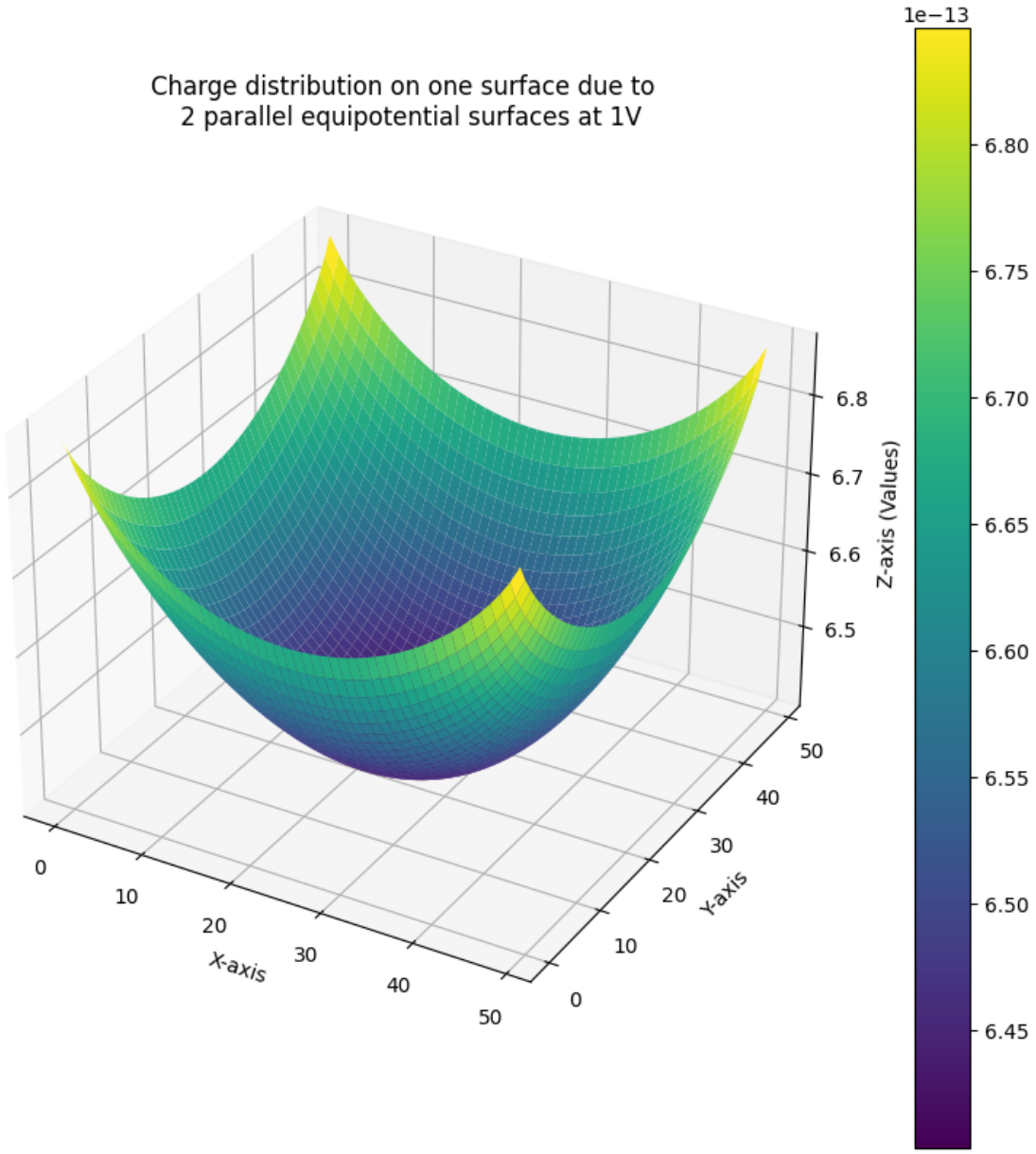

Figure 6: *Output of the above code blocks*

5.4 Considering 2 perpendicular equipotential surfaces.

Defining the 2 perpendicular planes:

```
x1 = np.linspace(0, 15, 50)
y1 = np.linspace(0, 15, 50)
z1 = np.zeros_like(x1)  # z is set to zero for the first plane

X, Y = np.meshgrid(x1, y1)
Z = np.zeros_like(X)  # z is set to zero for the first plane

x2 = np.linspace(0, 15, 50)
z2 = np.linspace(0, 15, 50)
y2 = 15.001 * np.ones_like(x2)  # y is set to a constant value for the second plane

X2, Z2 = np.meshgrid(x2, z2)
Y2 = 15.001 * np.ones_like(X2)  # y is set to a constant value for the second plane
```

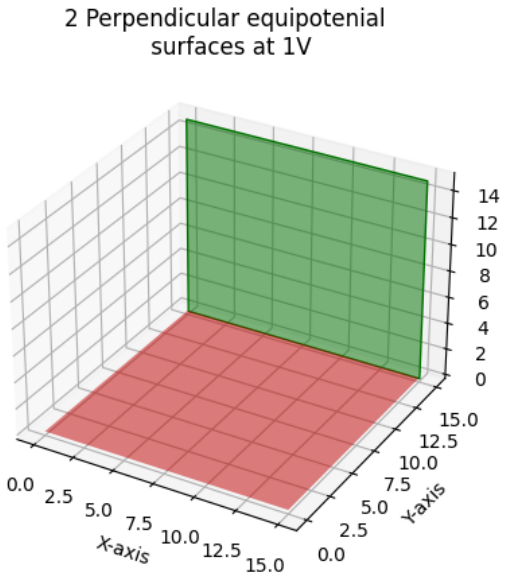

Figure 7: *Two perpendicular equipotential surfaces*

Finding the L matrix:

```
import numpy as np
L = []
k = 9000000000
eps = 8.854187817*(10**-12)
for i1 in range(50):
  for j1 in range(50):
    l=[]
    for i2 in range(50):
      for j2 in range(50): # L11
        if X[i1][j1]!=X[i2][j2] or Y[i1][j1]!=Y[i2][j2]:
          distance=((X[i1][j1]-X[i2][j2])**2 +
             (Y[i1][j1]-Y[i2][j2])**2)**(1/2)
          l.append(k*area_of_segment/distance)
        else:
          l.append(k*np.log(1+np.sqrt(2))*4*np.sqrt(c*d))
    for i2 in range(50):
      for j2 in range(50): # L12
        distance=((X[i1][j1]-X2[i2][j2])**2 +
           (Y[i1][j1]-Y2[i2][j2])**2 +
           (Z[i1][j1]-Z2[i2][j2])**2)**(1/2)
        l.append(k*area_of_segment/distance)
    L.append(l)
for i1 in range(50):
  for j1 in range(50):
    l=[]
    for i2 in range(50):
      for j2 in range(50): # L21
        distance=((X2[i1][j1]-X[i2][j2])**2 +
           (Y2[i1][j1]-Y[i2][j2])**2 +
           (Z2[i1][j1]-Z[i2][j2])**2)**(1/2)
        l.append(k*area_of_segment/distance)
    for i2 in range(50):
      for j2 in range(50): # L22
        if Y2[i1][j1]!=Y2[i2][j2] or
           Z2[i1][j1]!=Z2[i2][j2]:
          distance=((X2[i1][j1]-X2[i2][j2])**2 +
             (Y2[i1][j1]-Y2[i2][j2])**2 +
             (Z2[i1][j1]-Z2[i2][j2])**2)**(1/2)
          l.append(k*area_of_segment/distance)
        else:
          l.append(k*np.log(1+np.sqrt(2))*4*np.sqrt(c*d))
    L.append(l)
```

Defining the vector g for 2 parallel surfaces:

```
p_v = []
for i in range(5000):
  p_v.append(1)
```

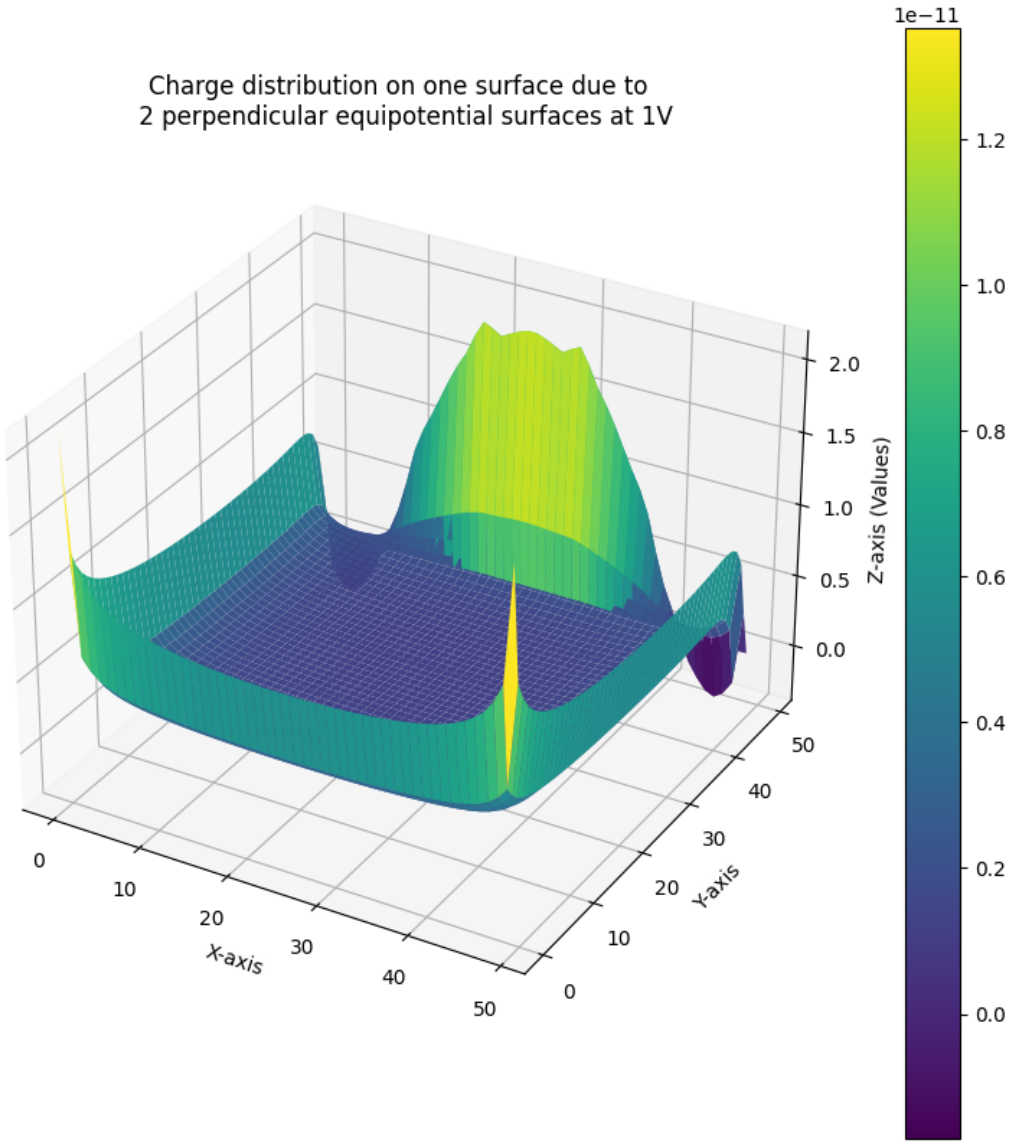

Figure 8: *Output of the above code blocks*

6 Final Results

The following plots portray the charge distribution on equipotential surfaces of the body under consideration. The first graph represents the charge distribution on a single isolated plate due to its own influence; the second graph shows the same for two such plates parallel to each other and the third plot shows the charge distribution for the plate when another plate is kept perpendicular to it, with one common edge.

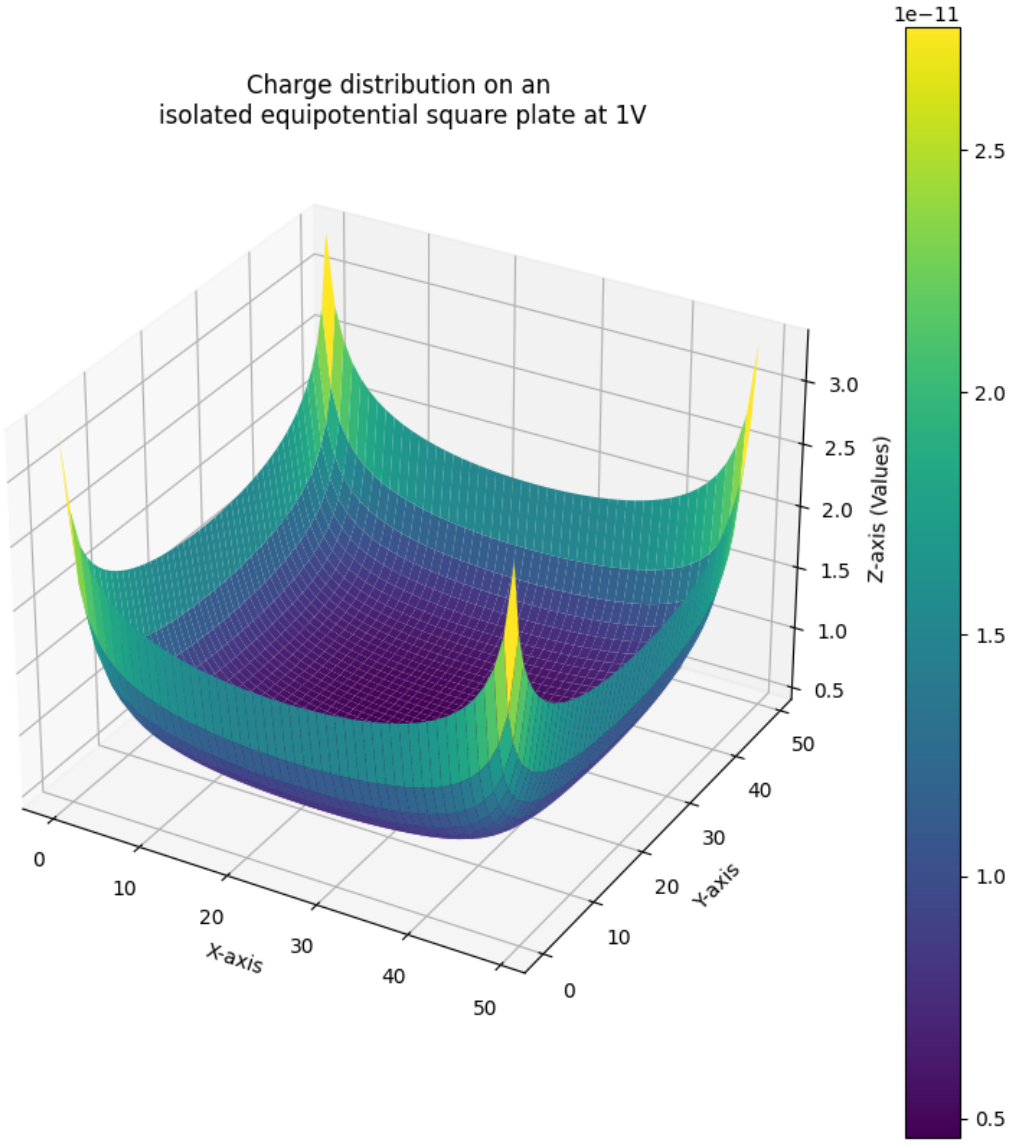

Figure 9: *Charge distribution on an isolated equipotential plate*

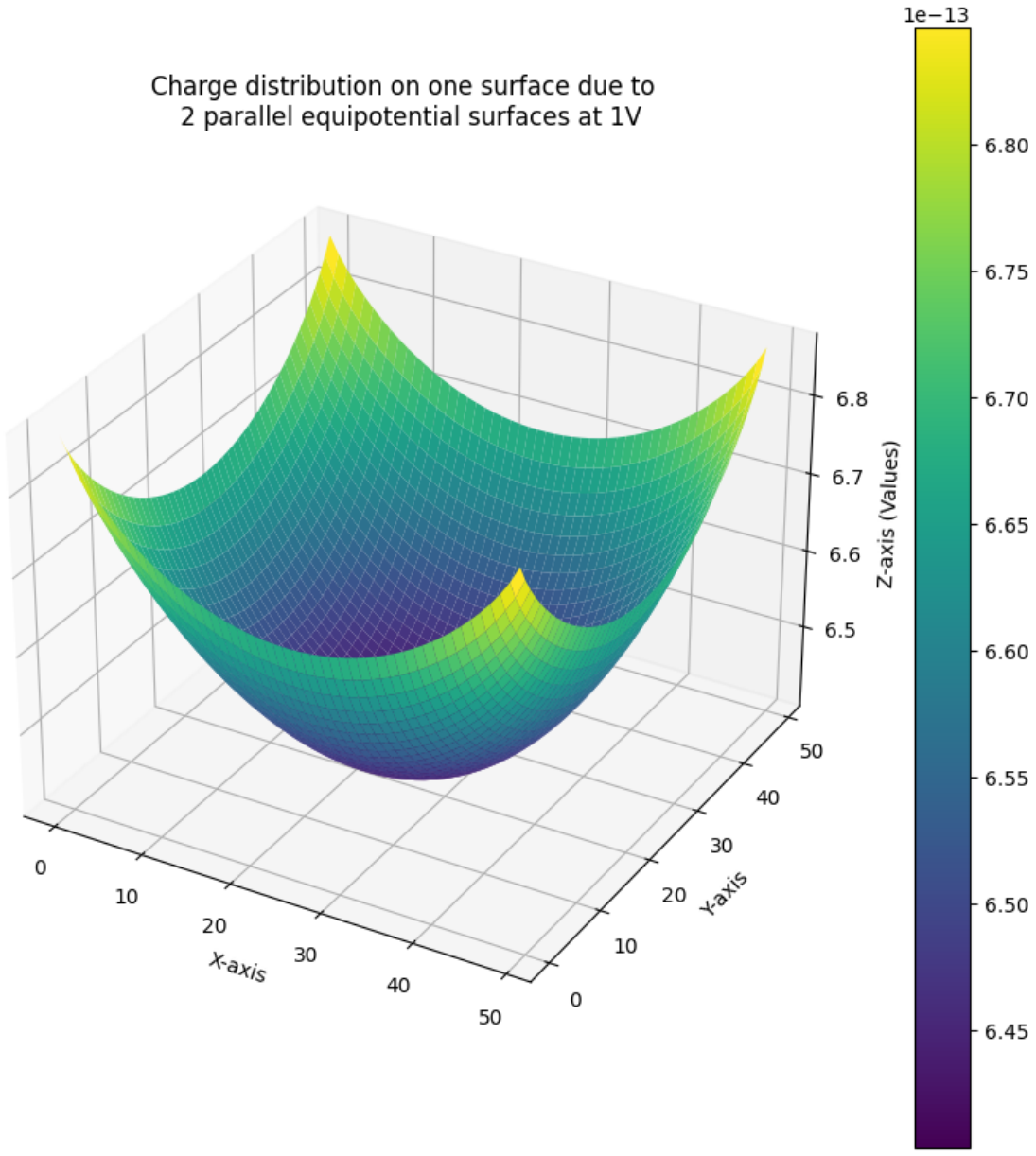

Figure 10: *Charge distribution on an equipotential plate, considering the electrostatic effects of an identical equipotential plate kept parallel to it*

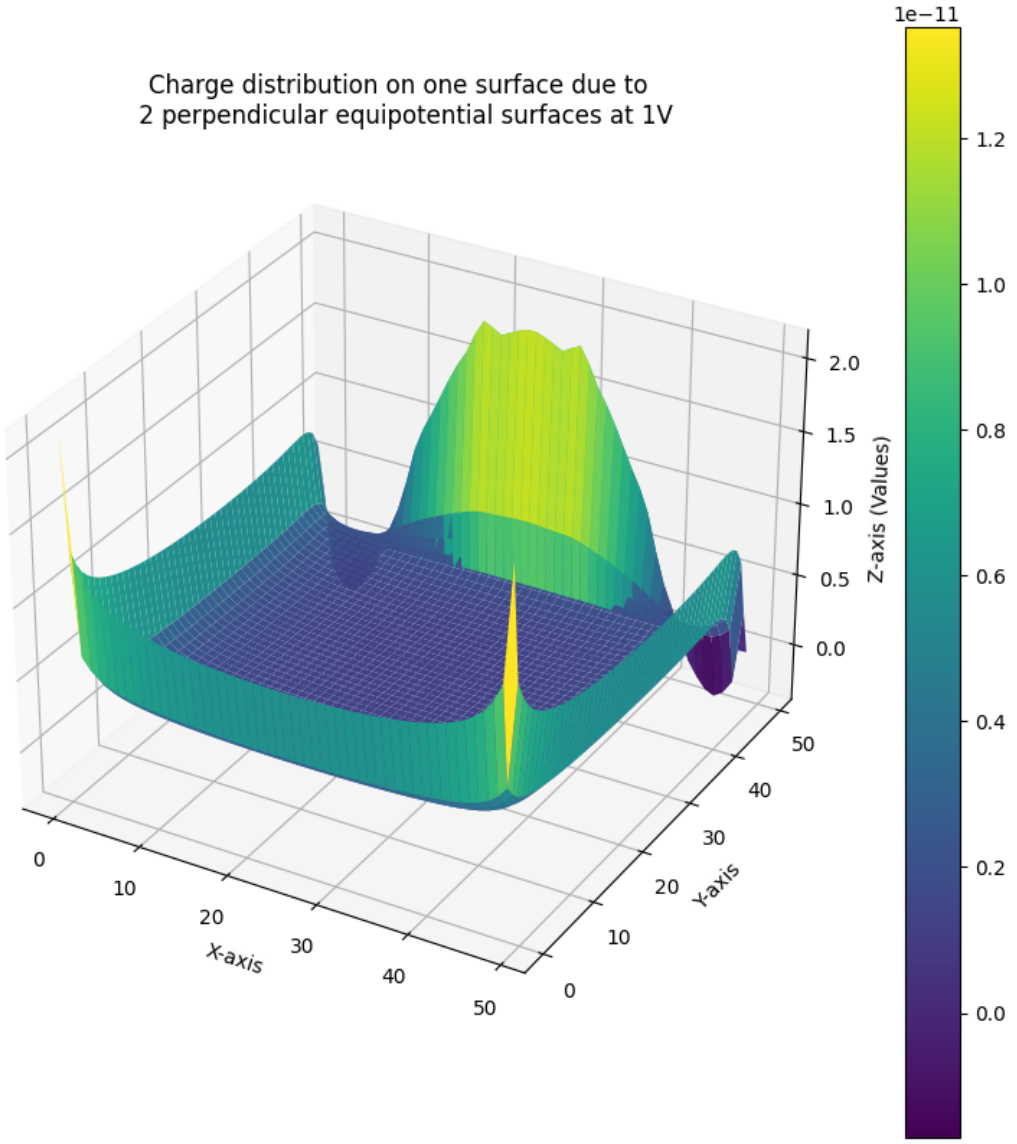

Figure 11: *Charge distribution on an equipotential plate, considering the electrostatic effects of an identical equipotential plate kept perpendicular to it*

The above were the graphs plotted for visualizing surface charge density. These effects will be visible on all 6 surfaces of our spacecraft.

7 Conclusion

The results obtained after implementing our method of mathematically modeling and visualising the charge distribution on the surface of a spacecraft in outer space orbits, clearly align with how it actually should be. The bowl-shaped plots indicate that most of the charge is concentrated on the edges of the surface, since like charges repel. These charge distribution plots will be instrumental in determining the design and assembly of the spacecraft.

Our group of five- Jiya, Kavya, Kishan, Nishi and Shrinivas, collaborated to successfully complete this project for the course MA:203 Numerical Methods. We chose a project topic that intrigued us while still providing a worthwhile challenge in utilizing our knowledge of numerical methods to solve complex problems.

The major takeaway from the project was learning how to develop a sense for visualising a continuous charge distribution as a discrete distribution, by numerically breaking it down into smaller parts. We also applied several concepts from the domains of physics and mathematics for this project. We spoke to our professors with expertise in this domain of spacecraft. It allowed us to explore and foray into the domain of outer space for further research if interested.

The project involved the formulation of the modeling equations and applying mathematical concepts to solve them. It also included Python programming for the application of numerical methods and output visualization.

Solving a challenging engineering problem and also gaining knowledge and experience during this project felt rewarding. We worked together as a team, managing clashing schedules and also distributing work effectively to complete the project at an even pace. We gained evidence of the real-life application of numerical methods. We are grateful for the opportunity to work on this project.

8 References

1. Alad, Rizwan H., and Soumyabrata Chakrabarty. “Electrostatic Analysis of an Artificial Orbiting Satellite for Absolute Charging.” IEEE Transactions on Plasma Science, vol. 43, no. 9, Sept. 2015, pp. 2887–93. DOI.org (Crossref), https://doi.org/10.1109/TPS.2015.2454054.

2. Harrington, Roger F. Field Computation by Moment Methods. IEEE Press, 1993.

3. Sekhon, Balbir S. and Bloom, Daniel J., ”Systems of Linear Equations and the Gauss-Jordan Method”, LibreTexts