
# Trabalho de Eletromagnetismo

Centro de Informática
Aluno : Marcos Heitor Carvalho de Oliveira
Turma : 2020.1 [ES201]

# Problema :

Uma placa condutora de lado maior L é mantida a um potencial $V_o$, como mostrado abaixo. Assuma que o meio seja espaço livre. Aplique o método dos momentos para determinar a distribuição superficial de carga na placa. Para tanto divida cada lado L em N segmentos (N par).

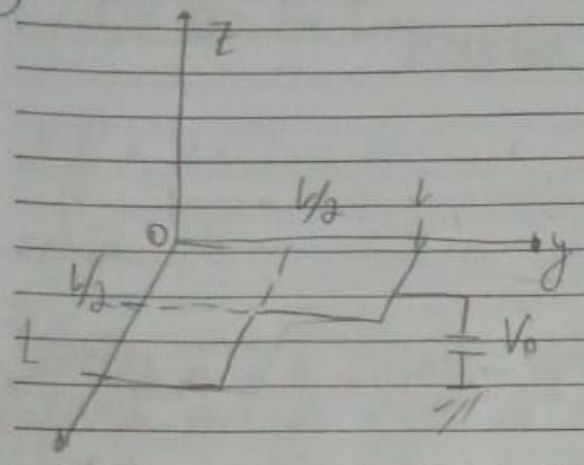
# Trabalho :

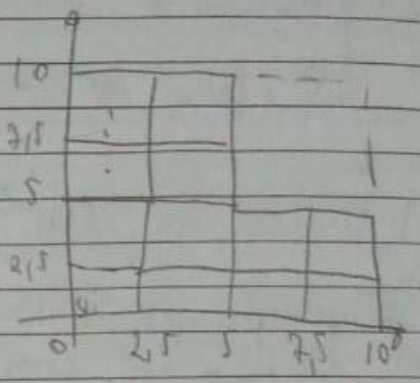
1. Expandindo a distribuição de carga em funções de base tipo pulso, determine as expressões para os elementos das matrizes de impedância e de tensão do método dos momentos.

.Resolução :

# Calculando Métodos Momentos

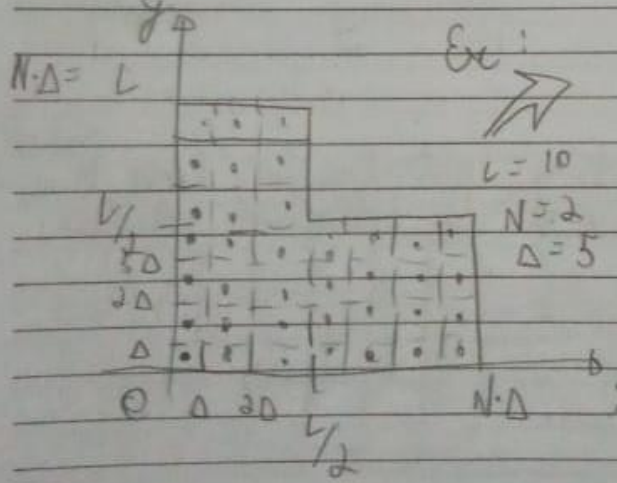
①



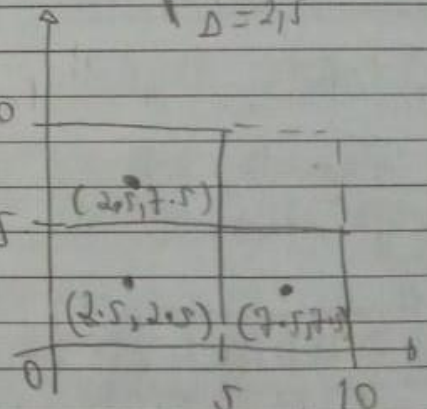

• Discretizando:

$L = 10$
$N = 4$
$\Delta = 2,5$




Ex:

$L = 10$
$N = 2$
$\Delta = 5$




Potencial constante da placa $V_0$:

$$V_0 = \frac{1}{4\pi\varepsilon_0} \iint_S \frac{\rho_s(\vec{r}')}{|\vec{r} - \vec{r}'|} ds' \quad \vec{r} \in \text{Placa}$$

Usando a condição de contorno no centro de cada quadrado:

$$m,\ i=1,2,\dots\left(N^2-\frac{L^2}{4}\right)$$

$$r_m=(x_p,y_p)\begin{cases}x_i=(p-1/2)\Delta\\ y_j=(q-1/2)\Delta\end{cases}\ \text{pontos centrais}$$

Usando a aproximação da distribuição de cargas, onde

$$P_0(x,y)=\sum_{n=1}^{(N^2-L^2/4)}a_n P_n(x,y)$$

$$V_0=\frac{1}{4\pi\varepsilon_0}\iint_{S'}\frac{P_0(\vec{r}^{\,0})}{|\vec{r}_m^{\,0}-\vec{r}^{\,0}|}ds'$$

$$V_0=\frac{1}{4\pi\varepsilon_0}\iint_{S'}\frac{\sum_{n=1}^{(N^2-L^2/4)}a_n P_n(x',y')}{|\vec{r}_m^{\,0}-\vec{r}^{\,0}|}ds'$$

$$V_0=\sum_{n=1}^{(N^2-L^2/4)}\frac{a_n}{4\pi\varepsilon_0}\int_{x_i-D/2}^{x_i+D/2}\int_{y_j-D/2}^{y_j+D/2}\frac{1}{|\vec{r}_m^{\,0}-\vec{r}^{\,0}|}dx'dy'$$

$$[Z]\begin{bmatrix}a_1\\ \vdots\\ a_{N^2-\frac{L^2}{4}}\end{bmatrix}=[V]$$

$$Z_{mn} = \frac{1}{4\pi\varepsilon_0} \int_{x_{ni}-\Delta/2}^{x_{ni}+\Delta/2} \int_{y_{nj}-\Delta/2}^{y_{nj}+\Delta/2} \frac{1}{|\vec{r}_m - \vec{r}'|} dx'dy'$$

Como $|\vec{r}_m - \vec{r}'| = \sqrt{(x_p - x')^2 + (y_q - y')^2}$

Então

$$Z_{mn} = \frac{1}{4\pi\varepsilon_0} \int_{x_{ni}-\Delta/2}^{x_{ni}+\Delta/2} \int_{y_{nj}-\Delta/2}^{y_{nj}+\Delta/2} \frac{1}{\sqrt{(x_p - x')^2 + (y_q - y')^2}} dx'dy'$$

Se $m \neq n$, então $\approx$ toda a carga em $n$ pode ser considerada como concentrada no centro; logo

$$Z_{mn} \approx \frac{1}{4\pi\varepsilon_0} \frac{\Delta^2}{\sqrt{(x_p - x_i)^2 + (y_q - y_j)^2}}$$

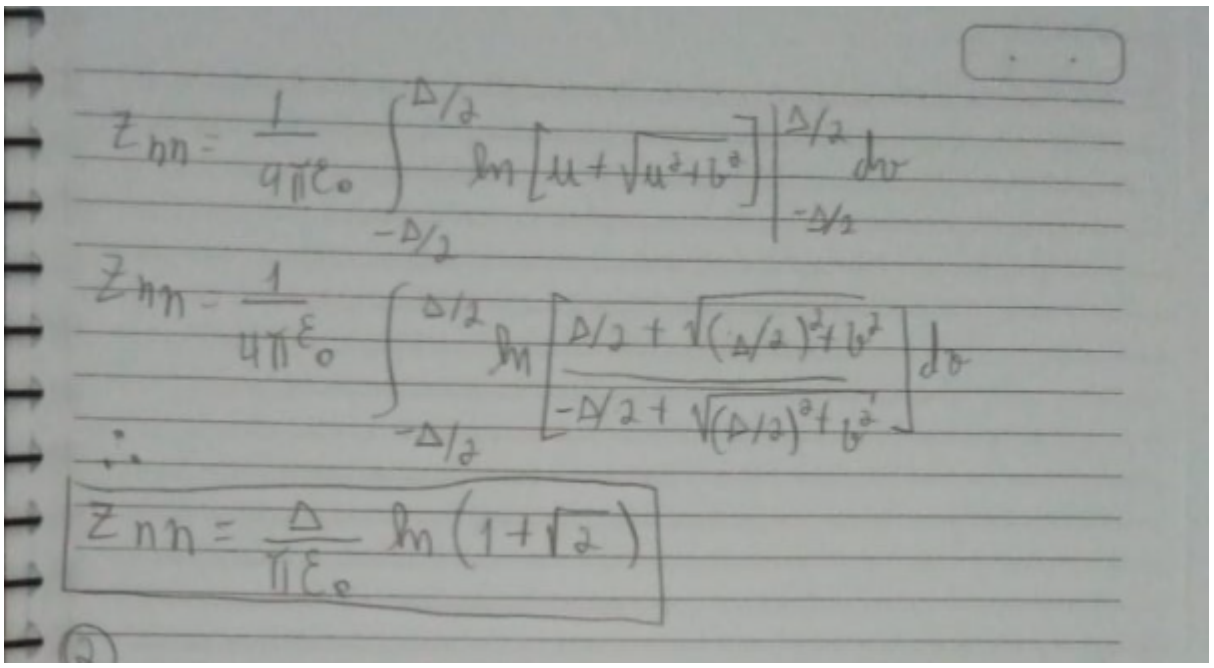
Se $m = n$, então calculamos a integral, onde

$u = x_p - x' \therefore du = -dx'$

$v = y_q - y' \therefore dv = -dy'$

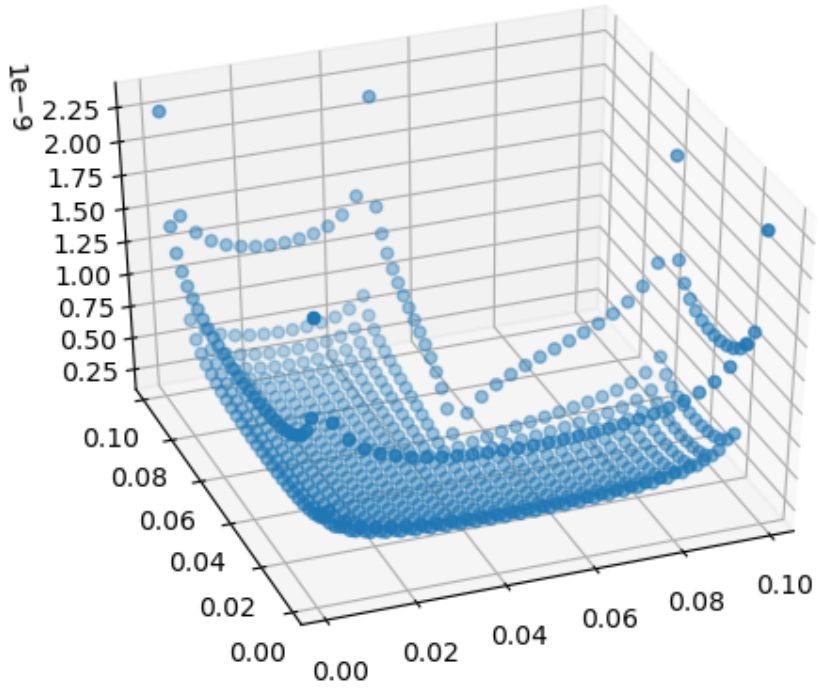
$\therefore$

$$Z_{nn} = \frac{1}{4\pi\varepsilon_0} \iint \frac{1}{\sqrt{u^2 + v^2}} du\,dv$$



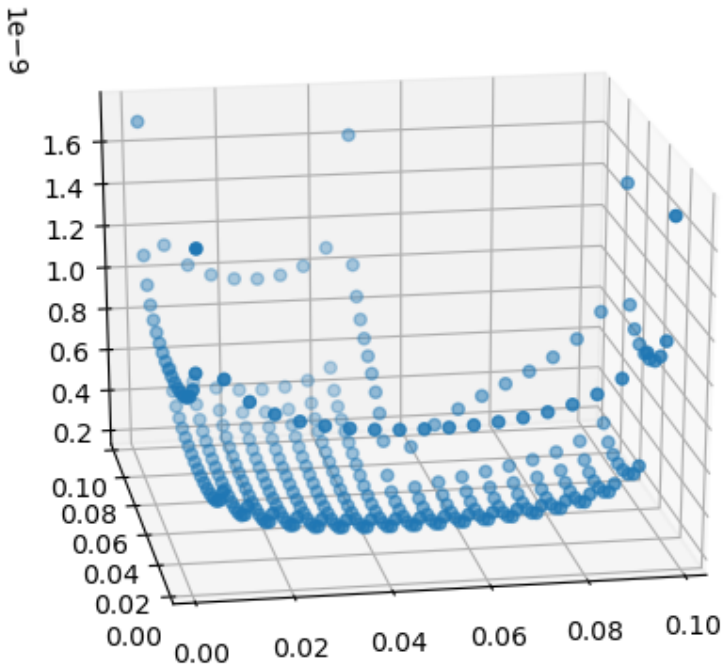

2. Para o caso em que L = 10 cm, e $V_0$ = 1 V, resolva o sistema linear para um valor de N específico (você escolhe, mas par). Determine as amplitudes dos pulsos, e obtenha uma aproximação para a distribuição de carga superficial na placa. Plote o resultado.

Usando a fórmula analitica obtida, plotamos as amplitudes dos pulsos como segue no gráfico ( Valor de N = 30) :

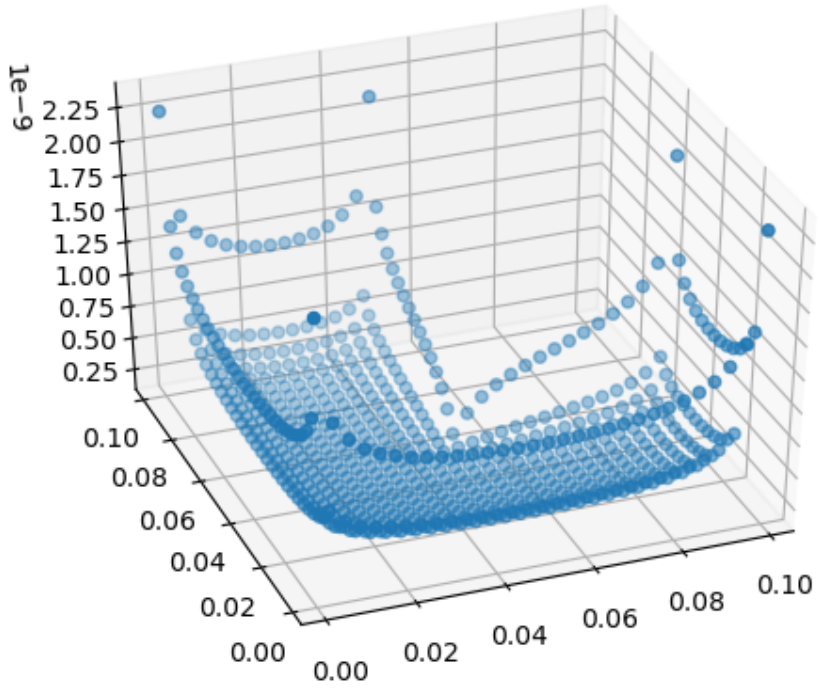



3. Resolva o problema e plote a distribuição superficial de carga para diferentes valores de N. Comente os resultados.

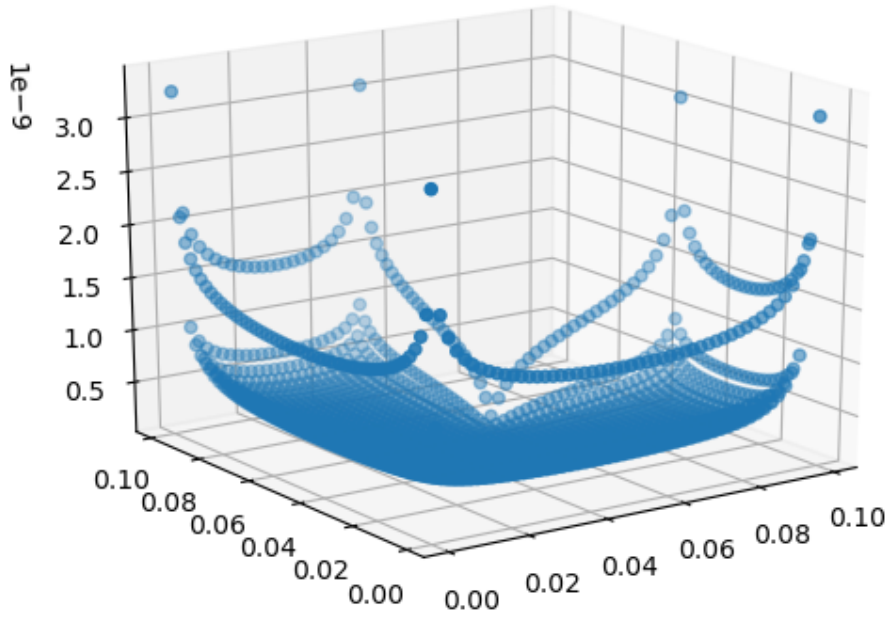
   Os resultados a seguir foram obtidos variando o valor de N de 20 à 80 com um passo de 10 :


1e−9
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.02
0.04
0.06
0.08
0.10

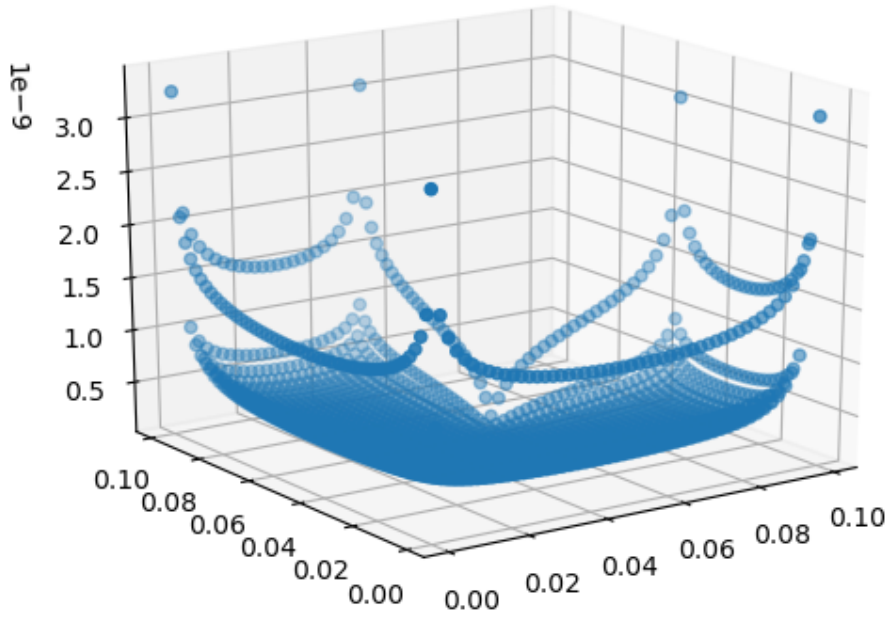
(N = 20)


1e−9
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.02
0.04
0.06
0.08
0.10

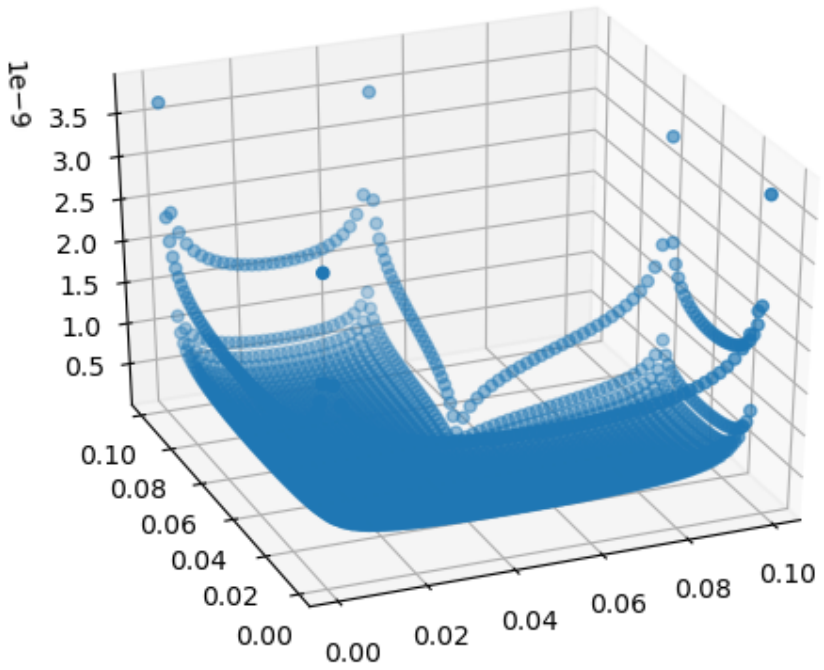
(N = 30)


1e−9
3.0
2.5
2.0
1.5
1.0
0.5
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.02
0.04
0.06
0.08
0.10

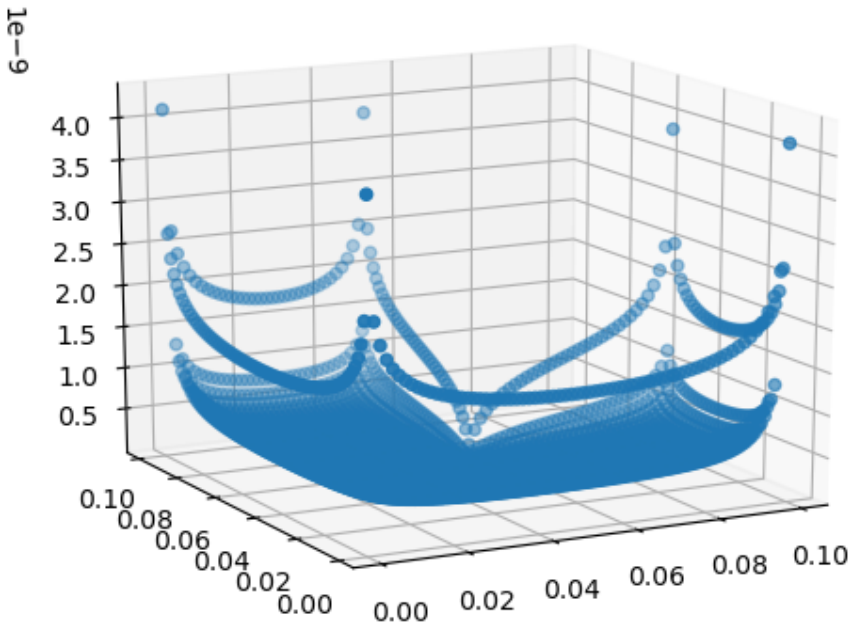
(N = 40)


1e−9
3.0
2.5
2.0
1.5
1.0
0.5
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.02
0.04
0.06
0.08
0.10

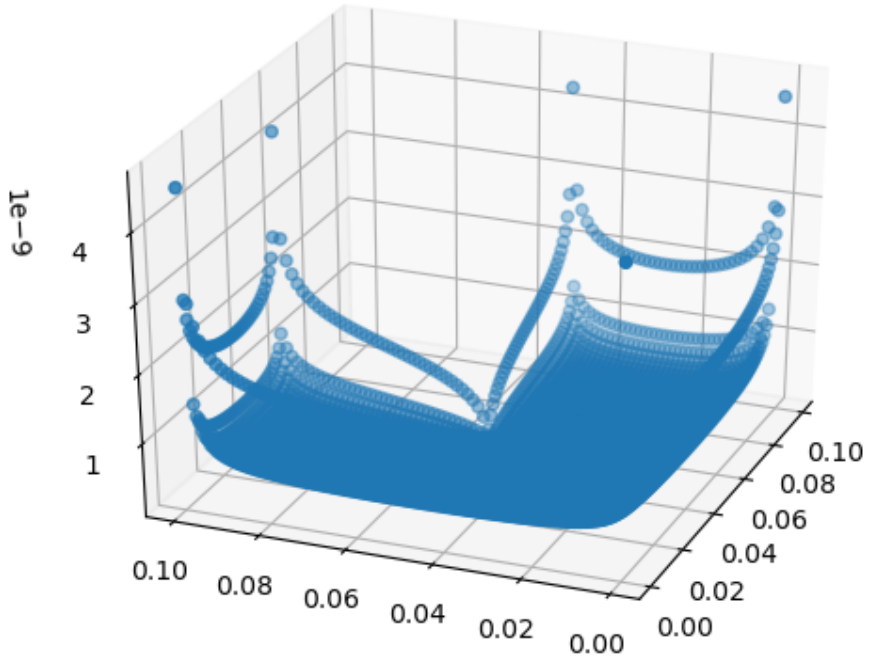
(N = 50)


1e−9
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.02
0.04
0.06
0.08
0.10

(N = 60)


1e−9
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.02
0.04
0.06
0.08
0.10

(N = 70)




(N = 80)

Baseado nos resultados, podemos observar que quanto mais aumentamos o numero de N (as subdivisões do método), notamos que temos uma aproximação melhor para o que seria a distribuição de carga sob a placa, é interessante notar que a distribuição da placa segue um determinado padrão, sendo ela bem distribuída nas regiões centrais e com picos de distribuições nas pontas, seguindo um padrão esperado com o do livro texto. Outra análise que podemos tirar é que, apesar de o aumento do N nos dar uma aproximação melhor para a distribuição de carga, isso infelizmente também traz um custo computacional bem maior, o que faz com o código demore mais.

4. Determine a carga total na placa. Varie N e observe a convergência.

A carga total pode ser obtida fazendo o somatório das distribuições de carga pela área das pequenas regiões que estamos aproximando, é fácil resolver isso computacionalmente, e observar que o valor tende em torno de 3,6 * 10 ^ 12 C, como mostra na figura a seguir com a execução dos testes.

```
marcos@LAPTOP-K3SSD7P6:/mnt/c/Users/Marcos/Documents/Minhas Imagens/Eletromag/MomentMethods$ python3 MomentMethods.py 0.
1 1 20 -q -nplot
Qtotal :  3.6169081006146795e-12
marcos@LAPTOP-K3SSD7P6:/mnt/c/Users/Marcos/Documents/Minhas Imagens/Eletromag/MomentMethods$ python3 MomentMethods.py 0.
1 1 30 -q -nplot
Qtotal :  3.635323869609814e-12
marcos@LAPTOP-K3SSD7P6:/mnt/c/Users/Marcos/Documents/Minhas Imagens/Eletromag/MomentMethods$ python3 MomentMethods.py 0.
1 1 40 -q -nplot
Qtotal :  3.6442808424384997e-12
marcos@LAPTOP-K3SSD7P6:/mnt/c/Users/Marcos/Documents/Minhas Imagens/Eletromag/MomentMethods$ python3 MomentMethods.py 0.
1 1 50 -q -nplot
Qtotal :  3.64952737372533e-12
marcos@LAPTOP-K3SSD7P6:/mnt/c/Users/Marcos/Documents/Minhas Imagens/Eletromag/MomentMethods$ python3 MomentMethods.py 0.
1 1 60 -q -nplot
Qtotal :  3.652951845457168e-12
marcos@LAPTOP-K3SSD7P6:/mnt/c/Users/Marcos/Documents/Minhas Imagens/Eletromag/MomentMethods$ python3 MomentMethods.py 0.
1 1 70 -q -nplot
Qtotal :  3.6553520953232846e-12
marcos@LAPTOP-K3SSD7P6:/mnt/c/Users/Marcos/Documents/Minhas Imagens/Eletromag/MomentMethods$ python3 MomentMethods.py 0.
1 1 80 -q -nplot
Qtotal :  3.657121737936611e-12
```

## O código

```python
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import sys

CONST_VACUUM_PERMITTIBITY = 8.854187812813e-12
#F · m−1

def estimated_full_charge(dist_consts, delta):
    qpa = sum(dist_consts)

    area = delta ** 2

    return qpa * area

def plot_surface(central_points, dist_consts):
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')

    xs = []
    ys = []
    zs = []

    for i in range(len(central_points)):
        xs.append(central_points[i][0])
        ys.append(central_points[i][1])
        zs.append(dist_consts[i])
```

```
    ax.scatter(xs, ys, zs)
    plt.show()

def find_dist_consts(impedance_matrix, tension_arr):
    # Z * a = V

    dist_consts = np.linalg.solve(impedance_matrix, tension_arr)

    return dist_consts

def build_impedance_matrix(central_points, delta):
    impedance_matrix = np.zeros((len(central_points), len(central_points)))

    for i in range(len(central_points)):
        for j in range(len(central_points)):
            if i != j:
                dx2 = (central_points[i][0] - central_points[j][0]) ** 2
                dy2 = (central_points[i][1] - central_points[j][1]) ** 2

                impedance_matrix[i][j] =  (delta * delta) / (4.0 * np.pi *
CONST_VACUUM_PERMITTIBITY * np.sqrt(dx2 + dy2))
            else:
                impedance_matrix[i][j] = (delta * np.log(1.0 + np.sqrt(2))) /
(np.pi * CONST_VACUUM_PERMITTIBITY)

    return impedance_matrix

def discretize(plate_lenght, delta):
    central_points = []

    for j in np.arange(0, plate_lenght, delta):
        for i in np.arange(0, plate_lenght, delta):

            x = i + (delta / 2.0)
            y = j + (delta / 2.0)

            if (x > plate_lenght / 2.0 and y > plate_lenght / 2.0):
                continue
            central_points.append([x, y])


    return central_points

def mm_help():
    print("HELP GUIDE : ")
    print("")
```

```
    print(".Usage :")
    print("")
    print("\tpython3 MomentMethods.py L V0 N [[-q] -nplot]")
    print("")
    print(".Details :")
    print("")
    print("L := The L length used for the metal plate (in m)")
    print("V0 := The constant tension used in the metal plate (in Volts)")
    print("N := A even number, represents the division in the metal plate")
    print("-q := Optional argument, with this set the code will print the code
will print the estimated full charge.")
    print("-nplot := Optional argument, if this is set with the '-q' arg, the
program won't plot the graph.")
    print("")
    print("--------------METAL PLATE--------------")
    print("")
    print("L ^ --------        ")
    print("  | --------        ")
    print("  | --------        =====> V0")
    print("  | --------'-------")
    print("  | ----------------")
    print("  | ----------------")
    print("0 | _________________>")
    print("  0        L/2      L")
    print(". L = N * delta")
    print(". ' = (L/2, L/2)")
    print("----------------------------------------")


def main():
    args_len = len(sys.argv)

    if args_len < 4:
        mm_help()
    else:
        print_plot = True
        print_q_total = False

        if args_len > 5:
            if(sys.argv[4] == "-q"):
                print_q_total = True
            else:
                print("Option " + sys.argv[4] + " not found.")

            if args_len == 6:
                if(sys.argv[5] == "-nplot"):
```

```
                    print_plot = False
                else:
                    print("Option " + sys.argv[5] + " not found.")

        plate_lenght = float(sys.argv[1])
        tension = float(sys.argv[2])
        sub_divisions = int(sys.argv[3])

        delta = plate_lenght / float(sub_divisions)

        central_points = discretize(plate_lenght, delta)

        impedance_matrix = build_impedance_matrix(central_points, delta)

        tension_arr = np.full(len(central_points), tension)

        dist_consts = find_dist_consts(impedance_matrix, tension_arr)

        if print_plot:
            plot_surface(central_points, dist_consts)

        if print_q_total:
            print("Qtotal : ", estimated_full_charge(dist_consts, delta))

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

Pequeno resumo, o código recebe como argumento os valores de N, L e V0 e executa a discretização, criação da matriz de impedâncias e resolução dos sistemas lineares, se o código não receber os argumentos ideais, ele mostra uma aba de ajuda ao usuário, se a opção ‘-q’ estiver setada na execução, o código calcula a carga total da placa como especificado, além disso se estiver setada a opção ‘-nplot’, o código realiza todos os cálculos mas não plota o gráfico. Todo o resto do código é feito de acordo com as técnicas do método momentos.