Numerical Capacitance Extraction using a Discretized Method

ECE 107 Fall 2021 Project

Hao Le

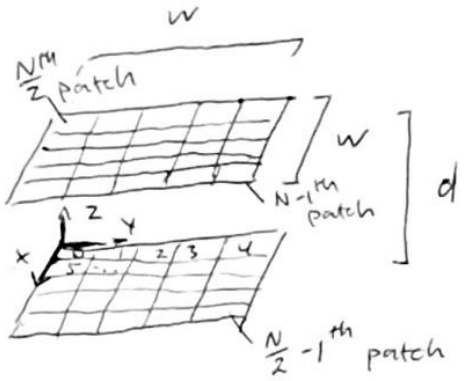
The objective of this project is to discretize the problem of finding the surface charge density of the plates of a parallel-plate capacitor, thus enabling the extraction of capacitance. We are given the voltages of the two plates: $V_o/2$ of the top plate and $-V_o/2$ of the bottom. Additionally, we also know the geometry of the plates: with negligible thickness, the width is *w*, and the distance between the plates is *d*.

The electric potential due to a point charge Q is given as:

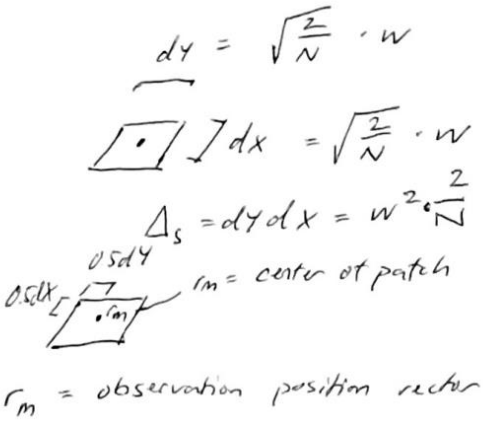
$$V_{\mathbf{E}} = \frac{1}{4\pi\varepsilon_0}\frac{Q}{r},$$

Equation taken from:
https://en.wikipedia.org/wiki/Electric_potential

Where r is the distance between the charge and the observation point. We can extend this to a surface charge distribution, where Q is now a function that involves multiplying infinitesimal area dA by the surface charge density evaluated at the area. If we discretized the plates' surfaces into squares:



Notice that the coordinate system is placed on the top left corner of the bottom plate, thus dictating that the patches are indexed as such in the diagram. For every patch, we can find the potential at its center r_m:



To find the potential for a patch, we first fix the observation point to be r_m. Then, using the formula above with the dA being discretized to delta S, we find the individual N potentials, from the other N charged patches including the observation patch, that contribute to the final potential of $V(r_m)$:

$$V(r_m) = \sum_{n=0}^{N-1} \frac{\rho_s(r_n)\,\Delta_s}{4\pi\varepsilon_0 |r_m - r_n|} = \begin{cases} \frac{V_0}{2} & r_m \in \text{top plate} \\ \frac{V_0}{-2} & r_m \in \text{bottom plate} \end{cases}$$

From the plate voltages given, we can form equalities for the potentials at each patch – those on the bottom plate are $-V_o/2$ and conversely those on the top plate are $V_o/2$. From these equalities, we can solve for the discrete charge density at each patch. Turning the problem into a matrix problem:

$$\begin{bmatrix} \frac{1}{4\pi\varepsilon_0|r_0-r_0|} & \frac{1}{4\pi\varepsilon_0|r_0-r_1|} & \cdots & \frac{1}{4\pi\varepsilon_0|r_0-r_{N-1}|} \\ \frac{1}{4\pi\varepsilon_0|r_1-r_0|} & \frac{1}{4\pi\varepsilon_0|r_1-r_1|} & \cdots & \vdots \\ \vdots & & & \\ \frac{1}{4\pi\varepsilon_0|r_{N-1}-r_0|} & \cdots & \cdots & \frac{1}{4\pi\varepsilon_0|r_{N-1}-r_{N-1}|} \end{bmatrix} \begin{bmatrix} \rho_s(r_0)\Delta_s \\ \rho_s(r_1)\Delta_s \\ \rho_s(r_2)\Delta_s \\ \vdots \\ \rho_s(r_{N-1})\Delta_s \end{bmatrix} = \begin{bmatrix} -\frac{V_0}{2} \\ -\frac{V_0}{2} \\ \vdots \\ \frac{V_0}{2} \\ \frac{V_0}{2} \\ \vdots \\ \frac{V_0}{2} \end{bmatrix}$$

Z $(N \times N)$ $\quad Q$ $(N \times 1)$ $\quad V$ $(N \times 1)$

Q is the charge vector that describes the discrete charges of all the patches. By dividing the vector by delta S, we get the discrete charge densities. Notice that the voltage contribution of a charged patch an observation point that is located on/near itself can be approximated as follows:

$$\frac{1}{4\pi\varepsilon_0|r_n - r_n|} \simeq \frac{1}{2\varepsilon_0\sqrt{\pi\Delta_s}}$$

To solve for Q, multiply the inverse of Z with V:

$$Q = Z^{-1}V$$

To find the capacitance, we simply sum up the N /2 charges of the positively charged plate (indexed from N/2 to N-1), and then divide by V_o.

Computer code

The problem will be solved using an implementation in Python.

Functions used are defined:

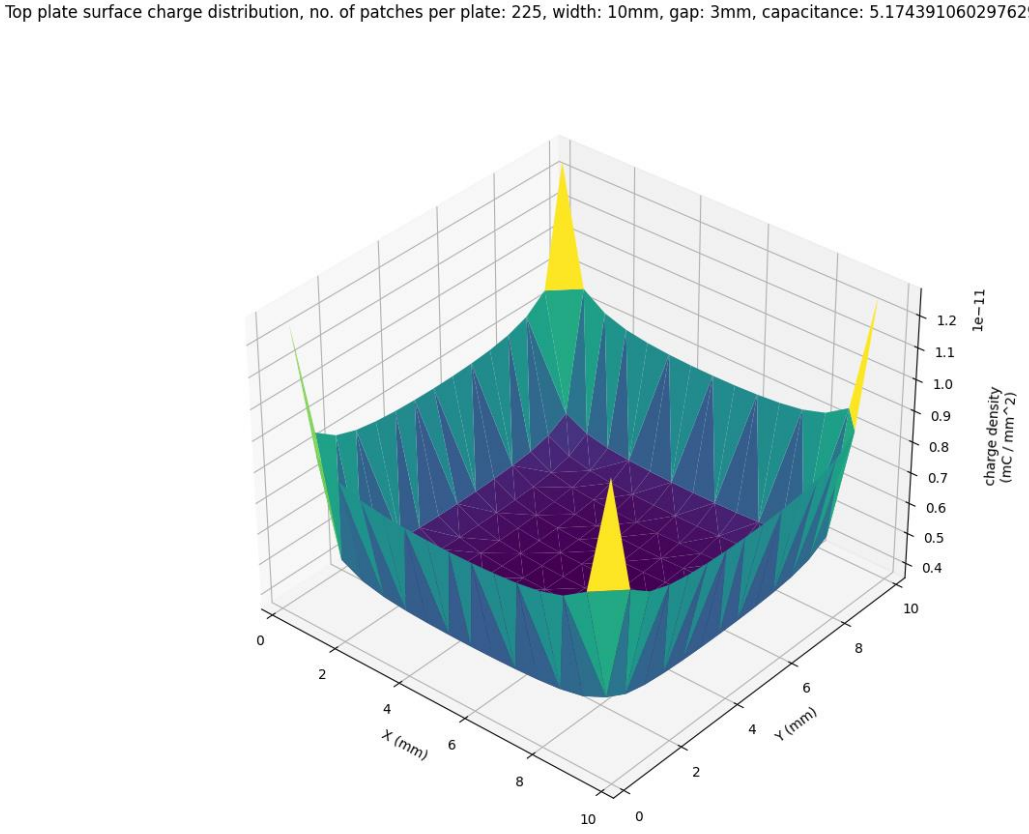
```python
def calculateZ(chargeLocation, observationLocation, delta_s): #calculate potential contribution of a charged patch to an observation point on the center of a patch.

    if np.array_equal(chargeLocation, observationLocation): #if charged patch is on/near the observation patch, approximate
        Z = 1 / (2 * epsilon_0 * np.sqrt(pi * delta_s))
        return Z
    else: #use potential formula
        distanceRmRn = np.linalg.norm(chargeLocation - observationLocation) #find magnitude of distance vector
        Z = 1 / (4 * pi * epsilon_0 * distanceRmRn)
        return Z


def getPatchVectorLocations(w, d, subdivisions, numberOfPatches): #get an array of all N 3d position vectors of the centers of the patches
    delta_w = w / subdivisions #get the increment between centers of patch
    patchLocations = np.zeros((numberOfPatches, 3)) #center vector locations of all N patches
    numberOfPatchesPerPlate = int(numberOfPatches / 2)

    for i in range(numberOfPatchesPerPlate): #bottom plate where z is 0
        yPatchIndex = i % subdivisions
        xPatchIndex = np.floor(i / subdivisions)

        x = 0.5 * delta_w + xPatchIndex * delta_w
        y = 0.5 * delta_w + yPatchIndex * delta_w
        z = 0

        locationVector = np.array([x, y, z])
        patchLocations[i] = locationVector

    for i in range(numberOfPatchesPerPlate): #top plate where z is d

        yPatchIndex = i % subdivisions
        xPatchIndex = np.floor(i / subdivisions)

        x = 0.5 * delta_w + xPatchIndex * delta_w
        y = 0.5 * delta_w + yPatchIndex * delta_w
        z = d

        locationVector = np.array([x, y, z])
        patchLocations[i + numberOfPatchesPerPlate] = locationVector

    return patchLocations
```

To find and plot the surface charge distribution of the positive plate, as well as extract the numerical capacitance, given w, d, and V_0:

```
#parameters

V_0 = 1 #positive plate is V_0 / 2
w = 10 #width of plates in mm
d = 3 #gap between plates in mm
subdivisions = 15 #how many discrete intervals along the sides of the plates. Each plate will be subdivisions x subdivisions in terms of patches

numberOfPatches = np.square(subdivisions) * 2 #calculate the total number of patches
delta_s = (2 / numberOfPatches) * np.square(w) #get delta_s discrete area

patchVectorLocations = getPatchVectorLocations(w, d, subdivisions, numberOfPatches)

Z = np.zeros((numberOfPatches, numberOfPatches)) #Z is N x N

for m in range(Z.shape[0]):

    observationLocation = patchVectorLocations[m] #every row of Z is a fixed observation point

    for n in range(Z.shape[1]): #vary where the charged plate is that contributes the potential

        Z_mn = calculateZ(patchVectorLocations[n], observationLocation, delta_s)
        Z[m][n] = Z_mn


V = np.zeros(numberOfPatches) #V is N x 1

for i in range(numberOfPatches):
    #construct V such that the first half elements are the voltages of the bottom plate's patches, -V_0 / 2
    #second half elements are voltages of top plate's patches, V_0 / 2
    if np.floor(i / (numberOfPatches / 2)) == 0:
        V[i] = - V_0 / 2
    else:
        V[i] = V_0 / 2


Z_inv = np.linalg.inv(Z)

Q = np.matmul(Z_inv, V) #Q charge vector is found with matrix algebra

numberOfPatchesPerPlate = int(numberOfPatches / 2)

#post process the Q vector for plotting distribution of one plate

x = np.zeros(numberOfPatchesPerPlate)
y = np.zeros(numberOfPatchesPerPlate)
surfaceChargeDensity = np.zeros(numberOfPatchesPerPlate)
totalChargePerPlate = 0
```

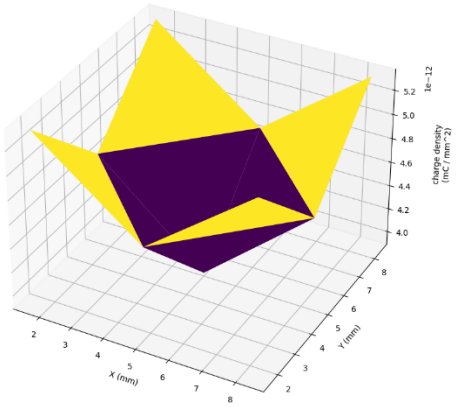
```
for i in range(numberOfPatchesPerPlate):
    locationVector = patchVectorLocations[i + numberOfPatchesPerPlate] #offsetting by N/2 since we want the top plate's positive charges
    x[i] = locationVector[0]
    y[i] = locationVector[1] #extract the x,y coordinate of the top plate patches
    surfaceChargeDensity[i] = Q[i + numberOfPatchesPerPlate] / delta_s #extract the discrete charge densities of top plate patches by dividing by patch area
    totalChargePerPlate = totalChargePerPlate + Q[i + numberOfPatchesPerPlate] #sum up all discrete charges to find plate's total charge


calculatedCapacitance = totalChargePerPlate / V_0 #with the summed up charges of one plate, divide by V_0 to extract capacitance


#plot the top plate's charge distribution

fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
fig.suptitle('Top plate surface charge distribution, no. of patches per plate: {}, width: {}mm, gap: {}mm, capacitance: {}mF'.format(numberOfPatchesPerPlate, w, d, calculatedCapacitance), fontsize=12)
surf = ax.plot_trisurf(x, y, surfaceChargeDensity, cmap=cm.viridis)
ax.set_xlabel('X (mm)')
ax.set_ylabel('Y (mm)')
ax.set_zlabel('\n\n\n\ncharge density\n(mC / mm^2)')
plt.show()
```

For width of 1cm and gap of 3mm, and with a patch subdivision of 15 (225 patches per plate), the code yields the following distribution plot:

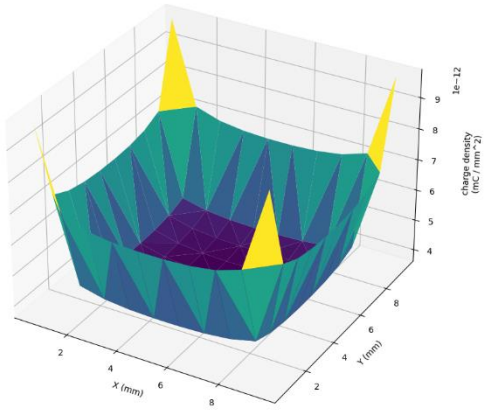


The plot appears probable, considering that the charges get denser near the edge of the plate, which is explained by the fringing effects. Experimenting with lower/higher subdivisions that yield coarser/finer distribution plots respectively as well as varying calculated capacitances:

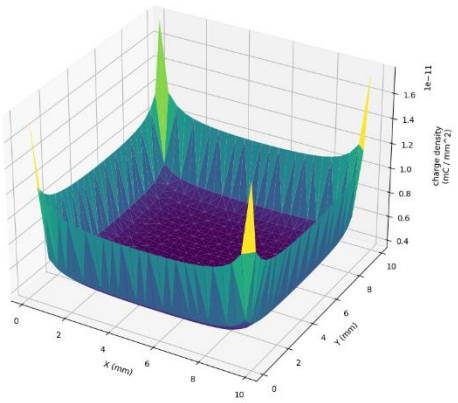
Top plate surface charge distribution, no. of patches per plate: 9, width: 10mm, gap: 3mm, capacitance: 4.88342968267649e-10mF



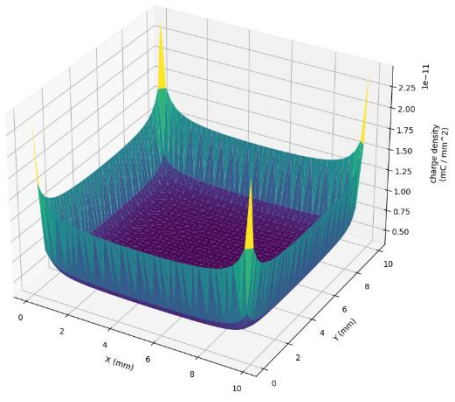
Top plate surface charge distribution, no. of patches per plate: 100, width: 10mm, gap: 3mm, capacitance: 5.123932726079905e-10mF



Top plate surface charge distribution, no. of patches per plate: 625, width: 10mm, gap: 3mm, capacitance: 5.215694129555363e-10mF



Top plate surface charge distribution, no. of patches per plate: 1600, width: 10mm, gap: 3mm, capacitance: 5.238624915232122e-10mF



The next experiment involves fixing the width of the plates to 1cm while varying the gap distance to compare the numerical capacitance extracted from our discrete method against using the approximation of:

$$C = (\varepsilon_0 (A^2)/ d$$

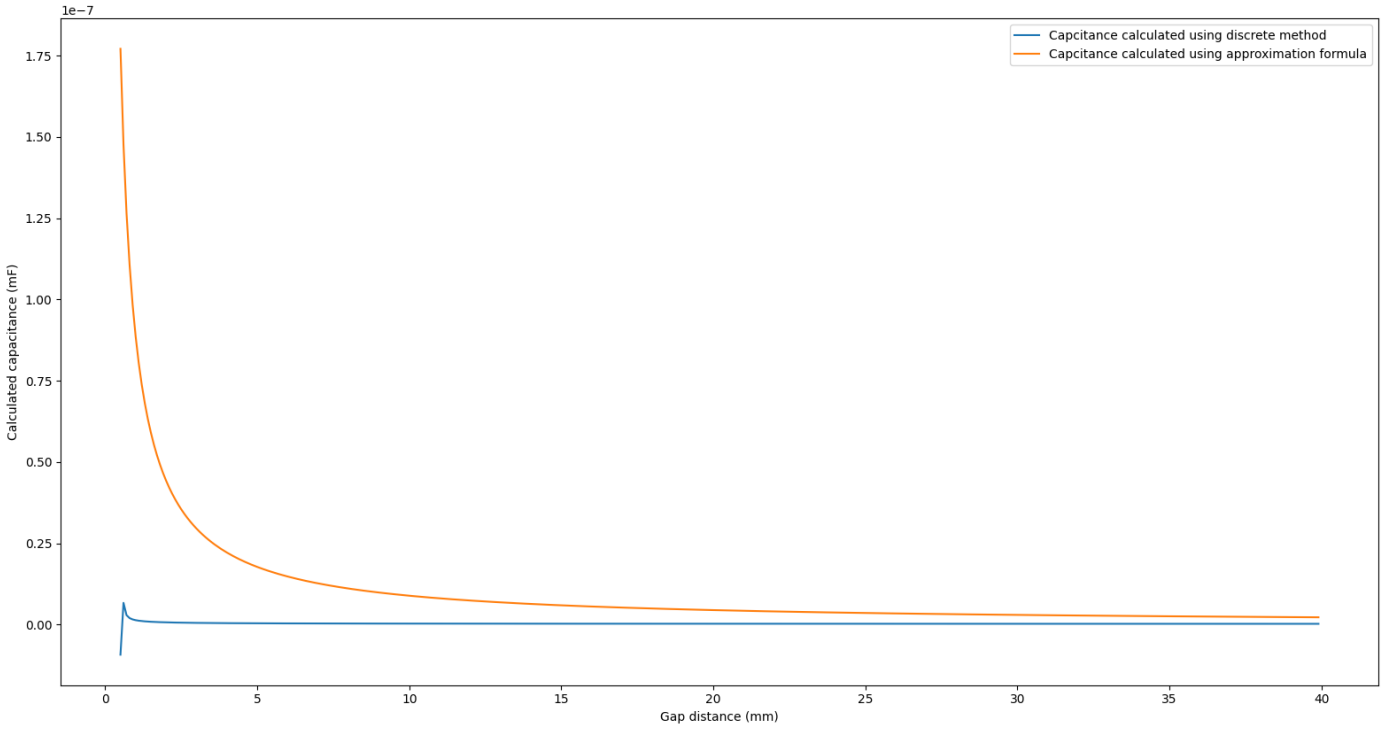
Where A is the area of a plate, which in our case is w*w.

Modification to the code involves adding the array of distances that are to be used for the capacitance calculation; we vary from 0.5mm up to 40mm, with 0.1mm increments. We fixed the subdivision number to 5 for faster computation time.

```
d_plot = np.arange(0.5, 40, 0.1) #distances from 1 to 40mm
w = 10 #fixed 1cm width
V_0 = 1
c_calc = [] #calculated capacitances using discretized method
c_formula = []#using approximation formula
subdivisions = 5
numberOfPatches = np.square(subdivisions) * 2
delta_s = (2 / numberOfPatches) * np.square(w)

for d in d_plot: #vary d and calculate capacitance using same discrete method

    print(d)

    patchVectorLocations = getPatchVectorLocations(w, d, subdivisions, numberOfPatches)

    Z = np.zeros((numberOfPatches, numberOfPatches))
```

```
    capacitance_calculated = totalChargePerPlate / V_0

    c_calc.append(capacitance_calculated)

    capacitance_formula = epsilon_0 * (np.square(w * w) / d)

    c_formula.append(capacitance_formula)


plt.plot(d_plot, c_calc, label = "Capcitance calculated using discrete method")
plt.plot(d_plot, c_formula, label = "Capcitance calculated using approximation formula")
plt.xlabel('Gap distance (mm)')
plt.ylabel('Calculated capacitance (mF)')
plt.legend()
plt.show()
```

The calculated capacitances are plotted against each other as a function of the gap distance



From observation, the discrete method's results converge towards the approximation as the gap distance gets larger. However, as the distance gets smaller, the approximation diverges from the discrete method's results.

It is worth noting that the discrete method fails at smaller gap distances and produces a negative, invalid capacitance likely due to floating point errors incurred by the nature of discretization. Furthermore, as the gap distance increases, the calculated capacitance increases steadily until it converges to a fixed capacitance. This is converse to the approximation that exhibits an exponential decrease until convergence.