ECE 107 Project

Numerical Capacitance Extraction

Finite Parallel Plate Capacitors

Daniel Peters

A10103585

Assignment

Consider a parallel plate capacitor shown in Fig.1(a). The side width of the plates is w, the thickness of the plates is negligibly small, and separation between the plates is d . The plates are made of perfect electric conductor and the space between them is filled with vacuum.

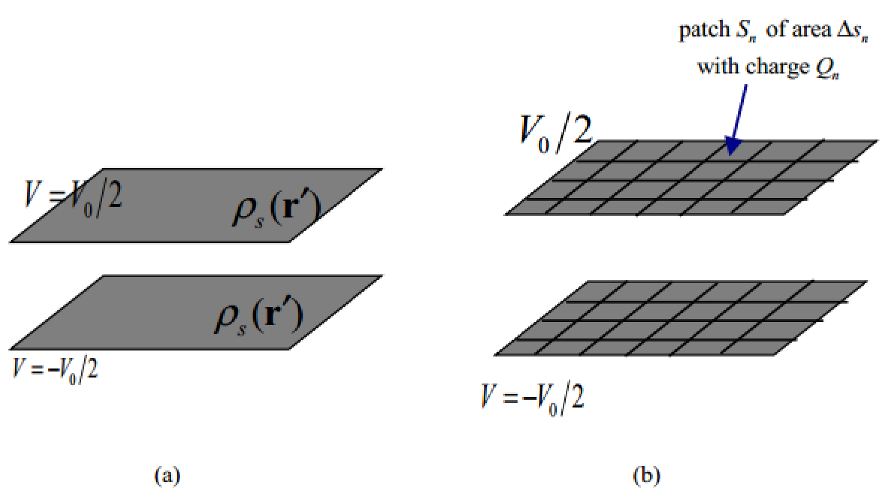


Fig. 1: (a) Parallel plate capacitor; (b) Discretized problem.

Method

I used the electric potential integral formula, that says voltage is the surface integral of surface charge divided by the distance between point on the surfaces and divided by the constants (4*pi*epsilon). Given that we know the voltage is Vo, and after calculating the distance between each point, we could use these matrices to solve for the unknown charge distribution.

$$\iint_{S^{(1)}+S^{(2)}} \underbrace{\frac{1}{4\pi\varepsilon_0 \,|\mathbf{r}-\mathbf{r}'|}}_{\text{Green's function}} \underbrace{\rho_s(\mathbf{r}')}_{\substack{\text{surface charge} \\ \text{distribution}}} \, dS' = \begin{cases} V_0/2; \mathbf{r} \in S_1 \\ -V_0/2; \mathbf{r} \in S_2 \end{cases}$$

By inverting the matrix of the distances between each point, and multiplying the matrix by the voltage at each point, we could arrive at an answer for the charge density at each point. Then, by adding up all of the charge and dividing by voltage, we got an approximation for the capacitance of the parallel plates.

$$\underline{\underline{Z}} : N \times N \text{ matrix}; Z_{mn} = \frac{1}{\Delta s_n} \iint_{S_n} \frac{ds'}{4\pi\varepsilon_0 |\mathbf{r}_m - \mathbf{r}'|} \Rightarrow \begin{cases} Z_{mn} \approx \dfrac{1}{4\pi\varepsilon_0 |\mathbf{r}_m - \mathbf{r}_n|}; \mathbf{r}_m \neq \mathbf{r}_n \\ Z_{mn} \approx \dfrac{1}{2\varepsilon_0 \sqrt{\pi \Delta s_n}}; \mathbf{r}_m = \mathbf{r}_n \end{cases}$$

$$\underline{Q} : N \text{ vector}; Q_n = \rho_s(\mathbf{r}_n)\Delta s_n$$

$$\underline{V} : N \text{ vector}; V_m = \begin{cases} V_0/2; \mathbf{r}_m \in S_1 \\ -V_0/2; \mathbf{r}_m \in S_2 \end{cases}$$

Implementation

I used C++ to write the program because I wanted to learn more about how that language works. I am a beginner, so it might not be the prettiest code. Also, I wrote the program from scratch – including the matrix inversion function (which utilized the Gaussian method for inverting a matrix). These implementations are not optimized, and are relatively slow. I wanted the whole project to be my own work. For the patch size, I was able to run the program with up to 20 patches per side, for a total of 400 patches. I did some simulations to see how much increasing the patch count changed the final result, and past 20 it didn't seem to have any huge effects, but rather a slow asymptotic decrease.

Parameters

Vo = 50V
PI= 3.14159
Eo = 8.854*10^(-12)

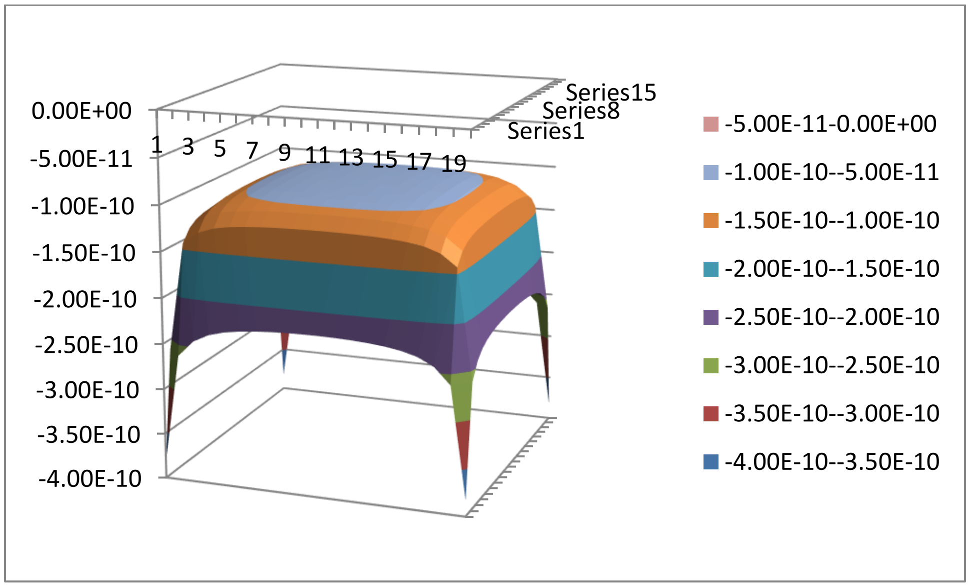
W=1cm and D=3mm

Vo = 50 D= 1.8 N=0   [N is the number of patches on each side]
Total Charge on One Plate: 5.20002e-008
Total Capacitance: 5.20002e-013
Estimated Capacitance: 2.95133e-013
Error Margin :1.76192

Each Unit is 1.5 mm
Each mm is 2 units
Distance Between = 6 Units = 3 mm
Length on Plate Side = 20 units = 10mm

This means there is 76% more capacitance than expected with this capacitor arrangement than if you used the C = EA/D formula.

Visual Simulation with N=40 (each unit is .5mm)

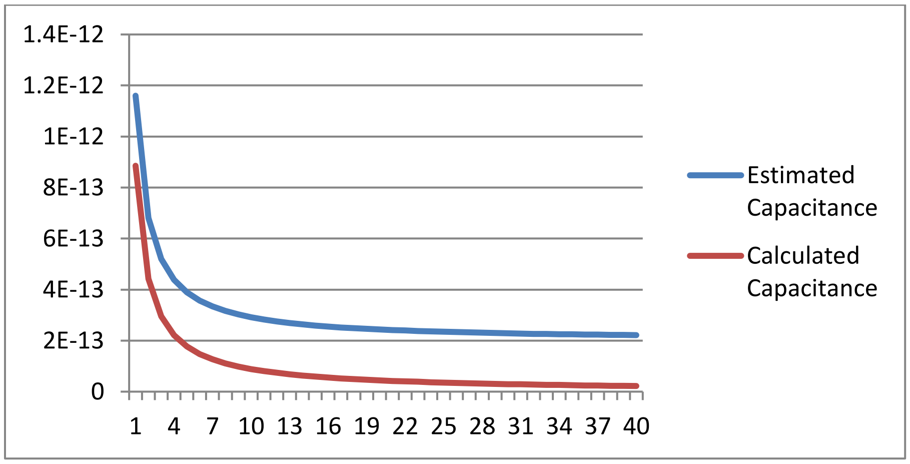


Description

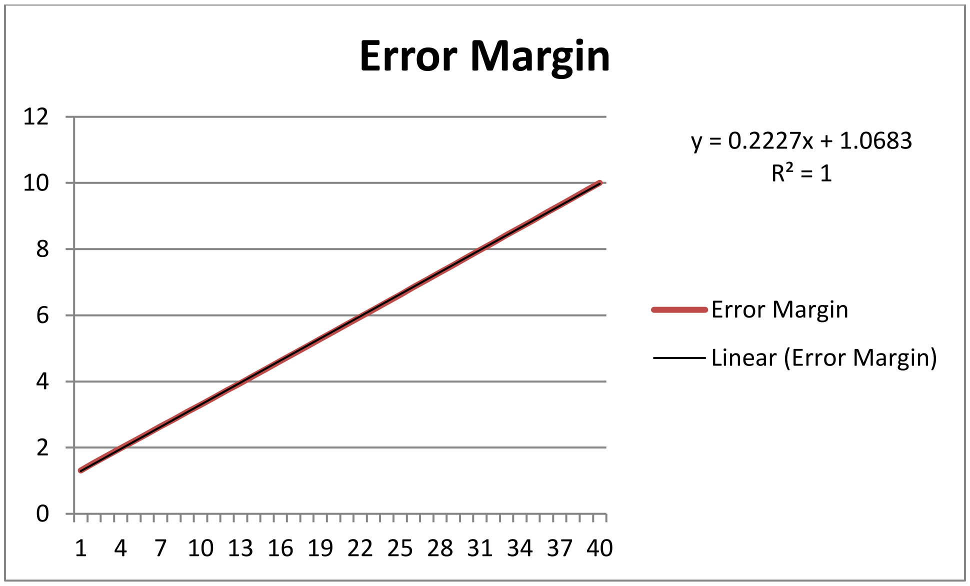
As we might expect, the charge is concentrated on the edges more than the center, and the corners are particularly dense with charge. This is because the electric field is highest on edges and sharp points, and the charge collects on these imperfections. The estimation equation for capacitance would predict the middle level, but since the plate is not infinite, charges unexpectedly build up on the edges and corners making the capacitance higher than expected.

Relationship with Distance

I ran a similar simulation for every integer value of d between 1 and 40 mm (4cm). This is the graph of the calculated capacitance using the general infinite parallel plate formula and then the estimated capacitance for our finite plate at using the matrix solving method.



It appears that both estimated and calculated capacitance has strong 1/d dependence with respect to distance. They both drop very quickly and then slow in their decent as d gets very large. However, the finite plate capacitance does not fall off as quickly as the infinite plate capacitance, leading to a higher charge on the plates. As discussed before, this is because of accumulation of charge on edges and corners. Let also take a look at the ration of calculated vs. estimated to get an idea of 1) what percentage higher it is at a given distance and 2) what relationship they may exhibit.



Wonderfully, they display a perfect linear dependence (R-squared is 1). This may seem surmising at first, but allow me to explain the reasoning for this. At small distances of separation, the edges are negligible because compared to the distance between plates; the size of the plate seems infinite. You could imagine a finite plate and infinitesimal distance producing the ideal 1:1 error margin. This makes sense since our line has a y-intercept of just about 1. However, as d gets larger, the two plates become more and more point like as the distance grows. Imagine that the distance is twice the width, now the width becomes almost negligible in the equation, and the entire plate seems like an "edge". Another way to explain this is to say that our infinite plate assumption works as long as the ratio of width to distance is very large. As that ratio decreases and goes to 0, the plates become simple points.

Code (in C++):

```
//BEGIN CODE

#include <iostream>
#include <vector>
#include <fstream>
#include <string>
#include <cmath>
#include <ctime>
#include <sstream>

//Distances in mm
#define SQRTN 20
#define N SQRTN*SQRTN
#define PI 3.1415926
#define Eo 8.854*pow(10,-12)
#define Vo 50

using namespace std;

float absval(float x){
if(x<=0){
   return -x;
}
else
   return x;
}

void MultiplyMatrices(vector<float>& I,float V[N*2],float Q[N*2]){
int i,j=0;

for(i=0;i<N*2;i++){
for(j=0;j<N*2;j++){
Q[i]+= I[2*N*i+j]*V[j];
}
}

return;
```

```
}

void inverse_gaussian(vector<float>& M,vector<float>& I){
  int k,l,i,j;
  float factor;
  float coefficient;

  for(k=0; k<N*2; k++){
     // cout << "Pivot = " << k << endl;
    //one in the pivot
    factor = M[2*N*k+k];
    for(i=0;i<N*2;i++){
      M[2*N*k+i] = M[2*N*k+i]/factor;
      I[2*N*k+i] = I[2*N*k+i]/factor;
        }
    //zeroing the column
    for(l=0; l<N*2; l++){
      coefficient = M[2*N*l+k];
      for(i=0;i<N*2;i++){
        if(k!=l){
        M[2*N*l+i]-=coefficient*M[2*N*k+i];
        I[2*N*l+i]-=coefficient*I[2*N*k+i];
        if(I[2*N*l+i]==0)
           I[2*N*l+i]=0;
        if(M[2*N*l+i]==0)
           M[2*N*l+i]=0;
        }
      }
    }
  }
  return;
}

float determinant(vector<float>& M, int x, int y, int Nt){
int i,j;
float det=0;

for(i=0; i<=Nt; i++){
```

```
det+=M[N*y+(x+i)]*determinant(M, x+1, y+1, Nt-1);
}



}

void inverse_analytical(vector<float>& M,vector<float>& I, int Nt){

  return;
}

float absdist(int i,int j,float k,int l,int m,float n){
float dist=0;
dist = sqrt(pow((i-l),2)+pow((j-m),2)+pow((k-n),2));
//cout  << i << " " << j << " " << k << "::" << l << " " << m << " " << n << "::" << dist << endl;
return dist;
}

void Identity(vector<float>& I){
int l,k;
for(k=0;k<N*2;k++){  //Create Identity Vector
for(l=0;l<N*2;l++){
   if(k==l){
      I[2*N*l+k]=1;
      }
   else{
      I[2*N*l+k]=0;
      }
  }
  }

}


int main(){

int i,j,k,l,m,n;  // i=x' j=y' k=z'   l=x m=y n=z
int counter1=0;   //Counter for the NxN Z array
int counter2=0;   //Counter for the NxN Z array

vector<float> Z(4*N*N);   // NxN Z Vector
float V[N*2]; // Nx1 Voltage (V) Vector
float Q[N*2]; // Nx1 Charge (Q) Vector
vector<float> I(4*N*N);  // Nx1 Identity Vector
float D;
time_t timer;
time(&timer);
string filename;
ofstream output, log;
std::ostringstream s;

log.open("log.txt");

float TotalQ;         // Sum of Q
float surface1[SQRTN][SQRTN];
float surface2[SQRTN][SQRTN];

for(int d=1; d<=40; d++)  //d is in mm
{
counter1=0;
counter2=0;

D = d*(((float)SQRTN)/(10));
cout << "D(in units)=" << D << endl;
cout << "Each unit is X mm:" << 10/((float)SQRTN) << endl;
cout << "Each mm is X units:" << ((float)SQRTN)/10 << endl;

s.str(std::string());
s <<  "Output_D_" << D << "_N_" << N << "__" << timer << ".txt";
filename=s.str();
cout << filename <<endl;
output.open(filename.c_str());  //Open Output File

Identity(I);  // Set Identity Vector

   for(n=0;n<=1;n++){
        for(l=0;l<SQRTN;l++){
          for(m=0;m<SQRTN;m++){  // rm = r
            for(k=0;k<=1;k++){
              for(i=0;i<SQRTN;i++){   //rn = r'
                for(j=0;j<SQRTN;j++){
                  if((i!=l)||(j!=m)||(k!=n)){
                    Z[counter1*2*N + counter2]=
1/(4*PI*Eo*absdist(i+.5,j+.5,k*D,l+.5,m+.5,n*D));  // rm=/=rn
                  }
                  else{
                    Z[counter1*2*N + counter2]= 1/(2*Eo*sqrt(PI));    // rm==rn
                  }
                  counter2++;
                  }
                }
              }
            if(n==1){        //If top plate, set V=Vo/2
            V[counter1]=Vo/2;
            }
            else         //If bottom plate, set V=-Vo/2
            {
            V[counter1]=-Vo/2;
            }
            Q[counter1]=0;
            counter1++;
            counter2=0;
          }
        }
    }


cout << "Initial Z Complete" << endl;

inverse_gaussian(Z,I);  // Invert Z

cout << "Inversion Complete" << endl;

MultiplyMatrices(I,V,Q);   //  Q = Z^-1 * V
```

```
cout << "Multiplication Complete" << endl;

TotalQ=0.0;        // Sum of Q

counter1=0;
counter2=0;

output << "Surface 1" << endl;

for(l=0;l<SQRTN;l++){
for(m=0;m<SQRTN;m++){
surface1[l][m]=Q[counter1];
output /*<< l <<" " << m << ":: "*/ << surface1[l][m] << ",";
counter1++;
}
output << endl;
}

output << endl << endl << endl << endl;

output << "Surface 2" << endl;
for(l=0;l<SQRTN;l++){
for(m=0;m<SQRTN;m++){
surface2[l][m]=Q[counter2+N];
output /*<< l <<" " << m << ":: "*/<< surface2[l][m] << ",";
counter2++;
}
output << endl;
}

output << endl << endl << endl << endl;


output << "Charge Layout" << endl;
for(i=0;i<N*2;i++){       //Sum Total Q
output << Q[i] << endl;
TotalQ+=absval(Q[i]);     //absolute value - to make positive
}

output << endl << endl << endl << endl;



output << "Current Run Time: " << timer << endl;
output << "Vo = " << Vo << "  D (in units)= " << D << "  N (in units)=" << N << endl;
output << "###################" << endl;
output << "Results" << endl;
output << "###################" << endl;
output << "Total Charge on One Plate: " <<TotalQ/2 << endl;      //Total Charge (One Plate)
output << "Total Capacitance: "<< TotalQ/(Vo*2)/(100*SQRTN) << endl;    //Total Capacitance
output << "Estimated Capacitance: "<< Eo*(N/D)/(100*SQRTN) << endl;       //Estimate
Capacitance - For Comparison
output << "Error Margin :"<< (TotalQ/(2*Vo))/(Eo*N/D) << endl;  //Error
log << "Each Unit is (in mm):" << 10/((float) SQRTN) << " " << "D (in mm): " << D/((float)
SQRTN/10) << " " <<"Error Margin :"<< (TotalQ/(2*Vo))/(Eo*N/D) << endl;  //Error

cout << "All Complete" << endl;

output.close();

}

log.close();

return 0;

}
```