Square Conductors and One Dimensional Electrostatics
Project #5

Inst. Pavel Snopok, Spring 2015

Michael Lee

April 26, 2015

Abstract

Partial differential equations are tools we use to describe the everyday world. In the real world, physical quantities change continuously in time and space. This make fields an invaluable mathematical instrument for us to understand nature. In this report, we explore electrostatic potentials to examine their behavior under certain boundary conditions. First, we solve the electrostatics problem of a box inductor sparking inside a larger grounded box. We then examine the one dimensional electrostatics case when moving in between a parallel plate capacitor.

1 Introduction

1.1 Partial Differential Equations

In a physical system, we use a field to describe that field by placing a value at each point in space and time. In electrodynamics, we use the field $U(x, y, z, t)$ to describe the *electrostatic potential* and the quantities are independent. The field itself, however, is affected by neighboring points as time evolves, so $U(x_0, y_0, z_0, t)$ affects $U(x_1, y_1, z_1, t)$ and so on.

Since the electrostatic potential field changes in accord to four *independent* variables, we must write the dynamic equations in terms of partial derivatives. For our case, however, we will only be examining the two dimensional case of a square conductor inside of a larger grounded conductor. The most general form of the potential in two dimensions is:

$$A\frac{\partial^2 U}{\partial x^2} + 2B\frac{\partial^2 U}{\partial x \partial y} + C\frac{\partial^2 U}{\partial y^2} + D\frac{\partial U}{\partial x} + E\frac{\partial U}{\partial y} = F$$

The coefficients describe the class of the partial differential equation and also describes if it is dependent on a first ordered derivative, second ordered derivative, or both.

Coefficient	Boundary Condition		Form
If $d = AC - B^2 > 0$	Elliptic	Poisson's	$\nabla^2 U(x) = -4\pi\rho(x)$
If $d = AC - B^2 = 0$	Parabolic	Heat	$\nabla^2 U(x) = -a\frac{\partial U}{\partial t}$
If $d = AC - B^2 < 0$	Hyperbolic	Wave	$\nabla^2 U(x) = \frac{1}{c^2}\frac{\partial^2 U}{\partial t^2}$

This table quickly demonstrates that: a parabolic equation contains a first order derivative in x or y and a second order derivative in the other, a hyperbolic equation contains second order derivatives of all the variables with opposite signs when placed on the same side of the equation, and a elliptical equation contains second ordered derivative of all the variables with all having the same sign when on the same side of the equation. For there to exist a unique solution for a given physical situation, there has to be sufficient boundary conditions to specify the problem.

Classification	Condition
Dirichlet	If the boundary condition is the value of the solution on a surrounding closed surface.
Neumann	If the boundary condition is the value of the normal derivative on the surrounding surface.
Cauchy	If the value of both the solution and the derivative are specified on a closed boundary.

These classifications gives us the type of problem we are facing regarding partial differential equations. By classifying them this way, we can use the proper theorems to attack the problem.

1.2 Electrostatics

The electric potential satisfies Poisson's equation from Maxwell's equations:

$$\nabla \cdot E = \frac{\rho}{\epsilon}$$

$$\nabla \cdot E = \nabla \cdot -(\nabla \cdot V) = -\nabla^2 V = -\frac{\rho}{\epsilon} = -4\pi\rho(x)$$

Poisson's equation, this is physical situation, describes a potential energy field due to a charge distribution. Since it is Poisson's equation, this means that the equation is parabolic and contains a first order derivative in time and contains a second order derivative in space. The first derivative term, of course, is the magnetic vector potential A and the second order derivative term is the charge distribution, $-4\pi\rho(x)$.

For the field that occurs from static charge, we have:

$$\nabla^2 U = -4\pi\rho(x)$$

For the charge free regions, the PDE satisfies Laplace's equations where:

$$\nabla^2 U = 0$$

since $\rho = 0$. For both cases, the potential depends simultaneously on x and y from the relationship in two dimensions:

$$\nabla^2 U(x,y) = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2}$$

The analytical solution to Laplace's equation exists as an infinite series and if we assume that we obtain solution as a product of U(x) and U(y), we get:

$$\frac{d^2U(x)}{U(x)dx^2} + \frac{d^2U(y)}{U(y)dy^2} = 0.$$

We can then separate the x and y variable terms since they are independent. Now, the only way that the two terms together can equal zero is if they both are constants with opposite signs.

$$-\frac{d^2U(x)}{U(x)dx^2} = \frac{d^2U(y)}{U(y)dy^2} = k^2$$

Then the solutions become:

$$\frac{d^2U(x)}{dx^2} + k^2U(x) = 0$$

$$\Rightarrow U(x) = A\cos(kx) + B\sin(kx)$$

$$\frac{d^2U(y)}{dy^2} - k^2U(y) = 0$$

$$\Rightarrow U(y) = Ce^{-kx} + De^{+kx}$$

This tells us that the solutions for U(x) are periodic and for U(y), exponential if x is along the charge distribution. For the case of the square box, this means that parallel to the box, the potential will have a periodic form. Perpendicular to the box, the form will be exponential.

The algorithm that we will use to solve the box potential problem is the finite-difference method, that is, solving the derivative numerically.

2 Analysis

2.1 Square Conductor

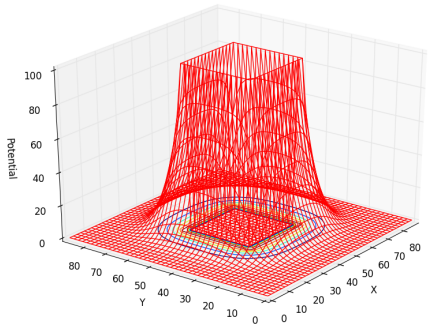

Figure 1: U(x,y) at V=100.

The potential is zero inside of the box because of Gauss's law. We see that the potential drops off the edge of the box like an exponential decay while parallel to the side, it looks like a sine curve. The electric field is most intense at the location of the greatest concentration of field lines. We see then from Figure 2 this is at the bottom corners of the box where the field line density is greatest. To reduce sparking from occurring, you would probably want to change the shape of the box into a *cylinder* instead so that the rounded edges do not contribute much to the electric field potential.

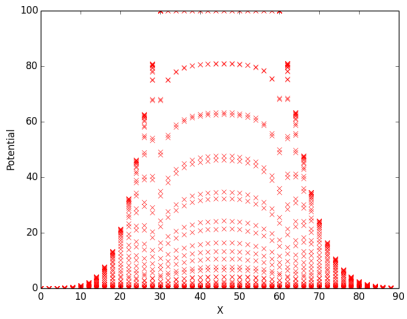

Figure 2: U(x) is periodic along x and exponential along y.

2.2 One Dimensional Electrostatics

We use the finite element method to find the approximate solution to a given one dimensional electrostatic potential. For the choice of parameters $a = 0$, $b = 1$, $U(a) = 0$, and $U(b) = 1$, we have the equation:

$$U(x) = \frac{-x \cdot (x-3)}{2}.$$

Taking the second derivative, we obtain:

$$\nabla U(x) = -1.$$

Observing the plots from $N = 3$, $N = 10$, $N = 100$, $N = 1000$, the general trend is that the finite element method correlate more and more to the analytical potential as N, the number of iterations, increases.

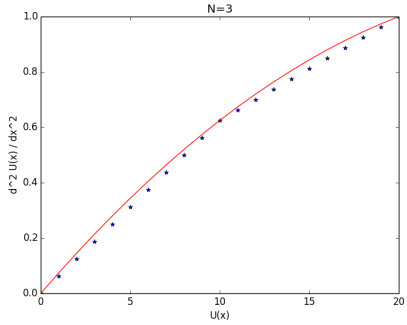

Figure 3: The FEM when N=3. $U(x) = \frac{-x*(x-3)}{2}$.

The error for $N = 3$ peaks at points 5 and 15. The error when $N = 10$ is erratic as shown by figure 7. Once $N = 100$ the error has even out to be similar to $N = 1000$, where the largest absolute error occurs at the 10th point of the finite element method.

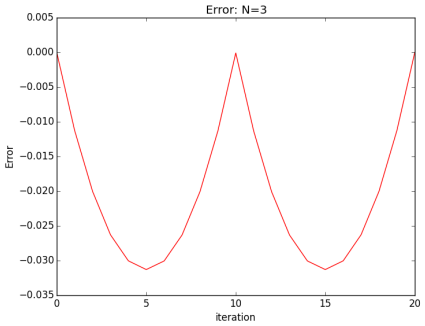

Figure 4: Error of N=3.

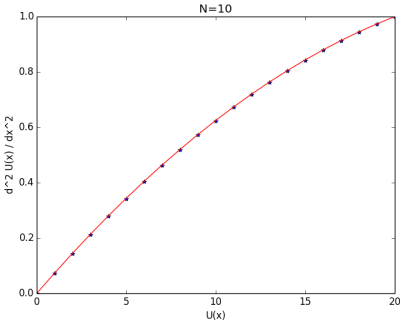

Figure 5: The FEM when N=10. $U(x) = \frac{-x*(x-3)}{2}$.

When the charge distribution is

$$\rho = \frac{\frac{1}{2} - x}{4\pi}$$

we obtain the graph in figure 10. At $N = 3$, the FEM plot almost fits the actual potential curve. It is the same case for $N = 1000$, but the relative error is high as shown in Figure 12. Interestingly, it has a local maximum absolute error at iteration 10, differing from when $N = 3$.

3 Conclusion

We found that the solution to the sparking problem was to change the geometry of the box into a cylinder. Using the finite element method, we were able to examine $\nabla U(x)$ as $U(x)$ changed as of function of x. This gives us a good picture of what happens when an object moves up and down in between two parallel plate capacitors and experiences the changes in the electric field.

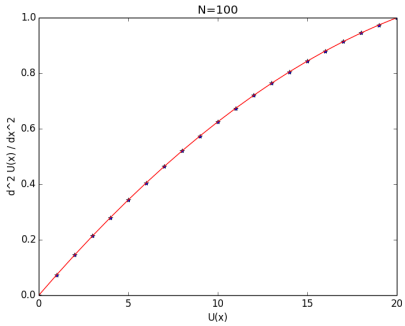

Figure 6: The FEM when N=100. $U(x) = \frac{-x*(x-3)}{2}$.

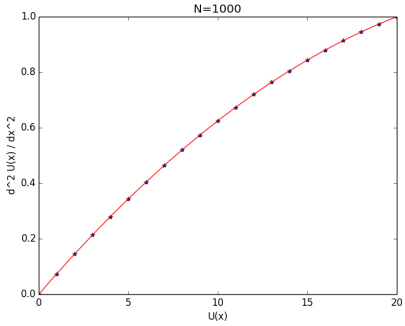

Figure 7: The FEM when N=1000. $U(x) = \frac{-x*(x-3)}{2}$.

References

[1] R. Landau, Manual J. Paez, Cristian C. Bordeianu. *A Survey of Computational Physics*, 2010: Princeton University Press.

[2] D. Beazley and Brian K. Jones. *The Python Cookbook*, 2013: O'Reilly.

[3] Newman, M. E. J. *Computational Physics*, 2013: Createspace.

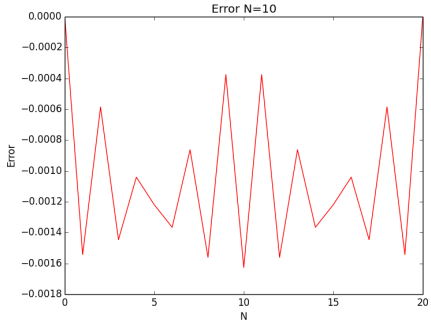

Figure 8: The relative error when N=10.

Figure 9: The relative error when N=100.

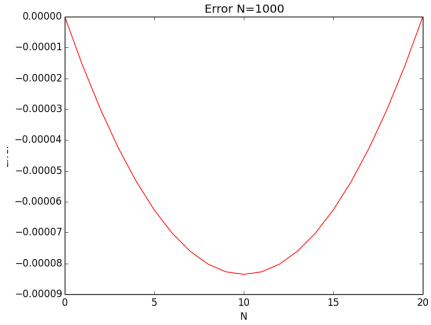

Figure 10: The relative error when N=1000.

4 Appendix

5 Laplace Forward Difference

```
"""
Written by Michael Lee.

PHYS440: Computational Physics
Project 5, Square Conductors and One Dimensional Electrostatics

"""

from numpy import *
import matplotlib.pylab as plt;
from mpl_toolkits.mplot3d import Axes3D

def functz(V):
    z = V[X,Y]
    return z

def voltage(volt, x_edge1, x_edge2, y_edge1, y_edge2):
    for j in range(y_edge1, y_edge2+1):
        V[x_edge1, j] = volt
        V[x_edge2, j] = volt
    for i in range(x_edge1, x_edge2+1):
        V[i, y_edge1] = volt
        V[i, y_edge2] = volt
    return V[i,j]

def finite_difference(V, x_edge1, x_edge2, y_edge1, y_edge2):
    for iter in range(Niter):
        if iter%10 == 0: print(iter)

        for i in range(x_edge1):
            for j in range(1, 91):
                V[i,j] = 0.25*(V[i+1,j]+V[i-1,j]+V[i,j+1]+V[i,j-1])

        for i in range(x_edge2+1, 90):
            for j in range(1, 91):
                V[i,j] = 0.25*(V[i+1,j]+V[i-1,j]+V[i,j+1]+V[i,j-1])

        for j in range(y_edge1):
            for i in range(x_edge1, x_edge2+1):
                V[i,j] = 0.25*(V[i+1,j]+V[i-1,j]+V[i,j+1]+V[i,j-1])

        for j in range(y_edge2+1, 90):
            for i in range(x_edge1, x_edge2+1):
                V[i,j] = 0.25*(V[i+1,j]+V[i-1,j]+V[i,j+1]+V[i,j-1])

    """
    The finite difference method takes points on the graph and calculates
    its potential, V[i,j], based on the potentials of its four nearest neighbors.
    If its neighbors have V[i,j] = 0, then it is also zero. Since we seeded the
    graph with lines of V = 100, we will get points of decreasing potential near those
    lines, since we take the average of the four values. We iterate the calculation of
    each point's potential for 'Niter' times, with the values V[i,j] still on the stack.

    The code below is probably more efficient since it contains one less
```

```
    nested for-loop. However, because the above uses the same pattern
    for X and Y, I think the above is easier to understand.

    """

        # for i in range(x_edge1, x_edge2):
        #     for j in range(1, y_edge1):
        #         V[i,j] = 0.25*(V[i+1,j]+V[i-1,j]+V[i,j+1]+V[i,j-1])
        #     for j in range(y_edge2+1, 90):
        #         V[i,j] = 0.25*(V[i+1,j]+V[i-1,j]+V[i,j+1]+V[i,j-1])

    return V[i,j]

if __name__ == '__main__':

  print("Initializing")
  Nmax = 100; Niter = 70; V = zeros((Nmax+1, Nmax+1), float)

  print("Working hard, wait for the figure while I count to 60")

  voltage(100, 30, 60, 30, 60)
  finite_difference(V, 30, 60, 30, 60)

  x = range(0, 90, 2);  y = range(0, 90, 2)
  X, Y = p.meshgrid(x,y)

  Z = functz(V)

  fig = plt.figure()
  ax = Axes3D(fig)
  ax.plot_wireframe(X, Y, Z, color='r')
  ax.contour(X, Y, Z, offset=0)
  ax.set_xlabel('X')
  ax.set_ylabel('Y')
  ax.set_zlabel('Potential')

  p.show()
```

6 Finite Element Method

```
"""
From "A SURVEY OF COMPUTATIONAL PHYSICS", Python eBook Version
by RH Landau, MJ Paez, and CC Bordeianu
Copyright Princeton University Press, Princeton, 2011; Book  Copyright R Landau,
Oregon State Unv, MJ Paez, Univ Antioquia, C Bordeianu, Univ Bucharest, 2011.
Support by National Science Foundation , Oregon State Univ, Microsoft Corp

Code adapted by Michael Lee.

PHYS440: Computational Physics
Project 5, Square Conductors and One Dimensional Electrostatics

"""

from numpy import *
from numpy.linalg import solve
import matplotlib.pyplot as p

```

```
19 N = 1000
20 h = 1./(N - 1)
21 u = zeros((N),float); A = zeros((N,N),float); b = zeros((N,N),float)
22 x2 = zeros((21),float); u_fem = zeros((21),float); u_exact = zeros((21),float)
23 error = zeros( (21), float); x = zeros( (N),  float)
24
25 x = arange(N)*h
26 for i in range(0, N):
27     x[i] = i*h
28
29 def lin1(x, x1, x2):
30     return  (x - x1)/(x2 - x1)
31
32 def lin2(x, x1, x2):
33     return (x2 - x)/(x2 - x1)
34
35 def f(x):
36     return 1
37
38 def int1 (min, max):
39     no = 1000
40     sum = 0.
41     interval = ( (max  - min) /(no - 1) )
42     for n in range(2, no, 2):                          # Loop odd points
43         x = interval * (n - 1)
44         sum   += 4 * f(x)*lin1(x, min, max)
45     for n in range(3, no, 2):                          # Loop even points
46         x = interval * (n - 1)
47         sum   += 2 * f(x)*lin1(x, min, max)
48     sum += f(min)*lin1(min, min, max) + f(max)*lin1(max, min, max)
49     sum *= interval/6.
50     return (sum)
51
52 def int2 (min, max):                                   # Simpson
53     no = 1000
54     sum = 0.
55     interval = ( (max  - min) /(no - 1) )
56     for n in range(2, no, 2):                          # Loop odd points
57         x = interval * (n - 1)
58         sum   += 4 * f(x)*lin2(x, min, max)
59     for n in range(3, no, 2):                          # Loop even points
60         x = interval * (n - 1)
61         sum   += 2 * f(x)*lin2(x, min, max)
62     sum   += f(min)*lin2(min, min, max) + f(max)*lin2(max, min, max)
63     sum *= interval/6.
64     return (sum)
65
66 def numerical(x, u, xp, N):                            # interpolate numerical solution
67     y = 0.
68     for i in range(0, N - 1):
69         if ( xp >= x[i] and xp <= x[i + 1] ):
70             y = lin2(xp,x[i],x[i+1])*u[i] + lin1(xp, x[i], x[i+1])*u[i+1]
71     return y
72
73 def exact(x):                                          # Analytic solution
74     u = -(x)*(x - 3.0) / 2.0
75     return u
76
77 if __name__ == '__main__':
```

```
78
79      for i in range(1, N):
80          A[i - 1, i - 1] = A[i - 1, i - 1] + 1./h
81          A[i - 1, i]  = A[i - 1, i]  -  1./h
82          A[i, i - 1]  = A[i - 1, i]
83          A[i, i]    = A[i, i] + 1./h
84          b[i - 1, 0]  = b[i - 1, 0] + int2(x[i - 1], x[i])
85          b[i, 0]    = b[i, 0] + int1(x[i - 1], x[i])
86
87      for i in range(1, N):                          # Dirichlet BC @ left end
88          b[i, 0] = b[i, 0] - 0.*A[i, 0]
89          A[i, 0] = 0.
90          A[0, i] = 0.
91
92      A[0, 0] = 1.
93      b[0, 0] = 0.
94
95      for i in range(1, N):                         # Dirichlet bc @ right end
96          b[i, 0] = b[i, 0] - 1.*A[i, N - 1]
97          A[i, N - 1] = 0.
98          A[N - 1, i] = 0.
99      A[N - 1, N - 1] = 1.
100     b[N - 1, 0] = 1.
101     sol = solve(A, b)
102
103     for i in range(0, N):     u[i] = sol[i, 0]
104     for i in range(0, 21):   x2[i] = 0.05*i
105     for i in range(0, 21):
106         u_fem[i] = numerical(x, u, x2[i], N)
107         u_exact[i] = exact(x2[i])
108         error[i] = u_fem[i] - u_exact[i]                      # Global error
109
110     figure = p.figure()
111     x = u_fem
112     y = u_exact
113     p.plot(x, 'b*', y, 'r')
114     p.title('Finite Element Method; N=%d' % N)
115     p.xlabel('U(x)')
116     p.ylabel('d^2 U(x) / dx^2')
117     p.show()
118
119     p.title('Error: N=%d' % N)
120     p.plot(error, 'r')
121     p.ylabel('Error')
122     p.xlabel('iteration')
123     p.show()
124     p.plot(u_fem,'r',u_exact,'b*')
```