Johnathan von der Heyde – 2019

Project 4 – Electrostatic Potential and The Laplace Equation

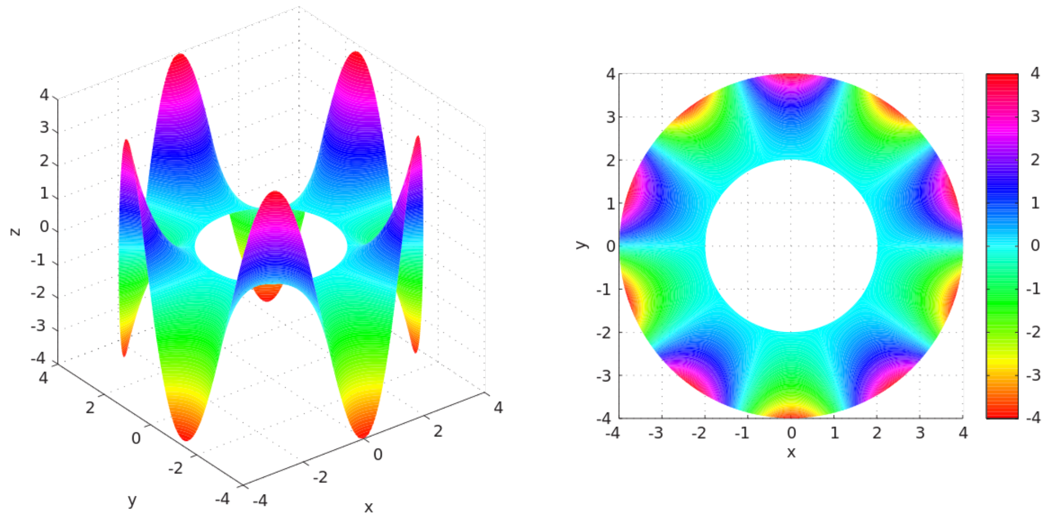
The Laplace equation, or Laplace's equation, comes up often in Physics – especially considering conserved quantities or, equivalently, isolated systems. It is a second order partial differential equation, usually written as

$$\nabla^2 f = 0$$

∇^2 is commonly referred to as the Laplace operator, and is equivalent to the divergence of the gradient of a function

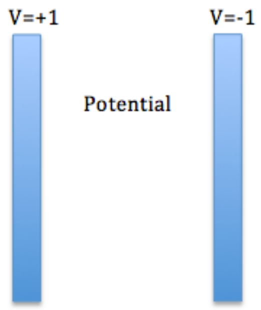
$$\nabla \cdot \nabla f$$

This value being set to zero is what defines the Laplace equation, which we can intuitively think of as a scalar function who's overall gradient (slope) is conserved. Hence why *Harmonic functions* obey Laplace's equation. Below is a visual example demonstrating the conserved slope of an annulus, courtesy of Wikipedia.



In this project we briefly explore how the Laplace equation can help us solve for the potential field between capacitor plates. Potential fields in general satisfy the Laplace equation, as they can always be expressed as differentiable manifolds which vanish relative to some zero-point. So it is a natural starting point for our project.

We begin by setting up an array grid for storing our potential values V[float][float]. It can be whatever size one wants. Then we initialize the Capacitor plates as left and right edges. The voltage for these plates is here modeled as +1 and -1, for the left and right plates respectively. Thus we have implicitly defined our zero-point, which will help us imagine what our graphs should look like. Below is a vague visualization of our setup.



The expanded Laplace equation, in 3D, is written as

$$\nabla^2 f(x,y,z) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} + \frac{\partial f}{\partial z} = 0$$

Which we must convert to a numerical solution. To do this, it helps to have a reasonable grip on what the Laplace equation is saying. Locally speaking, the Laplacian at (x, y, z) is *how different* that point is compared to its neighboring points – positive for being greater than its neighbors, and negative for being less than. In terms of the gradient, the Laplacian is positive within hills and negative in valleys. In terms of the divergence, the Laplacian is positive for *sources* and negative for *sinks*. And the fact that globally, we require this equation to be zero, means we're trying to *minimize the difference* between each point and its neighbors.

Of course this could be done trivially by having all points the same, but here we're insisting on at least two lines (capacitors) of distinct potential values. Therefore what we need is the next best thing, in terms of minimizing the difference between neighboring points. Numerically, this translates to the idea that each point is actually the *average* of its surrounding points.

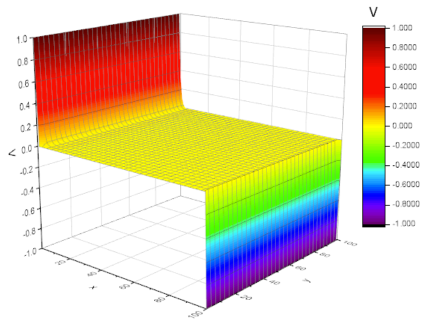
Thus we implement the following numerical version of the Laplace equation

$$V(x,y) = \frac{1}{4}(V[x+1][y] + V[x-1][y] + V[x][y+1] + V[x][y-1])$$

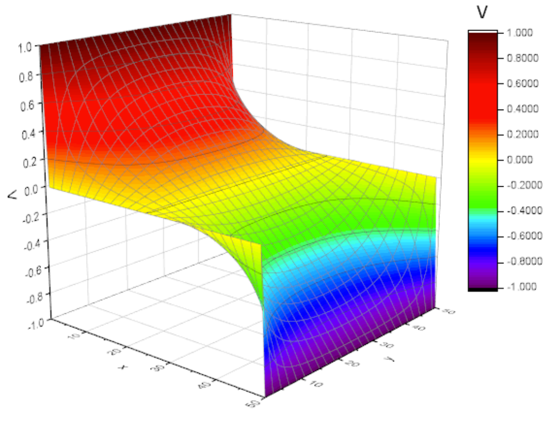
Here we also took advantage of the fact that our system is 2 dimensional. But before we can initialize all points with the average of their surrounding points, we must instill the boundary conditions. These include of course the capacitor plates,

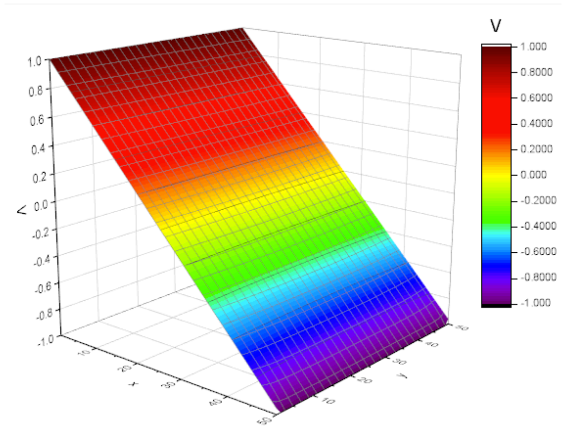
$$V_{c1}(x,y) = V[x_1][y_1] = +1 \ \forall \ x \wedge y \in c_1$$
$$V_{c2}(x,y) = V[x_2][y_2] = -1 \ \forall \ x \wedge y \in c_2$$

After initializing the plates and setting the rest of the values to zero, we run our equation and this is what we get



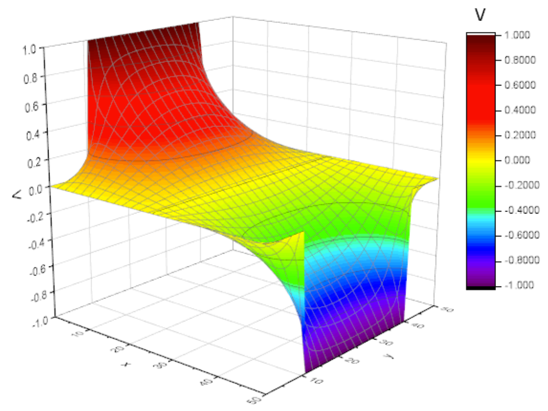
As we should have expected. However this is an inaccurate model of the *actual* potential between capacitor plates. What we have done is developed a 1st order solution to a 2nd order problem. The reason for this is twofold: First, we need more iterations in our calculation, so that we are averaging all the points until some convergence criteria is met (say, that the difference in slope between neighboring points is 0.001 – or something similar). Secondly, we need to give our calculation more to go on. This means we have to provide an *initial guess* for the values at the top and bottom edge, something other than zero. A natural and sufficient approach is to use a linear transition.

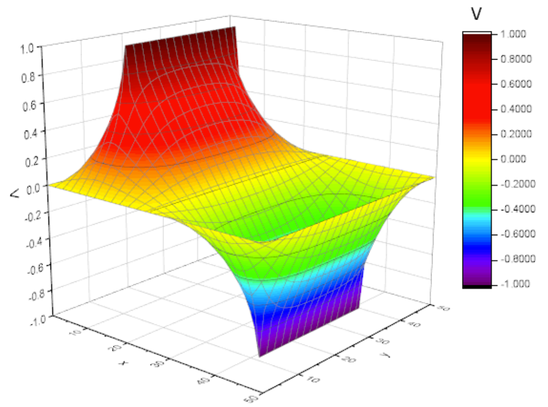




Above we see what happens after many iterations. The plot smooths out. On the left is the result when the top and bottom edges are left at zero, whereas on the right we see what happens when a linear guess is imposed at the top and bottom edge (connecting the capacitors).

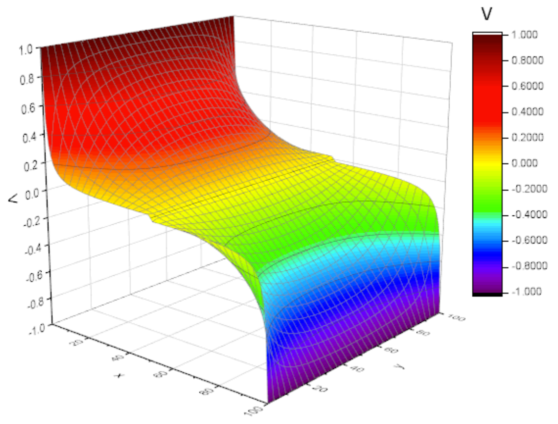
To get a more realistic model, we need to keep the starting assumptions, but also zoom out so that the zero-point is at infinity. The reason this is required is because otherwise our algorithm cannot adjust the initial edge guesses, since they lie on the boundary.

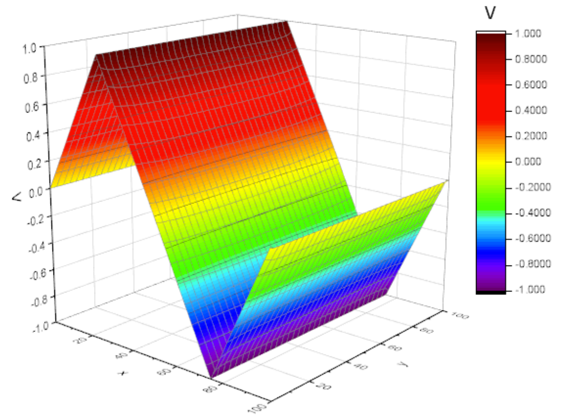




This is what happens when we shrink our capacitor width (left) and inset it (right).

Finally we have a good-looking model for the potential. Note that we could've imposed a $1/r$ guess for the edges connecting the capacitors, as opposed to a linear one. Yielding something like below (left). However, as it turns out, this initial guess isn't very important, nor need it be accurate – that's because after zooming out with plentiful iterations, the algorithm will nevertheless end up one of the above 2 plots.



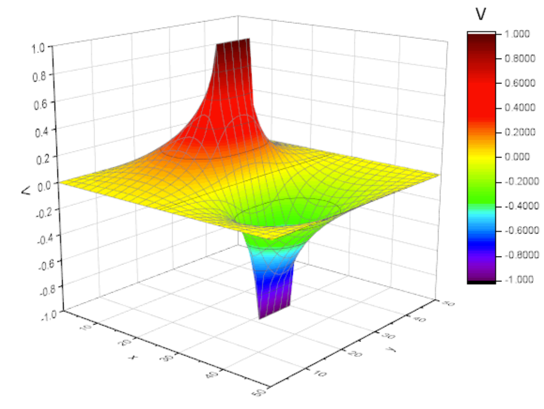


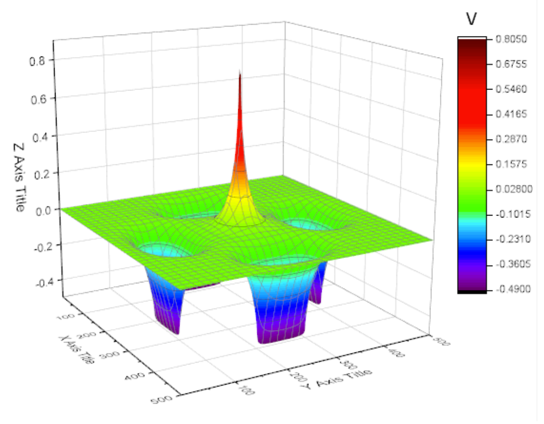
For fun on the above right, we see what happens when we don't zoom out, yet inset our capacitors. Again, the algorithm is not allowed to evaluate the boundaries, so it has no choice but to average according to their linear quality.

To conclude, it's important to realize all of the above plots satisfy the Laplace equation, as is evident by the fact that their area integral is zero. The method we used is actually the *Jacobi method*, which evolves the system according to the diffusion equation, so as to reach a time independent state. I find this all very fascinating and wish I had more time to explore. If I had more time I'd like to develop the associated derivative, i.e., in this context, the electric field. The numeric estimation equation is simple enough

$$E[x][y] = -((V[x][y+1] - V[x][y-1])/(2)) - ((V[x+1][y] - V[x-1][y])/(2))$$

However the boundary conditions proved too challenging for the time being.





Above are some supplemental plots showing what you can do with the Laplace equation, involving pseudo point charges etc., and below that is my C code implementation – completely adjustable.

```
//Johnathan von der Heyde 2019
//This program models the electrostatic potential
between adjustable capacitor plate bounds

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define max 100          //dimension of the grid
#define Lheight 1        //these ease calculations
#define Rheight -1       //height is value of capacitors
#define Linset 25        //inset is how close they are
#define Rinset max-1-25
#define width 25         //width is how wide (relative to
edge)

double V[max][max];              //index V[0-49][0-49]
int x, y, i, iter, count, Ldist, Rdist;    //coordinates and
iteration vars
double dv = 0, prev, final, last;          //for checking
convergence

void initialize();    //zeros the plane and sets boundary
conditions
void guessedge();     //makes starting assumptions for
edge values
void update(int i);   //averages coordinates and tracks
convergence
void printV();        //allows us to check on V before
plotting
void writeout();    //writes data to "laplace.dat"

int main() {
 //Input
  printf("Please Input # Iterations\n");
  scanf("%d", &iter);
 //Bounds
  initialize();
  guessedge();
 //Update
  for (i=1; i<=iter; i++) update(i);
  printV();
 //Convergence
  if (fabs(final - last) > 0.00001)
    printf("program failed to converge\n");
 //Output
  writeout();
  return 0;
}

void initialize() {
  for (x=0; x<max; x++) {
    for (y=0; y<max; y++) V[x][y] = 0;
  }
}

void guessedge(){ //edge values are guessed relative to
capacitor plates
 //0 to left capacitor
  Ldist = max/2-Linset;
  for (count=0, y=Linset; y>0; y--, count++)
    V[width/2][y]     =     V[max-1-width/2][y]     =
(double)Lheight*(1-((double)count/(double)Ldist));
 //left capacitor to middle
  for (count=0, y=Linset; y<max/2; y++, count++)
    V[width/2][y]     =     V[max-1-width/2][y]     =
(double)Lheight*(1-((double)count/(double)Ldist));
 //middle
  V[width/2][max/2] = V[max-1-width/2][max/2] = 0.0;
 //middle to right capacitor
  Rdist = Rinset-max/2;
  for (count=0, y=Rinset; y>max/2; y--, count++)
    V[width/2][y]     =     V[max-1-width/2][y]     =
(double)Rheight*(1-((double)count/(double)Rdist));
 //right capacitor to max
  for (count=0, y=Rinset; y<max-1; y++, count++)
    V[width/2][y]     =     V[max-1-width/2][y]     =
(double)Rheight*(1-((double)count/(double)Rdist));
}

void update(int i) {
 //laplace equation and convergence criteria
  prev = dv;
  for (x=1; x<max-1; x++) {
    for (y=1; y<max-1; y++) {
      V[x][y]        =        0.25*(V[x+1][y]+V[x-
1][y]+V[x][y+1]+V[x][y-1]);
      dv += fabs(V[x][y]);            //dv is total V per
iteration
    }
  }
 //convergence tracking
  if(i==iter-1) last = fabs(dv-prev);      //saves 2nd to
"last" delta V
  else final = fabs(dv-prev);              //compares that to
"final" delta V

 //capacitor plates
  for (x=width/2; x<max-width/2; x++) {       //range
modifies capacitor width
    V[x][Linset] = Lheight;               //inset must be <
height
    V[x][Rinset] = Rheight;
  }
}

void printV() {
  printf("\n----V\n");
  for (x=0; x<max; x++) {
    for (y=0; y<max; y++) {
      printf("%2.2lf\t", V[x][y]);
    } printf("\n");
  } printf("----V\n\n");
}

void writeout() {
  FILE *f;
  f = fopen("laplace.dat","w");
    for (x=0; x<max; x++) {
      for (y=0; y<max; y++) {
        fprintf(f, "%lf\t", V[x][y]);
      }
      fprintf(f, "\n");
    }
  fclose(f);
} //END
```