Semiconductor Device Modeling

1 Introduction

Semiconductors play an essential role in a number of electronic devices due to the enormous range of resistive values they can exhibit when tuned by carriers of electric charge. This tuning of electrical conductivity (called doping) occurs by adding impurities to a pure semiconducting material, such as silicon. This doping introduces a surplus of electrons (known as n carriers for negative charge) or holes (known as p carriers for positive charge) in the material, which in turn influence conductivity. One of the most important applications of semiconducting materials is the operation of metal-oxide semiconductor field effect transistors (MOSFETs), pictured below.

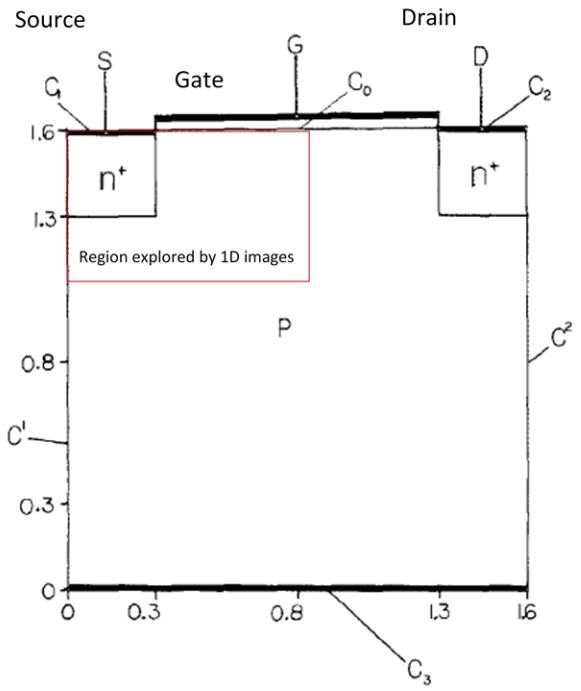


Figure 1: Image of a Two Dimensional MOSFET Model [1]

MOSFETs, like other field effect transistors, use electric fields in order to modulate the current that passes through the circuit component. For more information on semiconductors and field-effect transistors, additional resources have been included in the references section [2].

The ubiquity of MOSFETs and their dependence on electric fields make them a useful subject for the application of numerical methods to model the voltage potentials within the device. By solving Poisson's equation for electric potential while simultaneously enforcing the continuity condition for the movement of n and p type carriers, as outlined in I.D. Mayergoyz article in the Journal of Applied Physics [1], we can obtain the expression for a local potential. Generally, when handling Poisson's equation in a region, a numerical solution can be reached via the relaxation of the finite difference Laplace Equation [3] [4]. However, the continuity of the charge carriers cause the Poisson equation to become nonlinear, thus preventing an analytical solution of the potential

from being reached at a given interior point. In order to circumvent this issue, Newton's root-finding method can be used to evaluate the finite difference equations which govern the interior of the MOSFET.

By applying Newton's Root-Finding method and then utilizing Successive Relaxation, I aim to model the potentials inside of a MOSFET with the two dimensional geometric configuration shown in figure (). After finding a solution which agreed with Mayergoyz's results, I then sought an ideal relaxation parameter ω for which to perform further tests. Finally, by tuning the parameters of the code, I was able to model the behavior of the potentials within the MOSFET with respect to a variety of factors, such a carrier concentration and applied voltages.

2 Theory

Inside of a closed region governed by conservative forces, like those that arise from electrostatic charges contained within a MOSFET, the potential at a given point obeys Poisson's Equation. However, the flow of carriers inside of the MOSFET alter this equation by introducing terms that enforce the continuity condition of this current. This causes the potential ϕ at any given point within the region to obey the following nonlinear Poisson Equation [1]:

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} - \frac{qn_i}{\epsilon_s}(\exp(\phi/V_T) - \exp(-\phi/V_T) = \frac{qD}{\epsilon_s} \tag{1}$$

where q is the elementary charge of the electron, n_i is the intrinsic carrier concentration of silicon, V_T is thermal voltage, and D is the concentration of ionized impurities (doping).

Having obtained the behavior of the interior potentials, we then must establish the equations which govern the boundary potentials of the MOSFET. Assuming that the ohmic contacts of the MOSFET are electrically neutral and obey thermal equilibrium, then the potential is determined by:

$$\begin{aligned}
&(n - p - D) = 0 \\
&n = n_i \exp(\frac{\phi}{V_T}) \\
&p = n_i \exp(\frac{-\phi}{V_T}) \\
&\frac{D}{2n_i} = \sinh(\frac{\phi}{V_T}) \\
&\phi = V_T ln\left(D/2n_i + \sqrt{D^2/4n_i^2 + 1}\right)
\end{aligned} \tag{2}$$

where n and p are negative and positive charge respectively.

On the sides of the semiconductors, we hold the derivative of the potential with respect to the surface normal to zero in order to isolate the device ($\frac{\partial\phi}{\partial\nu} = 0$) Finally, at the MOSFET gate, I use the following boundary condition, as described by Mayergoyz [1]:

$$\phi_{i,j} = \frac{\phi_{i,j-1}\delta\epsilon_s + \delta Q\epsilon_d + V_G}{\epsilon_d + \delta\epsilon_s} \tag{3}$$

For the interior mesh points, the nonlinear Poisson equation becomes:

$$\begin{aligned} \psi_{i,j} &= 4\phi_{i,j} + 2B\sinh(\phi/V_T) - F_{i,j} = 0 \\ F_{i,j} &= (\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}) + \frac{-qD_{i,j}}{\epsilon_s} \end{aligned} \tag{4}$$

where the term $F_{i,j}$ contains none of the $\phi_{i,j}$ dependence (Note: For this program, the mesh is taken to be uniform. The influence of a non-uniform mesh on the computation slightly alters the finite difference terms [1]). Since this equation cannot be solved analytically in terms of $\phi_{i,j}$, I use Newton's Root-Finding Method in order to find the solution to Eq. 4 above. Newton's Method entails taking the slope of a function at a certain point, and then linearly extrapolating this slope down to where the function intersects the x-axis. In the context of this problem, we can use such a method to acquire a solution for $\phi_{i,j}$ at the bottom of Eq. 2. This process can be explicitly seen in the equations below:

$$\begin{aligned} \phi_{i,j}^{FD} &= \phi_{i,j}^{OLD} - \frac{\psi_{i,j}}{\psi'_{i,j}} \\ \frac{\psi_{i,j}}{\psi'_{i,j}} &= \frac{4\phi_{i,j} + 2B\sinh(\phi/V_T) - F_{i,j}}{4 - \frac{2B}{V_T}\cosh(\phi/V_T)} \end{aligned} \tag{5}$$

where $B = \frac{qn_i}{\epsilon_s}$. It should be noted that Newton's Method does not exhibit global convergence: the solution may approach different solutions depending on the starting point ($\phi_{i,j}^{OLD}$ in this case). However, as shown in the graph below, ψ is a monotonically increasing function with only one root, thus validating the choice of Newton's Method to acquire the solution for the nonlinear Poisson equation:

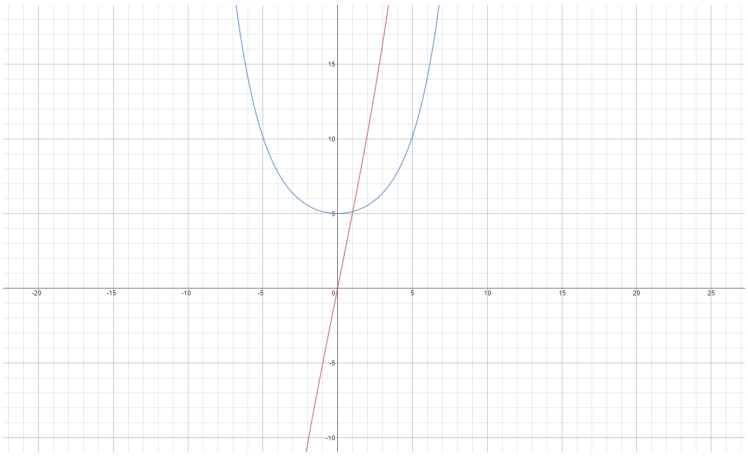


Figure 2: ψ (Red) and ψ' (Blue) vs ϕ. Note that ψ' always remains positive. Graph generated by Desmos Online Graphing Calculator

Once a certain updated value of $\phi_{i,j}$ is acquired, I then apply the Successive Relaxation Method in order to acquire the steady state solution for the system. In this method, I take a certain $\phi_{i,j}$, acquire the finite difference value through the equations outlined above, and then add

the difference of the two values, scaled by a relaxation parameter ω to yield a new value for that grid point [4]:

$$\phi_{i,j}^{NEW} = \phi_{i,j}^{OLD} + \omega(\phi_{i,j}^{FD} - \phi_{i,j}^{OLD}) \tag{6}$$

Using this method to update the grid points as a function of the neighboring points in the mesh (and itself in the case of the nonlinear Poisson Equation), I then iterate through this process until the maximum change in the system becomes less than a threshold value assigned by the user. Having explained and justified the equations used to acquire the steady state solution for the electric potentials inside of the MOSFET, I now explicitly outline the methods I use to set initial conditions that enforce the equations above and minimize runtime.

3 Implementation

First, I initialize two grids with the same dimensions: one labeled flags and one labeled grid. The flags array assigns a character value to each point on the grid, which determines the appropriate equations to a particular point. For example, a point (i, j) on the interior of the mesh is assigned the character s to indicate that it is an interior mesh point in the source carrier region of the MOSFET, and a such, should have positive doping, and obey the nonlinear Poisson equation (Eq. 1).

The grid array contains all of the values of the electric potential at any given time. It is updated as the program iterates through the relaxation routine, thus causing any particular electric potential value to be updated on average by half ϕ^{NEW} values and half ϕ^{OLD}.

Second, I initialize values in grid according to their location. Although the entire method is globally convergent, and thus would approach the same value regardless of initial conditions, I follow the advice outlined by Mayergoyz [1] by setting points in the n-region in figure() above to the potential enforced at the ohmic contacts at the source and drain (equation ()) and points in the p-region to the ohmic contact at C_3.

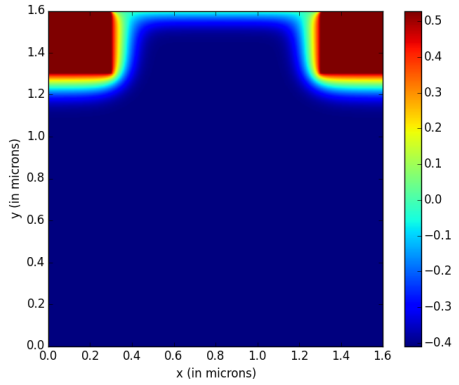
Third, after assigning constants, I determine an optimal relaxation parameter ω by testing the amount of steps required to complete the routine for any given value of ω until this number reaches a minimum.

Finally, I store the data into a .dat file, and plotted the data in Python using pylab's imshow() to generate 2D contours of the potentials, and 1-D cross-sections of lines parallel to the x-axis in the region from the source to halfway between the source and the drain at various y-values.

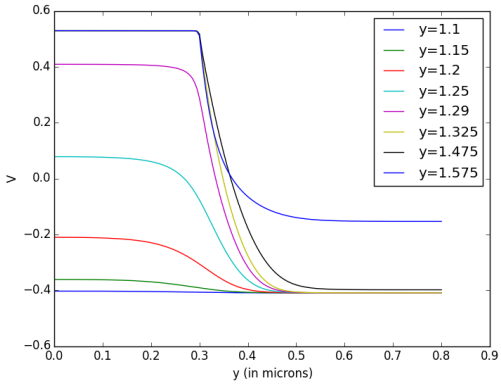
4 Results and Analysis

First, in order to ensure the accuracy of my code, I used the parameters listed in Section IV of Mayergoyz's paper to try to acquire the graph depicted in Figure 2 of that paper. Specifically, setting the doping in the n region to $10^{18}cm^{-3}$, the doping in the p region to $10^{-16}cm^{-3}$, $n_i = 1.45x10^{10}cm^{-3}$, $V_T = 0.026V$, $\delta = 5x10^{-}8m$, Q = 0 and using a 320x320 grid, I was able to obtain the graph on the left below in Figure 3:

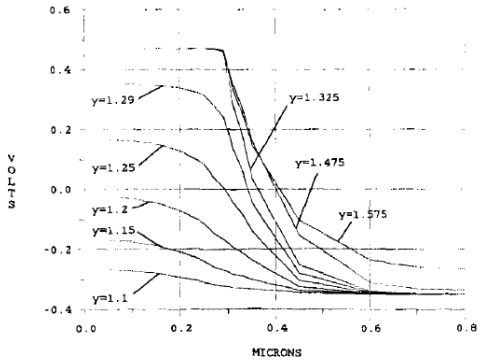
While there are a few minor differences between the two graphs, they appear to be generated from the same data. The few differences can be explained by a variety of factors. For instance, Mayergoyz uses a non-uniform mesh, meaning that additional coefficients have to be taken into



(a) Contour Map of Potential from Mayergoyz's Data



(b) 1-D Cross Section Obtained from Mayergoyz's Data



(c) Graphical Result from Journal of Applied Physics (1)

Figure 3: Comparison of Data Generated from Same Initial Conditions

account in order to adjust the finite difference equations. Also, while Mayergoyz's mesh contains 28 points along each axis, mine contains 320. This not only accounts for the greater smoothness exhibited by the graph on the left, but also small variations in the overall shape. Finally, while Mayergoyz uses a relaxation tolerance of the 0.5 percent of the maximum potential in the system (approximately 0.00025V), I use a relaxation tolerance of a much smaller value that is not pegged to the maximum voltage of the system (10^{-5} V). This high level of agreement, as well as other checks, such as how the contour map of the voltage largely resembles the actual geometry of the MOSFET, suggests the accuracy of the code.

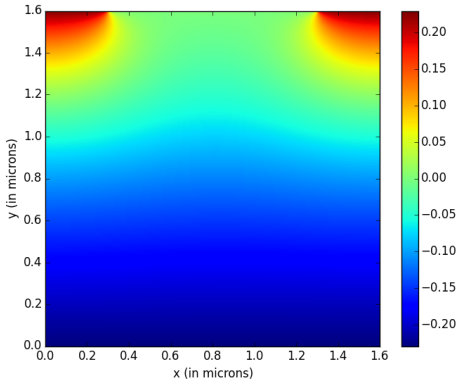
At this point, I sought to acquire an optimized relaxation parameter in order to minimize the number of iterations through the relaxation subroutine. By recording the number of iterations done within a call of the relax subroutine for a certain ω values, I was able to narrow down the optimal ω value to 1.86. The table below shows the number of runs required by various relaxation parameter candidates given a relaxation tolerance of 10^{-4} V:

Having acquired an optimal relaxation parameter, I then explored how a variety of changes in the parameters which govern the system would manifest in the plots. The next system I simulated is one in which the doping concentrations in the n and p region were not as disparate (usually, doping

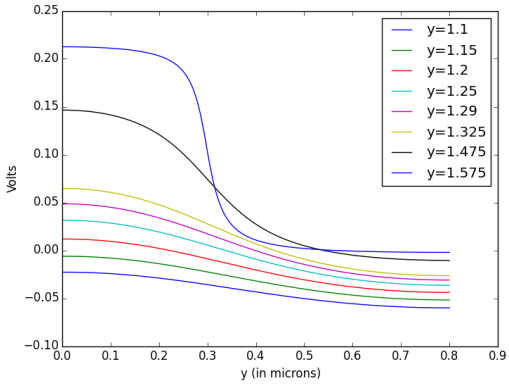
ω	Number of Iterations
0.50	1573
0.75	943
1.00	621
1.25	420
1.50	273
1.75	150
1.85	99
1.95	275
1.90	137
1.86	94
1.87	101

Table 1: Acquiring Optimal Relaxation Parameter

concentrations in silicon vary from 10^{13} to 10^{18} atoms per cubic centimeter). Shifting down the n and p concentrations to 10^{13} and -10^{13}, I obtained the following figures:



(a) Contour of Decreased Doping



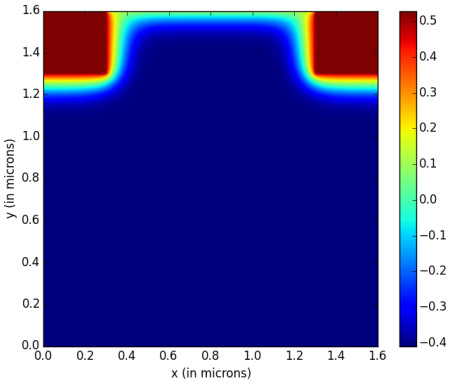
(b) 1-D Cross-Section of Decreased Doping

In these pictures, with the relative doping concentrations much lower than they were in the previous example, the voltage is more level throughout the region, as is expected.

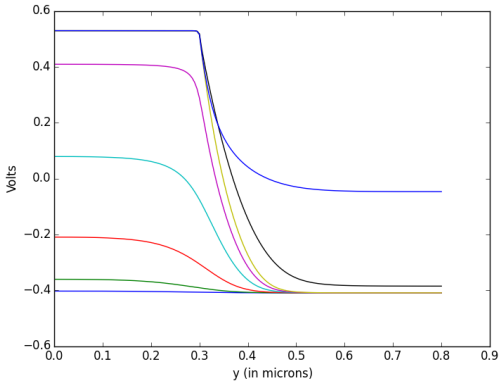
Next, I adjusted the gate voltage observe how that would influence the potentials the region of interest. The following graphs show how the region is affected by a change in gate voltage (n doping concentrations at both regions = 10^{18} and the p doping = -10^{16}:

As the gate voltage is increased, a region of high potential that connects the source and drain region is formed. This area is known as the conduction channel. When a voltage difference is applied between the source and the drain regions, this conduction channel provides a path by which charges can travel between the two parts of the MOSFET, thus generating a current [2]. As can be seen in the graphs above, as the gate voltage is raised, the conduction channel becomes larger as well, providing additional paths for charges to flow across.

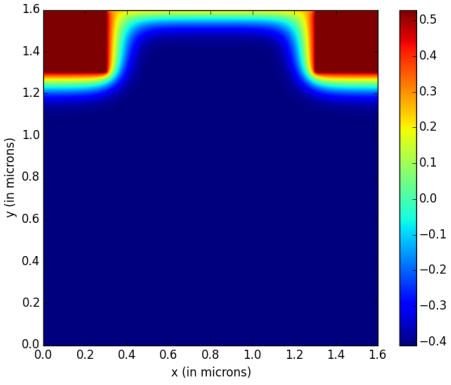
Finally, I looked at different doping concentrations in the source and drain regions, which



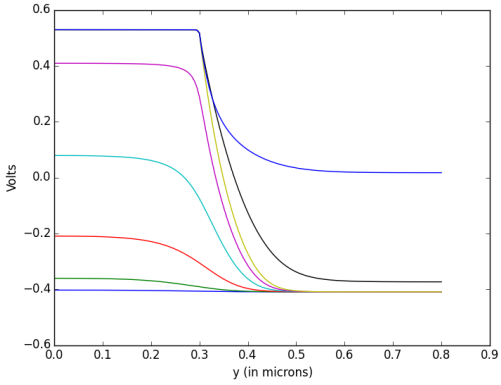
(a) Contour of Gate Voltage = 0.1 V



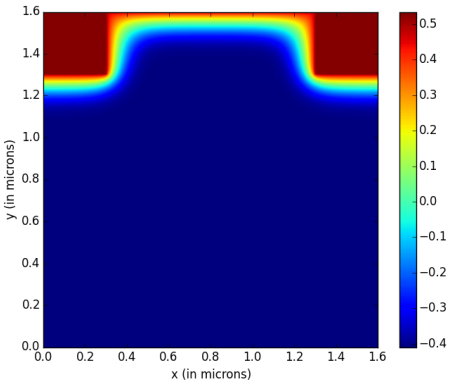
(b) 1-D Cross-Section of Gate Voltage = 0.1 V



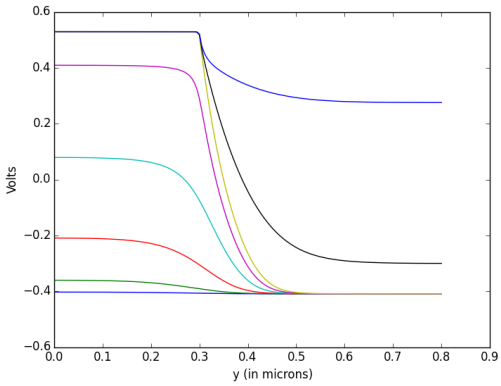
(c) Contour of Gate Voltage = 0.2 V



(d) 1-D Cross-Section of Gate Voltage = 0.2 V



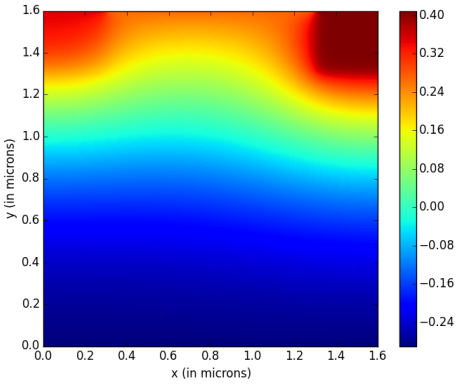
(e) Contour of Gate Voltage = 0.4 V



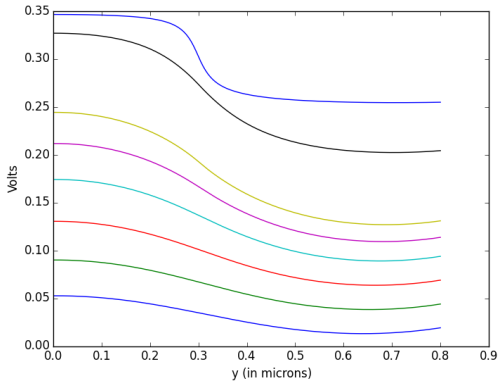
(f) 1-D Cross-Section of Gate Voltage = 0.4 V

in turn create a voltage difference between the two areas with a gate voltage fixed at 0.2V and a p-region doping concentration of -10^{14}:

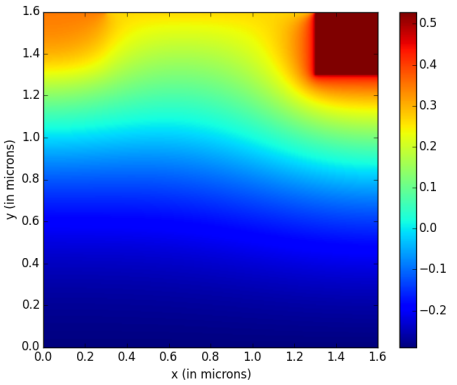
Although these pictures are not as explicit as a contour that took into account an explicit voltage difference between the drain and source, they do exhibit an interesting phenomenon in



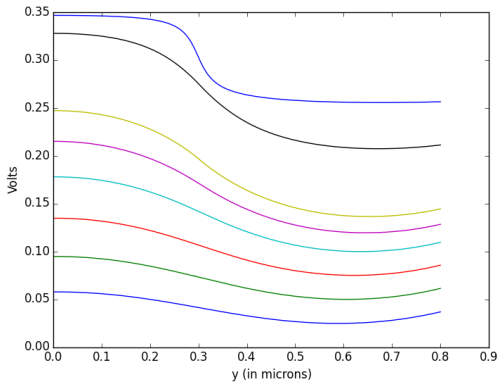
(a) Contour with Source Doping = 10^{15}, Drain Doping = 10^{16}



(b) 1-D Cross-Section of Source Doping = 10^{15}, Drain Doping = 10^{16}



(c) Contour with Source Doping = 10^{15}, Drain Doping = 10^{18}



(d) 1-D Cross-Section of Source Doping = 10^{15}, Drain Doping = 10^{18}

MOSFETS. The increased amount of concentration at the drain certainly created a higher voltage region around that region. However, this did not do much to change the nature of the potentials outside of that small region. Although it is not completely clear due to the change of scale in the contour, the potentials between the source and drain are largely unchanged, as shown by a comparison in the 1D cross-sections. This is indicative of a process called pinch-off in MOSFETs, which described the situation where additional voltage applied across the drain and the source fails to cause a significant change to the current between the two regions.

5 Conclusion

By using Newton's Method and the boundary conditions of an n-type MOSFET to solve the finite difference expression as outlined by Mayergoyz, I was then able to apply the technique of Successive Relaxation to obtain the potentials inside of the system. Once I tested the accuracy of my code by comparing it to Mayergoyz's results, I then altered the parameters to observe how they influenced the system. By doing this, I was able to observe and quantify phenomena essential to

the operation of the MOSFET as a function of parameters such as doping concentration and gate voltage.

The functionality of this program could be greatly expanded first by obtaining and graphing the x-component of the electric field between the source and the drain, thus yielding greater insight into the flow of charges between these two regions. This could be done by applying the finite difference equation to the relationship between voltage and current ($E_x = \frac{-\partial V}{\partial x}$). Furthermore, a greater variety of doping meshes could be used, rather than the uniform doping in the n and p regions that was used in the aforementioned examples. Finally, a non-uniform mesh, and their corresponding changes to the finite-difference equations, could further optimize the performance of the code and yield greater insights into the region of interest between the source and the drain.

6 Source Code and References

References

1. I.D. Mayergoyz, *Solution of the nonlinear Poisson equation of Semiconductor Device Theory*, J. Applied Physics, Vol. 59, 1986, p.195-199
2. P.R. Gray and R.G. Meyer, *Analysis and Design of Analog Integrated Circuits*, 5th edit., Wiley 2009, p38-49.
3. W. H. Press, S. A, Teukolsky, W. T. Vetterling and B. P. Flannery,
Numerical Recipes, The Art of Scientific Computing, 3rd edit., Camb. Univ. Press 2007.
4. AEP 4380 Website, HW 6. Fall 2015. https://courses.cit.cornell.edu/aep4380/secure/hw6f15.pdf//

Source Code Listing

apply.cpp

```
/*
AEP 4380 Final Project

Semiconductor Device Modeling

Jack Newman December 8, 2015

Last Edited: Decemeber 14, 2015

Intel(R) Core(TM) i7-4810MQ CPU @ 2.80 GHz

MinGW C++ compiler, GCC 4.8.1

OS: Windows 8.1

Note: Comments are either inline or below the lines they describe
*/
#include <cstdlib> // importing plain c
#include <cmath> // math library

#include <iostream> // stream IO
#include <fstream> // stream file IO
#include <iomanip> // to format the output
// define the following symbol to enable bounds checking
//
#define ARRAYT_BOUNDS_CHECK
#include "arrayt.hpp" //used with permission by Professor Earl Kirkland

int Nx = 320, Ny = 320; //End coordinates for boundaries

double D1 = 10E15, D2 = 10E15, D3 = -10E14;//Doping concentrations in cm^-3
double V_T = 0.026; //Thermal Voltage
double del = 5E-8;//Dioxide Thickness in m
double BigQ = 0; //Trapped Charges at the interface
double es = 11.68, ed = 3.9; //relative permittivity of silicon and silicon dioxide
//double q = 4.8032E-10; //charge of an electron in esu
double q = 1.60217662E-19;//charge of an electron in Couloumbs
double ni = 1.45E10; // intrinsic carrier concentration of silicon
double B = q*ni/es;
double steps = 0; //counter for use in relax method
double Vg = 0.2; //Applied Voltage at Gate

double initialgreater(double D){
        //helper function to initialize values on grid that have a doping
        //concentration greater than ni
        double phi = V_T*log(D/(2*ni) + sqrt(1 + D*D/(4*ni*ni)));
        return phi;
}

double initialless(double D){
        //helper function to initialize values on grid that have a doping
        //concentration less than ni
        double phi = V_T*log(1/(sqrt(1 + D*D/(4*ni*ni)) - D/(2*ni)));
        return phi;
```

```
}


void relax(arrayt<char>& flag, arrayt<double>& grid){
        //Relaxation method, which takes in a flag and grid array of equal
        //of equal dimensions. Will run until relaxation tolerance is reached.
        double w = 1.86; //relaxation parameter
        double diffmax = 0;//maximum change in potential for a given iteration
        double max = 0;//maximum potential on the grid
        double fd;//finite difference value
        double relaxtolerance;//relaxation tolerance
        int i, j, Nmax = 100000;
        //counters, and maximum steps to catch calls which fail to converge


        for(i = 0; i < (Nx + 1); i++){
                for(j = 0; j < (Ny + 1); j++){
                        if(flag(i,j) == '0'){
                                //finite difference equation on dioxide interface
                                fd = (grid(i, j-1)*del*es/ed+del*BigQ/ed+Vg)/(1+del*es/ed);
                        }

                        else if(flag(i,j) == '1'){
                                //finite difference equation on ohmic contact 1
                                fd = V_T*log((D1/2.0 + sqrt(D1*D1/4.0 + ni*ni))/ni);
                        }

                        else if(flag(i,j) == '2'){
                                //finite difference equation on ohmic contact 2
                                fd = V_T*log((D2/2.0 + sqrt(D2*D2/4.0 + ni*ni))/ni);
                        }

                        else if(flag(i,j) == '3'){
                                //finite difference equation on ohmic contact 3
                                fd = V_T*log((D3/2.0 + sqrt(D3*D3/4.0 + ni*ni))/ni);
                        }

                        else if(flag(i,j) == '4'){
                                //finite difference equation on left artificial
                                //zone boundary
                                fd = grid(i+1, j);
                        }

                        else if(flag(i,j) == '5'){
                                //finite difference equation on right artificial
                                // zone boundary
                                fd = grid(i-1, j);
                        }

                        else if(flag(i,j) == 's'){
                                //Application of Newton's Method for finite difference
                                //equation in the interior source region
                                double F = grid(i+1, j) + grid(i-1, j) + grid(i, j+1) +
                                        grid(i, j-1) + q*D1/es;
                                double psi = 4*grid(i,j)+2*B*sinh(grid(i,j)/V_T) - F;
                                double delpsi = 4+2*B/V_T*cosh(grid(i,j)/V_T);
                                double chi = -psi/delpsi;
                                double phinew = grid(i, j) + chi;
                                fd = phinew;
```

```
                }
                else if(flag(i,j) == 'd'){
                        //Application of Newton's Method for finite difference
                        //equation in the interior drain region
                        double F = grid(i+1, j) + grid(i-1, j) + grid(i, j+1) +
                                grid(i, j-1) + q*D2/es;
                        double psi = 4*grid(i,j)+2*B*sinh(grid(i,j)/V_T) - F;
                        double delpsi = 4+2*B/V_T*cosh(grid(i,j)/V_T);
                        double chi = -psi/delpsi;
                        double phinew = grid(i, j) + chi;
                        fd = phinew;
                }

                else if(flag(i,j) == '7'){
                        //Application of Newton's Method for finite difference
                        //equation in the interior hole carrier region
                        double F = grid(i+1, j) + grid(i-1, j) + grid(i, j+1) +
                                grid(i, j-1) + q*D3/es;
                        double psi = 4*grid(i,j)+2*B*sinh(grid(i,j)/V_T) - F;
                        double delpsi = 4+2*B/V_T*cosh(grid(i,j)/V_T);
                        double chi = -psi/delpsi;

                        double phinew = grid(i, j) + chi;
                        fd = phinew;
                }

                if(abs(grid(i,j) - fd) > diffmax){
                        //Obtains the maximum change for a given iteration
                        diffmax = abs(grid(i,j) - fd);
                }

                if(abs(grid(i,j)) > max){
                        //Obtains the maximum potential in a given iteration
                        max = abs(grid(i,j));
                }
                //Relaxation Equation
                grid(i,j) = grid(i,j) + w*(fd - grid(i,j));
        }
}
//cout << "max = " << max << endl;
if(steps > Nmax){
        //if the number of steps becomes greater than NMAX, throw error
        cout << "Max Step Error" << endl;
        exit(0);
}
relaxtolerance = 1/100000.0; //relaxation tolerance
//cout << "relaxtolerance = " << relaxtolerance << setw(15) << "diffmax = "
//      << diffmax << endl;
if(diffmax > relaxtolerance){
        //If the maximum potential change is greater than the relaxation
        //tolerance parameter, add one to steps and run relax again
        steps += 1;
        relax(flag, grid);
}
if(diffmax < relaxtolerance){
        //End relax call once diffmax is less than the
        // relaxation tolerance parameter
        cout << "DONE" << endl;
        cout << w << setw(15) << steps << endl;
```

```
                //print relaxation parameter and steps
        }
}

int main(){

        ofstream fp;
        ofstream gp;

        fp.open( "applycontour.dat" ); //open new file for contour data

        if( fp.fail() ) { // or fp.bad()
                cout << "cannot open file" << endl;
                // catches fp.open failure, and returns appropriate error
                return( EXIT_SUCCESS );
        }

        gp.open( "1dapply.dat" ); //open new file for 1d cross-section data

        if( gp.fail() ) { // or fp.bad()
                cout << "cannot open file" << endl;
                // catches fp.open failure, and returns appropriate error
                return( EXIT_SUCCESS );
        }


        int i, j; //counters
        arrayt<double> grid(Nx + 1, Ny + 1);//potential mesh
        arrayt<char> flags(Nx + 1, Ny + 1);//flag
        for(j = 0; j < (Ny + 1); j++){
                for(i = 0; i < (Nx + 1); i++){
                        if(j == Ny and i >= 60 and i <= (Nx - 60)){
                                //flags for dioxide interface
                                flags(i, j) = '0';
                        }
                        else if(j == Ny and i < 60){
                                //flag for ohmic contact 1
                                flags(i, j) = '1';
                        }
                        else if(j == Ny and i > (Nx - 60)){
                                //flag for ohmic contact 2
                                flags(i, j) = '2';
                        }
                        else if(j == 0){
                                //flag for ohmic contact 3
                                flags(i, j) = '3';
                        }
                        else if(i == 0){
                                //flag for left artifical boundary
                                flags(i, j) = '4';
                        }
                        else if(i == Nx){
                                //flag for right artifical boundary
                                flags(i, j) = '5';
                        }
                        else if(i <= 60 and j >= (Ny - 60)){
                                //flag for source region
                                flags(i, j) = 's';
                        }
```

```
                else if(i >= (Nx - 60) and j >= (Ny - 60)){
                        //flag for drain region
                        flags(i, j) = 'd';
                }
                else{
                        //flag for p region
                        flags(i, j) = '7';
                }
        }

}

for(j = 0; j < (Ny + 1); j++){
        for(i = 0; i < (Nx + 1); i++){
                if(i <= 60 and j >= (Ny-60)){
                        //Initialize source region potentials
                        grid(i,j) = initialgreater(D1);

                }
                else if(i >= (Nx-60) and j >= (Ny-60)){
                        //initialize drain region potentials
                        grid(i,j) = initialgreater(D2);

                }
                else{
                        //initialize p-region potentials
                        grid(i, j) = initialless(D3);

                }
        }
}
//Start Relaxation Method
relax(flags, grid);

//Save data for contour graph in applycontour
for(j = Ny; j > 0; j--){//altered for formatting in imshow function
        for(i = 0; i < (Nx + 1); i++){
                fp << grid(i, j) << setw(15);
        }
        fp << endl;
}
fp.close();

//Save data for cross-section graph
for(j = 0; j <= Ny/2; j++){
        gp << j*0.005 << setw(15) //x axis
        << grid(j, 220) << setw(15) <<//y=1.1
                grid(j, 230) << setw(15) << //y = 1.15
                grid(j, 240) <<//y = 1.2
                setw(15) << grid(j, 250) <<//y=1.25
                setw(15) << grid(j, 258) <<//y=1.29
                setw(15) << grid(j, 265) <<//y=1.325
                setw(15) << grid(j, 295) <<//y=1.475
                setw(15) << grid(j, 315) << endl;//y=1.575
}
gp.close();


return(EXIT_SUCCESS);
```

```
}//end main
```

plotapply.py

```
# uses numpy, scipy, matplotlib packages
"""
Plotting the results from the 320x320 grid
"""
from pylab import *
import matplotlib.pyplot as plt
import numpy as np

pixa = loadtxt( 'applycontour.dat', 'float' )
img = imshow( pixa, aspect='equal', extent=(0, 1.6 ,0 , 1.6)) #plot contour map
img.set_cmap( 'jet' )
colorbar()
xlabel( 'x (in microns)' )
ylabel( 'y (in microns)' )
show()
#imshow uses grid values
data = loadtxt( '1dapply.dat', 'float' )

fig = figure()
x = data[:,0]
y110 = data[:,1] #y = 1.1
y115 = data[:,2] #y = 1.15
y120 = data[:,3] #y = 1.20
y125 = data[:,4] #y = 1.25
y129 = data[:,5] #y = 1.29
y1325 = data[:,6] #y = 1.325
y1475 = data[:,7] #y = 1.475
y1575 = data[:,8] #y = 1.575
plot(x, y110, label= 'y=1.1')
plot(x, y115, label = 'y=1.15')
plot(x, y120, label = 'y=1.2')
plot(x, y125, label = 'y=1.25')
plot(x, y129, label = 'y=1.29')
plot(x, y1325, label= 'y=1.325')
plot(x, y1475, label = 'y=1.475')
plot(x, y1575, label = 'y=1.575')
#plt.legend()
#plt.colorbar()
xlabel("y (in microns)")
ylabel("Volts")
#tricontourf uses 3 columns: (x, y, psi(x,y))
show()
```