# A two-dimensional Self-Consistent Solution to Schrödinger -Poisson Equation Using Finite Element Method

by

Feras Aldahlawi

A thesis submitted in conformity with the requirements
for the degree of Masters of Science
Department of Physics
University of Chicago

## Abstract

# A two-dimensional Self-Consistent Solution to Schrödinger -Poisson Equation Using Finite Element Method

Feras Aldahlawi

Masters of Science

Graduate Department of Physics

University of Chicago

2014

In this thesis, we demonstrate a method of solving the Schrödinger equation and Poisson equation self-consistently using the Finite Element Method. We ran a simulation for a wafer of GaAs/AlGaA heterojunction that is doped with Si dopants. We achieve a convergence for the Fermi energy, $E_F$, in the range of energy between $E_f = (1000.0, 2000.0)$ eV.

# Contents

# Chapter 1

# Introduction

Modern day electronics have reached scales of extremely small magnitudes, typically, a few hundred nanometers. When we enter this regime, it is important to take into account quantum mechanical phenomenons as they have a larger impact on smaller scale electronics. These quantum effects will impose substantial amount of change on device performance. One way to understand the behavior of materials at such scales is to incorporate quantum mechanics along with semi-classical electrostatics. A complete picture requires a self-consistent solution of both Schrödinger and Poisson equation. These types of systems are extremely complex to be solved analytically. Therefore, we resort to numerical approximations. Ideally, one would device a numerical simulation flexible enough to capture wide range of applicability, while at the same time accurate to experimental data. A lot of literature have utilized a conventional approach to the solution of Schrödinger and Poisson equation called finite-difference method (FDM). In this paper, we use another method in finding a self-consistent solution called finite-element method (FEM). The reason why we decided to use FEM is that it is easy to have complex geometries, such as combinations of cylinders, spheres, and cubes. In chapter 1 we will discuss the formulation of Schrödinger and Poisson equations in the FEM regime. In chapter 2, we will discuss the theory behind our numerical simulation. Lastly, in chapter 3, we will discuss the results for a simple two-dimensional heterojunction.

# Chapter 2

# Formulation

## 2.1 Finite Element Method

More often than not, we need to numerically solve partial differential equations. In this paper, we will use the finite Element method to study the Schrödinger and Poisson equations. This method is powerful because allows us to solve PDEs for obscure geometries. For generic formulation for a PDE of the form

$$[D]u(x) = f(x) \in \Omega \tag{2.1}$$

Where D is an arbitrary operator, and $\Omega$ defines our geometry. We have to rewrite this equation in weak variational form so that we can solve it using FEM. Let us consider the boundary conditions

$$\begin{cases} u(x) = u_0(x) & \text{on } \Gamma_D \qquad (2.2) \\ \dfrac{\partial u}{\partial n} = g_0(x) & \text{on } \Gamma_N \qquad (2.3) \end{cases}$$

Where $\Gamma_D$ and $\Gamma_N$ signifies Dirichlet and Neumann boundary condition. The general idea of finite element method is to rewrite a PDE into a variational problem. The way to do that is to introduce an arbitrary function $v$ and multiply the PDE with $v$. Then we integrate over the domain $\Omega$ and separate every second-order derivative using integration by parts. The original PDE then can be written in the weak form as

$$\int_\Omega [D'](u \cdot v)\, d\Omega + \int_{\Gamma_N} gv\, \partial\Omega = \int_\Omega fv\, d\Omega \quad \forall v \in \hat{V} \tag{2.4}$$

Where $D'$ is the reduced operator after performing integration by part to the second-order derivatives, and $\hat{V}$ is the function space where our arbitrary function $v$ lives. The function $u$ lies in $V$, which could be different than $\hat{V}$. Now, to perform the calculation numerically, we need to reformulate the continuous variational problem to a discrete problem. We just need to define a discrete space $\hat{V}_d \subset \hat{V}$ and $V_d \subset V$ so that we can write our boundary problem as

$$\int_\Omega [D'](u_d \cdot v)\, d\Omega + \int_{\Gamma_N} gv\, \partial\Omega = \int_\Omega fv\, d\Omega \quad \forall v \in \hat{V}_d \subset \hat{V} \tag{2.5}$$

It convenient to use unified notation for linear weak forms

$$a(u, v) = L(v) \tag{2.6}$$

with

$$a(u,v) = \int_{\Omega} [D'](u_d \cdot v)\, d\Omega \tag{2.7}$$

$$L(v) = \int_{\Omega} fv\, d\Omega - \int_{\Gamma_N} gv\, \partial\Omega \tag{2.8}$$

## 2.2 Poisson in weak variational form

Here, we want to solve Poisson equation that arises in electrostatics. A general Poisson equation for electrostatics is giving by

$$\frac{d}{dx}\big(\epsilon_s(x)\frac{d}{dx}\big)\phi(x) = \frac{-q[N_D(x) - n(x)]}{\epsilon_0} \tag{2.9}$$

Where $\epsilon_s$ is the dielectric constant of the material, $N_D$ is the ionized donor concentration, $\phi$ is our electrostatic potential, and $n$ is the electron density. Here, we will only consider a piecewise dielectric constant, therefore, we only need to divide the domain into subdomain with different dielectric constants. The Poisson equation can be rewritten as

$$\epsilon_s \nabla^2 \phi(x) = \frac{-q[N_D(x) - n(x)]}{\epsilon_0} \tag{2.10}$$

Notice that our operator $[D]$ is replace with $\nabla^2$, and our source term $f(x)$ is replaced with the scaled difference of the ionized donor concentration and the electron density. It is pretty straight forward to reformulate the Poisson equation to the weak variational form, all we need to do is to follow the procedure explained in the previous section. We get our final weak variational as

$$\int_\Omega \epsilon_s \epsilon_0 \nabla\phi \cdot \nabla v \;\; d\Omega = \int_\Omega q[N_D(x) - n(x)]v(x) \;\; d\Omega \tag{2.11}$$

Now can use finite element method to solve for $\phi(x)$.

## 2.3 Schrödinger in weak variational form

To rewrite Schrdinger's equation in the variational form, we start with Schrödinger 's equation in differential form

$$-\frac{\hbar^2}{2}\frac{d}{dx}\left(\frac{1}{m^*(x)}\frac{d}{dx}\right)\psi(x) + V(x)\psi(x) = E\psi(x) \tag{2.12}$$

Where $m*(x)$ is the effective mass. Since we are going to have a constant mass within the region we are exploring, we can take that out of the derivative. With a little algebra we can rewrite the equation as

$$\frac{\hbar^2}{2m^*}\nabla^2\psi(x) + [E - V(x)]\psi(x) = 0 \tag{2.13}$$

Multiply both sides by a test function $v$. This function is arbitrary with the condition that it vanishes on the boundaries of our system.

$$\frac{\hbar^2}{2m^*}\nabla^2\psi(x)v(x) + [E - V(x)]\psi(x)v(x) = 0 \tag{2.14}$$

Integrate both terms over the domain and using integration by parts yields

$$\int_\Omega \frac{\hbar^2}{2m^*}\frac{\partial\psi}{\partial x}\frac{\partial v}{\partial x}d\Omega + \int_\Omega V(x)\psi(x)v(x)d\Omega = \int_\Omega E\psi(x)v(x)d\Omega \tag{2.15}$$

Notice that we drop the surface term because we require that our test function $v$ vanishes where $\psi$ is known.

# Chapter 3

# Numerical Simulation

## 3.1 Relevant Equations

In this section, we will cover the basic equations that we need to satisfy in our numerical simulation. The basic equations combines quantum mechanics with semi-classical physics. We need to solve Schrödinger and Poisson using

$$-\frac{\hbar^2}{2}\frac{d}{dx}\left(\frac{1}{m^*(x)}\frac{d}{dx}\right)\psi(x) + V(x)\psi(x) = E\psi(x) \tag{3.1}$$

$$\frac{d}{dx}\left(\epsilon_s(x)\frac{d}{dx}\right)\phi(x) = \frac{-q[N_D(x) - n(x)]}{\epsilon_0} \tag{3.2}$$

$$V(x) = -q\phi(x) + \Delta E_c(x) \tag{3.3}$$

$$n(x) = \sum_{k=1}^{m} \psi_k^*(x)\psi_k(x)n_k \tag{3.4}$$

$$n_k = \frac{m^*}{\pi\hbar^2}\int_{E_k}^{\infty} \frac{1}{1+e^{(E-E_F)/KT}}dE \tag{3.5}$$

Where $\Delta E_c(x)$ is the pseudopotential energy due to the band offset at the heterointerface, $n_k$ is the electron occupation number which can calculated by Fermi-Dirac distribution function with $E_F$ being the Fermi level, $\psi_k$ is the wavefunction in the $k^{th}$ state, and $E_k$ is the eigen energy in state $k$. These five equations have to be solved and produce a self-consistent solution in order for us to have a good approximation of the behavior of our system.

## 3.2 Approximations

In this section, we will cover the basic approximations that we employ in our numerical simulation. We first start with approximating the occupation number $n_k$. since the complete Fermi-Dirac integral is an exponential function

$$\mathcal{F}_j(x) = \frac{1}{\Gamma(j+1)}\int_0^{\infty} \frac{t^j}{exp(t-x)+1}dt \tag{3.6}$$

Where $x = E - E_F$. In this paper, we will only consider the two dimensional case, in which $j = -1/2$. Since we are taking the integral of $\mathcal{F}_j(x)$ from $E_k$ to $\infty$, we need the argument $E$ to be $E \leq 0$. Therefore the distribution function falls of exponentially as

$$\mathcal{F}_{-1/2}(E) \sim exp(-E), \quad E \gg 0 \tag{3.7}$$

Therefore, we only need to calculates relatively few quantum states the are lower than Fermi level. The second approximation involves treating $m^*(x)$ as a step function. Where it is (for two subdomains)

$$m^*(x) = \begin{cases} a_1 m_e & \in \quad \Omega_1 \quad (3.8) \\ a_2 m_e & \text{otherwise} \quad (3.9) \end{cases}$$

Where $m_e$ is the mass of electron ($\approx$ 0.511 MeV), and $a_1$ and $a_2$ are coefficients that depends on the materials.

## 3.3 Algorithm

In this section, we will describe the process of producing a self-consistent solution. We first start with an arbitrary potential $V(x)$ and plug it in equation 3.1 to solve for the wavefunctions. After we get the wavefunctions, we compute the electron density function and the occupation number using equations 3.4 and 3.5. Then we can solve Poisson equation because we have $n(x)$. Finally, we use the solution of 3.2, $\phi(x)$, to find the new corrected potential, $V(x)$, using 3.3. After finding the new potential and the new electron density, we compute the difference between them and their old values to test for convergence. We repeat this process until we achieve convergence, we choose our criteria for convergence to be on the order of $10^{-5}$ error.

## 3.4 Geometry

We will consider a GaAs/AlGaAs structure for our simulation. We chose GaAs/AlGaAs because it preforms better than most of semiconductors at high frequency. Also, GaAs/AlGaAs is pretty much insensitive to temperature. Lastly, GaAs/AlGaAs has a high electron mobility. Ideally, our simulation would work for all sorts of materials but we will only show GaAs/AlGaAs in this paper. Our wafer is depicted in Figure 3.1. Our wafer's boundary conditions at $\phi(0, y) = 0$ and $\phi(1, y) = 0$ at both ends on the $x$-axis, and is periodic, $\phi(x, 0) = \phi(x, 1)$ on the $y$-axis. Notice that we use normalized coordinate system in which our domain is $\Omega \in [0, 1] \times [0, 1]$. The last two boundary conditions take care of the fact that we dope the GaAs layer with $2 \times 10^{12}$ $cm^{-2}$.

## 3.5 Verification of the Simulation

Now, we try to verify the validity of our simulation. In order to do that, we try to reproduce Tan et al.'s paper [1]. They had the structure of quasi-one-dimensional channel depicted in 3.3. We use the same procedure as they do in the paper but with the extra dimension $y$. They use a reference Fermi level at zero. The number of bound states for their calculation is two, but for ours we get extra bound states because of the extra dimension. To characterize a bound states, the eigen energy has to be less than 1.8 $eV$ which is the psuedopotential energy at the heterointerface. They fix the temperature at

4 $K$. Figure 3.5 in is the electron density $n(x, y)$ following the work of Tan et al.. We can see that we get a similar behavior at the boundary of the substrate. This is done in two dimensions while Tan et al.'s work was done in one dimension, this means that we did not verify nor reproduced Tan et al.'s work, but we get a similar order of magnitude and similar form of the electron density.

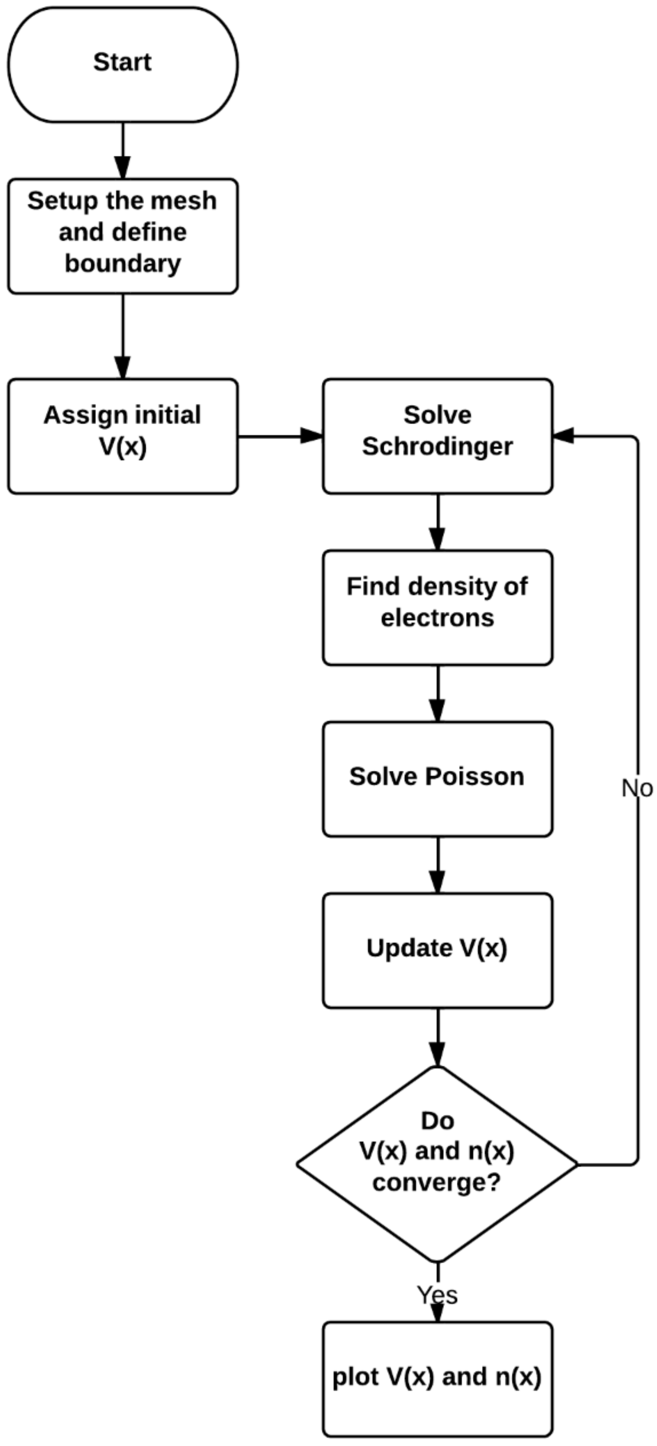


Figure 3.1: A flowchart of the self-consistent simulation

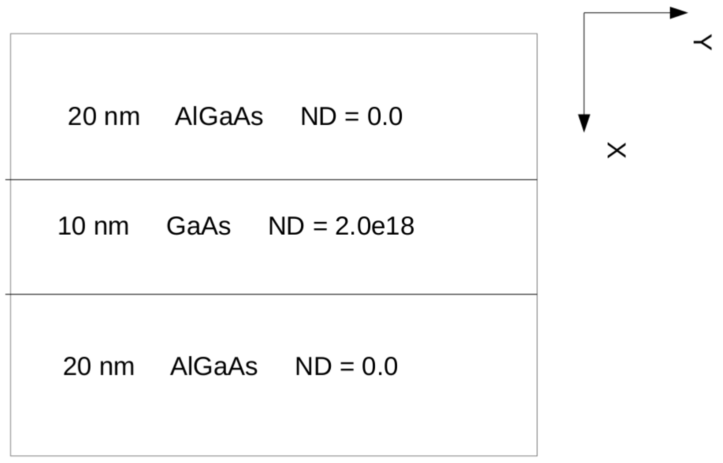


Figure 3.2: Structure of the wafer that we used for our simulation. Here, we use alloy fraction of GaAs/$Al_{0.3}Ga_{0.7}As$

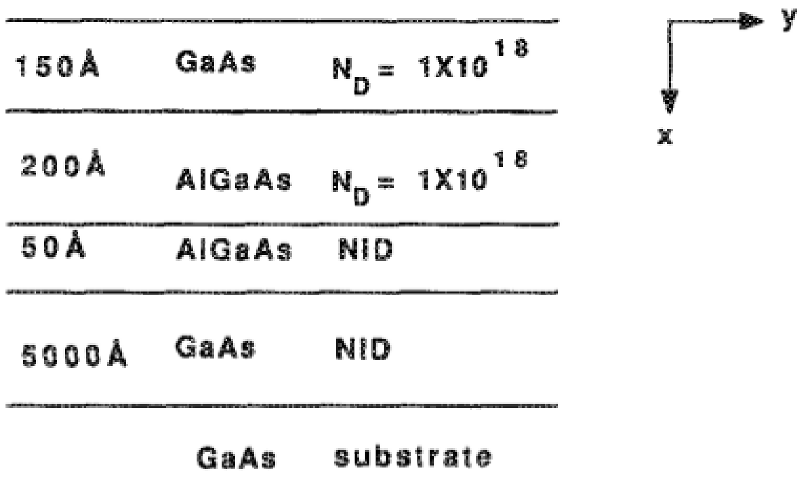


Figure 3.3: The quantum wafer that is used in Tan et al's paper.

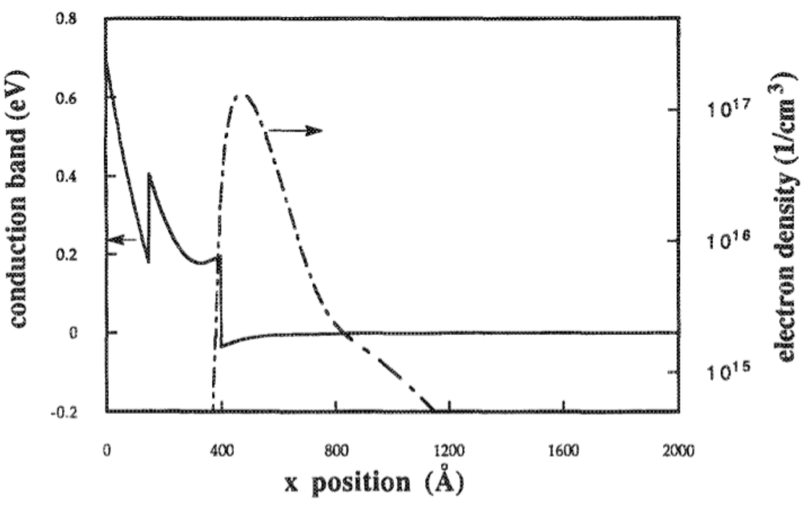


Figure 3.4: Conduction band and electron density simulation from Tan et al.'s paper.

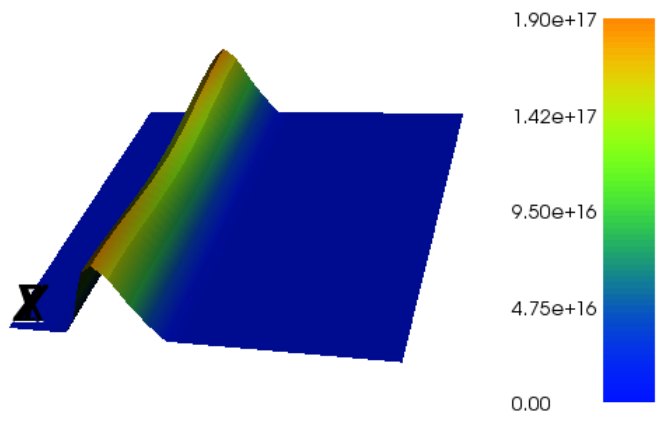


Figure 3.5: electron density on a 2D mesh, the lower left corner has coordinates of $(0, 0)$ and total area of $900 \times 900$. The density has dimensions of $cm^{-3}$

# Chapter 4

# Results

In this section, we will device our efforts to find the right Fermi level that will have a convergent solution while preserving the properties of the system. We start with Fermi Level of the order $\sim O(10^5)$. From figure 4.1.

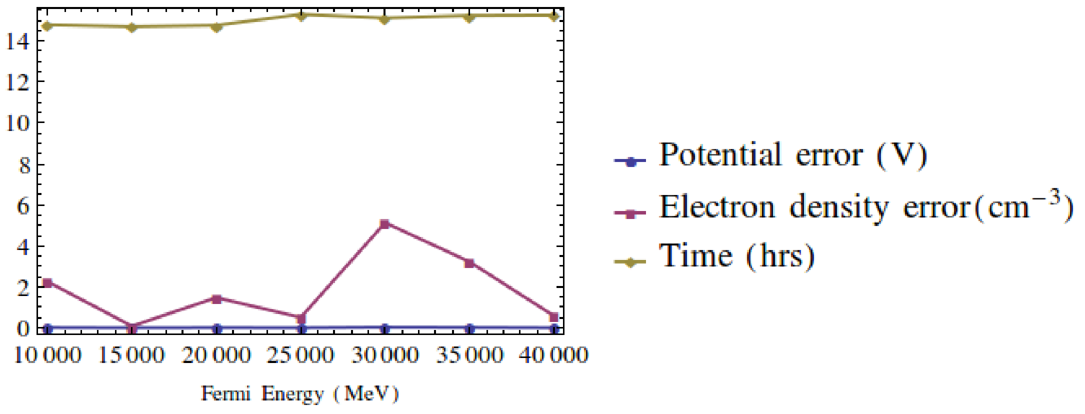


Figure 4.1: shown above is the time it took the simulation to run, the error in electron density, and the error in potential plotted against different values of Fermi level on the order of $10^5$

We can see that Fermi levels of the order $\sim O(10^5)$ are not good candidates for our wafer because they do not reach our $10^{-5}$ error requirement. Next, we try Fermi levels of the order $\sim O(10^4)$.

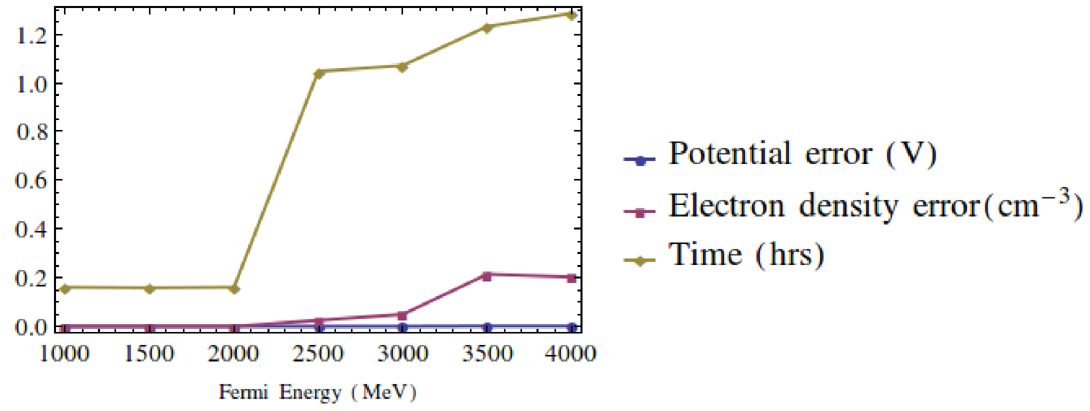
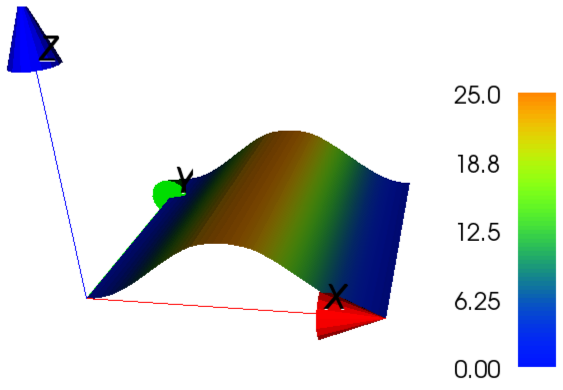
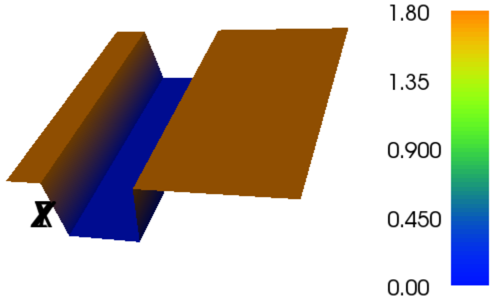


Figure 4.2: shown above is the time it took the simulation to run, the error in electron density, and the error in potential plotted against different values of Fermi level on the order of $10^4$. The values $E_f > 2000.0eV$ did not converge

For Fermi levels of the order $\sim O(10^4)$, we get some convergent solutions. Namely, for the Fermi levels $Ef = 1000.0$, 1500.0, and 2000.0 all in 4 iterations. Next, we plot the electron density and potential for a Fermi energy in the convergent range.

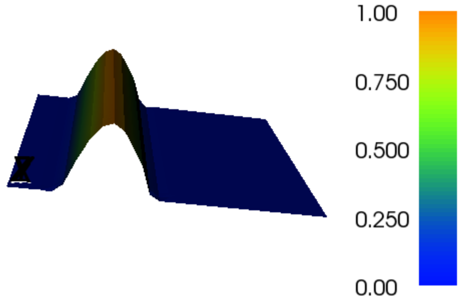


(a) Electron Density ($10^{11}/cm^3$) at room temperature with $E_f = 1020.0ev$

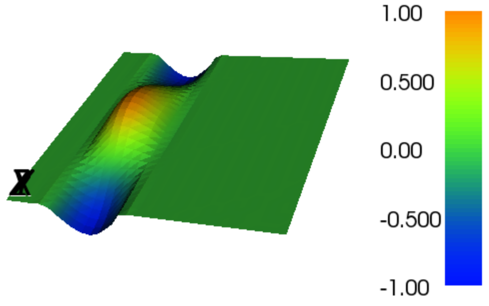


(b) Potential eV)

Figure 4.3: Convergent solution for electron density n(x,y) and potential V(x,y)

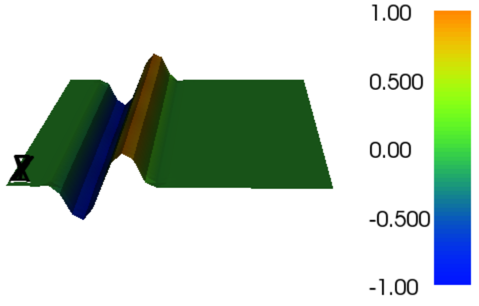


(a) wavefunction $k = 1$

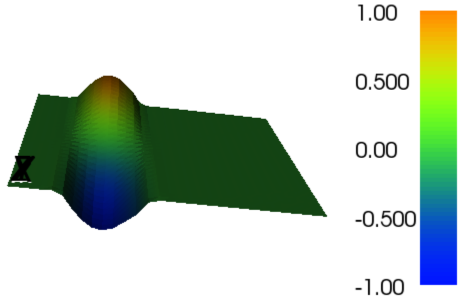


(b) wavefunction $k = 2$

Figure 4.4: First two eigenfuctions of $\psi(x, y)$



(a) wavefunction $k = 3$



(b) wavefunction $k = 4$

Figure 4.5: Third and forth eigenfuctions of $\psi(x, y)$

# Conclusion

As electronic chips becomes smaller, more work required to determine their electronic properties. When devices reaches the nanometer range, quantum effects become non-negligible. A self-consistent solution to the Schrödinger and Poisson equation is necessary to get accurate predictions of the system properties. Due to the complexity of the system, only numerical simulation are possible. We use FEM to simulate a self-consistent solution for a GaAs/AlGaAs quantum wafer. We achieve convergence for values of Fermi level ranging from $1000.0eV$ to $2000.0eV$

## Acknowledgements

I would like to thank Prof. Woowon Kang for his immense patience and allocating some of his precious time to sponsor and review this thesis. I would also like to thank the Saudi Arabian Cultural Mission (SACM) for sponsoring this research.

# Bibliography

[1] Tan, IH., G. L. Snider, L. D. Chang, and E. L. Hu. ”A self-consistent solution of Schrödinger-Poisson equations using a nonuniform mesh.” Journal of applied physics 68, no. 8 (1990): 4071-4076.

[2] Trellakis, Alex, Till Andlauer, and Peter Vogl. ”Efficient solution of the schrödinger-poisson equations in semiconductor device simulations.” In Large-Scale Scientific Computing, pp. 602-609. Springer Berlin Heidelberg, 2006.

[3] Serra, AM Cruz, and H. Abreu Santos. ”A onedimensional, self-consistent numerical solution of Schrödinger and Poisson equations.” Journal of applied physics 70, no. 5 (1991): 2734-2738.

[4] Huang, Chung-Kuang, and Neil Goldsman. ”2-D self-consistent solution of Schrödinger equation, Boltzmann transport equation, Poisson and current-continuity equation for MOSFET.” SISPAD 2001 (2001): 148-51.

[5] Ruic, Dino, and Christoph Jungemann. ”A self-consistent solution of the Poisson, Schrodinger and Boltzmann equations by a full Newton-Raphson

approach for nanoscale semiconductor devices."

[6] Kittel, Charles, and Paul McEuen. Introduction to solid state physics. Vol. 8. New York: Wiley, 1986.

[7] Simon, Steven H.. The Oxford solid state basics. Oxford: Oxford University Press, 2013.

# Chapter 5

# Appendices

In this appendex, we will assume that the reader is familiar with python. A basic tutorial for Fenics is available through their website. I will try to be as thorough as possible, but I highly recommend reading the tutorial before proceeding. Fenics package is extremely useful in solving partial differential equations with obscure geometries while minimizing the required work for setting the geometry up. The program we will describe here is designed to solve Schrödinger and Poisson equation self-consistently.

The program is divided into five files, main, meshCreator, electronDensity, poisson, and schrodinger. We will explain the functionality of each file in the following sections. To check if all the packages are installed correctly, unzip the program and type in the following command from a unix terminal window:

```
python main.py 0.0
```

Where the number after main.py signifies Fermi level. Running this should reproduce our result from the thesis.

## 5.1 Main

In main, we have to include the following files:

```
from dolfin import *

from meshCreator import *

from schrodinger import schrodingerEq

from poisson import poissonEq

from electronDensity import electronOccupationState, electronDensityFunction

from scipy import constants as pc

import numpy as np

import matplotlib.pyplot as plt

import time

import sys
```

*dolfin* is important so that we can use FEM in python, *meshCreator* file defines our function space, *schrodinger* takes care of the Schrödinger FEM formulation and solution. *poisson* is to solve Poisson in FEM formulation. *electronDensity* is to find the electron density. *scipy* and *numpy* are for defining physical constants and efficient arrays. *matplotlib* is used for plotting. *time* and *sys* are used for timing and writing the solution to a file.

Next, we proceed with our main loop. We start our loop by calling each file as follows

```
#Start of Iteration

k = 0

while k < 10:
  print str(k+1) + ' iteration'


  #get the eigenvalues and eigenfunctions
  eigenvalues, eigenvectors, u, rx_list = schrodingerEq(potential, meshArray,
     fermiEnergy)


  #find the electron density
  nk = electronOccupationState(eigenvalues, fermiEnergy)
  new_n = electronDensityFunction(eigenvectors, nk)


  #solve the poisson equation
  phi = poissonEq(new_n,meshArray)
```

This loop will run until we reach a convergent solution, or we reach our maximum number of iterations, give by

```
for i in range(len(potential.vector())):
   v_error.append((new_potential.vector()[i]-potential.vector()[i]))
   n_error.append((n.vector()[i]-new_n[i]))


  v_error = abs(sum(v_error)/len(v_error))
  n_error = abs(sum(n_error)/len(n_error))
  n_average = sum(new_n)/len(new_n)
  e_average = sum(eigenvalues)/len(eigenvalues)
  print 'v_error: ' + str(v_error)

  print 'n_error: ' + str(n_error) + '\n'

  print 'fermiEnergy (4 K) = ' + str(fermiEnergy)

  print 'average electron density = ' + str(n_average)

  print 'average eigenvalue = ' + str(e_average)



  #update potential, electron density and iteration

  potential = new_potential

  n.vector()[:] = np.array([j for j in new_n])

  k = k+1


  if(n_error < 10e-5 and v_error < 10e-5):

    print 'Convergence occured at k: ' + str(k)

    break
```

## 5.2 Creating a Mesh

Creating a mesh is not complicate as it might seem at first. We divide our subdomains by using the following code

```
# Define a MeshFunction over two subdomains

subdomains = MeshFunction('size_t', mesh, 2)
```

```
class Omega0(SubDomain):

    def inside(self, x, on_boundary):

        return (between(x[0], (x0, gaas_thickness1)))


class Omega1(SubDomain):

    def inside(self, x, on_boundary):

         return (between(x[0], (gaas_thickness1, gaas_thickness1 +

             algaas_thickness1)))


class Omega2(SubDomain):

    def inside(self, x, on_boundary):

         return (between(x[0], (gaas_thickness1 + algaas_thickness1,

             gaas_thickness1 + algaas_thickness1 + algaas_thickness2)))




class Omega3(SubDomain):

    def inside(self, x, on_boundary):

        return (between(x[0], (gaas_thickness1 + algaas_thickness1 +

            algaas_thickness2, xf)))


# Mark subdomains with numbers 0 and 1

subdomain0 = Omega0()

subdomain0.mark(subdomains, 0)

subdomain1 = Omega1()
```

```
subdomain1.mark(subdomains, 1)

subdomain2 = Omega2()

subdomain2.mark(subdomains, 2)

subdomain3 = Omega3()

subdomain3.mark(subdomains, 3)
```

Where $xf$ is our final $x$. We then have to allocate intrinsic properties of the subdomains. One of the properties is the psuedopotential energy. We do this by

```
class Step(Expression):

   def __init__(self, mesh):

      self.mesh = mesh

   def eval_cell(self, value, x, ufc_cell):

      cell = Cell(self.mesh, ufc_cell.index)

      if (between(cell.midpoint().x(), (gaas_thickness1, gaas_thickness1 +

         algaas_thickness1 + algaas_thickness2))):

        value[0] = 0.0

      else:

        value[0] = (1.0-0.3)*1.4 + (.3)*2.7


P = Step(mesh)

band_offset = interpolate(P, V0)
```

Now we can use the band offset in our Poisson function.

## 5.3 Schrodinger

Here, we have to define our matrix and then solve it using the boundary conditions. We can code it in Fenics by

```
# Define new measures associated with the interior domains and
 # exterior boundaries
 dx = Measure("dx")[domains]


 #define problem
 a = (inner(hb2m_g * grad(u), grad(v)) \
      + Vpot*u*v)*dx(1) + (inner(hb2m_a * grad(u), grad(v)) \
      + Vpot*u*v)*dx(2) + (inner(hb2m_a * grad(u), grad(v)) \
      + Vpot*u*v)*dx(3) + (inner(hb2m_g * grad(u), grad(v)) \
      + Vpot*u*v)*dx(4) + (inner(hb2m_g * grad(u), grad(v)) \
      + Vpot*u*v)*dx(0)
 m = u*v*dx(1) + u*v*dx(2) + u*v*dx(3) + u*v*dx(4) + u*v*dx(0)


 #assemble stiffness matrix
 A = PETScMatrix()
 M = PETScMatrix()
 _ = PETScVector()
 L = Constant(0.)*v*dx(0) + Constant(0.)*v*dx(1) + Constant(0.)*v*dx(2) +
    Constant(0.)*v*dx(3) + Constant(0.)*v*dx(4)
 assemble_system(a, L, A_tensor=A, b_tensor=_)
 assemble_system(m, L, A_tensor=M, b_tensor=_)


 #create eigensolver

 eigensolver = SLEPcEigenSolver(A,M)

 eigensolver.parameters['spectrum'] = 'smallest magnitude'

 eigensolver.parameters['solver'] = 'lapack'

 eigensolver.parameters['tolerance'] = 1.e-15


 #solve for eigenvalues

 print 'solving Schrodinger\'s equation...'

 eigensolver.solve()
```

## 5.4 Electron Density

In this function, find both the state occupation number and electron density function. First we find the occupation number by using the eigenvalues in the following code

```
#function to return the electron occupation state nk

def electronOccupationState(eigenvalues, fermiEnergy):

 print 'Finding n_k...'

 nk = []

 for i in range(0,len(eigenvalues)):

  ek = float(eigenvalues[i])

  result = mpmath.quad(lambda x:
    nk_factor/(1+mpmath.exp((x-fermiEnergy)/T/K)), [ek, 2*ek + fermiEnergy])
  print float(result)
  nk.append(float(result))

 return nk
```

We then use that and pass it to the electron density function

```
def electronDensityFunction(eigenvectors, nk):
 print 'Finding the electron density function n(x)'
 result = []
 for i in range(len(eigenvectors)):
   kth_term = [(j * j * nk[i]) for j in eigenvectors[i]]
   result.append(kth_term)

 n = np.sum(result, axis=0)
 return n
```

Thus, we obtain the electron density function. Now we can solve the Poisson equation.

## 5.5 Poisson

Here, we follow a similar procedure to the Schrödinger function. We define our problem and use boundary condition to find a solution.

```
# Define new measures associated with the interior domains and
# exterior boundaries

dx = Measure("dx")[domains]

ds = Measure("ds")[boundaries]


# Define variational form

a = inner(e_g*grad(u), grad(v))*dx(1) + inner(e_a*grad(u), grad(v))*dx(2) +

    inner(e_a*grad(u), grad(v))*dx(3) + inner(e_g*grad(u), grad(v))*dx(4)


L = qe0*(N_D-n)*v*dx(1) + qe0*(N_D-n)*v*dx(2) - qe0*n*v*dx(3) -

    qe0*n*v*dx(4)


# Compute solution

u = Function(V)

solve(a == L, u, bcs)
```

That concludes our program. The program is available on request via email at feras@uchicago.edu