# SOLVING SCHRÖDINGER EQUATION FOR A ONE-DIMENSIONAL SYSTEM IN A HARMONIC POTENTIAL WELL USING ARTIFICIAL NEURAL NETWORK

**CH-800**

**M.Sc. THESIS RESEARCH PROJECT(STAGE-II)**

**By**

**Soumya Shriwastava**

**(2003131024)**


**Department of Chemistry**

**Indian Institute of Technology**

**Indore**

**MAY 2022**

**INDIAN INSTITUTE OF TECHNOLOGY**

**INDORE**

**CANDIDATE'S DECLARATION**

I hereby certify that the work which is being presented in the thesis entitled "**SOLVING SCHRÖDINGER EQUATION FOR ONE DIMENSIONAL SYSTEM IN A HARMONIC POTENTIAL WELL USING ARTIFICIAL NEURAL NETWORK"** and submitted in the **DEPARTMENT OF CHEMISTRY, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from August 2021 to May 2022 under the supervision of Dr. Satya S. Bulusu, Associate Professor, Department of Chemistry, Indian Institute of Technology, Indore.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

**Soumya Shriwastava**

---

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

**Dr. Satya S. Bulusu**

---

**SOUMYA SHRIWASTAVA** has successfully given her M.Sc. Oral Examination held on 24/5/22

Signature of Supervisor M.Sc. thesis

Date: 27/5/22

Convenor, DPGC

Date: 27.05.2022

Signature of PSPC member 1

Date: 27/05/2022

Signature of PSPC member 2

Date:

# Acknowledgment

I would like to thank my supervisor, Dr. Satya S. Bulusu, for his constant support, guidance, and encouragement. He has assisted me throughout this project. I have been extremely lucky to have a supervisor who cared so much about my work and promptly responded to my questions and queries. I would also like to thank my PSPC members.

I would like to acknowledge this assistance given to me by my lab senior Ms. Aparna Gangwar. She helped me a lot by giving me guidance and encouragement whenever I was stuck in any problem. Their contribution was precious to me in this project. I feel lucky to work under her guidance.

I would like to thank all my classmates, labmates, friends. Finally, I would like to thank my parents who have always been a constant source of support for me throughout my life.

Soumya Shriwastava

## Abstract

It is well known that the solution of the Schrödinger equation beyond the hydrogen atom is a non-trivial problem. Our main objective in this report is to use artificial neural network to solve Schrödinger equation. In the study we considered one dimensional Schrödinger wave equation in harmonic potential well as a model system. Also, the ANN is faster than the Numerov method. In our project, we have considered two and three non-interacting fermion systems. We have got the wavefunctions and probability densities for the two and three fermion systems using the Numerov algorithm. Our target is fitting of kinetic energy density functional using artificial neural network. For this, we made a dataset of about 2000 probability densities using Numerov method. The dataset is obtained by randomly changing the value of the force constant in harmonic potential well. Also, the input of the artificial neural network would be the probability densities and output would be the total energy of the system. Such models can be used to calculate the energies of one-dimensional Schrödinger equation in a similar way but unknown potential energy well without really solving the Schrödinger equation analytically.

## Table of Contents

# 1. LIST OF FIGURES

# 2. Symbols Representation

| S. No. | Symbols | Name of the Symbol |
|---|---|---|
| 1. | E | Energy |
| 2. | h | Planck's Constant |
| 3. | m | Mass (kg) |
| 5. | J | Joule |
| 6. | V | Potential |
| 7. | Ψ | Wavefunction |

# 3. Acronyms

| S.No. | Acronyms | Name |
|---|---|---|
| 1. | ANN | Artificial Neural Network |
| 2. | DFT | Density Functional Theory |
| 3. | MD | Molecular Dynamics |
| 4. | NN | Neural Network |
| 5. | PES | Potential Energy Surface |

## Chapter 1

# Introduction

### 1.1 General Introduction:

Artificial neural network is used to approximate density functionals. It is a powerful tool for finding patterns in high dimensional data. It employs algorithms by which computer learns from empirical data via induction and it has been very successful in many computer applications [1]. In artificial neural network, mean errors can be systematically reduced with the increasing number of inputs. Artificial neural network are capable of solving quantum mechanical problems.

The total ground state energy of many electrons system can be expressed as the functional of ground state density and the ground state energy satisfies the energy stationery principle resulting in Euler – Lagrange equation given by:

$$\mu = \frac{\delta E[n]}{\delta n(r)}$$

Where *n(r)* is the electron density and μ is the chemical potential for the system.

The ground state energy functional E[n] is given as:

$$E[n] = T[n] + E_{ext}[n] + E_{coul}[n] + E_{XC}[n]$$

Where T[n] is the non-interacting kinetic energy, $E_{ext}[n]$ is the interaction energy of electron density with the external potential, $E_{coul}[n]$ is the classical coulomb repulsion energy, $E_{XC}[n]$ is the exchange correlation energy.

Artificial neural network, which is inspired by biological neural network, is the most important methods in machine learning technique [3]. Artificial neural networks are massively parallel interconnected networks of simple (usually adaptive)

elements and their hierarchical organizations which are intended to interact with the objects of the real world in the same way as biological nervous systems do. Examples of artificial neural networks are feed-forward neural network, radial basis function network, and restricted Boltzmann machine. As a universal approximator, an artificial neural network can be used to represent functions. Naturally it is possible to use artificial neural network as a representation of the wavefunction in a quantum system. Researchers have been trying to combine neural network theory and quantum mechanics. For example, using neural network in the real space to solve differential equations, especially Schrodinger equation in some specific potential. On using neural network, there is no need to change the input to get the desired output [2]. In this method, input values are not changing for a particular result. Only the modification of weights connection between the neurons of a specified network is required. The NNs energy expression is unbiased, which means it does not require any type of system modifications generally applicable to all types of bonding. In this project, we are using machine learning technique, artificial neural networks to map probability densities with energies [4]. Here, we are using probability density as input for artificial neural networks, and the output will be the total energies of the system.

Our main motive in this thesis is to use artificial neural network to solve Schrodinger equation. In this study we considered one-dimensional Schrodinger wave equation in harmonic potential well as a case study to use artificial neural network. This project has been taken as a prototype for typical problems.

The potential which is being used is $V = \frac{1}{2}kx^2$

Also, the potential $V = \frac{1}{2}k(x-a)^2$ which would be represented with shifted harmonic potential throughout the thesis.

## 1.2 Organization of the report:

- **Chapter 2: Theory**
- **Chapter 3 Numerov C code**
- **Chapter 4: Results and Discussion**
- **Chapter 5: Problems faced**
- **Chapter 6: Conclusion**
- **Chapter 7: References**

## Chapter 2

### Theory

### 2.1 Time Independent Schrodinger Equation:

$$\frac{-\hbar^2}{2m}\frac{d^2Y(x)}{dx^2} + V(x)Y(x) = EY(x) \qquad (1)$$

Kinetic energy operator (K.E) = $\frac{-\hbar^2}{2m}\frac{d^2}{dx^2}$

Potential energy operator (P.E) = V($x$)

It can be rewritten as:

$$\left(\frac{-\hbar^2}{2m}\frac{d^2}{dx^2} + V(x)\right)\ Y(x) = \mathrm{E}Y(x) \qquad (2)$$

$$(\mathrm{K.E} + \mathrm{P.E})\ Y(x) = \mathrm{E}Y(x) \qquad (3)$$

Where, $Y(x)$ is the wave function

Or,

$$\mathrm{H}Y(x) = \mathrm{E}Y(x) \qquad (4)$$

Here, H is the Hamiltonian operator which is equal to the sum of kinetic energy and potential energy. E is the eigenvalue and represents total energy of the system.

## 2.2 Schrödinger Equation for Harmonic Oscillator:

From equation (1)

$$\frac{-\hbar^2}{2m}\frac{d^2Y}{dx^2} + V(x)Y(x) = \mathrm{E}Y(x)$$

Here, $V(x) = \frac{1}{2}kx^2$

Above equation can rewrite as -

$$\frac{d^2Y}{dx^2} = -\frac{2m}{\hbar^2}(\mathrm{E} - \frac{1}{2}kx^2)\,Y(x) = 0 \qquad (7)$$

For adimensional results we are introducing adimensional variables x and e.

Where,

$$x = \sqrt[4]{\frac{mk}{\hbar^2}} * x$$

And $e = \frac{E}{\hbar w}$ $\left(w\sqrt{\frac{k}{m}}\right)$

'k' denotes the force constant and *w* frequency of classical oscillator.

equation (2) can be rewritten by using adimensional variables a and b and the formula is given below:-

$$\frac{d^2Y}{dx^2} - 2(e - \frac{x^2}{2})\,Y(x) = 0 \qquad (8)$$

On solving above we get the eigenvalues for harmonic oscillator:

$$e = (n + \frac{1}{2}) \qquad n = 0, 1, 2\ldots\ldots\ldots$$

Here, n is the quantum number, and the value of n is varied from 0 to ∞ . There is no unit of energy.

## 2.3 Numerov's method:

Equation for 1-D box with potential is shown in equation (1), and it can be rewritten as given below-

$$\frac{d^2Y}{dx^2} + k^2\mathrm{Y(x)} = 0 \qquad (9)$$

where,

$$k^2(\mathrm{x}) = \frac{2m}{\hbar^2}[E - V(x)]$$

A numerical method is a method which is used to solve the second-order ordinary differential equation in which the first-order term does not appear. Numerov's method [17] is a suitable algorithm to determine this type of problem because Numerov's method is simpler and one order higher (fifth) than RK4.

From equation (9)

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

The above equation is linear in *Y*, and there is no term involving the first derivative. So, Numerov's method is a suitable algorithm for this type of problem. So, we have used this method to calculate wave functions and densities.

**2.4 Description of Numerov's method:**

To describe the Numerov's method, firstly, we will write the Talor series for Ψ(x+h).

So,

$$Y(x+h) = Y(x) + h\,Y'(x) + \frac{h^2}{2}Y''(x) + \frac{h^3}{6}Y'''(x) + \frac{h^4}{24}Y''''(x) + \ldots\ldots\ldots \quad (10)$$

On adding Y(x+h) and Y(x-h) all the h of odd powers will be terminate

$$Y(x+h) + Y(x-h) = 2Y(x) + h\,Y''(x) + \frac{h^4}{24}\,Y''''(x) + O(h^6) \quad (11)$$

Subsequently, we can write second order Schrödinger equation as given below-

$$Y''(x) = \frac{(Y(x+h)+Y(x-h)-2Y(x))}{h^2} - \frac{h^2}{12}Y''''(x) - O(h^4) \quad (12)$$

We want to estimate the term including the 4th derivative so, for this; we will work on Eq. (1) with $1 + \frac{h^2}{12}\frac{d^2Y}{dx^2}$ which gives

$$Y''(x)+\frac{h^2}{12}Y''''(x)+k^2(x)Y(x) + \frac{h^2}{12}\frac{d^2Y}{dx^2}[k^2(x)\,Y(x)] = 0 \quad (13)$$

Here,

$k^2 =$

Substituting for $Y''(x) + \frac{h^2}{12}Y''''(x)$ from equation 13 in to 12

$$Y(x+h)+Y(x-h)-2Y(x)+h^2k^2(x)Y(x) + \frac{h^4}{12}\frac{d^2Y}{dx^2}[k^2(x)Y(x)] + O(h^6) = 0 \quad (14)$$

So, now we evaluate $\frac{d^2}{dx^2}[k^2(x)\,Y(x)]$ by using elementry difference formula

$$\frac{d^2}{dx^2}[k^2(x)Y(x)] \sim \frac{\left(k^2(x+h)Y(x+h)+k^2(x-h)Y(x-h)-2k^2(x)Y(x)\right)}{h^2} \quad (15)$$

Now equation (12) is substituting in equation (11) and rearranging, after that on assuming $x_\circ = x_n = x_\circ + nh$ and defining $k_n = k(x_n)$ we get

**Final equation:-**

$$Y_{n+1} = \frac{2\left(1-\frac{5}{12}h^2k_n^2\right)Y_n - \left(1+\frac{1}{12}h^2k_{n-1}^2\right)Y_{n-1}}{1+\frac{1}{12}h^2k_{n+1}^2} \qquad (16)$$

Equation (13) which is given above is used to determine $Y_n$ for n= 2, 3, 4…But there is a condition two initial values $Y_0$, and $Y_1$ should be given.

## 2.5 Working of Artificial neural network:

An artificial neural network consists of a network of artificial neurons, or we can say that artificial neural networks are parallel computing devices, which is basically an attempt to make a computer model of the brain [11]. A simple example of a feed-forward neural network that consists of three layers of artificial neurons is given below. Each neuron is represented by a circle. Suppose we have N inputs denoted by $x_i^{(1)}$. i = 1, 2... N. These inputs are represented by N neurons in the input layer. The input layer can be fed to the hidden layer through the relation.

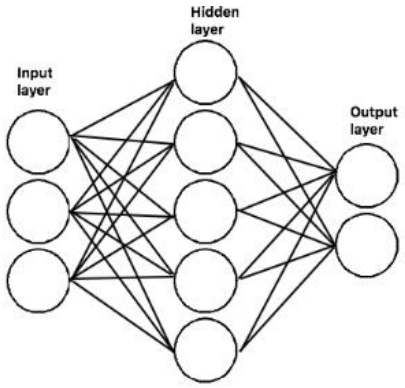



**Fig1**. Feed- forward three layer neural network.

$$Y_j^1 = \sum_i w_{ji}^{(1)} x_i^{(1)} + b_j^{(1)} \quad (17)$$

Here, $w_{ji}^{(1)}$ is called weight and $b_j^{(1)}$ is called bais. j=1, 2..., M is the index labeling the hidden layer and M is the number of neurons in the hidden layer. In the hidden layer, each$Y_j^{(1)}$ is transformed to the input of the next layer through the activation function

$$x_j^{(2)} = a*(y_j^{(1)} \quad (18)$$

It has been reported that mostly sigmoid function is used as activation function.

$$\text{Sigmoid (x)} = \sigma(x) = \frac{1}{1+e^{-x}} \quad (19)$$

After the activation function, $x_j^{(2)}$ is fed to the output layer through

$$Y_k^{(2)} = \sum_j w_{kj}^{(2)} x_j^{(2)} + b_k^{(2)} \quad (20)$$

Here, k labeling the output layer. Then $Y_k^{(2)}$ is trans-formed to the final output of the neural network through the activation function

$$Z_k = \sigma^*( \quad |k^{(2)}) \tag{21}$$

The learning process can be carried out by minimizing the error function

$$E(w, b) = \frac{1}{2}\sum_{i=1}^{N_t}|Z(x_i; w, b) - Z_o(x_l)|\ v^2 \tag{22}$$

In this equation w, b are the weights and biases in the neural network. $N_{t,}$ denotes number of elements in the training set, and Z, $Z_0$ is the output of the neural network the measured value in the training set respectively [15]. The main objective of the learning process is to find out the optimal w and b so that the error function E(w,b) become less.

## Chapter 3

### Methods

3.1). Simple Harmonic Oscillator potential is given by

$$V = \frac{1}{2}kx^2 \qquad (23)$$

Where k is the force constant.

3.2). For the two electron and three electron systems, we have used the Numerov algorithm to get the wavefunctions and the electron densities respectively. The C code that we are using for it is:

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

static double pi = 3.14159265358979323846;
void numerov(double k1, int nodes,double xa[1050], double ya[1050], double e[1],
double v[1050]);

int main()
{
   char fileout[80];
   FILE *out, *in, *out1;
   double totts[1050],tsyterm1,tsyterm2,dx,xmax,e[6],v[1050],tottsvW[1050];
   double xa[1050], ya[1050], tts, toten, dtottsvW[1050], tsyterm3, k1;
   double dens01[6][1050],ts[6][1050],totdens[1050], avgdens[1050],tsy,chempot;

   int num, occu,ith,j, nd, nodes, npart;

   int mesh=250; xmax =6.0;
   dx=xmax/mesh;
```

```
//Here, mentioning the parameter file name


 out=fopen("h_density2.dat", "w");
out1=fopen("h_avgdens2.dat","w");

//in =fopen("gparameters_1dpot",  "r");
in =fopen("hparameters2",  "r");

for (j=0; j < 2*mesh; j++){
avgdens[j] =0.0;
}
for (num=1; num < 4; num++)
{
  fscanf(in,"%d\n",&ith);
  //fscanf(in,"%lf%lf%lf\n",&pa1,&pa2,&pa3);
  //fscanf(in,"%lf%lf%lf\n",&pb1,&pb2,&pb3);
 // fscanf(in,"%lf%lf%lf\n",&pc1,&pc2,&pc3);
  tsy=0.0;
  tts=0.0;
  toten=0.0;
  nodes=1;
  npart=2;

  for (nd=0; nd<nodes+1; nd++)
  {
   for (j=0; j < 2*mesh; j++)
```

```
    {
      dens01[nd][j]=0.0;
      ts[nd][j]=0.0;
      ya[j]=0.0;
      totts[j]=0.0;
      totdens[j]=0.0;
    }
   }

   for (nd=0; nd<nodes+1; nd++)
   {

    numerov(pa1,pa2,pa3,pb1,pb2,pb3,pc1,pc2,pc3,nd,xa,ya,e,v);
    if (nd == 0) { occu = 1;}
    else if (nd >= 1) { occu = 1;}

    for (j=0; j <= 2*mesh; j++)
    {
     if  (j == 0 || j == 2*mesh) {tsyterm1 =0.0; tsyterm2=0.0;}
     else { tsyterm1=(ya[j+1] - ya[j-1])/(2.0*dx);
          // tsyterm2=ya[j]*(2*ya[j] - ya[j+1] - ya[j-1])/(2.0*dx*dx);
           }
        ts[nd][j]=occu*(tsyterm1*tsyterm1);
        dens01[nd][j]=occu*ya[j]*ya[j];
        //if (xa[j] >= -3.0 && xa[j] <= 3.0)
{printf("%7.3f%16.8e\n",xa[j],dens01[nd][j]);}
    }
```

```
 toten = toten + occu * e[0];
 //printf("%4d%16.5f%16.5f\n", num, toten, e[0]);

}
chempot = toten/npart;
//c1 =c1 + num;
for (j=0; j <= 2*mesh; j++)
{
    if(j>0 && ts[0][j] ==0){ts[0][j] = 0.5*(ts[0][j-1]+ts[0][j+1]);}
  if(j>0 && ts[1][j] ==0){ts[1][j] = 0.5*(ts[1][j-1]+ts[1][j+1]);}

  totts[j] = 0.5*(ts[0][j] + ts[1][j]);
  totdens[j]=(dens01[0][j] + dens01[1][j]);
  //printf("%7.3f%16.8e%16.8e\n",xa[j],totts[j], totdens[j]);

}
//discrete data integration
double integral=0.0;
for (j=0; j <= 2*mesh-1; j++)
{
 if (xa[j] >= -6.0 && xa[j] <= 6.0){
 integral=integral + ((dx/2.0)*(totts[j] + totts[j+1]));}
}
 for (j=0; j <= 2*mesh; j++)
{
 if  (j == 0 || j == 2*mesh) {tottsvW[j]=0.0;dtottsvW[j]=0.0;}
```

```
      else
      { tsyterm1=(totdens[j+1] - totdens[j-1])/(2.0*dx);
        tottsvW[j] = (1.0/8.0)*(tsyterm1 * tsyterm1 / totdens[j]);
        tsyterm2= ((totdens[j+1] - 2*totdens[j] + totdens[j-1])/(dx*dx));
        dtottsvW[j] = (-1.0)*(((-1.0/4.0)*(tsyterm2/totdens[j])) +
(tottsvW[j]/totdens[j]));
        dtottsvW[j]=dtottsvW[j];
      }
      }

     //printf("%4d%16.5f\n", num, integral);
       fprintf(out,"%4d%16.5f\n", num, integral);
     //printf("\n");
     // fprintf(out,"\n");
     for (j=0; j <= 2*mesh; j++)
     {
      //if (xa[j] >= -9.0 && xa[j] <= 9.0)
{fprintf(out,"%7.3f%16.8e%16.8e%16.8e\n",xa[j],totts[j],totdens[j],tottsvW[j]);}
       if (xa[j] >= -6.0 && xa[j] <= 6.0)
{fprintf(out,"%7.3f%16.8e%16.8e%16.8e\n",xa[j],totts[j],(-
chempot+v[j]),totdens[j]);}
      //if (num==3){
       // if (xa[j] >= -6.0 && xa[j] <= 6.0)
{fprintf(out,"%7.3f%16.8e\n",xa[j],totdens[j]);}
       //printf("%7.3f%16.8e\n",xa[j],totdens[j]);
     // if (xa[j] >= -6.0 && xa[j] <= 6.0)
{fprintf(out,"%7.3f%16.8e%16.8e%16.8e\n",xa[j],v[j],chempot,(-chempot+v[j]));}
```

```
      avgdens[j] = avgdens[j] + totdens[j];
     }
     for (j=0; j <= 2*mesh; j=j+10){
    //     fprintf("%7.3f%16.8e\n",xa[j],totdens[j]);
     }

   }
   for (j=0; j <= 2*mesh; j++){
      avgdens[j] = avgdens[j]/3;
      if (xa[j] >= -6.0 && xa[j] <= 6.0)
{fprintf(out1,"%7.3f%16.8e\n",xa[j],avgdens[j]);}
   }
   //printf("%16.5e\n", c1/13999);
   fclose(in);
   fclose(out1);
   fclose(out);
}//end of main

////////////////////////////////////start of function////////////////////////////////
void numerov(double k1, int nodes, double xa[1050],double ya[1050], double
energy[0], double v[1050])
{
    char fileout[80];
    FILE *out2;
    int mesh, ii,i, icl, imin, imax, k;
    int  hnodes, ncross, parity, kkk, n_iter;
    double xmax, dx, ddx12, xmcl, norm, arg, yicl, djump, xmin;
```

```
    double elw, eup, tsyyterm, tsyterm;
   // double *x, *y, *p, *vpot, *f, *xx, *yy, *vpot1;
    double                                                    x[1050],y[1050],
p[1050],vpot[1050],f[1050],xx[1050],yy[1050],vpot1[1050];
   // double ext1, ext2, ext3, e;
    double e;

    mesh=250; xmax=6.0;
/*  Allocate arrays (from 0 to mesh), Initialize grid */
/*       x = (double *) malloc( (mesh+50) * sizeof (double));
      xx= (double *) malloc( (mesh+50) * sizeof (double));
      yy= (double *) malloc( (mesh+50) * sizeof (double));
      y = (double *) malloc( (mesh+50) * sizeof (double));
      p = (double *) malloc( (mesh+50) * sizeof (double));
      f = (double *) malloc( (mesh+50) * sizeof (double));
   vpot = (double *) malloc( (mesh+50) * sizeof (double));
   vpot1= (double *) malloc( (mesh+50) * sizeof (double)); */


   dx = xmax / mesh;
   ddx12 = dx * dx / 12.;

/*  set up the potential (must be even w.r.t. imin) */

   for (i = 0; i <= mesh; ++i)
   {
        x[i] = (double) i * dx;
```

```
    //ext1 = ((x[i]-b1)*(x[i]-b1))/(2*(c1*c1));
    //ext2 = ((x[i]-b2)*(x[i]-b2))/(2*(c2*c2));
    //ext3 = ((x[i]-b3)*(x[i]-b3))/(2*(c3*c3));
    vpot[i] = 0.5*k1*(x[i])*(x[i]);

    //printf("%7.3f%16.8f\n", x[i],vpot[i]);
}

eup = vpot[mesh];
elw = eup;
for (i = 0; i <= mesh; ++i) {
     if ( vpot[i] < elw ){
        elw = vpot[i];
        xmin=x[i];
        imin=i;
        f[i]=0.0;
        y[i]=0.0;}}
k=0;
for (i=imin-1; i >= -mesh; --i){
    xx[k] = (double)i*dx;
   vpot1[k] = 0.5*k1*(xx[k])*(xx[k]);



    //printf("%7.3f%16.8f\n", xx[k],vpot1[k]);
    k=k+1;
    }
```

```
  e=0.;
if (e == 0.) { /* search eigenvalues with bisection (max 1000 iterations) */
    e = 0.5 * (elw + eup);
    n_iter = 1000;
} else {     /*  test a single energy value */
    n_iter = 1;
}

for (kkk = 0; (kkk < n_iter) && (eup-elw > 1.e-10); kkk++) {

    f[imin] = ddx12 * (2.*(vpot[imin] - e));
    icl = -1;
    for (i = imin+1; i <= mesh; ++i)
  {
    f[i] = ddx12 * 2. * (vpot[i] - e);
      if (f[i] == 0.)   {
          f[i] = 1e-20; }
      if (f[i] != copysign(f[i],f[i-1])) {
        icl = i;
      }
    }


    /*   f(x) as required by the Numerov algorithm  */

    for (i = imin; i <= mesh; ++i) {
    f[i] = 1. - f[i];
```

```
}
for (i = imin; i <= mesh; ++i) {
y[i] = 0.;
}
hnodes = nodes / 2;
if (2*hnodes == nodes) {
  y[imin] = 1.;
  y[imin+1] = 0.5 * (12. - f[imin] * 10.) * y[imin] / f[imin+1];
} else {
  y[imin] = 0.;
  y[imin+1] = dx;
}
ncross = 0;
for (i = imin+1; i <= icl-1; ++i) {
y[i + 1] = ((12. - f[i] * 10.) * y[i] - f[i - 1] * y[i - 1])
    / f[i + 1];
if (y[i] != copysign(y[i],y[i+1]))
    ++ncross;
}

yicl = y[icl];

if (2*hnodes == nodes) {
ncross = 2*ncross;
} else {
ncross = 2*ncross+1;
}
```

```
if (n_iter > 1) {
if (ncross != nodes) {
    if ( kkk == 1) {
    }
    if (ncross > nodes) {
      eup = e;
    } else {
      elw = e;
    }
    e = 0.5 * (eup + elw);
}
} else {
}
if ( n_iter == 1 ||  ncross == nodes ) {
  y[mesh] = dx;
  y[mesh - 1] = (12. - 10.*f[mesh]) * y[mesh] / f[mesh-1];

  /*     Inward integration */
  for (i = mesh - 1; i >= icl+1; --i) {
    y[i-1] = ((12. - 10.*f[i]) * y[i] - f[i+1] * y[i+1]) / f[i-1];
  }
  /*     Rescale function to match at the classical turning point (icl) */
  yicl /= y[icl];
  for (i = icl; i <= mesh; ++i) {
    y[i] *= yicl;
  }
```

```
      if (n_iter > 1) {
        i = icl;
        djump = (y[i+1] + y[i-1] - (14. - 12.*f[i]) * y[i]) / dx;
       // fprintf(stdout, "%5d%25.15e%5d%14.8f\n", kkk, e, nodes, djump);
        if (djump*y[i] > 0.) {
          /*            Energy is too high --> choose lower energy range */
          eup = e;
        } else {
          /*            Energy is too low --> choose upper energy range */
          elw = e;
        }
        e = 0.5 * (eup + elw);
      }
    } /* end if (ncross==nodes) */
} // end of iterations

norm = 0.;
for (i = imin; i <= mesh; ++i) {
   norm += y[i]*y[i];
 }
 norm = dx * (2* norm);
 norm = sqrt(norm);
 for (i = imin; i <= mesh; ++i) {
 y[i] /= norm;
 }
 k=0;
 for (ii = imin-1; ii >=-mesh; ii=ii-1)
```

```
 {
    i = (2*(k+1)) + ii; yy[k] = y[i];
 // printf("%4d%4d%4d\n", ii, i, k);
  k=k+1;
 }
 //exit(0);
 i=k-1;
 int j;
 //printf("\n");
 // applying parity for the wavefunction
 if (hnodes << 1 == nodes)
  parity = +1;
else
  parity = -1;
 out2=fopen("ground", "w");
 for (j=0;j<=k-1;j++)
 {
     xa[j]=xx[i];
     ya[j]=parity*yy[i];
     v[j] =vpot1[i];
       i=i-1;
 //  fprintf(out2,"%7.3f\t%16.8e\n",xa[j],ya[j]);
 }
for (i = imin;i<=mesh;++i)
{
    xa[j]=x[i];
    ya[j]=y[i];
```

```
        v[j]=vpot[i];
//        printf("%4d%4d\n", i, j);
    //  fprintf(out2,"%7.3f\t%16.8e\n",xa[j],ya[j]);
        j++;
    }
    for (i =mesh; i >=0; --i) {
     xa[j]=x[i];
      ya[j]=parity*y[i];
       v[j]=vpot[i];
     //fprintf(out2, "%7.3f %16.8e\n", x[i], parity*y[i]);

     // j++;
    }


    fclose(out2);
energy[0]=e;
//free(vpot);free(f);free(p);free(y);free(x);free(vpot1);free(xx),free(yy);
}
```

3.3) To train the neural network, we have considered the system of N non interacting fermions where we have taken (N = 2,3).

For the 2SL system we have considered N = 2, where the two fermions, we have considered to be spinless and are filled following the Pauli's exclusion principle.

One electron is filled in the ground state and the second electron is filled in the first excited state.

For N =3, we have considered two configurations arising from the choice of fermions:

The first configuration is 3S, in which the two fermions are filled in the lowest energy state occupying the opposite spin and the third electron is in the first excited state.

The second configuration is 3SL in which the three fermions are considered to be spinless, in which all the three energy levels i.e. ground state, first excited state and the second excited states are singly occupied.

The system is confined to the coordinate values of x ranging from -3 to +3.

This gives the wavefunction and corresponding eletron densities for the two and three electrons systems from -x to +x coordinates.

3.4) Now, we have shifted the potential from 0 to 6 coordinates (i.e. in positive x coordinates). So, for this, we would do some changes in the potential of the system. So, the potential corresponding to the changes for the system would be:

$$V = \frac{1}{2}k(x-a)^2 \qquad (24)$$

where

k is the force constant.

a is the mean value.

For this system, we have used Numerov algorithm corresponding to the changes. Similarly, here also, we have got the wavefunction and corresponding densities for the two electron and three electron systems (i.e. 2SL, 3S, 3SL).

For this the Numerov algorithm, we have used the above C code, the only change in it is the potential mentioned n equation

3.5) By randomly changing the value of force constant between (1.08 to 1.2), we have got the dataset of densities for 2000 potentials. For each data set there would be individual densities and single kinetic energy associated with each data point.

3.6) This output file of the numerical densities would be used as the fitting into the orthogonal basis functions. So, to use the numerical densities as the descriptors, we fitted them with the mathematical functions under the constraint $\int n(x)\,dx = N$. Then we expanded each density in the basis function as:

$$n(x) = \sum_{i=0}^{M} a_i \psi_i(x) \qquad (25)$$

So, for it, we considered a wavefunction to be:

$$\psi_i(x) = H_n(x) exp(-bx^2) \qquad (26)$$

Where,

$Hn(x)$ is Hermite polynomial which is being used for the orthogonality of the wavefunction, $a_i$ and $\psi_i(x)$ are the variational parameters.

$e^{-ax^2}$ is the exponential form of the orthogonal basis function.

We have used the orthogonal basis function ranging the value of i from 0 to 5. Correspondingly, the values of Hermite polynomials are as follows:

$$Ho(x) = 1$$

$$H_1(x) = 2x$$

$H_2(x) = 4x^2 - 2$

$$H_3(x) = 8x^3 - 12x$$

$$H_4(x) = 16x^4 - 48x^2 + 12$$

$$H_5(x) = 32x^5 - 160x^3 + 120x$$

After few trials, we fixed the value of b to be 3.0 and the value of M to be 5. The linear fitting coefficients are obtained using constraint minimization using Sequential Least-Squares Programming (SLSQP) method in Scipy optimization module. Then, we selected the best values of linear fitting coefficients. This was determined by the chi-square test. Through this test, we set the value of b to be less than 0.1, so that, we get the best fitting. We did this for 2SL, 3S and 3SL fermions system. For these, we took the 2000 data set of densities and for each dataset, there are individual densities known as data points.

The linear fitting coefficients, thus obtained, used as the input to train the artificial neural network for fitting of kinetic energy.

3.7) The python code which is used for fitting is as follows:

```
####################################
# For any fitting procedure check the arguments and based on the structure of arguments
# it is good to make repective functions
# Two examples are given 1. cureve fit and 2. leastsq
# Here we will deal with minimize program
#
import numpy as np
import math
```

```
import matplotlib.pyplot as plt
from scipy import asarray as ar, exp, sqrt
from scipy.optimize import curve_fit, minimize, leastsq, basinhopping
import statistics
from scipy.stats import chisquare
from itertools import islice
from scipy.integrate import quad, simps
################################################
#np.seterr(all="raise")
#b =p/2
n=252
#p=1.5
#b=2*p
b= 2.0
a= 3.0
c= 1.00
file1=open("2_1.dat","w")
#with open('hdensity.dat', 'r') as f:
with open('temp.dat', 'r') as f:
        while True:
                next_n_lines = list(islice(f, n))
                if not next_n_lines:
                  break
                #print(next_n_lines[0], file=file1)
               # print(next_n_lines[0])
                #try:
                next_n_lines.pop(0)
                lines_array=[]
                lines_array = np.array(next_n_lines)
                X=[]
                Y=[]
                Y1=[]
                Y2=[]
                y00=[]
                y01=[]
                y02=[]
                y03=[]
                y04=[]
                y05=[]
                co0=[]
```

```
co1=[]
co2=[]
co3=[]
co4=[]
co5=[]
for line in lines_array:
  line = line.rstrip('\n')
  values= line.split()
  X.append(float(values[0]))
  Y.append(float(values[1]))
  Y1.append(float(values[2]))
  Y2.append(float(values[3]))

#print(Y1, sep="\n",file=file1)
X=np.asfarray(X)
Y=np.asfarray(Y)
Y1=np.asfarray(Y1)
Y2=np.asfarray(Y2)
#data to be fitted and initial parameters
a0  = 1.0
a1  = 1.0
a2  = 1.0
a3  = 1.0
a4  = 1.0
a5  = 1.0

def n00(X,b,a,c):
   ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
   psi0 =ex
   return(psi0**2)
res0= quad(n00,-math.inf,math.inf, args=(b,a,c))

def n01(X,b,a,c):
   ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
   psi1=2.0*(math.sqrt(b)*((X-a)))*ex
   return (psi1**2)
res1= quad(n01,-math.inf,math.inf, args=(b,a,c))

def n02(X,b,a,c):
   ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
```

```
        psi2=(4.0*(b*((X-a)**2))-2.0)*ex
        return (psi2**2)
    res2= quad(n02,-math.inf,math.inf,args=(b,a,c))

    def n03(X,b,a,c):
        ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
        psi3=ex*(8.0*(math.sqrt(b**3)*((X-a)**3))-(12.0*(math.sqrt(b)*((X-
a)))))
        return (psi3**2)
    res3= quad(n03,-math.inf,math.inf,args=(b,a,c))

    def n04(X,b,a,c):
        ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
        psi4=ex*((16.0*(b**2)*((X-a)**4))-(48.0*(b*((X-a)**2)))+12.0)
        return(psi4**2)
    res4= quad(n04,-math.inf,math.inf,args=(b,a,c))

    def n05(X,b,a,c):
        ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
        psi5=ex*((32.0*(math.sqrt(b**5))*((X-a)**5))-
(160.0*(math.sqrt(b**3))*((X-a)**3))+(120.0*(math.sqrt(b)*(X-a))))
        return(psi5**2)
    res5= quad(n05,-math.inf,math.inf,args=(b,a,c))


    ri00 = math.sqrt(1/res0[0])
    ri01 = math.sqrt(1/res1[0])
    ri02 = math.sqrt(1/res2[0])
    ri03 = math.sqrt(1/res3[0])
    ri04 = math.sqrt(1/res4[0])
    ri05 = math.sqrt(1/res5[0])
    print(ri00,ri01,ri02,ri03,ri04,ri05)
    #exit()
    print("normalization")
    def np0(X,b,a,c,ri00):
        return((ri00*(np.exp(-(b*((X-a)/c)**2)*0.5)))*(ri00*(np.exp(-(b*((X-
a)/c)**2)*0.5))) )
    psi0_n = quad(np0,0,math.inf,args=(b,a,c,ri00))[0]
    print(psi0_n)
    #exit()
```

```
        def np1(X,b,a,c,ri01):
            ex=(np.exp(-(b*((X-a)/c)**2)*0.5) )
            return((ri01*(2.0*(math.sqrt(b)*(X-a))*ex))* (ri01*(
2.0*(math.sqrt(b)*(X-a))*ex)))
        psi1_n =quad(np1,0,math.inf,args=(b,a,c,ri01))[0]
        print(psi1_n)
        #exit()
        def np2(X,b,a,c,ri02):
            ex=(np.exp(-(b*((X-a)/c)**2)*0.5) )
            return((ri02*((4.0*(b*(X-a)**2)-2.0)*ex))*(ri02*((4.0*(b*(X-a)**2)-
2.0)*ex)))
        psi2_n =quad(np2,0,math.inf,args=(b,a,c,ri02))[0]
        print(psi2_n)
        #exit()
        def np3(X,b,a,c,ri03):
            ex=(np.exp(-(b*((X-a)/c)**2)*0.5) )
            return((ri03*((8.0*(math.sqrt(b**3)*((X-a)**3)))-
(12.0*(math.sqrt(b)*(X-a))))*ex)*(ri03*((8.0*(math.sqrt(b**3)*((X-a)**3)))-
(12.0*(math.sqrt(b)*(X-a))))*ex))
        psi3_n =quad(np3,0,math.inf,args=(b,a,c,ri03))[0]
        print(psi3_n)
        #exit()
        def np4(X,b,a,c,ri04):
            ex=(np.exp(-(b*((X-a)/c)**2)*0.5) )
            return((ri04*ex*((16.0*(b**2)*((X-a)**4))-(48.0*(b*((X-
a)**2)))+12.0))*(ri04*ex*((16.0*(b**2)*((X-a)**4))-(48.0*(b*((X-
a)**2)))+12.0)))
        psi4_n = quad(np4,0,math.inf,args=(b,a,c,ri04))[0]
        print(psi4_n)
        #exit()
        def np5(X,b,a,c,ri05):
            ex=(np.exp(-(b*((X-a)/c)**2)*0.5) )
            return((ri05*ex*((32.0*(math.sqrt(b**5))*((X-a)**5))-
(160.0*(math.sqrt(b**3))*((X-a)**3))+(120.0*(math.sqrt(b)*(X-
a)))))*(ri05*((32.0*(math.sqrt(b**5))*((X-a)**5))-(160.0*(math.sqrt(b**3)*((X-
a)**3)))+(120.0*(math.sqrt(b)*(X-a))))*ex))
        psi5_n = quad(np5,0,math.inf,args=(b,a,c,ri05))[0]
        print (psi5_n)
        #exit()
        #print(psi0_n, psi1_n, psi2_n, psi3_n, psi4_n, psi5_n, psi6_n)
```

```
print("orthogonal")
def no01(X,b,a,c,ri00,ri01):
    ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
    return((ri00*ex)*(ri01*ex*(( 2.0*(math.sqrt(b)*(X-a))))) )
psi0_1= quad(no01,0,math.inf,args=(b,a,c,ri00,ri01))[0]
print(psi0_1)
#exit()
def no02(X,b,a,c,ri00,ri02):
    ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
    return((ri00*ex)*(ri02*((4.0*(b*(X-a)**2)-2.0)*ex)))
psi0_2 =quad(no02,0, math.inf,args=(b,a,c,ri00,ri02))[0]
print(psi0_2)
#exit()
def no03(X,b,a,c,ri00,ri03):
    ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
    return((ri00*ex)*(ri03*ex*((8.0*(math.sqrt(b**3)*(X-a)**3)-
(12.0*math.sqrt(b)*(X-a)))) ) )
psi0_3 =quad(no03,0,math.inf,args=(b,a,c,ri00,ri03))[0]
print(psi0_3)
#exit()
def no04(X,b,a,c,ri00,ri04):
    ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
    return((ri00*ex)*(ri04*(ex*((16.0*(b**2*(X-a)**4)-(48.0*(b*(X-
a)**2))+12.0)))) )
psi0_4 =quad(no04,0,math.inf,args=(b,a,c,ri00,ri04))[0]
print(psi0_4)
#exit()
def no05(X,b,a,c,ri00,ri05):
    ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
    return((ri00*ex) *(ri05*ex*(((32.0*(math.sqrt(b**5)*(X-a)**5)-
(160.0*(math.sqrt(b**3)*(X-a)**3))+(120.0*math.sqrt(b)*(X-a))) ))))
psi0_5 =quad(no05,0,math.inf,args=(b,a,c,ri00,ri05))[0]
print(psi0_5)
#exit()
def no12(X,b,a,c,ri01,ri02):
    ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
    return((ri01*ex*(( 2.0*math.sqrt(b)*(X-a))))*(ri02*ex*((4.0*(b*(X-
a)**2)-2.0))))
psi1_2 =quad(no12,0,math.inf,args=(b,a,c,ri01,ri02))[0]
print(psi1_2)
```

```
            #exit()
            def no13(X,b,a,c,ri01,ri03):
                ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
                return((ri01*ex*(( 2.0*math.sqrt(b)*(X-
a))))*(ri03*ex*((8.0*(math.sqrt(b**3)*(X-a)**3)-(12.0*math.sqrt(b)*(X-a))))) )
            psi1_3 =quad(no13,0,math.inf,args=(b,a,c,ri01,ri03))[0]
            print(psi1_3)
            #exit()
            def no14(X,b,a,c,ri01,ri04):
                ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
                return((ri01*(ex*( 2.0*math.sqrt(b)*(X-
a))))*(ri04*(ex*((16.0*(b**2*(X-a)**4)-(48.0*(b*(X-a)**2)))+12.0))))
            psi1_4=quad(no14,0,math.inf,args=(b,a,c,ri01,ri04))[0]
            print(psi1_4)
            #exit()
            def no15(X,b,a,c,ri01,ri05):
                ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
                return((ri01*ex*(( 2.0*math.sqrt(b)*(X-
a))))*(ri05*ex*(((32.0*(math.sqrt(b**5)*(X-a)**5)-(160.0*(math.sqrt(b**3)*(X-
a)**3))+(120.0*math.sqrt(b)*(X-a))) ))))
            psi1_5=quad(no15,0,math.inf,args=(b,a,c,ri01,ri05))[0]
            print(psi1_5)
            #exit()
            def no23(X,b,a,c,ri02,ri03):
                ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
                return((ri02*ex*((4.0*(b*(X-a)**2)-
2.0)))*(ri03*(ex*(8.0*(math.sqrt(b**3)*(X-a)**3)-(12.0*math.sqrt(b)*(X-a))))) )
            psi2_3 =quad(no23,0,math.inf,args=(b,a,c,ri02,ri03))[0]
            print(psi2_3)
            #exit()
            def no24(X,b,a,c,ri02,ri04):
                ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
                return((ri02*ex*((4.0*(b*(X-a)**2)-
2.0)))*(ri04*ex*((16.0*(b**2*(X-a)**4)-(48.0*(b*(X-a)**2))+12.0))) )
            psi2_4=quad(no24,0,math.inf,args=(b,a,c,ri02,ri04))[0]
            print(psi2_4)
            #exit()
            def no25(X,b,a,c,ri02,ri05):
                ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
```

```
            return((ri02*ex*((4.0*(b*(X-a)**2)-
2.0)))*(ri05*ex*(((32.0*(math.sqrt(b**5)*(X-a)**5)-(160.0*(math.sqrt(b**3)*(X-
a)**3))+(120.0*math.sqrt(b)*(X-a))) ))))
        psi2_5 =quad(no25,0,math.inf,args=(b,a,c,ri02,ri05))[0]
        print(psi2_5)
        #exit()
        def no34(X,b,a,c,ri03,ri04):
            ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
            return((ri03*ex*((8.0*(math.sqrt(b**3)*(X-a)**3)-
(12.0*math.sqrt(b)*(X-a)))))  *(ri04*ex*((16.0*(b**2*(X-a)**4)-(48.0*(b*(X-
a)**2))+12.0))))
        psi3_4=quad(no34,0,math.inf,args=(b,a,c,ri03,ri04))[0]
        print(psi3_4)

        def no35(X,b,a,c,ri03,ri05):
            ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
            return((ri03*ex*((8.0*(math.sqrt(b**3)*(X-a)**3)-
(12.0*math.sqrt(b)*(X-a)))))*(ri05*ex*(((32.0*(math.sqrt(b**5)*(X-a)**5)-
(160.0*(math.sqrt(b**3)*(X-a)**3))+(120.0*math.sqrt(b)*(X-a)) )))))
        psi3_5= quad(no35,0,math.inf,args=(b,a,c,ri03,ri05))[0]
        print(psi3_5)
        #exit()
        def no45(X,b,a,c,ri04,ri05):
            ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
            return((ri04*ex*((16.0*(b**2*(X-a)**4)-(48.0*(b*(X-
a)**2))+12.0)))  *(ri05*ex*(((32.0*(math.sqrt(b**5)*(X-a)**5)-
(160.0*(math.sqrt(b)*(X-a)**3))+(120.0*math.sqrt(b)*(X-a))) ))))
        psi4_5 = quad(no45,0,math.inf,args=(b,a,c,ri04,ri05))[0]
        print(psi4_5)
        #exit()

        def ni0(X,b,a,c,ri00):
            ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
            npsi0=ri00*ex
            return(npsi0)
        in0= quad(ni0,0,math.inf,args=(b,a,c,ri00))[0]

        def ni1(X,b,a,c,ri01):
            ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
            npsi1=ri01*(2.0*(math.sqrt(b)*(X-a))*ex)
```

```
        return(npsi1)
    in1=quad(ni1,0,math.inf,args=(b,a,c,ri01))[0]

    def ni2(X,b,a,c,ri02):
        ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
        npsi2=ri02*((4.0*(b*(X-a)**2)-2.0)*ex)
        return(npsi2)
    in2=quad(ni2,0,math.inf,args=(b,a,c,ri02))[0]

    def ni3(X,b,a,c,ri03):
        ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
        npsi3=(ri03*((8.0*(math.sqrt(b**3)*(X-a)**3))-
(12.0*(math.sqrt(b)*(X-a))))*ex)
        return(npsi3)
    in3=quad(ni3,0,math.inf,args=(b,a,c,ri03))[0]

    def ni4(X,b,a,c,ri04):
        ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
        npsi4=(ri04*ex*((16.0*(b**2)*(X-a)**4)-(48.0*(b*(X-
a)**2))+12.0))
        return(npsi4)
    in4 = quad(ni4,0,math.inf,args=(b,a,c,ri04))[0]

    def ni5(X,b,a,c,ri05):
        ex=(np.exp(-(b*((X-a)/c)**2)*0.5))
        npsi5=(ri05*ex*((32.0*(math.sqrt(b**5))*(X-a)**5)-
(160.0*(math.sqrt(b**3))*(X-a)**3)+(120.0*math.sqrt(b)*(X-a))))
        return(npsi5)
    in5=quad(ni5,0,math.inf,args=(b,a,c,ri05))[0]

    #psi0 = ni0(X,b,a,c,ri00)
    #psi1 = ni1(X,b,a,c,ri01)
    #psi2 = ni2(X,b,a,c,ri02)
    #psi3 = ni3(X,b,a,c,ri03)
    #psi4 = ni4(X,b,a,c,ri04)
    #psi5 = ni5(X,b,a,c,ri05)
    #psi6 = ni6(X,b,a,c,ri06)
    #print(in0,in1,in2,in3,in4,in5)
    #exit()
    off =0.0
```

```
            ###################################################
             def multigaus(xx, ri00, ri01, ri02, ri03, ri04, ri05, b,a,c, params1):
                a0, a1, a2, a3, a4, a5 = params1
                ex=(np.exp(-(b*((xx-a)/c)**2)*0.5))
                psi0=ex*ri00
                psi1=ex*ri01*(2.0*(math.sqrt(b)*(xx-a)))
                psi2=ex*ri02*((4.0*(b*((xx-a)**2))-2.0))
                psi3=ex*ri03*((8.0*(math.sqrt(b**3)*((xx-a)**3)))-
(12.0*(math.sqrt(b)*(xx-a))))
                psi4=ex*ri04*((16.0*(b**2)*((xx-a)**4))-(48.0*(b*((xx-
a)**2)))+12.0)
                psi5=ex*ri05*((32.0*(math.sqrt(b**5))*((xx-a)**5))-
(160.0*(math.sqrt(b**3))*((xx-a)**3))+(120.0*(math.sqrt(b)*(xx-a))))
                return (a0*psi0 + a1*psi1 + a2*psi2 + a3*psi3 + a4*psi4 + a5*psi5)
            ###################################################
             def penal(params1, ri00, ri01, ri02, ri03, ri04, ri05,b,a,c,x, y):
                (a0, a1, a2, a3, a4, a5) = params1
                residual= sum((y- multigaus(x, ri00, ri01, ri02, ri03, ri04, ri05, b, a, c,
(a0,a1,a2,a3,a4,a5)))**2)
                return residual
            ###################################################
             def constraint(params1):
                (a0, a1, a2, a3, a4, a5) = params1
                #penalty = ((a0*1.4306128055802485)+(a1*1.106551718532689e-
06)+(a2*1.0115902662668896)+(a3*2.4846131574496333e-
05)+(a4*0.8759782176040072)+(a5*0.00028435409468241524)) -2
                penalty =
((a0*1.5832159998788626)+(a1*0.00011023504751145334)+(a2*1.119035081899
96617)+(a3*0.0016651209787289058)+(a4*0.9646574849507779)+(a5*0.012387
598577603937)) -2


                return penalty
            ###################################################
             params1 =(a0, a1, a2, a3, a4, a5)
             con = {'type':'eq','fun':constraint}
             #bounds=((0,50),(0,50),(0,50),(0,50),(0,50),(0,50),(-10,50))
             #minimizer_kwargs = {"method":"SLSQP","args":(ri00, ri01, ri02, ri03,
ri04, ri05, ri06, b,a,c, X, Y2),"constraints":con, "bounds":bounds}
```

```
            minimizer_kwargs = {"method":"SLSQP","args":(ri00, ri01, ri02, ri03,
ri04, ri05, b,a,c,X,Y2),"constraints":con}
            fit2 = basinhopping(penal, (a0,a1,a2,a3,a4,a5),
minimizer_kwargs=minimizer_kwargs,niter=100)
            yfit2 = multigaus(X, ri00, ri01, ri02, ri03, ri04, ri05, b,a,c,fit2.x)
            #res= quad(multigaus,ri00, ri01, ri02, ri03, ri04, ri05,b,-
math.inf,math.inf, args=(fit2.x))
            res= simps(yfit2,X)
            #pa = (fit2.x[0], fit2.x[1], fit2.x[2])
            #pb = (fit2.x[3], fit2.x[4], fit2.x[5])
            #cfita=gaus(X, (fit2.x[0], fit2.x[1], fit2.x[2]))
            #cfitb=gaus(X, (fit2.x[3], fit2.x[4], fit2.x[5]))
            print(*fit2.x, sep=' ', file=file1)
            print(fit2.fun, file=file1)
            print(res, file=file1)
            #print(fit2.x, fit2.fun, res1)
            #print(p)
            #print(nin0, nin1, nin2, nin3, nin4, nin5)
            #print(i00, i01, i02, i03, i04, i05)
            #print(res1)
            plt.plot(X, Y2, 'bo', label="y-original")
            #plt.plot(X, yfit1, color="orange", label="unconstrained mimimize")
            plt.plot(X, yfit2, color="green", label="constrained mimimize")
            #plt.plot(X, cfita, color="red", label="cfita")
            #plt.plot(X, cfitb, color="black", label="cfitb")
            plt.xlabel('X')
            plt.ylabel('Y2')
            plt.legend(loc='best', fancybox=True, shadow=True)
            plt.grid(True)
            plt.show()
            #exit()
            #except:
            #print ("error")
file1.close()
```

Now, as mentioned above, we have also considered a system in which we have shifted the coordinates of the box. Now, the coordinate of the box is changed from (-3 to +3) to (0 to +6). The corresponding potential is as follows:

$$V = \frac{1}{2}k(x-a)^2 \qquad (27)$$

Now for such systems, we have randomly changed the value of the parameter k from 1.08 to 1.4 and then correspondingly, we have generated the 2000 potentials. For each potential, the Schrödinger equation is solved numerically to get the energy eigen values and the wavefunctions. This has been done with the Numerov C code which is mentioned above.

3.8) The electron density is employed from the following function:

$$n(x) = \sum_{i=1}^{N} f_i |\phi_1 \cdot (x)|^2 \qquad (28)$$

Where $f_i$ denotes the occupation number.

As kinetic energy is a function of density which is a function of wave function, so, the kinetic energy functional can be written in the form of:

$$T[n] = \int_{-\infty}^{\infty} \tau(x)\, dx \qquad (29)$$

Where $\tau(x)$ denotes the kinetic energy density, which is also obtained by employing the function as follows:

$$\tau(x) = \frac{1}{2}\sum_{i=1}^{N} |\nabla \phi_i(x)|^2 \qquad (30)$$

Here, we generated the 2000 data set of densities. These numerical densities are used as the descriptors of artificial neural network, so we fit them with the mathematical functions under the constraints $\int n(x)\, dx = N$. Now, similarly as mentioned above, we expanded each density in terms of basis functions as:

$$n(x) = \sum_{i=0}^{M} a_i \psi_i(x) \qquad (31)$$

For it, we considered wavefunction to be:

$$\psi_i(x) = H_n(x)exp(-b(x-a)^2) \quad (32)$$

Where a denotes the mean value.

$a_i$ and b denotes the variational parameters.

$Hn(x)$ is the Hermite polynomial which is being used for the orthogonality of the wavefunctions.

$e^{-a(x-a)^2}$ is the exponential part of the wavefunction.

The value of hermite polynomials ranging from n = 0 to 5 is as follows:

$$Ho(x) = 1$$

$$H_1(x) = 2(x-a)$$

$H_2(x) = 4(x-a)^2 - 2$

$$H_3(x) = 8(x-a)^3 - 12(x-a)$$

$$x - a$$

$$H_4(x) = 16(x-a)^4 - 48$$

$$H_5(x) = 32(x-a)^5 - 160(x-a)^3 + 120(x-a)$$

Now, after so many trials, the value of b is set to be 3.0. The value of a is set to be 3.0.

The linear fitting coefficients are obtained using constraint minimization using Sequential Least-Squares Programming (SLSQP) method in Scipy optimization module. Then, we selected the best values of linear fitting coefficients. This was

determined by the chi-square test. Through this test, we set the value of b to be less than 0.1, so that, we get the best fitting. We did this for 2SL, 3S and 3SL fermions system. For these, we took the 2000 data set of densities and for each dataset, there are individual densities known as data points.

The linear fitting coefficients, thus obtained, used as the input to train the artificial neural network for fitting of kinetic energy.

The python code which is used for fitting is mentioned in point (3.8). Only the difference is that we have used the above Hermite polynomial and exponential function in the basis set.

**3.9) Similarly, we are using the Pöschl Teller potential, which is given by:**

$$v_{(x)} = \frac{-\hbar^2}{2m}[l(l+1)sech^2(x)] \qquad (33)$$

Where $l$ is 1,2, 3,…is a natural number. Here, we have obtained the family of potentials, by changing the value of $l$.

For this system, we have got the wavefunction and densities for two electron and three electron system (i.e. 2SL, 3S, 3SL) as mentioned above.

We have used the Numerov C code for getting wavefunction. Randomly changing the value of l between the range (3 to 5), we have generated data sets for numerical densities for 2000 potentials.

These numerical densities are used as the descriptors of artificial neural network, so we fit them with the mathematical functions under the constraints $\int n(x)\,dx = N$. Now, similarly as mentioned above, we expand each density in terms of basis functions as:

$$n(x) = \sum_{i=0}^{M} a_i \psi_i(x) \qquad (34)$$

So, for it, we would consider a wavefunction to be:

$$\psi_i(x) = H_n(x)exp(-bx^2) \quad (35)$$

Where,

$Hn(x)$ is Hermite polynomial which is being used for the orthogonality of the wavefunction, $a_i$ and $\psi_i(x)$ are the variational parameters.

$e^{-ax^2}$ is the exponential form of the orthogonal basis function.

### 3.10) Fitting of Kinetic energy T[n]:

For the training of kinetic energy, we used feed forward artificial neural network with 5 nodes in each layer. We have used the inear ftting coefficients as the input for the artificial neural network.

Out of the 2000 data sets we divided them in such a way that 1000 datasets used for raining purpose and rest (999) for testing purpose.

The ANN KE expression used is given by:

$$T_j = \sum_{l=1}^{hn} W_{l1}^{23}.af_l^2\left[Wb_l^2 + \sum_{k=1}^{hn} W_{kl}^{12}.af_k^1\left(Wb_k^1 + \sum_{i=0}^{M} W_{ik}^{01}.a_{i,j}\right)\right]$$

where, $T_j$ is energy of $j^{th}$ density and $h_n$ is number of hidden nodes. $a_{i,j}$ is input descriptor of length M for $j^{th}$ density. $W^{01}{}_{ik}$, $W^{12}{}_{kl}$ and $W^{23}{}_{l1}$ are weights connecting from input layer to hidden layer one, hidden layer one to hidden layer two and hidden layer two to output layer respectively. $af^{2}{}_{l}$ and $af^{1}{}_{k}$ represents sigmoid activation functions of network and $Wb^{2}{}_{l}$, $Wb^{1}{}_{k}$ are bias weights.

The network had total 360 weights which were optimized using Global Advanced Extended Kalman Filter. The average root mean square error was calculated and validated the training weights iteratively at each point.

$$RMSE(Energy) = \sqrt{\frac{1}{Nt}\sum_{k=1}^{Nt}\left(T_k^{Exact} - T_k^{ANN}\right)^2}$$

Where Nt is the number of datapoints in the test set.

The exact KE is obtained by the following expression:

$$T_{Exact} = \int_{-3}^{3}\sum_{i=1}^{N}(\nabla|\phi_i(x)|)^2 dx$$

## Chapter 4

# Results and Discussion

4.1) We have solved the one-dimensional Schrödinger equation in harmonic potential well by using Numerov method. We have calculated the wavefunction, probability density for the ground state, first excited state and the second excited state respectively. The value of energy for the ground state, first excited state and the second excited state are 0.5 hv, 1.5hv, 2.5hv respectively. The Numerov method

is used to solve the 1- dimensional Schrodinger wave equation in harmonic potential well.

4. 2) The Simple Harmonic Oscillator potential that we are using is:

$$v(x) = \frac{1}{2}kx^2$$

**4.2.a) The plot of harmonic potential is as follows:**

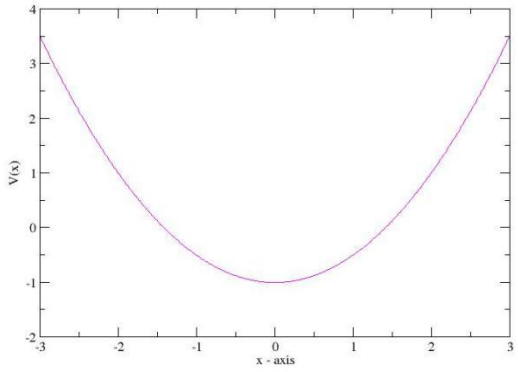



Figure (4.2.a) Plot for harmonic potential

We have solved the Schrödinger equation for a one-dimensional system in Harmonic potential well by using the Numerov algorithm. We have calculated the wavefunctions and probability densities for two electron and three electron system (i.e., 2SL, 3S, 3SL). Also, we generated the datasets for densities by randomly changing the value of force constant for 2000 potentials.

The values of force constant k have been taken in the range of 1.08 to 1.2.

Then, we fitted the probability densities with the analytical equations and generated the values of linear fitting (i.e., $a_0$, $a_1$, $a_2$, $a_3$, $a_4$, $a_5$). Firstly, we generated the linear fitting coefficients for one dataset and then we generated for the 2000 dataset of densities.

**4.2.b) The wavefunction for ground state harmonic oscillator wavefunction is as follows:**

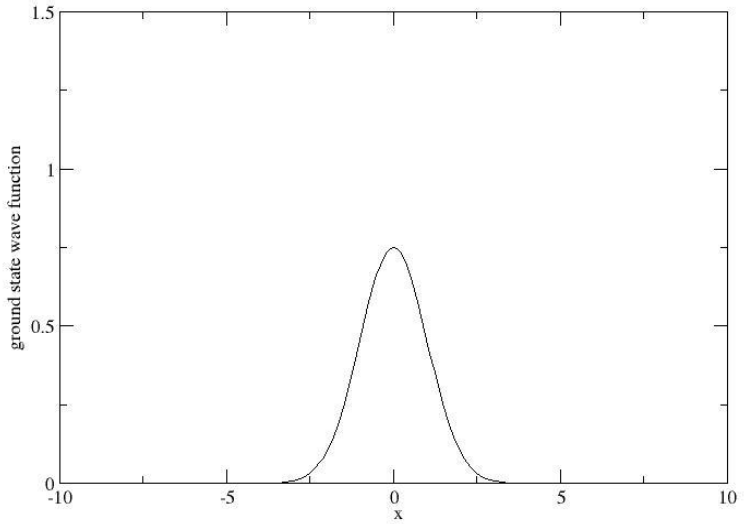



Figure (4.2.b) Wavefunction for ground state harmonic oscillator potential

**4.2.c) The wavefunction for first excited state harmonic oscillator is as follows:**

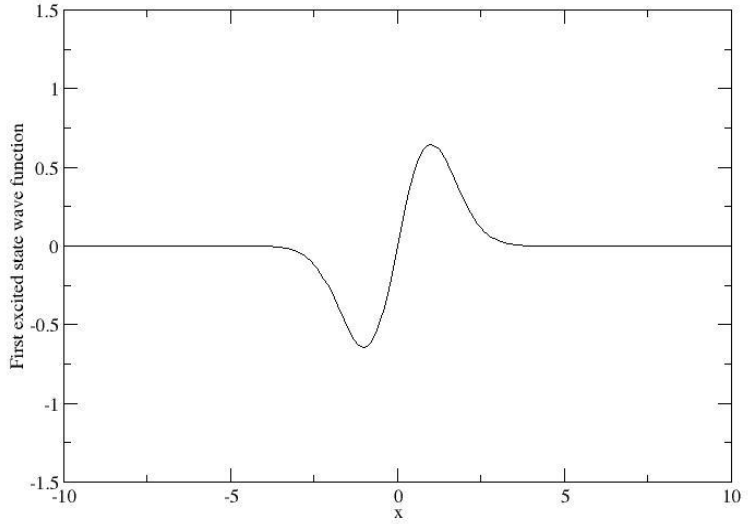



Figure (4.2.c) Wavefunction for first excited state harmonic oscillator potential

**4.2.d) The wave function for second excited state potential is as follows:**

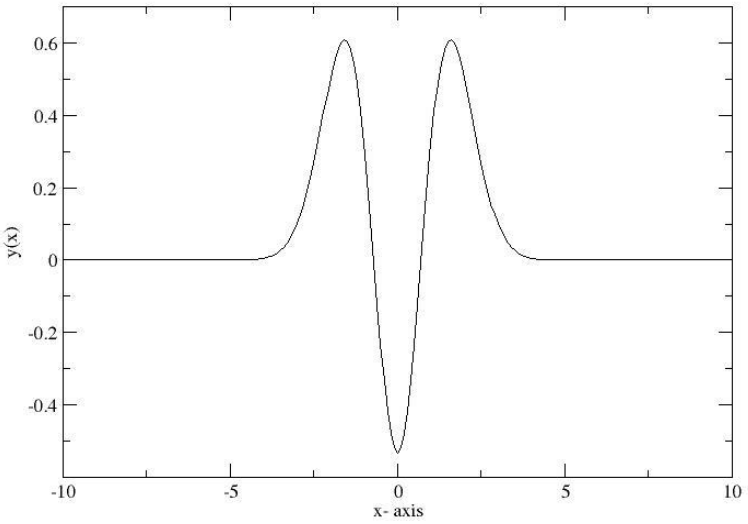



Figure (4.2.d) Wavefunction for second excited state harmonic oscillator potential

**4.3) Data generation:**

As already mentioned, we have generated the datasets for 2000 potentials for all the three systems by randomly changing the value of the force constant and got the 2000 datasets for densities.

The plots for 2000 datasets of densities are as follows:

**4.3.a) Plots for densities for two electron and one node system (i.e. 2SL)**

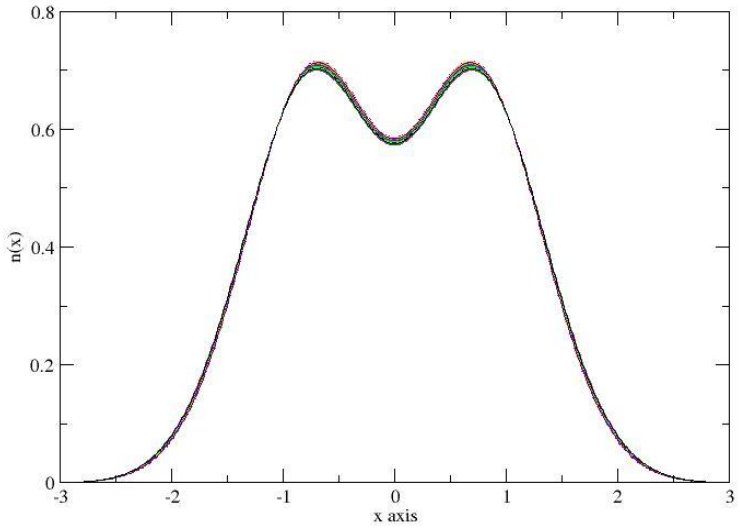



Figure (4.3.a) Plots for 2000 dataset of densities for 2SL system

**4.3.b.) Plots for the dataset of densities for three electron and 1 node system (i.e. 3S)**

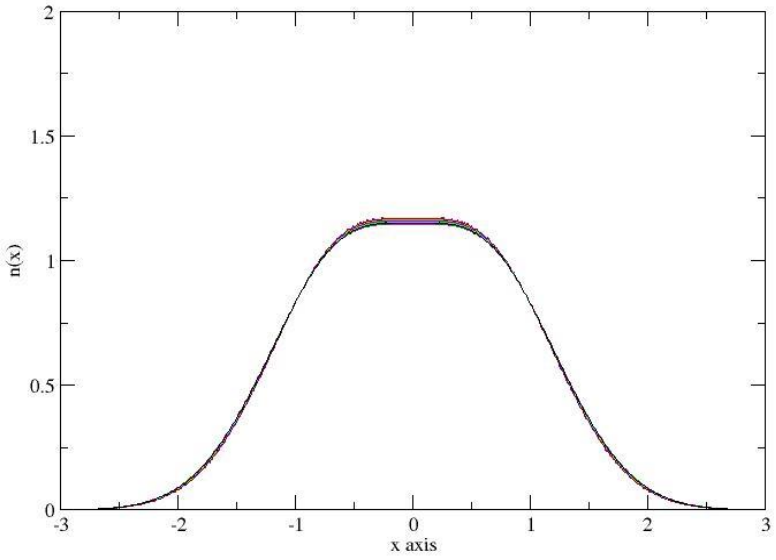



Figure (4.3.b) Plots for 2000 dataset of densities for 3S system

## 4.3.c.) Plot for the dataset of densities for the three electron 2 node system (3SL)

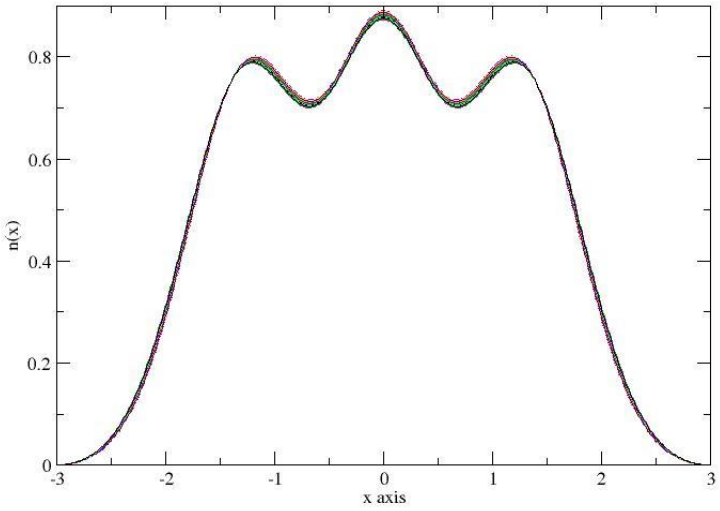



Figure (4.3.c) Plots for 2000 dataset of densities for 3SL system

Now, the plots for first datapoint in the density dataset for each system is shown as follows:

## 4.3.e) Average density plot for the 2SL system:

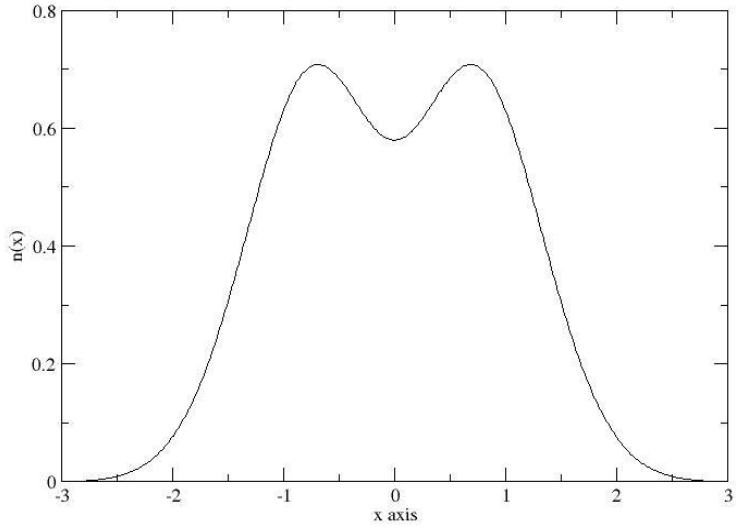



Figure (4.3.e) Average density plots for 2S system for harmonic oscillator potential

## 4.3.f.) Average density plot for the 3S sytem:

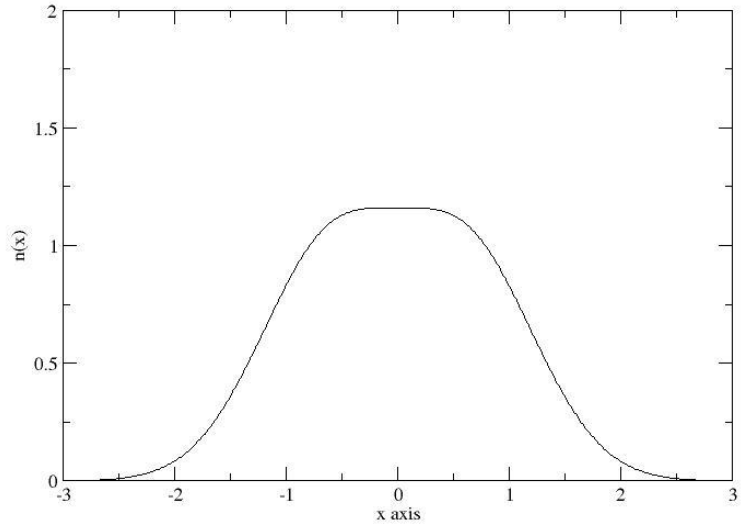



Figure (4.3.f) Average density plots for 3SL system for harmonic oscillator potential

## 4.3.g.) Average density plot for the 3SL system:

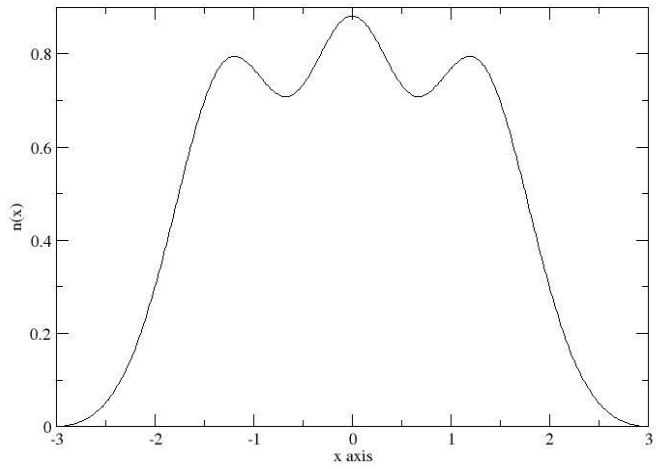



Figure (4.3.g) Average density plots for 3SL system for harmonic oscillator potential

Similarly for the harmonic potential which is being shifted from 0 to positive x coordinates, i.e.

$$V = \frac{1}{2}k(x-a)^2$$

**4.3.h) The plot of harmonic potential for such system is as follows:**

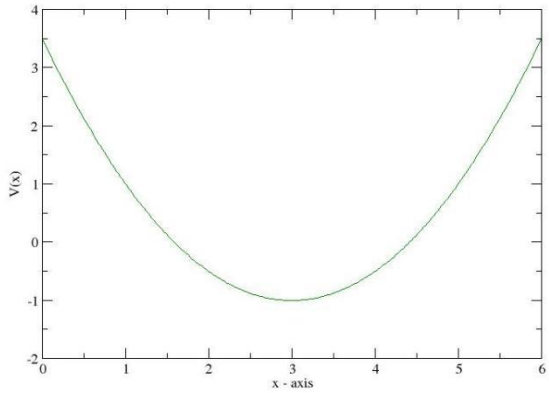



Figure (4.3.h) Plot for shifted harmonic oscillator potential

Similarly, for the 2SL, 3S and 3SL system plots for datasets of densities obtained from 2000 potentials is as follows:

### 4.3.i) Plots for numerical densities dataset for 2SL system

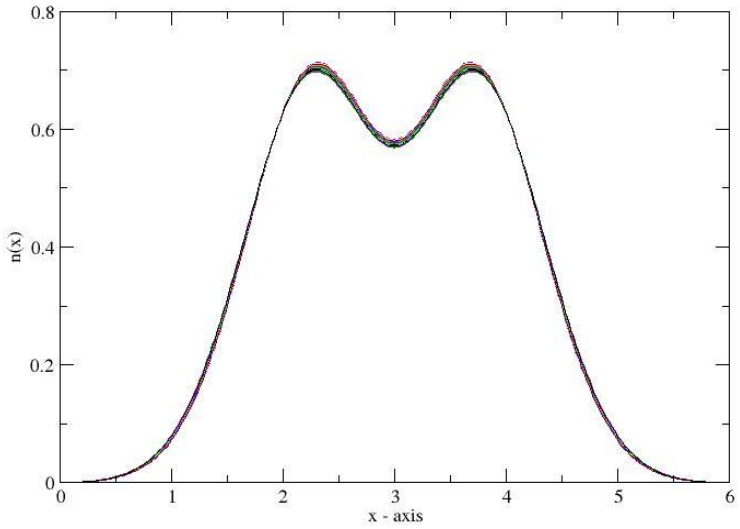



Figure (4.3.i) Plots for 2000 dataset of densities for 2SL system

### 4.3.j) Plots for numerical densities dataset for 3S system:

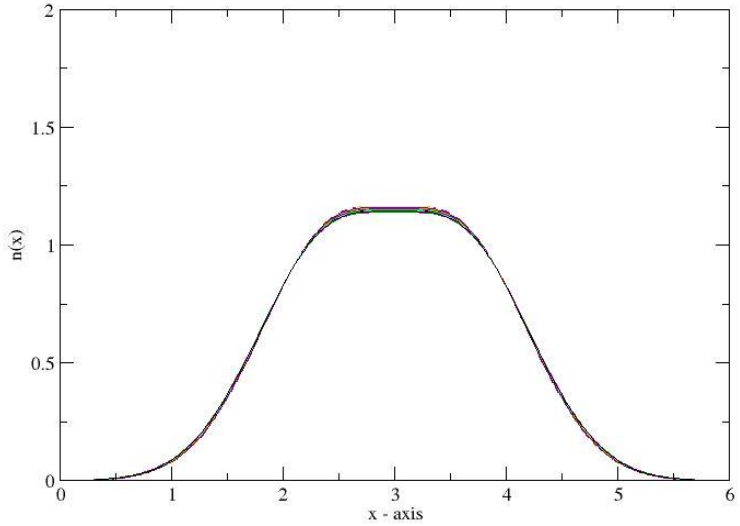



Figure (4.3.j) Plots for 2000 dataset of densities for 3S system

### 4.3.j) Plots for numerical dataset ofdensities for 3SL system

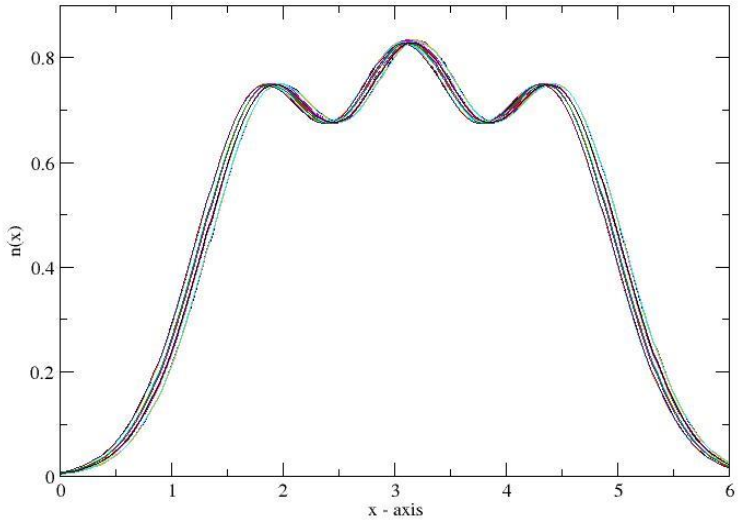



Figure (4.3.J) Plots for 2000 dataset of densities for 3SL system

Similarly for average densities plots for all the three systems are as follows:

## 4.3.k). Average density plots for 2SL system:

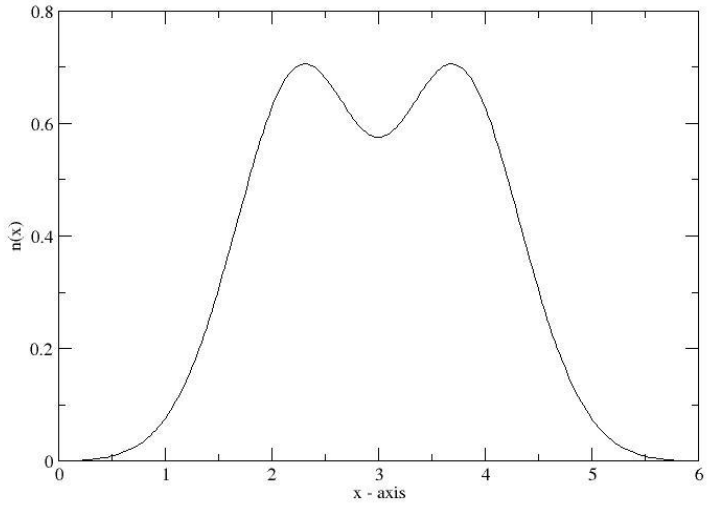



Figure (4.3.k) Average density plots for 2SL system for harmonic oscillator potential

## 4.3.l) Average density plot for 3S system:

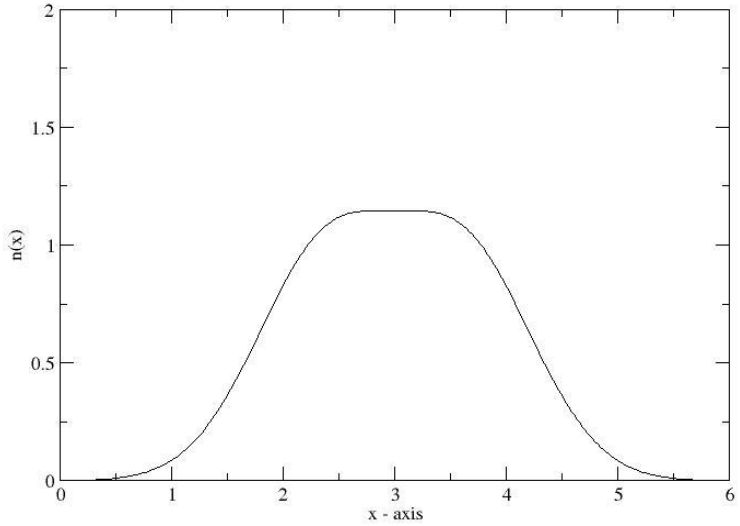



Figure (4.3.l) Average density plots for 3S system for harmonic oscillator potential

### 4.3.m) Average density plots for 3SL system

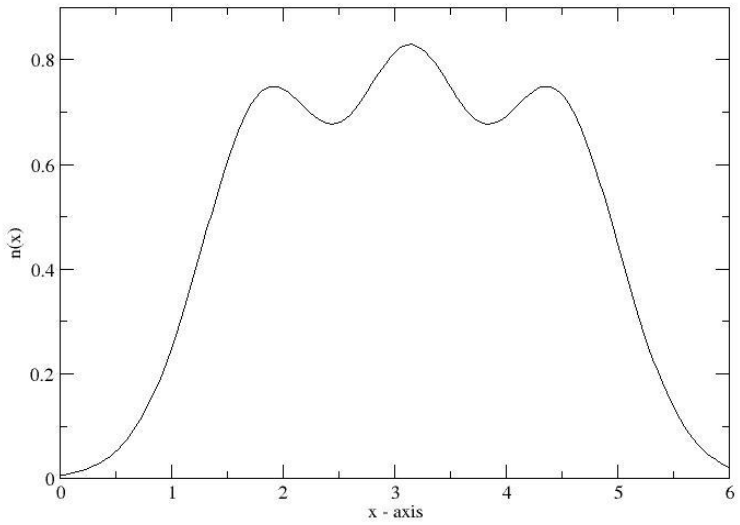



Figure (4.3.m) Average density plots for 3SL system for harmonic oscillator potential

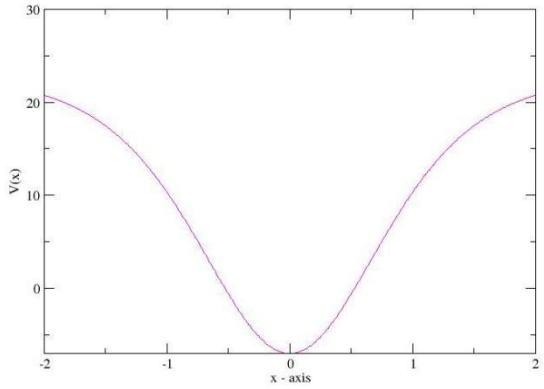
4.4) By using numerov algorithm, we have also solved the Schrödinger equation for the Pöschl Teller potential. We have got the wavefunction and corresponding probability densities for two electron and the three electron systems.

Pöschl Teller potential is expressed as :

$$v_{(x)} = -\frac{\hbar^2}{2m}[l(l+1)\,\mathrm{sech}^2\,(x)]$$

Where l can take 1,2,3…..N integer values.

**The plot for Pöschl Teller potential is as follows:**




Figure(4.4) Plot for Pöschl Teller potential

Similarly as mentioned above, the probability densities plots for dataset of 2000 potentials by randomly changing the value of the parameter l.

**4.4.a.) Plot for the two electron, 1 node system dataset of densities:**

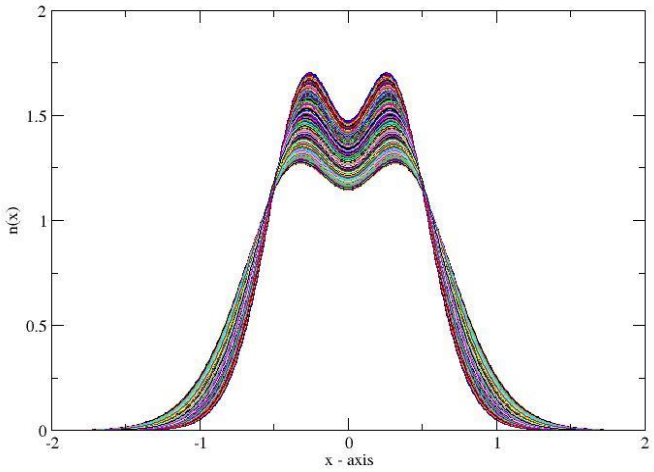



Figure (4.4.a) Plot for 2000 dataset of densities for 2SL

### 4.4.b.) plot for the three elctron and one node system datset of densities:

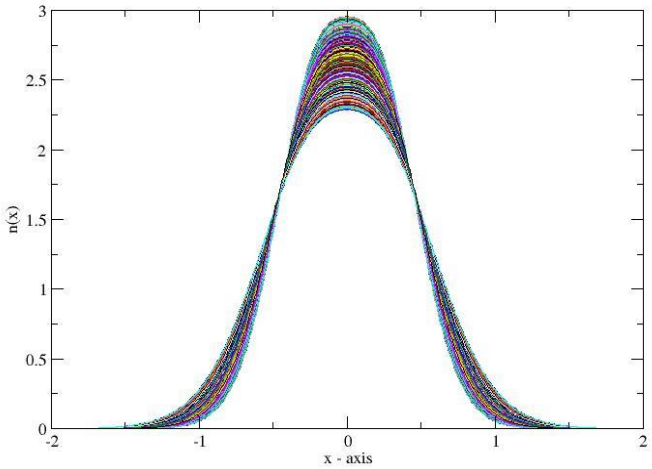



Figure (4.4.b) Plot for 2000 dataset of densities for 3S

### 4.4.c.) Plot for the three electron and two node system dataset of densities:

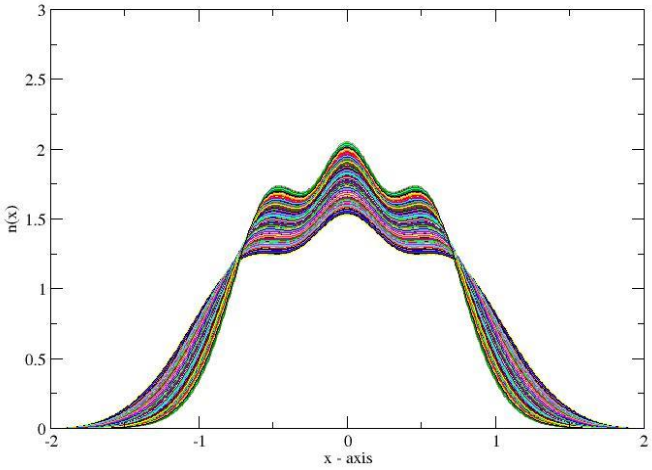



Figure (4.4.c) Plot for 2000 dataset of densities for 3SL

We have also plotted the first data point of the density dataset in the following figure.

## 4.4.d) Average density plot for 2SL system:

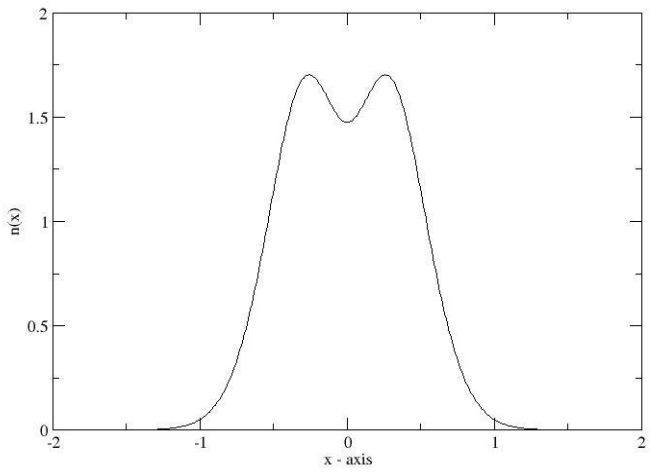



Figure (4.4.d) Plot for average density for 2SL

## 4.4.e) Average density plots for 3S system:

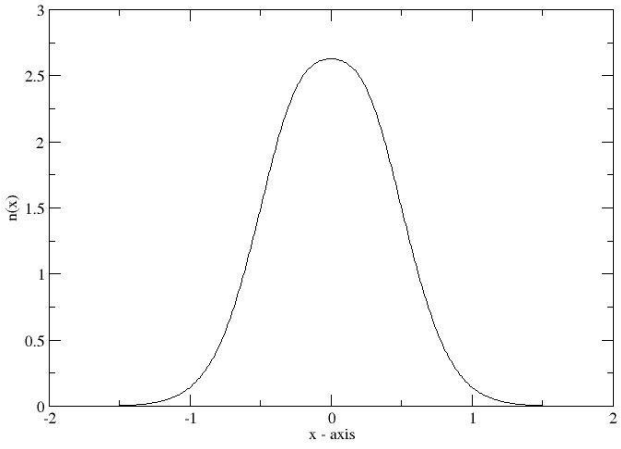



Figure (4.4.e) Plot for average density for 3S

### 4.4.f) Average density plots for 3SL system:

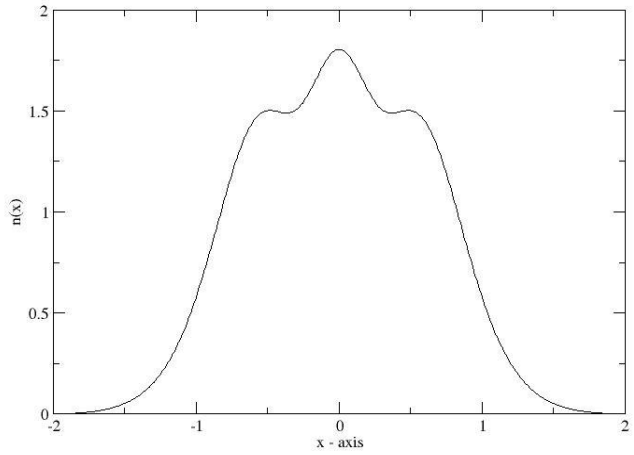



Figure (4.4.f) Plot for average density for 3SL

### 4.5) Fitting equation and plots for all the three systems:

Here, we have established a general form of the probability density to use linear fitting coefficients as input for the artificial neural network. We need to ensure that the fitting coefficients which we are using are relatively independent. So, we use hermite polynomials, which shows the orthogonality of the wavefunction. That means the two wavefunctions are independent of each other. Also, in our analytical equation we have used the exponential term,

So, from here, we obtained the linear fitting coefficients using constraint minimization using sequential least square programming (SLSQP) method. We select the best fitting coefficients obtained through constraint minimization and characterized by the chi-square test. We choose the chi square value less than 0.1 as

the best fit for all the systems. The set of optimized coefficients $a_i$ forms the input for the artificial neural network.

The fitting equations for all the systems are as follows:

$$n(x) = [a_0 * H_0(x) + a_1 * H_1(x) + a_2 H_2(x) + a_3 * H_3(x) + a_4 * H_4(x) + a_5 * H_5(x)] e^{-b\frac{x^2}{2}}$$

Where n(x) is total probability density of all electrons in the system $a_0$, $a_1$, $a_2$, $a_3$, $a_4$, $a_5$ are fitted coefficients. $H_0(x)$, H1(x), H2(x), $H_3(x)$, $H_4(x)$, $H_5(x)$ are Hermite polynomials up to order 5.

The values of fitting coefficients obtained for the 2000 datasets of numerical densities which we got from the randomly generating data set of potential (i.e.,$v = \frac{1}{2}kx^2$) are as following:

| Fitted coefficient | Value of fitted coefficient |
|---|---|
| $a_0$ | 0.9100 |
| $a_1$ | 0.0000 |
| $a_2$ | 0.5571 |
| $a_3$ | 0.0000 |
| $a_4$ | 0.1533 |
| $a_5$ | 0.0000 |

Table1: Values of fitting coefficients for harmonic potential

The corresponding plot for the fitting for the above mentioned numerical densities for the 2SL system is:

### 4.5.a) Plot for fitting of densities with original density for 2SL system:

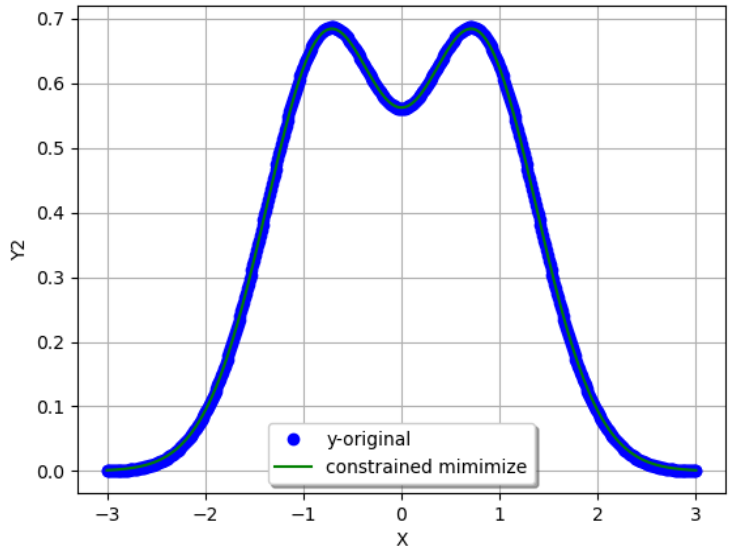



Figure (4.5.a) Plot for the fitting of density with original density for 2SL system

Now, similarly for the 3S and 3SL systems, the values of fitting coefficients obtained from the fitting of numerical densities for the above systems would be:

| Fitted coefficient | Value of fitted coefficient |
|---|---|
| $a_0$ | 1.5499 |
| $a_1$ | 0.0000 |
| $a_2$ | 0.6363 |
| $a_3$ | 0.0000 |
| $a_4$ | 0.1667 |
| $a_5$ | 0.0000 |

Table2: Fitting coefficients for numerical densities for 3S system

| Fitted coefficient | Value of fitted coefficient |
|---|---|
| $a_0$ | 1.2077 |
| $a_1$ | 0.0000 |
| $a_2$ | 0.6720 |
| $a_3$ | 0.0000 |
| $a_4$ | 0.3495 |
| $a_5$ | 0.0000 |

Table3: Fitting coefficients for numerical densities obtained for 3SL system

Following are the plots for the fitting of original probability density and fitted probability densities for all the three systems (i.e., 3S, 3SL)

**4.5.b.) Plot for fitting of densities with the original densities for 3S system:**

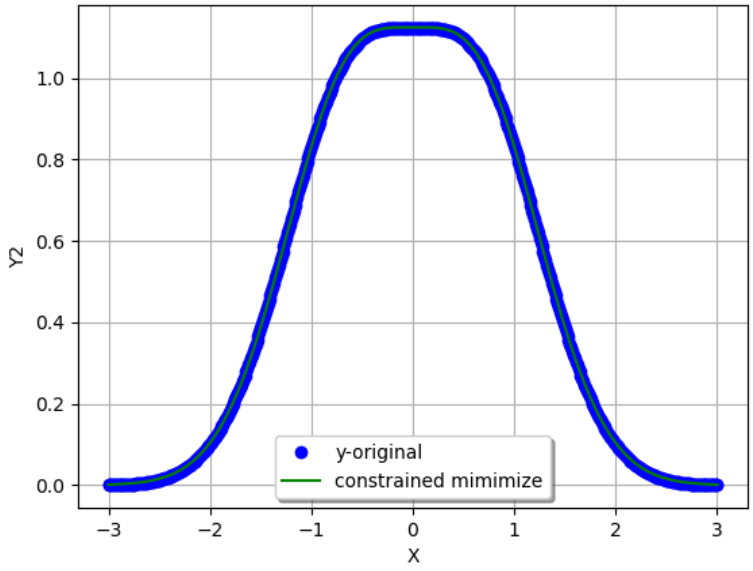



Figure (4.5.b) Plot for the fitting of density with original density for 3S system

### 4.5.c.) Plot for fitting of densities with original density for 3SL system:

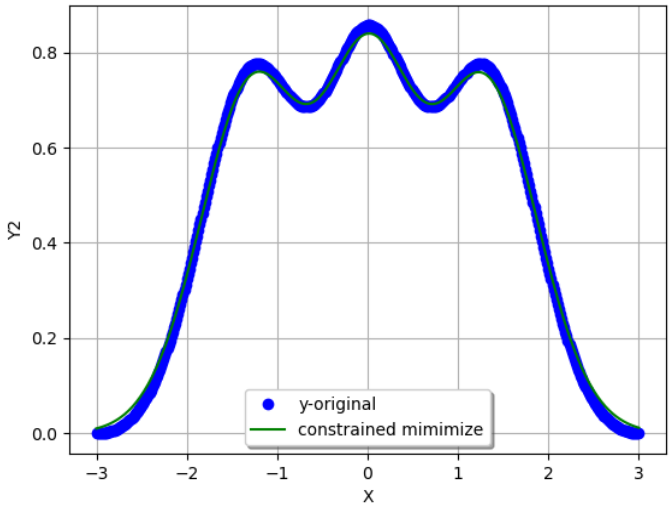



Figure (4.5.c) Plot for the fitting of density with original density for 3SL system

For the harmonic oscillator potential, which is being shifted from 0 to positive x coordinates, the potential i.e., $V = \frac{1}{2}k(x-a)^2$, the fitting has been done for all the three systems.

For such systems, the fitting equation has been taken as:

$$n(x) = [a_0 * H_0(x) + a_1 * H_1(x) + a_2 H_2(x) + a_3 * H_3(x) + a_4 * H_4(x) + a_5 * H_5(x)]e^{\frac{-b(x-a)^2}{2}}$$

Here, we have taken 'a' as the mean value.

Where n(x) is total probability density of all electrons in the system.

$a_0, a_1, a_2, a_3, a_4, a_5$ are fitted coefficients.

$H_0(x)$, H1(x), H2(x), $H_3(x)$, $H_4(x)$, $H_5(x)$ are the shifted Hermite polynomials up to order 5, that means we have taken the Hermite polynomial corresponding to the coordinates of the system.

For such system, after so many trials, we have chosen the value of b to be 3.0

Similarly, the value of linear fitting coefficients has been obtained for the dataset of numerical densities.

| Fitted coefficient | Value of fitted coefficient |
|---|---|
| $a_0$ | 0.9459 |
| $a_1$ | 0.0000 |
| $a_2$ | 0.4443 |
| $a_3$ | 0.0000 |
| $a_4$ | 0.1026 |
| $a_5$ | 0.0000 |

Table4: Linear fitting coefficients for the 2-electron system (2SL)

| Fitted coefficient | Value of fitted coefficient |
|---|---|
| $a_0$ | 1.4336 |
| $a_1$ | 0.0000 |
| $a_2$ | 0.5627 |
| $a_3$ | 0.0000 |
| $a_4$ | 0.1041 |
| $a_5$ | 0.0000 |

Table 5: Linear fitting coefficients for the three-electron system (3S)

| Fitted coefficient | Value of fitted coefficient |
|---|---|
| $a_0$ | 1.482 |
| $a_1$ | 0.0000 |
| $a_2$ | 0.6757 |
| $a_3$ | 0.0000 |
| $a_4$ | 0.3836 |
| $a_5$ | 0.0000 |

Table 6: Linear fitting coefficients for the three electron system(3SL)

Following are the plots for the fitting of densities obtained from the mathematical functions and the numerical densities for all the three systems (i.e. 2SL, 3S, 3SL).

### 4.5.d) Plot for the fitting of density with original density for 2SL system for shifted harmonic potential

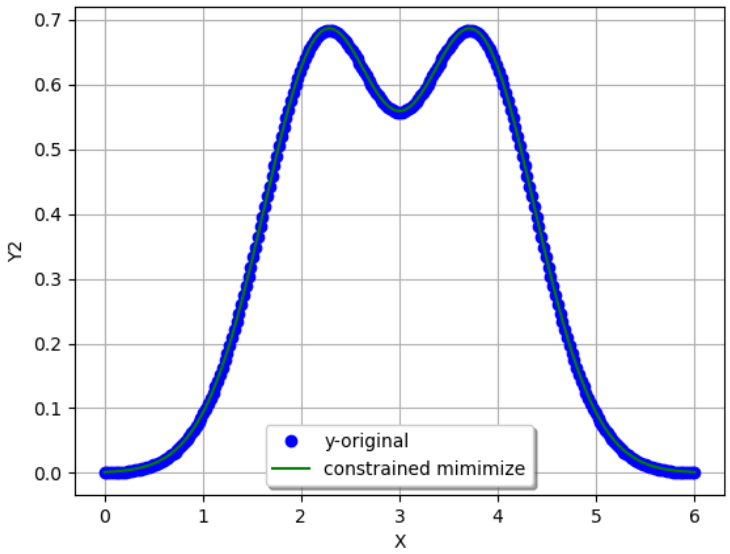



Figure (4.5.d) Plot for the fitting of density with original density for 2SL system

**4.5.e.) Plots for the fitting of density with original density for 3S sytem for shifted Harmonic potential:**

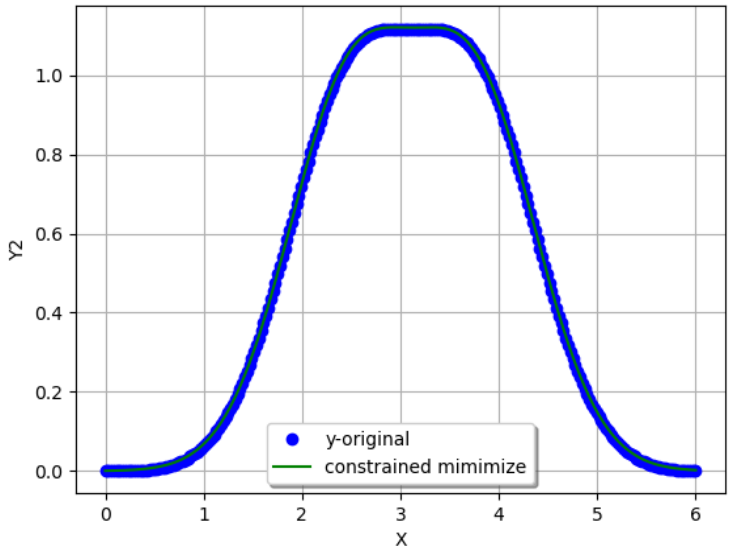



Figure (4.5.e) Plot for the fitting of density with original density for 3S system

**4.5.f.) Plots for the fitting of density with original density for 3SL system for shifted harmonic potential:**

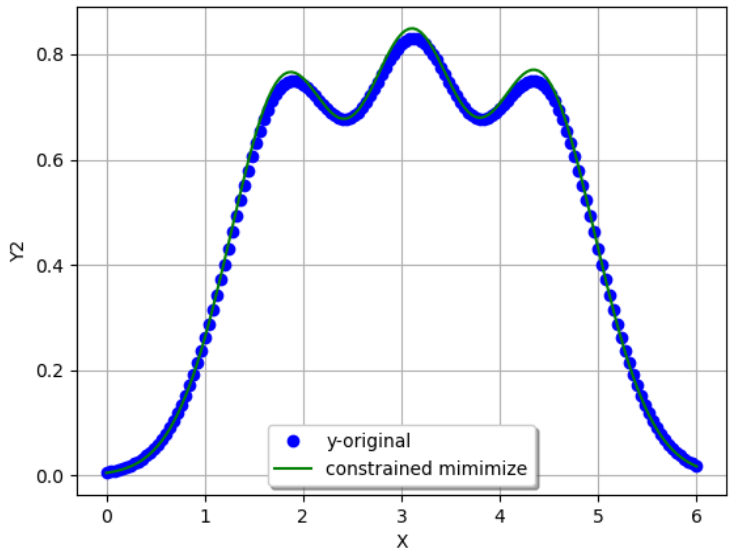



Figure (4.5.f) Plot for the fitting of density with original density for 3SL system

Now, in the similar fashion, we have also done fitting for the densities obtained from the mathematical functions with the original densities and obtained the linear fitting coefficients for all the three systems of two electrons and three electrons respectively.

Following are the plots for the fitting of density obtained from the mathematical functions and the original density for all the three systems (i.e. 2SL, 3S, 3SL).

For the fitting, after many trials, we have set the value of b to be 8.0 for all the systems in our mathematical expression.

## 4.6) Fitting of Kinetic Energy :

For the fitting of kinetic energy, linear fitting coefficients and the single kinetic energy associated with each data point is given as input. The following correlation plots have been obtained for the two electron system. We could plot only for two electron system. Plot (4.6.a) is for 2SL system for which the harmonic potential is being shifted from ) to positive x coordinates and then got the numerical densities as input to train ANN.

### 4.6.a.) Correlation plot for Exact KE versus ANN KE for 2SL:

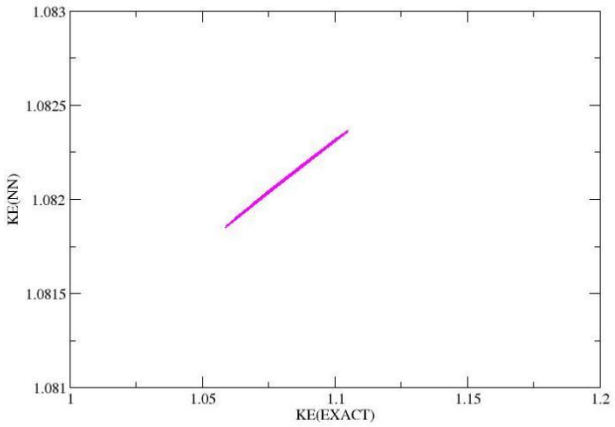



Figure (4.6.a) Correlation plot for Exact KE versus ANN KE for 2SL

### 4.6.b.) Correlation plot for Exact KE versus ANN KE for 2SL for shifted harmonic potential:

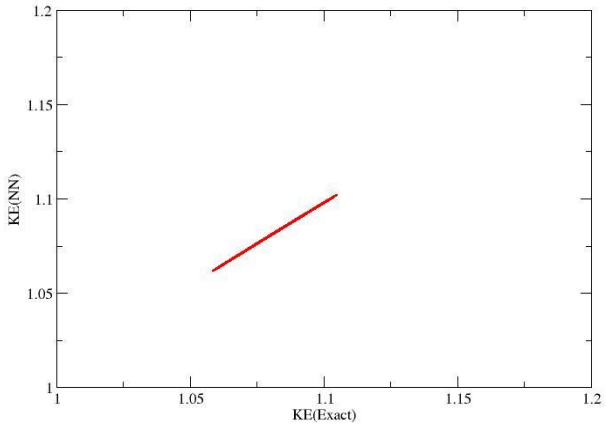



Figure (4.6.b) Correlation plot for Exact KE versus ANN KE for 2SL for shifted harmonic potential

| Error | 2SL |
|---|---|
| RMSE in T (a) | 0.000043 |
| RMSE in T (b) | 0.000107 |

Table 3: RMSE for Exact KE versus ANN KE

## Chapter 5

### Problems faced:

Understanding the Numerov C code initially. Then shifting the potential and wavefunction for the harmonic potential from 0 to positive x coordinates took much time. Once parity in the wavefunction was introduced, the problem got solved.

During the fitting of the original density with the mathematical function , we initially considered wavefunction for the system to be the product of normalization constant and the exponential function($e^{-b\frac{x^2}{2}}$), but, this wavefunction was not satisfying the condition for orthonormality So to introduce orthogonality to the wavefunction, we added the Hermite polynomial to the wavefunction as it shows the independency that is they are orthogonal to each other.

Also, for the fitting of densities with the reference numerical dataset of densities obtained from the 2000 set of potential which is being shifted from 0 to positive x coordinates, for such potential, we consider our mathematical wavefunction to be the same in terms of $e^{-b\frac{x^2}{2}}$initially, but it was not to be done this way. As the Gaussian distribution term is $e^{-b*(x-a)*(x-a)/c}$ so the mean value of the function has to be also mentioned and standard deviation has to be mentioned. But in our case, we have taken the value of standard deviation to be the same as the deviation for -x to +x coordinates because standard deviation is just the deviation from its mean position. Now, still the mathematical wave function, we considered was not coming to be orthogonal. Hence, we multiplied it with Hermite polynomial. Also, in the Hermite polynomial, we added the mean value term in such a way that the polynomial became consistent with the coordinates of system. Then the mathematical wavefunctions became orthogonal and normalized.

**Chapter 6**

# Conclusion

In this project, we solved Schrödinger Equation for one-dimensional system in a Harmonic potential well by using the artificial neural network. So, from the future perspective, such models can be used to calculate energies of 1-dimensional Schrödinger equation in a similar way but for unknown potential energy well without really solving the Schrödinger equation analytically. We considered two and three non-interacting fermion systems and filled the available energy states giving three types of unique total densities for the system. Each density is expanded in terms of Hermite basis function. Our aim was to map the probability densities to energies using artificial neural networks. For this we made a dataset of about 2000 probability densities using the Numerov method and trained these probability densities to the known energies obtained. We have fitted kinetic energy density functional using the feed forward artificial neural network for the 2SL system.

## References


1. Born, M.; Oppenheimer, R.; *Ann. Phys*., 1927, **389**, 457.
2. Car, R.; Parrinello, M.; *Phys. Rev. Lett.,* 1985, **55**, 2471.
3. Behler, J.; *Int. J. Quantum Chem.,* 2015, **115**, 1032.
4. I.E. Lagaris; A. Likas; D.I. Fotiadis; *Comput. Phys. Commun.,* 1997,**104**, 359.
5. Park, J.; Sandberg, I. W.; *Neural Computation*, 1991, **3**, 246.
6. Synder, J. C.; Rupp, M.; Hansen, K.; Muller, K. R.; Burke, K.; *Phys. Rev. Lett.*, 2012, **108**, 253002.
7. Behler, J.; *J. Phys*., 2014, **26**, 183001.
8. McCulloch, W. S.; Pitts, W.; *Bull. Math. Biophys*, 1943, **5**,115.
9. T. B. Blank et al.; *J. Chem. Phys*., 1995, **103**, 4129.
10. Schwenker, F.; Kestler, H. A.; Palm, G.; *Neural Networks*, 2001, **14**, 439.
11. Scarselli, F.; Tsoi, A. C.; *Neural Networks*, 1998, **11**, 15.
12. Lagaris, I.; Likas, A.; Fotiadis, D.; *Comput. Phys. Commun.,* 1997, 1**04**, 1.
13. Da Silva, A. J.; Ludermir, T. B.; De Oliveira, W.R.; *Neural Networks*, 2016, **76**, 55.
14. Peter, Y.; *J. Phys*., 2009, **115**, 242.
15. McQuarrie, D. A.; Simon, J. D.; *J. Chem. Educ*., 1998, **75**, 545.