UNIVERSITY OF SOUTH AFRICA

MASTERS THESIS

Solving Differential Equations in Quantum Mechanics using Sinc Functions in one and two dimensions, employing Python and Numpy.

Author:
Obiageli Lovenda EZENWACHUKWU

Supervisor:
Professor Moritz BRAUN

A thesis submitted in fulfillment of the requirements for the degree of Masters of Science in Physics

in the

Department of Physics

at the

University of South Africa

October 7, 2021

Declaration of Authorship

I declare that the thesis entitled “Solving Differential Equations in Quantum Mechanics using Sinc Functions in one and two dimensions, employing Python and Numpy ” is my own work, and that all sources used have been acknowledged by means of complete references. I understand and adhere to the Ethical Code as presented by College of Science, Engineering and Technology, University of South Africa.

Signed: Obiageli Ezenwachukwu

October 7, 2021

Acknowledgements

I would like to express my deep and sincere appreciation, gratitude and thanks to God Almighty, for the gift of life He has given me through His only begotten Son, Jesus Christ. He gave me the ability, strength and courage to start and complete this thesis. His Grace was so awesome during the course of this work.
I sincerely appreciate my supervisor, Professor Moritz Braun, for guidance, patience and encouragement throughout this work. My thanks and appreciate goes to the University of South Africa, for the financial support that helped me in the completion of this program.
I would like to express my appreciation to my family; my husband and daughter for their support, encouragement and understanding while going through this program. It was worth the efforts!

Abstract

In this contribution the sinc basis functions are used to numerically solve the Schrödinger equation in one and two dimensions for a number of potentials. The calculations are done using the Python and Numpy modules. Convergence is found to be fast for the harmonic oscillator. For the Morse potential it agrees with the theoretically expected behaviour. In the two dimensional case code optimization leads to a large speed-up. We also present the results of calculations for the ground state energy of the hydrogen molecular ion employing Sinc functions as a basis set. Modifications are required to make the basis functions suitable for calculating the ground state energy of the hydrogen molecular ion with the application of the cusp factor formalism . Finally the resulting energies are investigated as a function of the number of basis functions and double-logarithmic fits are performed.

Key Terms:
Sinc functions, Numerical Methods, Python, Numpy, Scipy, Morse potential, Eigenvalue Problem, Computational Physics, Cusp Factor, Hydrogen Molecular Ion, Least Square Fits.

This work is dedicated to God Almighty, the Father of our Lord Jesus Christ, for His love for me, salvation of my soul and filling me with His Holy Spirit. To Him alone be all the glory. . . .

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Our world consists of many particles interacting with one another while some do not. To accurately and adequately describe these interacting systems requires the inclusion of a potential [1]. These interacting systems can be considered as governed by the laws of quantum mechanics [2]. The interacting systems could be atomic systems, molecular systems, or nuclear systems [2]. Inside these quantum mechanical systems, particles are moving and interacting with each other. Physicists and engineers describe the properties and behaviour of the interacting quantum systems by the wave function, which defines the state of the system at each position and time.

The Schrödinger equation is one of the fundamental equations of physics for describing quantum mechanical behaviour. In quantum mechanics, the Schrödinger equation is an essential and powerful tool for investigating and understanding quantum processes. The quantum processes within the systems are described by the Schrödinger equation which determines the wave function $\Psi(x,t)$ [3].

The numerical methods for solving the full range of partial and ordinary differential equations are of interest to scientists and engineers. This is because in science and engineering, differential equations occur very often. To solve these differential equations, both linear and non-linear, many different methods have to be employed. Numerical methods have proven to be of great importance to get accurate solutions [4, 5].

Sinc functions as a basis set have been used extensively for obtaining the approximate solutions of ordinary differential equations, partial differential equations and integral equations [6, 7]. The sinc numerical method is easily implemented and the results obtained are quite accurate [8, 9]. The approximation by sinc functions takes care of singularities in problems [10]. The sinc methods are less prone to common instability problems compared to other numerical methods due to their rapid convergence [10]. It has been shown that sinc numerical methods are distinguished by exponentially decaying errors [11, 12]; they have convergence rates of $O(\exp(-k\sqrt{N}))$ with some $k > 0$, [13] where N is the number of nodes or basis functions used in the methods.
Many engineers, mathematicians and physicists, handling an infinitely long discrete signal, or differential equations have employed the sinc method in order to analyse and solve these equations [4, 8].

Different numerical techniques have been employed to solve the differential equations governing few-body quantum mechanical systems such as the Euler method, (both implicit and explicit) [14, 15, 16], the Numerov algorithm [17], the Chebyshev collocation method [18], and the Runge-Kutta methods [16]. The Euler method is only of first order, and thus while consistent, it has a slow convergence rate [15, 16]. The Runge-Kutta methods are of limited use on an irregular grid [16]. Also most numerical approximation procedures,

such as interpolation, quadrature, finite difference approximation, finite element methods and variational methods are based on exact relationships that polynomials satisfy [13]. These procedures generally do very well in a region where the function to be approximated is analytic, and perform poorly in a region of a singularity of the function [13]. Some of these above-mentioned methods employed to solve the differential equations of quantum mechanical systems have instability problems and are usually truncated before they are treated numerically [19, 10], leading to loss of accuracy.
Even though the Chebyshev method provides a robust, efficient and accurate tool for the calculation of three dimensional nuclear wave functions [17], it is polynomially based because it is defined over bounded domains [11].

The above has led to more investigations of other numerical methods for the solutions of quantum mechanical problems. The robustness, efficiency and accuracy of the solutions will definitely depend on the choice of the basis function [19]. Hence this work on solving the differential equations in quantum mechanics using the sinc basis functions is of relevance.
In this thesis, we will apply the sinc basis functions method to a variety of quantum mechanical problems in one and two dimensions. The results obtained as well as the efficiency are compared to a known numerical method.

Chapter 2

Theory

Differential equations appear in many areas of science and technology, precisely whenever a deterministic relation involving some continuously varying quantities [20], and their rates of change in the space and/or time (expressed as derivatives) is known or postulated. They are widely used in pure and applied mathematics, physics, and engineering. All of these disciplines are interested in the properties of differential equations of various types. Pure mathematics focuses on the existence and uniqueness of solutions [21], while applied mathematics emphasizes the rigorous justification of the methods for approximating solutions.
Differential equations play an important role in investigations of virtually every physical, technical, or biological process, from celestial motion to bridge design, and to interactions between neurons [22]. Differential equations such as those used to solve real-life problems may not necessarily be directly solvable [23] since they do not have closed form solutions. Instead, such solutions can be approximated using numerical methods. The sinc method is an attractive alternative for numerical solutions to the problems with no closed form [19]. Differential equations of quantum mechanical systems modelled by the Schrödinger equation can be solved exactly for some types of potentials giving a detailed account of interactions in the quantum systems[2]. Atomic, molecular and nuclear systems are examples of quantum mechanical systems [2], which are described by the Schroedinger equation.

Theoretical studies of quantum mechanical systems are based on the solution of the few-body Schrödinger equation for the systems in question [2]. Investigations of these quantum systems are based on those solutions which depend on [2] the potentials associated with the interactions in the quantum systems. Subsequently, only numerical solutions are the appropriate techniques for the differential equations of the systems. The solutions to these quantum mechanics problems use approximation methods [24]. Just as stated by Zettili [24], most problems encountered in quantum mechanics cannot be solved exactly hence one resorts to approximation methods for their solutions.

The sinc method has been studied extensively [8] and is said to be a highly efficient numerical technique [9], especially with problems having singular solutions and independent of boundary conditions. The sinc method, based on the sinc functions, has been found to be among the few reliable methods for obtaining numerical solutions [8], combining convolution with the boundary integral equation (IE) approach, and it brings about exponentially fast convergence for solutions of differential equations [25]. In other words, the sinc method approximation yields both an effective and rapidly convergent scheme for solving those problems, and so circumvents the instability problems that one typically encounters in some other methods [19]. Many researchers have come to the realization that the sinc function plays an important role in many areas of science and its application.

2.1 Background on Sinc Function

The sinc function is a sine wave multiplied by 1/x. The sinc method is connected to a family of approximation formulas [26]. It is widely used in various fields of numerical analysis [10] such as interpolation, quadrature, approximation of transforms, the solution of integral equations, ordinary differential equations(ODE), and partial differential equations(PDE) [10, 6]. The sinc function method has been used for solving a wide range of linear and non-linear problems which emanate from scientific and engineering applications [10], like oceanographic problems with boundary layers [27], astrophysical equations, heat distribution, and two-point boundary value problems. Volterra's population model, Hallen's integral equation, third-order boundary value problems, and systems of second-order boundary value problems are other applications of the sinc method [10]. The sinc method has also been applied to elasto-plastic problems [28], fourth-order boundary value problems, the inverse problem, integro-differential equations [6, 29], and optimal control [10].

The sinc function dates back to the works of Whittaker [30], Lund [7], and Stenger [26]. Many more related works have been carried out by other researchers such as Chen Li [4], Mehdi [8], Sugihara [12] and Abbas [10]. Whittaker happened to be the first to make a connection with analytic functions, where the sinc expansion was being used, together with the cardinal function $C(f, h)$, while Stenger and his school developed different sinc numerical methods for solving differential equations [12]. The term sinc was introduced by Phillip M. Woodward in his 1952 paper "Information theory and inverse probability in telecommunication" [31], and also in his 1953 book "Probability and Information Theory with Applications to Radar" [32], in which he said the function "occurs so often in Fourier analysis and its applications that it definitely seems to merit some notation of its own".

Recent developments have established that the sinc numerical methods can achieve convergence rates of $O(\frac{exp(-k^*N)}{\log N})$ [12] with some $k^* > 0$ for a smaller but still practically meaningful class of problems, and that these convergence rates are best possible. Here N is the number of nodes or basis functions used in the methods.
In the solution of inverse problems, the sinc methods have been employed as forward solvers [9]. At the same time, solving the problems with non-homogeneous mixed boundary conditions directly seems to be difficult using the sinc method [4].
Derivations of sinc approximations are usually done using complex variables, while some are done using Fourier Transforms. Sinc approximations have been derived, for a finite interval (a,b), and the semi-finite interval $(0, \infty)$, the whole real line $(-\infty, \infty)$ and generally over arcs in the complex plane [25].

The sinc function is used with the variational method [33] which is often discretized in a grid in order to make it applicable. Using these grids or stepsize methods for the solution of differential equation are currently regarded as one of the most powerful tools of the numerical methods for solving Schroedinger equation both for the time independent and time dependent systems [34]. It provides the necessary accuracy and computational efficiency when compared with the traditional variational techniques [34].
Presently, the sinc functions are widely used for getting the approximate solution of ODEs, PDEs and integral equations [35, 7]. It is frequently used in Fourier analysis, which is a good technique for solving differential equations. It has a smoothing effect on Fourier analysis in order to improve convergence [33], and also handles the Gibbs' phenomenon, which is the tendency of a truncated function or series to display oscillations near points of discontinuity. Therefore this smoothing effect of the sinc functions helps the truncated series to improve convergence [34, 33].

2.2 Properties of Sinc Function

We will concentrate on the properties of the sinc function that are useful to our study of solving the Schrödinger equation. More detailed properties of the sinc function can be found in [11, 7, 25].
The sinc function is defined as

$$\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x}, \tag{2.1}$$

and satisfies the following properties:

$$\int_{-\infty}^{\infty} \operatorname{sinc}(x)\, dx = 1, \tag{2.2}$$

$$\int_{-\infty}^{\infty} \operatorname{sinc}^2(x)\, dx = 1. \tag{2.3}$$

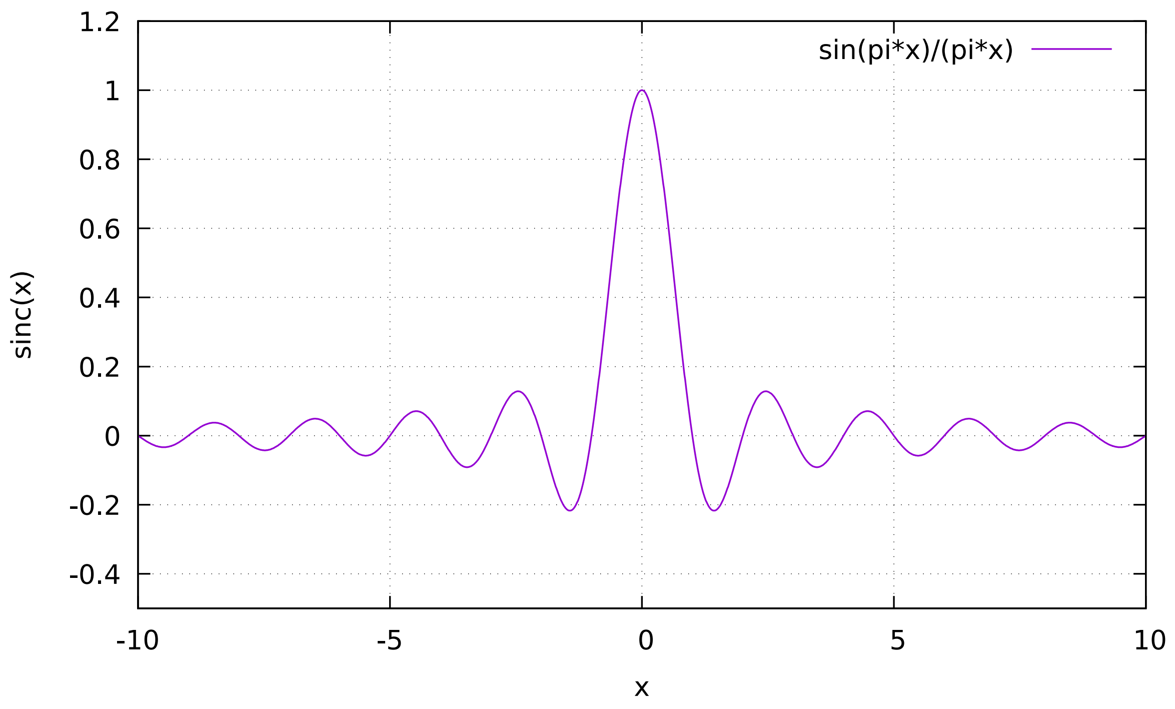

FIGURE 2.1: Plot of $\frac{\sin(\pi x)}{\pi x}$.

In figure 2.1 the sinc function is shown.

Another useful relation is, that the sinc function is the Fourier transform of the rectangular function or pulse with no scaling. It is used when reconstructing a continuous band-limited signal from uniformly spaced samples of that signal [35]. Thus we have

$$\mathrm{rect(t)} = \begin{cases} 1, & \text{-}1/2 \leq t \leq 1/2; \\ 0, & |t| > \frac{1}{2}. \end{cases} \tag{2.4}$$

$$\int_{-\infty}^{\infty} \mathrm{rect(t)} e^{-i\pi f t} f t = \frac{\sin(\pi f)}{\pi f} = \mathrm{sinc(f)}. \tag{2.5}$$

The sinc function is analytic everywhere, even in the complex plane and hence an entire function.
For $h > 0$, and $k = 0, \pm 1, \pm 2, ...,$ the translated sinc functions with evenly spaced nodes, are given by [13, 12]

$$S(k,h)(x) = \mathrm{sinc}\left(\frac{(x-kh)}{h}\right) = \begin{cases} \frac{\sin \frac{\pi}{h}(x-kh)}{\frac{\pi}{h}(x-kh)}, & x \neq kh; \\ 1, & x = kh. \end{cases} \tag{2.6}$$

The translated sinc functions form an interpolatory set of functions, i.e,

$$S(k,h)(jh) = \delta_{kj} = \begin{cases} 1, k = j, \\ 0, k \neq j. \end{cases} \tag{2.7}$$

Here k and j belong to $\mathbb{Z}$.
A function $f(x)$ which is analytic on a rectangular strip centered on the real axis can be approximated in terms of sinc functions as [26, 36]

$$f(x) = \sum_{-\infty}^{\infty} f(kh) S_k(h,x). \tag{2.8}$$

If a function $f(x)$ is defined on the real axis, then for $h > 0$ the series

$$C(f,h)(x) = \sum_{k=-\infty}^{\infty} f(kh)\mathrm{sinc}\left(\frac{(x-kh)}{h}\right), \tag{2.9}$$

which can also be written as

$$C(f,h)(x) = \sum_{k=-\infty}^{\infty} f(kh) S(k,h)(x), \tag{2.10}$$

is called the Whittaker Cardinal expression [30] of f whenever this series converges for $h > 0$, where h is the stepsize.

Chapter 3

Numerical Methods

3.1 The Variational Method

The variational method begins by evaluating the expectation value of the Hamiltonian and norm of the wave function.
Time -independent Schrödinger equation is

$$H\psi = E\psi, \tag{3.1}$$

where H is the Hamiltonian, and E is the total energy of the system, which is dependent on the wave function of the system. The variational principle states [37] that the ground-state energy, E_0, is always less than or equal to the expectation value of H calculated with the trial wave function. We can express the wave function, even though unknown, as an infinite linear combination of basis functions [37];

$$\psi = \sum_{i=-\infty}^{\infty} c_i^n S(k,h), \tag{3.2}$$

where S is the basis functions and c_i is the weighted coefficients.

3.2 The Basis Functions

For the remainder of this document the one dimensional basis functions are

$$s_i(x) = \frac{1}{\sqrt{h}} \frac{\sin(\pi(\frac{x}{h} - i))}{\pi(\frac{x}{h} - i)}, \tag{3.3}$$

where h is the distance between zeroes of the transformed sinc functions, and i is the index running in principle over all integer values from $-\infty$ to ∞. The first factor on the right hand side provides for normalization. However, for our calculations we have to restrict this index to a range from $-n$ to n. We then define

$$f_j(x) = s_{-n+j-1}(x), \tag{3.4}$$

with

$$j = 1, 2, ...2n + 1, \tag{3.5}$$

as a basis with $N = 2n + 1$ members. The choice of n which is equivalent to a cut-off at $x_{max} = + nh$ and $x_{min} = -nh$, influences the accuracy of the result. It should be noted, that the above basis set is approximate because of the finite step width h and the cut-off at $\pm x_{max}$.
We express the wave function as a finite linear combination of basis functions.
Thus the numerical ansatz for the wave function of the ν-th state becomes

$$\psi_\nu(x) = \sum_{i=1}^{N=2n+1} c_i^\nu f_i(x), \tag{3.6}$$

where c_i^ν are the coefficients and f_j are the sinc basis functions.

3.2.1 Expectation values and Kinetic matrix elements k_{ij}

The I-D Schrödinger equation for this is given by

$$ \tag{3.7} $$

$$\left\{-\frac{d^2}{dx^2}+V(x)\right\}\psi_\nu(x)=e_\nu\psi_\nu(x). \tag{3.8}$$

This is solved via the variational method [37], leading to the eigenvalue problem

$$H\mathbf{u}_\nu=e_\nu\mathbf{u}_\nu, \tag{3.9}$$

where u_v is the vector of expansion coefficients,

with

$$h_{ij}=k_{ij}+v_{ij}\,, \tag{3.10}$$

$$h_{ij}=\int[f_i'(x)f_j'(x)+f_i(x)V(x)f_j(x)]dx=k_{ij}+v_{ij} \tag{3.11}$$

where h_{ij} is the matrix elements of H, k_{ij} are the elements of kinetic energy matrix while v_{ij} are the elements of potential matrix.

Analytical evaluation of the first term k_{ij} in h_{ij} can be carried out using the Fourier expansion of the sinc function and the Parseval theorem with the following result

$$k_{ij}=\begin{cases}\frac{\pi^2}{3h^2} & i=j;\\ (-1)^{|i-j|}\frac{1}{h^2}\frac{2}{|i-j|^2} & i\neq j.\end{cases} \tag{3.12}$$

The second term v_{ij} is evaluated numerically via repeated Gauss-Legendre integration [38, 39, 40] as described in the following below.

3.2.2 Evaluation of Potential matrix elements v_{ij}

It is convenient to use Gauss-Legendre integration for each interval of width h since the basis functions vanish at known points. Thus we obtain

$$v_{ij}=\sum_{k=-n}^{n}\sum_{l=1}^{N_{GL}}hw_lf_i(kh+hx_l)V(kh+hx_l)f_j(kh+hx_l), \tag{3.13}$$

where x_l and w_l are the Gauss-Legendre points and weights on the unit interval.

Rewriting Eq (3.13), we can suitably define global points y_m and weights u_m such that

$$v_{ij} = \sum_{m=1}^{N \times N_{GL}} u_m f_i(y_m) V(y_m) f_j(y_m), \tag{3.14}$$

where

$$y_m = kh + hx_l, \tag{3.15}$$

and

$$u_m = hw_l, \tag{3.16}$$

with

$$k = -n + \left[\frac{m-1}{\mathrm{N_{GL}}} \right], \tag{3.17}$$

and here the brackets [] refer to the integer part of what is between them.

Eq (3.14) is much more suitable for numerical purposes since it can be evaluated efficiently as a scalar product of vectors.
For this integration we used N_{GL} = 10 points and weights in each interval, resulting in a total number of Integration points of 20n.

3.3 Programming languages and libraries used

To perform the above calculations we use Python [41], Numpy [42], Scipy [43] and Matplotlib [44]. All the source code used is given in Chapter 6.

Chapter 4

Applications and Discussion

4.1 Numerical Results for One Dimension

4.1.1 The harmonic oscillator

For testing purposes we first considered the harmonic oscillator. The quantum harmonic oscillator is of unique importance in quantum mechanics, since it is one of the few problems that can be solved both in closed form, and in approximations.
The Hamiltonian for the harmonic oscillator is given by:

$$H = -\frac{d^2}{dx^2} + x^2. \tag{4.1}$$

The energy eigenvalues E_ν of the quantum harmonic oscillator are

$$E_\nu = 2\nu + 1, \quad \nu \geq 0. \tag{4.2}$$

This means that energy eigenvalues are all odd numbers 1, 3, 5, 7. Please note, that these eigenvalues obtained are twice the standard values for the harmonic oscillator due to the definition of H, i.e. no factor of $1/2$.

4.1.2 Numerical calculations for harmonic oscillator

Employing the Python [41, 43] code with the numerical extension Numpy [42] listed in section 6.1, and using the sinc basis functions for $x_{\max} = 8$, the numerical results shown in the table below were obtained. Here, E_0, E_1, E_2 and E_3 are the energies of the ground state, first excited state, second state, and third excited states respectively. The known lowest eigenvalues $1, 3, 5, 7$ are obtained very accurately. Convergence is very fast as function of n.

TABLE 4.1: Results of Numerical Calculations for 1-D Harmonic Oscillator; n, h and $x$max;

n	h	E_0	E_1	E_2	E_3
8	1.0000	1.00013274618	3.00388708934	5.01965658349	7.1426916954
9	0.8889	1.00001043728	3.00040366538	5.00262094231	7.026576271
10	0.8000	1.00000059941	3.00002946274	5.00023921435	7.00326643666
11	0.7273	1.00000002524	3.00000153079	5.00001518651	7.00026666304
12	0.6667	1.00000000078	3.00000005692	5.00000067774	7.00001475924
13	0.6154	1.00000000002	3.00000000152	5.00000002142	7.00000056346
14	0.5714	1.00000000000	3.00000000003	5.00000000048	7.000000015
15	0.5333	1.00000000000	3.00000000000	5.00000000001	7.00000000028
16	0.5000	1.00000000000	3.00000000000	5.00000000000	7.00000000000

4.1.3 Morse Potential

The Morse potential is used to model the vibrational excitations of a chemical bond for diatomic molecules.
The potential energy of this diatomic molecule is described by the Morse potential as [45]

$$V(x) = D(e^{-2x/a} - 2e^{-x/a}). \tag{4.3}$$

The Hamiltonian is

$$H = \frac{p^2}{2\mu} + D\left(e^{-2x/a} - 2e^{-x/a}\right), \tag{4.4}$$

where x = length of chemical bond, reduced mass μ=1/2, and D = measure of strength of chemical bond. For $\hbar = 1$ and $a = 1$, the eigenvalues E_ν of H are given by [45]

$$E_\nu = -D\left\{1 - \frac{1}{\sqrt{D}}\left(\nu + \frac{1}{2}\right)\right\}^2, \quad \nu \geq 0\,. \tag{4.5}$$

for ν such that the curly bracket is positive.

4.1.4 Numerical calculations for Morse Potential

Employing the Python code [41, 42, 43] listed in section 6.2, for the sinc basis functions for $x_{\max} = nh = 15$, and $D = 9$ using Numpy, we calculated the three lowest eigenvalues.

E_0 is the ground state energy level, E_1 is first excited state, and E_2 is the second excited state. The results obtained are shown in the table below.

TABLE 4.2: Results of Numerical Calculations for 1-D Morse Potential; n, h and x_{max}, E_0, E_1 and E_2 with $D = 9$;

n	h	E_0	E_1	E_2
10	1.5000	−3.76038976403	−0.250055663018	0.0733336499528
15	1.0000	−5.61144760855	−1.35904155742	0.00106155076298
16	0.9375	−5.78207245684	−1.54867449554	−0.0242262756023
17	0.88235	−5.9116427122	−1.70946234508	−0.0551345268337
18	0.83333	−6.00860002465	−1.84134344147	−0.0879500590189
19	0.78947	−6.08010573628	−1.94651057482	−0.119474470539
20	0.7500	−6.13196406853	−2.02827460821	−0.148096434547
22	0.68182	−6.19502991317	−2.13662852199	−0.192096781641
25	0.6000	−6.23360618925	−2.21183371107	−0.228296935064
27	0.55555	−6.24283368003	−2.23225185845	−0.239467707444
30	0.5000	−6.24797717715	−2.24473113548	−0.246788915935
50	0.3000	−6.25000348798	−2.25000886772	−0.249990616734

For the ν-th eigenpair resulting from calculation we expect from the theoretical convergence studies [13] the following to hold for the ν_{th} eigenvalue resulting from the calculations as function of n.

$$E_\nu(N) = E_\nu + b_\nu \exp(-c_\nu\sqrt{N}), \tag{4.6}$$

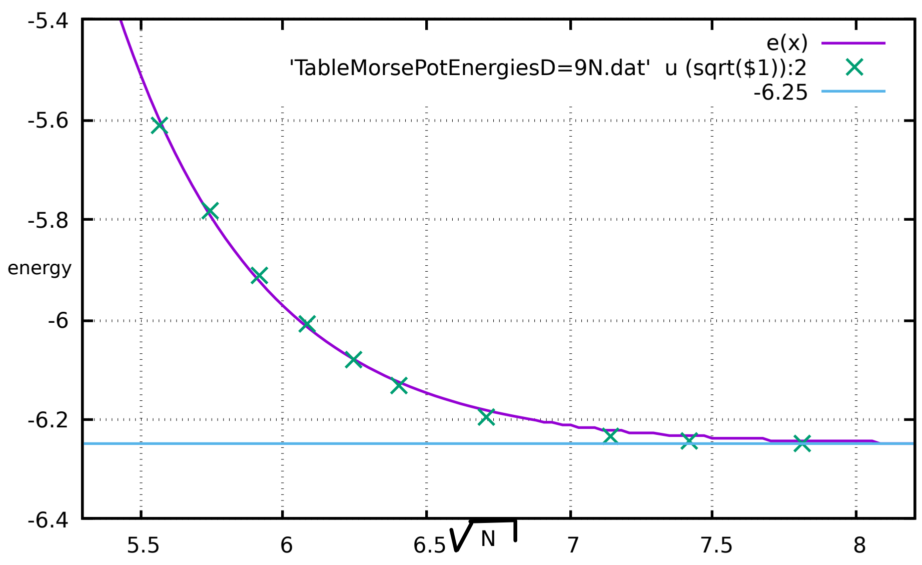

FIGURE 4.1: Convergence for ground state of Morse potential

where N indicates the number of basis functions.

For a constant $x_{\max} = 15$, a Gnuplot was used both for creating the figures as well as fitting the results to the expected convergence behaviour. The fits are done in terms of $\sqrt{N}$. On the following two pages, plots for ground state and first excited state are shown. From figure 4.1 and figure 4.2 it is evident, that the eigenvalues agree reasonably well with the fits to equation (4.6).

4.1.5 Discussion on one-dimensional results

The calculation for harmonic oscillator was very fast. Convergence for the ground state energy is obtained when $n = 14$, $h = 0.5714$ and $E_0 = 1.00000000000$. The first excited state energy level converges at $n = 15$, $h = 0.5333$ and $E_1 = 3.00000000000$. Similarly we have convergence for the second and third excited states when $n = 16$, $h = 0.5000$, i.e. $E_2 = 5.00000000000$ and $E_3 = 7.00000000000$.

For the Morse potential, the error of eigenvalues showed good agreement with the theoretically expected behaviour. The ground state energy converged to $E_0 = -6.25$, while the first excited state converged to $E_1 = -2.25$. These two values are in agreement with the theoretical prediction.

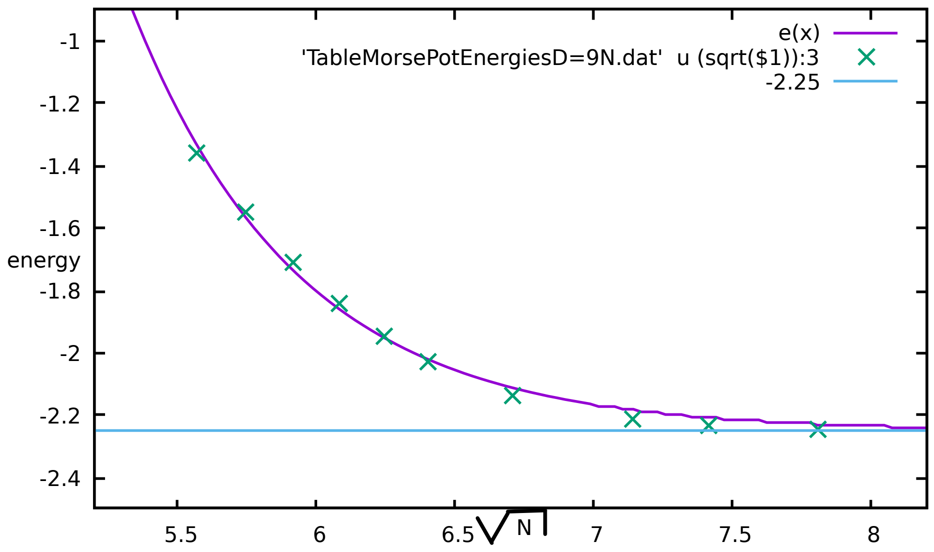

FIGURE 4.2: Convergence for first excited state of Morse potential

4.2 Two-Dimensional Schrödinger equation

The 2-D Schrödinger equation is given by

$$H\psi_i(x,y) = E_i\psi_i(x,y). \tag{4.7}$$

To simplify the calculation we assume that the distance between nodes of our basis functions is the same in both dimensions, i.e.

$$h = h_x = h_y. \tag{4.8}$$

Therefore, the basis functions used in x and y are identical, the domain of expansion is $[-x_{\max}, x_{\max}] \times [-x_{\max}, x_{\max}]$, and the total number of basis functions becomes N^2.
In two dimensions the Hamiltonian is given by

$$H = T + V = -\frac{\partial^2}{\partial x^2} - \frac{\partial^2}{\partial y^2} + V(x,y). \tag{4.9}$$

Evaluation of the matrix elements of the Hamiltonian, then proceeds analytically for the kinetic energy term and numerically for the potential energy term in a similar fashion as in one dimension. It should be noted, that the numerical integration requires much more CPU time, since there are now $4n^2N_{\text{GL}}^2$ integration points. Applying the variational principle, the two dimensional Schrödinger equation was solved numerically by expanding the wave function $\psi(x,y)$ in terms of products of one dimensional basis functions, i.e.

$$\psi(x,y) = \sum c_\alpha g_\alpha(x,y)\,. \tag{4.10}$$

Here the two dimensional basis functions are

$$g_\alpha(x,y) = f_{i(\alpha)}(x) f_{j(\alpha)}(y) \tag{4.11}$$

where i_α and j_α are the suitably defined functions of α.

The wave function in terms of the two dimensional basis functions is

$$\psi(x,y) = \sum c_\alpha f_{i(\alpha)}(x) f_{j(\alpha)}(y) \tag{4.12}$$

and the matrix elements of the Hamiltonian, H, are given by

$$h_{\alpha\beta} = \left\langle g_\alpha(x,y) \middle| -\nabla^2 + V(x,y) \middle| g_\beta(x,y) \right\rangle. \tag{4.13}$$

The eigenvalue problem will then be

$$H\mathbf{u}_\nu = \lambda_\nu \mathbf{u}_\nu, \tag{4.14}$$

with

$$h_{\alpha\beta} = \int\int g_\alpha(x,y) \left(-\nabla^2 + V(x,y)\right) g_\beta(x,y) dxdy, \tag{4.15}$$

$$h_{\alpha\beta} = \int\int g_\alpha(x,y) \left(-\frac{\partial^2}{\partial x^2} - \frac{\partial^2}{\partial y^2} + V(x,y)\right) g_\beta(x,y) dxdy, \tag{4.16}$$

$$h_{\alpha\beta} = t_{\alpha\beta} + v_{\alpha\beta}. \tag{4.17}$$

$$t_{\alpha\beta} = \int\int g_\alpha(x,y) \left(-\frac{\partial^2}{\partial x^2} - \frac{\partial^2}{\partial y^2}\right) g_\beta(x,y) dxdy, \tag{4.18}$$

$$v_{\alpha\beta} = \int\int g_\alpha(x,y) V(x,y) g_\beta(x,y) dxdy, \tag{4.19}$$

$$t_{\alpha\beta} = t^{(x)}_{\alpha\beta} + t^{(y)}_{\alpha\beta}. \tag{4.20}$$

Substituting Eq (4.11) into Eq (4.18) gives;

$$t^{(x)}_{\alpha\beta} = \int\int f_{i(\alpha)}(x) \left(-\frac{\partial^2}{\partial x^2}\right) f_{i(\beta)}(x) f_{j(\alpha)}(y) f_{j(\beta)}(y) dxdy, \tag{4.21}$$

$$t^{(x)}_{\alpha\beta} = \int dx f_{i(\alpha)}(x) \left(-\frac{\partial^2}{\partial x^2}\right) f_{i(\beta)}(x) \int dy f_{j(\alpha)}(y) f_{j(\beta)}(y), \tag{4.22}$$

$$t^{(x)}_{\alpha\beta} = k_{i(\alpha)i(\beta)} \int f_{j(\alpha)}(y) f_{j(\beta)}(y) dy, \tag{4.23}$$

$$t^{(x)}_{\alpha\beta} = k_{i(\alpha)i(\beta)} \delta_{j(\alpha)j(\beta)}. \tag{4.24}$$

Similarly the $t^{(y)}_{\alpha\beta}$ is obtained as:

$$t^{(y)}_{\alpha\beta} = \int dx f_{i(\alpha)}(x) f_{i(\beta)}(x) \int dy f_{j(\alpha)}(y) \left(-\frac{\partial^2}{\partial y^2}\right) f_{j(\beta)}(y), \tag{4.25}$$

$$t^{(y)}_{\alpha\beta} = \delta_{i(\alpha)i(\beta)} k_{j(\alpha)j(\beta)}. \tag{4.26}$$

The detailed matrices are given by:

$$t^{(x)}_{\alpha\beta} = k_{i(\alpha)i(\beta)} \delta_{j(\alpha)j(\beta)}, \tag{4.27}$$

$$t^{(y)}_{\alpha\beta} = \delta_{i(\alpha)i(\beta)} k_{j(\alpha)j(\beta)}, \tag{4.28}$$

$$v_{\alpha\beta} = \int\int f_{i(\alpha)}(x) f_{j(\alpha)}(y) V(x,y) f_{i(\beta)}(x) f_{j(\beta)}(y) dx dy. \tag{4.29}$$

In the above potential energy equation $v_{\alpha\beta}$ is a true double integral which can be evaluated using two dimensional Gauss-Legendre integration.

4.2.1 Numerical results for two dimensions

Numerical Calculations were done for the two dimensional harmonic oscillator, for different parameters such as for x_{max} = y_{max} = 8, and for 10 Gauss-Legendre points. The Python code employed is listed in section 6.3.

In table 4.3, E_0, E_1, $\cdots$, E_5 are the eigen energies obtained via the calculations.

TABLE 4.3: Results of numerical calculations for 2-D harmonic oscillator;

n	h	E_0	E_1	E_2	E_3	E_4	E_5
4	2.0000	2.303871	5.385916	5.385916	8.096441	8.152794	8.442909
5	1.6000	2.080862	4.473939	4.473939	6.858751	6.998085	7.023671
6	1.3333	2.016334	4.131066	4.131066	6.243554	6.363701	6.369892
7	1.1429	2.002428	4.027192	4.027192	6.051474	6.099747	6.100884
8	1.0000	2.000262	4.003994	4.003994	6.007651	6.019370	6.019519
9	0.8889	2.000021	4.000412	4.000412	6.000794	6.002584	6.002597
10	0.8000	2.000001	4.000030	4.000030	6.000058	6.000236	6.000237
11	0.7273	2.000000	4.000002	4.000002	6.000003	6.000015	6.000015
12	0.6667	2.000000	4.000000	4.000000	6.000000	6.000000	6.000000
13	0.6154	2.000000	4.000000	4.000000	6.000000	6.000000	6.000000
14	0.5714	2.000000	4.000000	4.000000	6.000000	6.000000	6.000000
15	0.5333	2.000000	4.000000	4.000000	6.000000	6.000000	6.000000
16	0.5000	2.000000	4.000000	4.000000	6.000000	6.000000	6.000000

We can see from the above table 4.3, that as n increases, h decreases, and the energy levels also decrease, until convergence is achieved. When $n = 12$ and $h = 0.6667$, convergence was achieved for ground state energy level, E_0 = 2.000000. Also, we observed convergence for E_1 and E_2 when $n = 13$ and $h = 0.6154$, while the eigenvalue = 4.000000. Finally, when $n = 14$ and $h = 0.5714$, convergence was achieved for E_3, E_4 and E_5 with eigenvalue = 6.000000.

Considering the fully converged results in the last line of the above table we recover both the eigenvalues 2, 4 and 6 and the degeneracies 1, 2 and 3 for the 2-D harmonic oscillator. Please note again that our energies are twice those known for 2-D harmonic oscillator.

TABLE 4.4: Parameters of numerical calculations for harmonic oscillator for 10 Gauss-Legendre points,

n	h	N_{eig}	N_{int}	T_{bf}	T_{vmat}	T_{EVP}
4	2.0000	81	6400	2.1270	0.0648	0.0044
5	1.6000	121	10000	5.0457	0.2248	0.0123
6	1.3333	169	14400	10.2311	0.6282	0.0420
7	1.1429	225	19600	18.5929	1.4714	0.1197
8	1.0000	289	25600	31.1685	3.1726	0.1915
9	0.8889	361	32400	48.4563	6.2601	0.3513
10	0.8000	441	40000	73.5441	11.5954	0.6294
11	0.7273	529	48400	109.2896	20.1906	1.0501
12	0.6667	625	57600	154.8675	33.5829	1.7290
13	0.6154	729	67600	211.3824	53.5678	2.6635
14	0.5714	841	78400	279.7736	82.5839	3.9894
15	0.5333	961	90000	389.9374	123.8101	5.9517
16	0.5000	1089	102400	476.6943	181.5246	8.5615

In Table 4.4, we define the following parameters;
N_{eig} is the dimension of the eigenvalue problem.
N_{int} is the number of integration points.
T_{bf} is the time for calculating the basis functions.
T_{vmat} is the time for calculations of potential matrices.
T_{EVP} is the time for calculating the eigenvalue problems.

TABLE 4.5: Results of numerical calculations for harmonic oscillator for 10 Gauss-Legendre points, after code optimization;

n	h	N_{eig}	N_{int}	T_{bf}	T_{vmat}	T_{EVP}	E_0	E_1	E_2	E_3
4	2.0000	81	6400	0.0323	0.0672	0.0046	2.303871	5.385916	5.385916	8.096441
5	1.6000	121	10000	0.0603	0.2237	0.0132	2.080862	4.473939	4.473939	6.858751
6	1.3333	169	14400	0.1003	0.6186	0.0440	2.016334	4.131066	4.131066	6.243554
7	1.1429	225	19600	0.1613	1.4806	0.0969	2.002425	4.027162	4.027162	6.051418
8	1.0000	289	25600	0.2341	3.1731	0.1933	2.000262	4.003994	4.003994	6.007651
9	0.8889	361	32400	0.3343	6.2837	0.3545	2.000021	4.000412	4.000412	6.000794
10	0.8000	441	40000	0.4597	11.5784	0.6382	2.000001	4.000030	4.000030	6.000058
11	0.7273	529	48400	0.6081	20.1427	1.0403	2.000000	4.000002	4.000002	6.000003
12	0.6667	625	57600	0.8030	33.3005	1.6976	2.000000	4.000000	4.000000	6.000000
13	0.6154	729	67600	1.0367	53.4464	2.6106	2.000000	4.000000	4.000000	6.000000
14	0.5714	841	78400	1.3088	82.2574	3.9483	2.000000	4.000000	4.000000	6.000000
15	0.5333	961	90000	1.6377	122.7418	5.8455	2.000000	4.000000	4.000000	6.000000
16	0.5000	1089	102400	2.0274	180.1188	9.2350	2.000000	4.000000	4.000000	6.000000

Tables 4.5 and 4.6, show the effects of using 10 and 4 Gauss- Legendre integration points. We observed that the 4 Gauss-Legendre integration points reduced the number of integration points, time taken for the calculations of the basis functions, time for the calculation of the potential matrices and finally the time for the calculation of the eigenvalue problems, while maintaining the accuracy of the fast convergence of the energy levels.

TABLE 4.6: Results of numerical calculations for harmonic oscillator for 4 Gauss-Legendre points, after code optimization

n	h	N_{eig}	N_{int}	T_{bf}	T_{vmat}	T_{EVP}	E_0	E_1	E_2	E_3
4	2.0000	81	1024	0.0377	0.0121	0.0182	2.303268	5.386119	5.386119	8.092750
5	1.6000	121	1600	0.0240	0.0361	0.0126	2.080690	4.474059	4.474059	6.859193
6	1.3333	169	2304	0.0405	0.1016	0.0441	2.016295	4.131116	4.131116	6.243699
7	1.1429	225	3136	0.0626	0.2395	0.0954	2.002419	4.027176	4.027176	6.051453
8	1.0000	289	4096	0.0934	0.5131	0.1923	2.000261	4.003997	4.003997	6.007657
9	0.8889	361	5184	0.1260	1.0041	0.3553	2.000021	4.000412	4.000412	6.000795
10	0.8000	441	6400	0.1699	1.8476	0.6252	2.000001	4.000030	4.000030	6.000058
11	0.7273	529	7744	0.2202	3.2080	1.0613	2.000000	4.000002	4.000002	6.000003
12	0.6667	625	9216	0.2857	5.3602	1.7419	2.000000	4.000000	4.000000	6.000000
13	0.6154	729	10816	0.3621	8.4907	2.6021	2.000000	4.000000	4.000000	6.000000
14	0.5714	841	12544	0.4483	13.1560	4.0709	2.000000	4.000000	4.000000	6.000000
15	0.5333	961	14400	0.5563	19.5872	5.8239	2.000000	4.000000	4.000000	6.000000
16	0.5000	1089	16384	0.6818	28.7472	8.5508	2.000000	4.000000	4.000000	6.000000

4.2.2 Discussion on two dimensional harmonic oscillator results

In general, there is fast convergence of the energy levels. From the results of our calculations in table 4.3 we observe that the ground state energy, E_0 converged to 2.000000 when $n = 12$ and $h = 0.6667$. The first excited state energy, E_1 and second excited state energy E_2 converged to 4.000000 when $n = 13$ and $h = 0.6154$. This is a degeneracy in the energy levels. Similarly, E_3, E_4 and E_5 converged to 6.000000 when $n = 14$ and $h = 0.5714$. It shows another degeneracy in the energy levels.

We observed that a lot of time was taken for the evaluation of the basis functions after all the parameters were considered. The two dimensional Python and Numpy code was accelerated via unrolling of a large loop, i.e. splitting the large loop into two small loops for code optimization in order for evaluations to be faster. The time used for evaluating the basis functions is now reduced. The time for evaluating the potential energy matrices remains almost the same.

We see that when 4 Gauss-Legendre integration points were used, the time taken by the calculations process was reduced drastically, while the energy values remain the same. We can opt for 4 Gauss-Legendre integration points in order to reduce the CPU time taken.

4.2.3 Comparison of the sinc function and finite element method (FEM) results for 1-D harmonic oscillator

We compare the results of the calculations for the energy eigenvalues for the quantum mechanical simple harmonic oscillator using FEM [46] and the sinc function.

TABLE 4.7: Results of Numerical Calculations for Quantum Mechanics for 1-D Harmonic Oscillator, using Sinc function and FEM with linear interpolation [46] functions

Energy level	Energy Obtained with Sinc Function	Energy obtained with FEM	Exact Eigenvalue
E_0	1.00000000000	1.00693	1.00000000000
E_1	3.00000000000	3.03439	3.00000000000
E_2	5.00000000000	5.08875	3.00000000000
E_3	7.00000000000	7.16942	7.00000000000

TABLE 4.8: Results of Numerical Calculations for Quantum Mechanics for 1-D Harmonic Oscillator, using Sinc function and FEM using Hermite interpolation with two nodes [46] of three degrees of freedom

Energy level	Energy Obtained with Sinc Function	Energy obtained with FEM	Exact Eigenvalue
E_0	1.00000000000	1.000000	1.00000000000
E_1	3.00000000000	3.000000	3.00000000000
E_2	5.00000000000	5.000000	3.00000000000
E_3	7.00000000000	7.000001	7.00000000000

The numerical calculations obtained using the sinc basis functions and the finite element method (FEM) as shown in tables 4.7 and 4.8. It is obvious from the results that the sinc basis set has an edge over the finite element method (FEM).

4.3 Hydrogen Molecular Ion, H_2^+

The use of a basis set in molecular calculations [47] is becoming common in recent years. We now proceed to test the accuracy of the sinc basis functions for calculating of the ground state energy of H_2^+. The volume element in these coordinates is defined by

$$dV = \rho d\rho d\phi dz. \tag{4.30}$$

The two dimensional wave function for the hydrogen molecular ion in the ground state is

$$\psi = \psi(\rho, z). \tag{4.31}$$

The wave function of the ground state depends on ρ and z, but not on φ, because separation of variables yields a differential equation in φ, whose lowest eigenvalue is zero.
The Hamiltonian is given by

$$H = -\frac{1}{\rho}\frac{\partial}{\partial\rho}\rho\frac{\partial}{\partial\rho} - \frac{\partial^2}{\partial z^2} + V(\rho, z), \tag{4.32}$$

$$V(\rho, z) = -\frac{2}{\sqrt{\rho^2 + (z+1)^2}} - \frac{2}{\sqrt{\rho^2 + (z-1)^2}}. \tag{4.33}$$

4.3.1 Symmetrical Sinc function basis set

The ground state wave function ψ of the hydrogen molecular ion satisfies the boundary condition

$$\frac{\partial\psi(0, z)}{\partial\rho} = 0\,. \tag{4.34}$$

This can be shown by expanding the wave function in powers of ρ for $\rho \to 0$. We now also extend ψ to negative values of ρ, yielding a function even with respect to $\rho = 0$. It should be noted, that ρ is now just a variable but not a radius.
For simplicity we first consider a one-dimensional function $\phi(\rho)$ on the interval $[0, x_{\text{max}}]$ under the boundary condition

$$\frac{\partial\phi}{\partial\rho} = 0 \text{ at } \rho = 0\,. \tag{4.35}$$

The basis functions on the interval $[-x_{\text{max}}, x_{\text{max}}]$ used in this thesis are

$$f_j(x) = s_{-n+j-1}(x), \quad j = 1, \cdots, 2n+1\,. \tag{4.36}$$

with

$$s_i(x) = \frac{1}{\sqrt{h}}\text{sinc}(x - ih), \quad i = -n, \cdots, n\,. \tag{4.37}$$

In order to satisfy the boundary condition at $\rho = 0$ another set of basis functions is required whose derivative vanishes at $\rho = 0$. It is evident, that the derivative of the basis functions whose maxima lie on the right hand side of zero does not vanish. Thus another basis set is needed. The elements of this basis set are obtained by combining those f_j, whose origins are at $\rho = \pm ih$ and including the function f_{n+1}, whose derivative at the origin vanishes, to yield a basis set g_j of $n+1$ elements.
Denoting the expansion coefficients of a function F in the basis f_j by c_i and those in the symmetrized basis set g_j by d_i we have

$$\mathbf{c} = D\mathbf{d}, \tag{4.38}$$

with D shown below for $n = 4$

$$D = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

In general the matrix has dimensions (N_2, N_1) and it is used to get the function f to satisfy the non-symmetric boundary condition.
With $N_1 = n+1$ and $N_2 = 2n+1$ and those elements d_{ij} that satisfy $i + j = n+2$ or $i - j = n$ are unity.

In order to simplify the calculation of matrix elements as required for the variational method we use Eq (4.38) to obtain

$$\langle g_i|A|g_j\rangle = \sum_{kl} d_{ik}\langle f_k|A|f_l\rangle d_{lj}, \tag{4.39}$$

for an operator A. The functions g_i and g_j are the symmetrized basis functions.
Here the scalar product $\langle a|A|b\rangle$ is an abbreviation for

$$\int_0^{\text{xmax}} \rho a(\rho) A b(\rho) d\rho\,, \tag{4.40}$$

since we are using cylindrical coordinates.

We will now expand the ground state wave function in terms of products of sinc functions f_i in z and the functions g_j derived above, i.e.

$$\psi(\rho, z) = \sum_a c_a F_a(\rho, z), \tag{4.41}$$

with

$$F_a(\rho, z) = g_{i(a)}(\rho) f_{j(a)}(z), \tag{4.42}$$

and certain suitably defined $i(a)$ and $j(a)$.

To facilitate the evaluation of matrix elements with respect to the product basis we define

$$C = \mathbf{1} \bigotimes D\,,$$

where $\bigotimes$ stands for the tensor product between two matrices. Using this definition the matrix elements of an operator B which depends both on ρ and z can be written as

$$\langle F_a|B|F_b\rangle = \sum_{kl} c_{ak}\langle G_k|B|G_l\rangle c_{lb}. \tag{4.43}$$

Here the basis G_a is defined by

$$G_a = f_{k(a)}(\rho) f_{l(a)}(z), \tag{4.44}$$

with suitably defined functions $k(a)$ and $l(a)$ and the scalar products are defined with respect to the domain $[0, x_{\max}] \times [-x_{\max}, x_{\max}]$. Thus this product basis combines the sinc functions in both ρ and z.

The variational principle is applied, the two dimensional Schrödinger equation is solved numerically by expanding the wave function $\psi(\rho, z)$ in terms of products of one dimensional basis functions that constitute a new basis set defined by using thus

$$\psi(\rho, z) = \sum c_\alpha g_{i_\alpha}(\rho) f_{j_\alpha}(z)\,, \tag{4.45}$$

are the matrix elements of the basis functions. Thus we define Q_α in equation on line below as thus; Q_α = $g_{i(\alpha)}(\rho) f_{j(\alpha)}(z)$.

The Hamiltonian matrix becomes

$$h_{\alpha\beta} = \int\int Q_\alpha(\rho, z) \left(-\frac{1}{\rho}\frac{\partial}{\partial\rho}\rho\frac{\partial}{\partial\rho} - \frac{\partial^2}{\partial z^2} + V(\rho, z) \right) Q_\beta(\rho, z)\rho d\rho dz. \tag{4.46}$$

Some of the integrals resulting from the above equation cannot be evaluated analytically and the overlap integrals are not diagonal. Therefore the structure of $h_{\alpha\beta}$ is more complex than for the isotropic harmonic oscillator. This matrix is the sum of kinetic and potential energy matrices, which are defined by

$$h_{\alpha\beta} = t_{\alpha\beta} + V_{\alpha\beta}, \tag{4.47}$$

$$V_{\alpha\beta}=\int\int Q_{\alpha}(\rho,z)V(\rho,z)Q_{\beta}(\rho,z)\rho d\rho dz, \tag{4.48}$$

$$t_{\alpha\beta} = \int\int Q_{\alpha}(\rho,z)\left(-\frac{1}{\rho}\frac{\partial}{\partial\rho}\rho\frac{\partial}{\partial\rho} - \frac{\partial^2}{\partial z^2}\right)Q_{\beta}(\rho,z)\rho d\rho dz. \tag{4.49}$$

Substituting the expressions for the basis functions, we have

$$V_{\alpha\beta} = \int\int g_{i(\alpha)}(\rho)f_{j(\alpha)}(z)V(\rho,z)g_{i(\beta)}(\rho)f_{j(\beta)}(z)\rho d\rho dz. \tag{4.50}$$

$$t_{\alpha\beta} = \int\int g_{i(\alpha)}(\rho)f_{j(\alpha)}(z)\left(-\frac{1}{\rho}\frac{\partial}{\partial\rho}\rho\frac{\partial}{\partial\rho} - \frac{\partial^2}{\partial z^2}\right)g_{i(\beta)}(\rho)f_{j(\beta)}(z)\rho d\rho dz. \tag{4.51}$$

$$\begin{aligned} t_{\alpha\beta} &= \int_{-z_{\max}}^{z_{\max}}\int_0^{\rho_{\max}} \frac{\partial g_{i(\alpha)}(\rho)}{\partial\rho}\frac{\partial g_{i(\beta)}(\rho)}{\partial\rho} f_{j(\alpha)}(z)f_{j(\beta)}(z)\rho d\rho dz \\ &+ \int_{-z_{\max}}^{z_{\max}}\int_0^{\rho_{\max}} g_{i(\alpha)}(\rho)g_{i(\beta)}(\rho)\frac{\partial f_{j(\alpha)}(z)}{\partial z}\frac{\partial f_{j(\beta)}(z)}{\partial z}\rho d\rho dz. \end{aligned} \tag{4.52}$$

In contrast to the two dimensional harmonic oscillator the potential matrix cannot be simplified further and it reads,

$$\begin{aligned} V_{\alpha\beta} &= \int_{-z_{\max}}^{z_{\max}}\int_0^{\rho_{\max}} g_{i(\alpha)}(\rho)f_{j(\alpha)}(z)\left(\frac{-2}{\sqrt{\rho^2+(z+1)^2}}\right)g_{i(\beta)}(\rho)f_{j(\beta)}(z)\rho d\rho dz \\ &+ \int_{-z_{\max}}^{z_{\max}}\int_0^{\rho_{\max}} g_{i(\alpha)}(\rho)f_{j(\alpha)}(z)\left(\frac{-2}{\sqrt{\rho^2+(z-1)^2}}\right)g_{i(\beta)}(\rho)f_{j(\beta)}(z)\rho d\rho dz. \end{aligned}$$

The evaluation of the above potential matrix $v_{\alpha\beta}$ thus requires a true double integral which can be evaluated using two dimensional Gauss-Legendre integration.

4.3.2 Cusp factor ansatz

The Hydrogen molecular ion, H_2^+, poses a challenge when solving the Schrödinger equation, using the sinc basis functions in cylindrical coordinates. The resulting eigenvalues show no convergence, since the ground state wave function exhibits a cusp [48] at the coordinates of the nuclei. In order to overcome this problem, we write the wave function as a product [47, 49] of a cusp factor F and a function ϕ, i.e.

$$\psi(\mathbf{r}) = F(\mathbf{r})\phi(\mathbf{r}). \tag{4.53}$$

The suitable choice of F must achieve the required cusp behaviour. In this the Hamiltonian for a molecule with N_a nuclei at R_i, $i = 1, ..., N_a$ the ansatz ij

$$F(r) = 1 + \sum_{i=1}^{N_a} c_i exp(-2Z_i r_i), \tag{4.54}$$

with

$$r_i = |\mathbf{r} - \mathbf{R_i}|. \tag{4.55}$$

For F to satisfy the cusp conditions at all the nuclei, we require

$$\lim_{r_i \to 0} \frac{d\bar{F}}{dr_i} = -Z_i F(\mathbf{R_i}), \tag{4.56}$$

leading to a linear system of equations of the form

$$\mathbf{Ac} = \mathbf{1}, \tag{4.57}$$

with

$$a_{ij} = \sum_{j=1}^{N_a} \left[\delta_{ij} - (1 - \delta_{ij}) exp(-2Z_i |\mathbf{R_i} - \mathbf{R_j}|)\right], \tag{4.58}$$

from which the coefficients c_i are obtained.
The above product ansatz is substituted into the Schrödinger equation to obtain the expectation value of the Hamiltonian as

$$\langle h \rangle = \int \left[\nabla(F\phi).\nabla(F\phi) + F\phi V F\phi\right] d^3r. \tag{4.59}$$

Using integration by parts we can simplify the above to

$$\langle h \rangle = \int \left[F^2 |\nabla\phi|^2 + F^2 W(r) \phi^2\right] d^3r, \tag{4.60}$$

where

$$W_r = V(r) - \frac{\nabla^2 F}{F}. \tag{4.61}$$

Thus the sinc basis functions technique is applied as usual starting from the Hamiltonian $\langle h \rangle$ expression, under the normalization condition;

$$\langle F\phi | F\phi \rangle = 1, \tag{4.62}$$

$$\langle \psi | \psi \rangle = \int F^2 \phi^2 d^3r. \tag{4.63}$$

Expanding ϕ_i in terms of our chosen sinc basis functions f_i,

$$\phi_i(\mathbf{r}) = \sum_{\mathbf{i}} \mathbf{c_j^i f_j}(\mathbf{r}), \tag{4.64}$$

the vectors of the expansion coefficients for the orbitals ϕ satisfy the generalized eigenvalue problem

$$H(\mathbf{v_i}) = \mathbf{e_i}\mathbf{U}\mathbf{v_i}, \tag{4.65}$$

where the elements of the matrices above are given by

$$h_{\alpha\beta} = \int \nabla f_\alpha . \nabla f_\beta F^2 \rho d\rho dz + \int f_\alpha W f_\beta F^2 \rho d\rho dz, \tag{4.66}$$

$$u_{\alpha\beta} = \int f_\alpha f_\beta F^2 \rho d\rho dz. \tag{4.67}$$

The kinetic matrices are given by

$$t_{\alpha\beta} = \int \nabla f_\alpha . \nabla f_\beta F^2 \rho d\rho dz, \tag{4.68}$$

while the potential matrices are given as

$$V_{\alpha\beta} = \int f_\alpha W f_\beta F^2 \rho d\rho dz. \tag{4.69}$$

4.3.3 Numerical calculations for hydrogen molecular ion

The properly defined and symmetrized sinc basis functions were used to solve the Schrödinger equation, employing the Python code as listed in section 6.4. Due to the slow convergence rate, a cusp factor was incorporated into the Python code to obtain convergence. Numerical calculations were done using 6 Gauss-Legendre points.

TABLE 4.9: Ground State Energy for Hydrogen Molecular ion H_2^+ for 6 Gauss-Legendre points;

n, x_{max}	8	9	10	11
6	−2.18872648	−2.17524035	−2.15527998	−2.12653719
7	−2.19949231	−2.19141089	−2.18023681	−2.16627527
8	−2.20343190	−2.19988067	−2.19342910	−2.18387560
9	−2.20475349	−2.20333783	−2.20017701	−2.19492593
10	−2.20500928	−2.20462627	−2.20376415	−2.20041787
11	−2.20527325	−2.20499988	−2.20451046	−2.20320160
12	−2.20514592	−2.20517149	−2.20499862	−2.20440769
14	−2.20523021	−2.20519125	−2.20527236	−2.20501570
16	−2.20526185	−2.20523723	−2.20526246	−2.20517891
18	−2.20525576	−2.20526165	−2.2052487	−2.20528852
20	−2.20525665	−2.20526009	−2.20526160	−2.20525917
21	−2.20527435	−2.20526499	−2.20526058	−2.20525889
22	−2.20526327	−2.20526561	−2.20526514	−2.20526157
24	−2.20526752	−2.20527148	−2.20527585	−2.20526825
26	−2.20526600	−2.20526787	−2.20527008	−2.20527292
27	−2.20527050	−2.20526740	−2.20526671	−2.20526312
28	−2.20526565	−2.20526715	−2.20526555	−2.20525896
30	−2.20526705	−2.20526766	−2.20526730	−2.20526399
32	−2.20526826	−2.20526630	−2.20526826	−2.20526716

n is the number of intervals; and $h = \mathrm{x}_{\mathrm{max}}/n$

4.3.4 Results for the hydrogen molecular ion

Using the sinc basis functions to calculate the ground state energy of the molecule H_2^+, employing Python and Numpy, the resulting eigenvalues show no convergence.
As we introduce the cusp factor at the coordinates of the nuclei, fast convergence of E_n is achieved, provided n is restricted to be a multiple of x_{max}. Properly symmetrized sinc basis functions were used to solve the Schrödinger equation. Since the memory requirements of the calculations exceeded those provided by a laptop, the calculations had to be performed serially on a HPC Cluster with Intel-Xeon-2.60GHz processors.

4.3.5 Results of hydrogen molecular ion (H_2^+) for different multiple values of x_{max}

TABLE 4.10: Results of numerical calculations for H_2^+ for $x_{max} = 8$;

n	E_n	T_{eig}	T_{CPU}
8	-2.2034319021	0.0381	7.80012×10^0
16	-2.2052618504	0.5150	2.05465×10^3
24	-2.2052675161	4.9273	2.35813×10^4
32	-2.2052682560	24.7067	1.14732×10^5

TABLE 4.11: Results of numerical calculations for H_2^+ for $x_{max} = 9$;

n	E_n	T_{eig}	T_{CPU}
9	-2.2033378271	0.1292	1.96519×10^1
18	-2.2052616520	1.0058	1.49664×10^3
27	-2.2052674016	9.6054	1.43385×10^4
36	-2.2052681547	48.0883	2.36921×10^5

E_n is the ground state energy for n interval. T_{eig} is the time for the evaluation of the eigenvalues. T_{CPU} is the CPU time taken for the system calculation

TABLE 4.12: Results of numerical calculations for H_2^+ for $x_{\text{max}} = 10$;

n	E_n	T_{eig}	T_{CPU}
10	-2.2032641495	0.1435	4.06690×10^1
20	-2.2052616023	1.7179	3.34859×10^3
30	-2.2052673892	16.7811	3.13161×10^4
40	-2.2052681480	92.3637	4.42660×10^5

TABLE 4.13: Results of numerical calculations for H_2^+ for $x_{\text{max}} = 11$,

n	E_n	T_{eig}	T_{CPU}
11	-2.2032016043	0.0893	7.47470×10^1
22	-2.2052615698	2.8618	4.41524×10^3
33	-2.2052673873	28.7552	6.86511×10^4
44	-2.2052681474	163.7395	8.64033×10^5

4.3.6 Least square fits for H_2^+ energies

Defining

$$\Delta E_n = E_n - E_{\text{gs}},$$

making the ansatz

$$\Delta E_n = \left(\frac{1}{n}\right)^a \exp(b),$$

and taking the logarithm of both sides we obtain with $x = \log 1/n$,
$y = \log \Delta E_n$,
the equation of a straight line

$$y = ax + b. \tag{4.70}$$

The Python code employed with the numerical extension Numpy is listed in section 6.5.

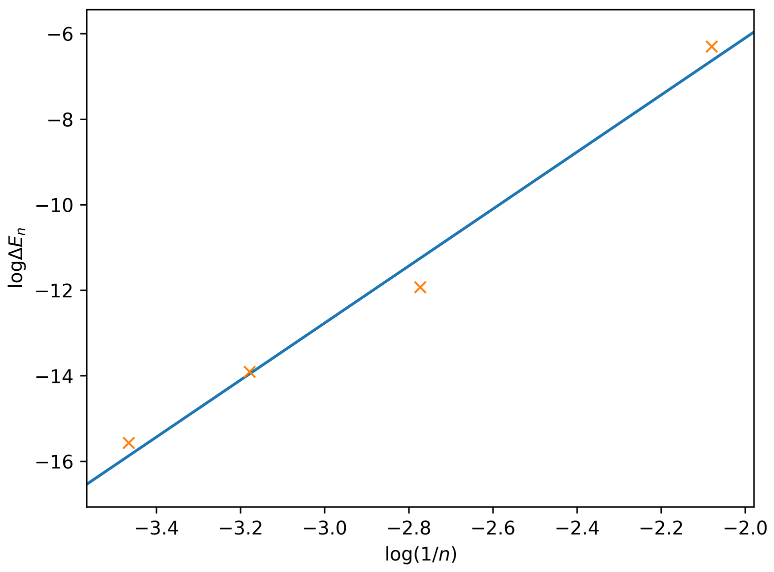

FIGURE 4.3: Least Squares Fit of $\log(\Delta E_n)$ versus $\log(1/n)$ for $x_{max} = 8$

TABLE 4.14: Parameters obtained from Least Squares Fits for H_2^+;

x_{max}	a	b
8	6.667	7.233
9	6.422	7.449
10	6.433	8.182
11	6.454	8.873

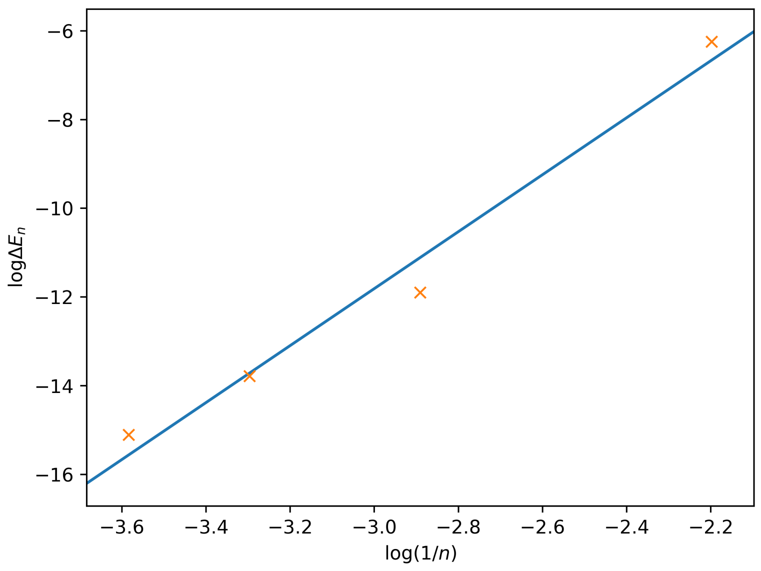

FIGURE 4.4: Least Squares Fit of $\log(\Delta E_n)$ versus $\log(1/n)$ for $x_{max} = 9$

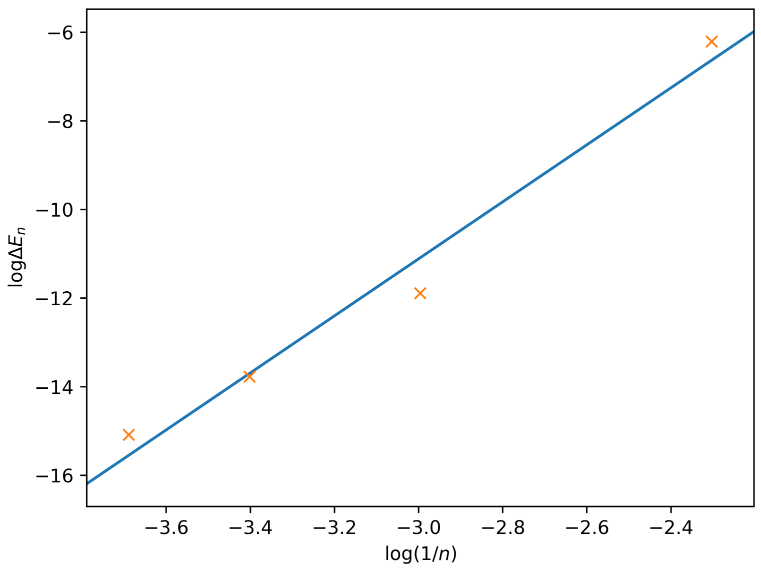

FIGURE 4.5: Least Squares Fit of $\log(\Delta E_n)$ versus $\log(1/n)$ for $x_{max} = 10$

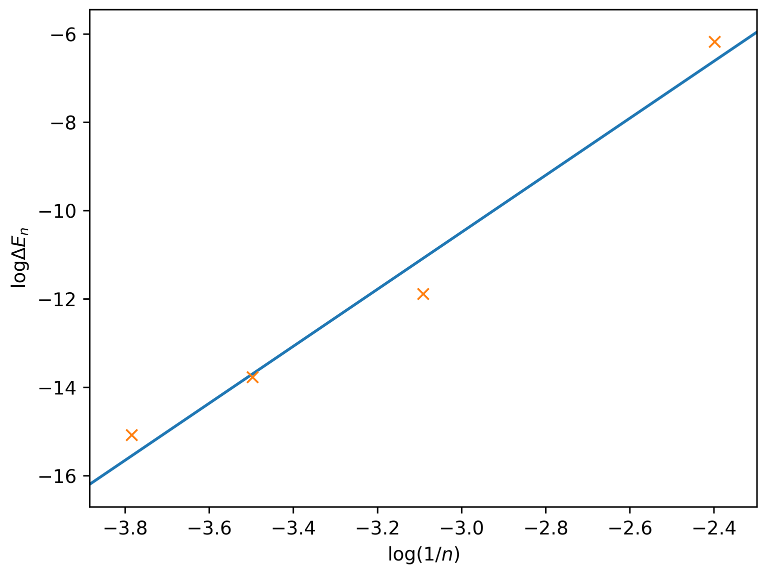

FIGURE 4.6: Least Squares Fit of $\log(\Delta E_n)$ versus log(1/n) for $x_{max} = 11$

4.3.7 Discussion on Convergence of H_2^+

The least square fits to $y = ax + b$ were performed for x_{max} = 8,9,10 and 11. We plotted $y = \log(E_n - E_{gs}) = log\Delta E_n$ versus $x = log(1/n)$ and the data points were nearly on a straight line, as seen in Figures 4.3 − 4.6.
The double-logarithmic least squares fits of ΔE_n versus $1/n$ were done to obtain the values of a and b as shown in the table 4.14

The parameters resulting from least squares fits of $\log(\Delta E_n)$ versus log(1/n) are shown above in Table 4.14:
From this table it is evident, that the convergence order is at least 6.

Chapter 5

Conclusions

In this thesis, it has been demonstrated that using sinc basis functions the convergence of the energy levels of ground state, first, second and third excited states for the one dimensional quantum harmonic oscillator was very fast. When comparing the results obtained for the numerical calculations of the sinc functions and the finite element method (FEM) [46] it is evident that the sinc basis set has an edge over the finite element method.

For the two dimensional harmonic oscillator, we also observed fast convergence to the theoretical values, also including the degeneracies.

For the Morse Potential the energy error of the eigenvalues was in good agreement with the theoretically expected behaviour and the eigenvalues converged to the theoretically expected values.

For the hydrogen molecular ion, H_2^+, it was shown that using a Cusp factor at the location of the nuclei resulted in fast convergence. With respect to $x = 1/n$ the order of convergence is approximately 6. With respect to $x = 1/N$ where N is the number of degrees of freedom the convergence order is approximately 3. For comparison we consider the finite element calculation by Braun [49] where a convergence order of approximately 4 was obtained.

All the codes used were written in Python, with the numerical extension Numpy and Scipy modules. In future, additional applications should be found to observe how the sinc basis functions will perform on other systems.

Bibliography

[1] A. A Rajabi. “Exact Analytical Solution of the Schrödinger Equation for an N-Identical Body-Force System”. In: *Few-Body Systems* 37 (2005), 197–213.

[2] Gaotsiwe J Rampho. “The Schroedinger equation on a Lagrange mesh”. In: *Journal of Physics; Conf. Series 905* (2017).

[3] Jr. Jones Marvin Quenten. “Numerical Methods in Quantum Mechanics: Analysis of Numerical Schemes on One-Dimensional Schrodinger Wave Problems”. In: *Masters Abstracts International* Volume: 52-04.; (2013), p. 121.

[4] Xionghua Wu Chen Li. “Numerical solution of differential equations using Sinc method based on the interpolation of the highest derivatives”. In: *Applied Mathematical Modelling, Issue 1* 31 (2007), pp. 1–9.

[5] A. Pirkhedri K. Parand Mehdi Dehghan. “The Sinc-collocation method for solving the Thomas-Fermi equation”. In: *ComputationalandApplied Mathematics* 237 (2012), pp. 244–252.

[6] Esmail Hesameddini and Elham Asadolahifard. “Solving systems of linear volterra integro-differential equations by using Sinc-collocation method”. In: *International Journal of Mathematical Engineering and Science* 2.Issue 7 (2013), pp. 1–9.

[7] John Lund and Kenneth L. Bowers. *Sinc Methods for Quadrature and Differential Equations*. P 101-174. ISBN, 1992.

[8] Abbas Saadatmandi Mehdi Dehghan. “The numerical solution of a nonlinearsystem of second-order boundary value problems using the sinc-collocation method”. In: *Mathematical and Computer Modelling, Issue 11-12* 46 (2007), pp. 1434–1441.

[9] Jennifer L. Mueller and Thomas S. Shores. “A new Sinc-Galerkin method for convection-diffusion equations with mixed boundary conditions”. In: *Computers & Mathematics with a Applications* 47.Issues 4-5 (2004), P 803–822.

[10] Abbas Saadatmandi and Mehdi Dehghan. “The use of Sinc-collocation method for solving multi-point boundary value problems”. In: *Communications in Nonlinear Science and Numerical Simulation* 17 (2012), pp. 593–601.

[11] Frank Stenger. “Summary of sinc numerical methods”. In: *Journal of computational and applied mathematics* 121 (2000), pp. 379–420.

[12] T. Matsuo M. Sugihara. "Recent developments of the Sinc numerical methods". In: *Journal of Computational and Applied Mathematics* V 164–165, (2003), P 673 – 689.

[13] Frank Stenger. "Numerical Methods based on Whittaker Cardinal, or Sinc Functions". In: *Society for Industrial and Applied Mathematics* 23.2 (1981), pp. 165–224.

[14] Timothy Bui. "Explicit and Implicit Methods In Solving Differential Equations". In: *Digital Commons Uconn* Honors Sch Theses.119 (2010).

[15] David Stewart Kendall Atkinson Weimin Han. *Numerical Solution of Ordinary Differential Equations*. John Wiley & Sons, Inc, 2009.

[16] Ashok Kumar and T.E. Unny. "Application of Runge-Kutta method for the solution of non-linear partial differential equations". In: *Applied Mathematical Modelling* 1 (1977), pp. 199–204.

[17] Helmut Grubmuller Timo Graen. "NuSol - Numerical solver for the 3D stationary nuclear Schroedinger equation". In: *Computer Physics Communications* 198 (2016), pp. 169–178.

[18] Aysegul Akyuz and Mehmet Sezer. "A chebyshev collocation method for the solution of linear integro-differential equations". In: *International Journal of Computer Mathematics* 72 (1999), pp. 491–507.

[19] Kamel Al-Khaled Sababheh MS Abdul-Majid Nusayr. "Some convergence results on sinc interpolation". In: *Journal of Inequalities in Pure and Applied Mathematics* 4, Issue 2.Article 32 (2003).

[20] CTI Reviews. *Calculus - Single Variable*. Cram101 Textbook Reviews, 2016.

[21] Waldemar Dos Passos. *Numerical Methods, Algorithms and Tools in C Plus Language*. CRC Press, 2016.

[22] Mahtab Uddin. "Study on different numerical methods for solving differential equations". In: *Academia* (2011).

[23] Cheng Yung Ming. "Solution of differential equations with applications to engineering problems". In: *InTechOpen Book Series* (2017).

[24] Nouredine Zettili. *Quantum Mechanics Concepts and Applications*. Second Edition. John Wiley and Sons Ltd, 2009.

[25] Frank Stenger. *Handbook of Sinc Numerical Methods*. CRC Press - Taylor & Francis Group, 2011.

[26] Frank Stenger. "Matrices of Sinc methods". In: *Journal of Computational and Applied Mathematics* 86 (1997), pp. 297–310.

[27] John Lund DF Winter Kenneth L. Bowers. "Wind-driven currents in a sea with a variable eddy viscosity calculated via a Sinc–Galerkin technique". In: *International Journal for Numerical Methods in Fluids* 33 (2000), pp. 1041–1073.

[28] I. Kucuk K. Abdella X. Yu. "Application of the Sinc method to a dynamic elasto-plastic problem". In: *Journal of Computational and Applied Mathematics* 223.2 (2009), 626–645.

[29] J Rashidinia and M Zarebnia. “The numerical solution of integro-differential equation by means of the Sinc method”. In: *Applied Mathematics and Computation* 188.Issue 2 (2007), pp. 1124–1130.

[30] E.T. Whittaker. “On the functions which are represented by expansion of the interpolation theory”. In: *Proceedings of the Royal Society of Edinburgh* 35 (1915), pp. 181–194.

[31] & I.L. Davies P.M. Woodward. “Information Theory and Inverse Probability in Telecommunication”. In: *Radio & Communication Engineering* 99.58 (1952), pp. 37–44.

[32] P. M. Woodward. *Probability and Information Theory, with Applications to Radar. 2nd Edition*. Ed. by D. W. Fry W. Higinbotham. Vol. 3. International Series of Monographs on Electronics and Instrumentation. Pergamon, 1953.

[33] William B. Gearhart Harris S. Shultz. “The Function Sinc”. In: *The College mathematics Journal* 21.2 (1990), pp. 90 –99.

[34] Raul Guantes and Stavros C. Farantos. “High order finite difference algorithms for solving the Schroedinger equation in molecular dynamics II. Periodic variables”. In: *Journal of Chemical Physics* 113.23 (2000), P 10429–10437.

[35] F. Stenger. *Numerical Methods based on Sinc and Analytic Functions*. P 131-310. Springer New York, 1993.

[36] Damian Trip. “On Sinc Methods for partial differential equations”. In: *Fixed Point Theory* 3 (2002), pp. 173–182.

[37] Ralph Baker Kearfott. “A sinc approximation for the indefinite integral”. In: *Mathematics of computation* 41.164 (1983), pp. 559–572.

[38] I. Bogaert. “Iteration-free computation of Gauss-Legendre quadrature nodes and weights”. In: *SIAM Journal on Scientific Computing* 36 (2014), A1008–A1026.

[39] Gene H. Golub and John H. Welsch. “Calculation of Gauss quadrature rules”. In: *Math. Comp.* 23 (1969), pp. 221–230.

[40] Alex Townsend. “The race for high order Gauss-Legendre quadrature”. In: *SIAM News* 48 (2015), pp. 1–3.

[41] Guido Van Rossum and Fred L Drake Jr. “Python tutorial”. In: *Technical Report CS-R9526* (1995).

[42] Millman K.J. van der Walt-S.J. et al Harris C.R. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362.

[43] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272.

[44] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95.

[45] Philip M. Morse. “Diatomic molecules according to the wave mechanics 11 vibrational levels”. In: *Phys.Rev.* 34 (1929), pp. 57–64.

[46] Don Dossa J. Shertzer L. R. Ram-Mohan Sunil Saigal. “The finite-element method for energy eigenvalues of quantum mechanical systems”. In: *AIP Conference Proceedings* (1989).

[47] Moritz Braun and Kingsley Onyebuchi Obodo. “Finite element density functional calculations for light molecules using a cusp factor to mitigate the Coulomb potential”. In: *The European Physical Journal* 230 (2019), p 100310–6.

[48] Claudio Attaccalite. “RVB phase of hydrogen at high pressure:towards the first ab-initio Molecular Dynamicsby Quantum Monte Carlo”. PhD thesis. Sissa (International School for Advanced Studies), 2005.

[49] Moritz Braun. “Finite element calculations for systems with multiple Coulomb centers”. In: *Journal of Computational and Applied Mathematics* 236 (2012), pp. 4840–4845.

Chapter 6

Python Source Code

6.1 Solve 1-D Schrödinger Harmonic Oscillator

Solve1dSchroedingerHOUsingSinc.py

```
from numpy import *
from numpy.polynomial.legendre import leggauss
import sys
import time
xg,wg=leggauss(20)
xg=0.5*(xg+1)
wg=0.5*wg
def V(x):
    #return -2*exp(-2*x*x)
    return x*x
    #return  D*(exp(-2*x) - 2*exp(-x))
h,n=map(eval,sys.argv[1:3])
nf=1.0/sqrt(h)
xmin=-n*h
xmax=n*h
print xmin,xmax
ndof=2*n+1
def s(x,i):
    t=pi*(x/h-i)
    if t==0:
        return nf*1
    else:
        return nf*sin(t)/t
def f(x,k):
    i=k-n
    return s(x,i)
import pylab
xi=linspace(xmin,xmax,200*ndof+1)
si=array([s(x,0) for x in xi])
pylab.plot(xi,si,"-")
pylab.grid()
pylab.savefig("s0.png")
pylab.close()
fi=array([f(x,n) for x in xi])
pylab.plot(xi,fi,"-")
pylab.grid()
pylab.savefig("fn.png")
pylab.close()
#####
yi=[]
wi=[]
for k in range(ndof-1):
    tmpy=list((k-n)*h+h*xg)
    tmpw=list(h*wg)
    yi+=tmpy
    wi+=tmpw
yi=array(yi)
wi=array(wi)
fi_atyi=[]
for i in range(ndof):
    tmp=[f(y,i) for y in yi]
    fi_atyi.append(tmp)
fi_atyi=array(fi_atyi)
vi_atyi=array([V(y) for y in yi])
vw=diag(vi_atyi*wi)
t1=time.time()
vmat=dot(fi_atyi,dot(vw,fi_atyi.T))
t2=time.time()
#print t2-t1
kmat=zeros((ndof,ndof),"d")
for i in range(ndof):
    for j in range(ndof):
        if i == j:
            kmat[i,i]=pi**2/3.0/h**2
        else:
            d=abs(i-j)
            kmat[i,j]=(-1)**d*(2.0/d**2)/h**2
#hmat=kmat+vmat
hmat=kmat+vmat
evals,evecs=linalg.eig(hmat)
from operator import itemgetter
tmp=zip(evals,evecs.T)# important to use Transpose here!
tmp1=sorted(tmp,key=itemgetter(0))
evals=[tmp1[i][0] for i in range(ndof)]
evecs=[tmp1[i][1] for i in range(ndof)]
print evals[:4]
evec2=evecs[2]
C="""
def wave0(x):
    tmp=0
    for i in range(ndof):
        tmp=tmp+evec2[i]*f(x,i)
    return tmp
xp=linspace(-5,5,1001)
wavei=array([wave0(x) for x in xp])
waveana=exp(-xp*xp/2.0)/pi**0.25
#pylab.plot(xp,wavei-waveana,"-")
pylab.plot(xp,wavei,"-")
pylab.grid()
pylab.savefig("wave0.png")
pylab.close()"""
```

6.2 Solve 1-D Schrödinger Morse Potential

Solve1dSchroedingerMorsePotUsingSinc.py

```
from numpy import *
from numpy.polynomial.legendre import leggauss
import sys
import time
xg,wg=leggauss(20)
xg=0.5*(xg+1)
wg=0.5*wg
def V(x):
    #return -2*exp(-2*x*x)
    #return x*x
    return  D*(exp(-2*x) - 2*exp(-x))
h,n,D=map(eval,sys.argv[1:4])
nf=1.0/sqrt(h)
xmin=-n*h
xmax=n*h
print xmin,xmax
ndof=2*n+1
def s(x,i):
    t=pi*(x/h-i)
    if t==0:
        return nf*1
    else:
        return nf*sin(t)/t
def f(x,k):
    i=k-n
    return s(x,i)
import pylab
xi=linspace(xmin,xmax,200*ndof+1)
si=array([s(x,0) for x in xi])
pylab.plot(xi,si,"-")
pylab.grid()
pylab.savefig("s0.png")
pylab.close()
fi=array([f(x,n) for x in xi])
pylab.plot(xi,fi,"-")
pylab.grid()
pylab.savefig("fn.png")
pylab.close()
#####
yi=[]
wi=[]
for k in range(ndof-1):
    tmpy=list((k-n)*h+h*xg)
    tmpw=list(h*wg)
    yi+=tmpy
    wi+=tmpw
yi=array(yi)
wi=array(wi)
fi_atyi=[]
for i in range(ndof):
    tmp=[f(y,i) for y in yi]
    fi_atyi.append(tmp)
fi_atyi=array(fi_atyi)
vi_atyi=array([V(y) for y in yi])
vw=diag(vi_atyi*wi)
t1=time.time()
vmat=dot(fi_atyi,dot(vw,fi_atyi.T))
t2=time.time()
#print t2-t1
kmat=zeros((ndof,ndof),"d")
for i in range(ndof):
    for j in range(ndof):
        if i == j:
            kmat[i,i]=pi**2/3.0/h**2
        else:
            d=abs(i-j)
            kmat[i,j]=(-1)**d*(2.0/d**2)/h**2
```

```
#hmat=kmat+vmat
hmat=kmat+vmat
evals,evecs=linalg.eig(hmat)
from operator import itemgetter
tmp=zip(evals,evecs.T)# important to use Transpose here!
tmp1=sorted(tmp,key=itemgetter(0))
evals=[tmp1[i][0] for i in range(ndof)]
evecs=[tmp1[i][1] for i in range(ndof)]
print evals[:4]
C="""
evec2=evecs[2]
def wave0(x):
    tmp=0
    for i in range(ndof):
        tmp=tmp+evec2[i]*f(x,i)
    return tmp
xp=linspace(-5,5,1001)
wavei=array([wave0(x) for x in xp])
waveana=exp(-xp*xp/2.0)/pi**0.25
#pylab.plot(xp,wavei-waveana,"-")
pylab.plot(xp,wavei,"-")
pylab.grid()
pylab.savefig("wave0.png")
pylab.close()"""
```

6.3 Solve 2-D Schrödinger Harmonic Oscillator

Solve22dSchroedingerUsingSinc.py

```
from numpy import *
from numpy.polynomial.legendre import leggauss
import sys
from time import time
xg,wg=leggauss(10)
xg=0.5*(xg+1)
wg=0.5*wg
def V(x,y):
    #return -2*exp(-2*x*x)
    return x*x + y*y
h,N=map(eval,sys.argv[1:3])
nf=1.0/sqrt(h)
nf2 = nf**2
xmin=-N*h
xmax=N*h
ymin=-N*h
ymax=N*h
#print xmin,xmax,ymin,ymax
ndof=2*N+1
ndof2=ndof**2

def sinc(t):
    if t==0:
        return nf
    else:
        return nf*sin(t)/t
def s(x,y,i,j):
    tx=pi*(x/h-i)
    ty=pi*(y/h-j)
    return sinc(tx)*sinc(ty)
def f(x,y,k):
    ik=k % ndof
    jk=k // ndof
    return s(x,y,ik-N,jk-N)

#def f(x,y):
#    return sin(x)*cos(2*y)
#xi = linspace(-4,4,101)
#for x in xi:
#    for y in xi:
#        print x,y,f(x,y,ndof**2/2)
#    print
#C="""
C1="""
def psi(x,y):
    for tmps in range((ndof)**2)
    tmp=tmp+ci(i),f(x,y,i)
    return tmp"""

yi=[]
wi=[]
for k in range(ndof-1):
    tmpy=list((k-N)*h+h*xg)
    tmpw=list(h*wg)
    yi+=tmpy
    wi+=tmpw
yi=array(yi)
wi=array(wi)
nint=len(wi)
nint2=nint**2
t1=time()
q=0
uq= []
vq=[]
rq=[]
for i in range(nint):
    for j in range(nint):
        uq.append(yi[i])
        vq.append(yi[j])
        rq.append(wi[i]*wi[j])
        q=q+1
uq=array(uq)
vq=array(vq)
rq=array(rq)
t2=time()
print "time␣to␣create␣2d␣int␣grid=",t2-t1
fi_atyi=zeros([ndof2,nint2])
print ndof2
print nint2
t1=time()
tx=zeros(nint)
ty=zeros(nint)
for k in range(ndof2): # alpha in thesis
    #for q in range(nint2):
    #    #print k,q
    #    fi_atyi[k,q]=f(uq[q], vq[q], k)
    ik=k % ndof # adjust to arrays starting at 1
    jk=k // ndof #
    for i in range(nint):
        tx[i]=pi*(yi[i]/h-(ik-N))
    for j in range(nint):
        ty[j]=pi*(yi[j]/h-(jk-N))
    stx=array([sinc(t) for t in tx])
    sty=array([sinc(t) for t in ty])
    tmp=outer(stx,sty)
    tmp.shape=nint2
    fi_atyi[k]=tmp.copy()
t2=time()
print "time␣to␣evaluate␣all␣2d␣basis␣functions␣at␣all␣integration␣points=",t2-t1
t1=time()
vi_atyi=array([V(xx,yy) for (xx,yy) in zip(uq,vq)])
d=vi_atyi*rq
t2=time()
print "time␣to␣evaluate␣potential␣at␣all␣int␣points=",t2-t1
t1=time()
tmpmat=(d*fi_atyi).T
vmat=dot(fi_atyi,tmpmat)
t2=time()
print "time␣to␣evaluate␣product␣of␣three␣matrices␣giving␣vmat=",t2-t1
t1=time()
tx=zeros((ndof2,ndof2))
ty=zeros((ndof2,ndof2))
ttot=zeros((ndof2,ndof2))
kmat=zeros((ndof,ndof),"d")
for i in range(ndof):
    for j in range(ndof):
        if i == j:
            kmat[i,i]=pi**2/3.0/h**2
        else:
            d=abs(i-j)
            kmat[i,j]=(-1)**d*(2.0/d**2)/h**2
for ia in range(ndof):
    for ib in range(ndof):
         for j in range(ndof):
             k=ndof*ia + j
             l=ndof*ib +j
             tx[k,l]=kmat[ia,ib]
for i in range(ndof):
    for ja in range(ndof):
        for jb in range(ndof):
            k=ndof*i + ja
            l=ndof*i + jb
            ty[k,l]=kmat[ja,jb]
ttot=tx+ty
t2=time()
print "time␣to␣calculate␣matrix␣for␣kinetic␣energy=",t2-t1
hmat=ttot+vmat
t1=time()
evals,evecs=linalg.eig(hmat)
```

```
t2=time()
print "time␣to␣solve␣eigenvalue␣problem=",t2-t1
evals.sort()
print evals[:10]

c="""
from operator import itemgetter
tmp=zip(evals,evecs.T)# important to use Transpose here!
tmp1=sorted(tmp,key=itemgetter(0))
evals=[tmp1[i][0] for i in range(ndof)]
evecs=[tmp1[i][1] for i in range(ndof)]
print evals[:4]
evec0=evecs[0]
def wave0(x):
    tmp=0
    for i in range(ndof):
        tmp=tmp+evec0[i]*f(x,i)
    return tmp
xp=linspace(-5,5,1001)
wavei=abs(array([wave0(x) for x in xp]))
waveana=exp(-xp*xp/2.0)/pi**0.25"""
```

6.4 Solve H_2^+ with Cusp Function

SolveH2CuspFunc26March2020.py

```
from numpy import *
from numpy.polynomial.legendre import leggauss
from scipy.linalg import eig,eigh
import pylab
import sys
from time import time
from CuspPotFact import cuspPotWfFact
xg,wg=leggauss(6)
xg=0.5*(xg+1)
wg=0.5*wg
e=0.0
NA=2  # fixed number of atoms
ZI=[1,1]
RI=[array([0,0,-1]), array([0,0,1])]  # numpy arrays for nuclei
poteff,wf1,ci=cuspPotWfFact(NA,ZI,RI)
def wf(rho,z):
    return wf1(rho,0.0,z)
#def wf(rho,z):
#    return 1.0
def V(rho,z):
    #return rho*rho+z*z
    return poteff(rho,0.0,z)
xmax,N=map(eval,sys.argv[1:3])
h=xmax/(N*1.0)
nf=1.0/sqrt(h)
nf2 = nf**2
xmin=0
#xmax=N*hn
ymin=-xmax
ymax=xmax
#print xmin,xmax,ymin,ymax
ndofrho = N+1
ndofrho1=2*N+1
ndofz=2*N+1
ndof2=ndofz * ndofrho
ndof21=ndofz * ndofrho1
def sinc(t):
    if t==0:
        return nf
    else:
        return nf*sin(t)/t

def dsinc(t):
   if t==0:
        return 0.0
   else:
        return nf*pi/h*(cos(t)/t-sin(t)/t**2)

def s(x,y,i,j):
    tx=pi*(x/h-i)
    ty=pi*(y/h-j)
    return sinc(tx)*sinc(ty)
def f(x,y,k):
    ik=k % ndofrho1
    jk=k // ndofrho1
    return s(x,y,ik-N,jk-N)
# define derivative of 1D rho and z functions
def frho(rho,i):
    t=pi*(rho/h-i+N)
    return sinc(t)
def dfrho(rho,i):  # changed to cover interval -N*h to N*h
    return dsinc(pi*(rho/h-i+N))
N1=N+1   # dimension of final basis set
N2=2*N+1 # dimension of initial basis set
D=zeros((N2,N1))
for i in range(N1-1):
    D[i,N-i]=1.0
for i in range(0,N1):
    D[i+N,i]=1.0
#######
yirho=[]
wirho=[]
for k in range(ndofrho-1):
    tmpy=list(k*h+h*xg)
    tmpw=list(h*wg)
    yirho+=tmpy
    wirho+=tmpw
yirho=array(yirho)
wirho=array(wirho)
nintrho=len(wirho)
print "nintrho=",nintrho
yiz=[]
wiz=[]
for k in range(ndofz-1):
    tmpy=list((k-N)*h+h*xg)
    tmpw=list(h*wg)
    yiz+=tmpy
    wiz+=tmpw
yiz=array(yiz)
wiz=array(wiz)
nintz=len(wiz)
#print "yiz=",yiz
print "nintz=",nintz
nint2=nintrho*nintz
print "nint2=",nint2
t1=time()
q=0
###
uq=[]
vq=[]
rq=[]
for i in range(nintrho):
    for j in range(nintz):
        uq.append(yirho[i])
        vq.append(yiz[j])
        rq.append(wirho[i]*wiz[j])
        q=q+1
uq=array(uq)
vq=array(vq)
rq=array(rq)
t2=time()
print "time␣to␣create␣2d␣int␣grid=",t2-t1
fi_atyi=zeros([ndof21,nint2])
t1=time()
tx=zeros(nintrho)
ty=zeros(nintz)
for k in range(ndof21):
    ik=k % ndofrho1
    jk=k // ndofrho1
    for i in range(nintrho):
        tx[i]=pi*(yirho[i]/h-(ik-N))
    for j in range(nintz):
        ty[j]=pi*(yiz[j]/h-(jk-N))
    stx=array([sinc(t) for t in tx])
    sty=array([sinc(t) for t in ty])
    tmp=outer(stx,sty)
    tmp.shape=nint2
    fi_atyi[k]=tmp.copy()
t2=time()
print "time␣to␣evaluate␣all␣2d␣basis␣functions␣at␣all␣integration␣points=",t2-t1
t1=time()
vi_atyi=array([xx*wf(xx,yy)*V(xx,yy) for (xx,yy) in zip(uq,vq)])
d=vi_atyi*rq
t2=time()
D2=kron(identity(ndofz),D)
print "time␣to␣evaluate␣potential␣at␣all␣int␣points=",t2-t1
t1=time()
tmpmat=dot(D2.T,d*fi_atyi).T
```

```
vmat=dot(dot(D2.T,fi_atyi),tmpmat)
t2=time()
print "time␣to␣evaluate␣product␣of␣three␣matrices␣giving␣vmat=",t2-t1
# code for umatrho
## code kmatrho
t1=time()
wf_atyi=array([xx*wf(xx,yy) for (xx,yy) in zip(uq,vq)])
d=wf_atyi*rq
t2=time()
print "time␣to␣evaluate␣weight␣function␣at␣all␣int␣points=",t2-t1
t1=time()
tmpmat=dot(D2.T,d*fi_atyi).T
umat=dot(dot(D2.T,fi_atyi),tmpmat)
t2=time()
print "time␣to␣evaluate␣product␣of␣three␣matrices␣giving␣umat=",t2-t1
##########################
drhofi_atyi=zeros([ndof21,nint2])
t1=time()
tx=zeros(nintrho)
ty=zeros(nintz)
for k in range(ndof21):
    ik=k % ndofrho1
    jk=k // ndofrho1
    for i in range(nintrho):
        tx[i]=pi*(yirho[i]/h-(ik-N))
    for j in range(nintz):
        ty[j]=pi*(yiz[j]/h-(jk-N))
    dstx=array([dsinc(t) for t in tx])
    sty=array([sinc(t) for t in ty])
    tmp=outer(dstx,sty)
    tmp.shape=nint2
    drhofi_atyi[k]=tmp.copy()
t2=time()
print "time␣to␣evaluate␣all␣2d␣basis␣functions␣for␣kinetic␣energy␣in␣rho␣at␣all␣integration␣points=",t2-t1
dzfi_atyi=zeros([ndof21,nint2])
t1=time()
tx=zeros(nintrho)
ty=zeros(nintz)
for k in range(ndof21):
    ik=k % ndofrho1
    jk=k // ndofrho1
    for i in range(nintrho):
        tx[i]=pi*(yirho[i]/h-(ik-N))
    for j in range(nintz):
        ty[j]=pi*(yiz[j]/h-(jk-N))
    stx=array([sinc(t) for t in tx])
    dsty=array([dsinc(t) for t in ty])
    tmp=outer(stx,dsty)
    tmp.shape=nint2
    dzfi_atyi[k]=tmp.copy()
t2=time()
print "time␣to␣evaluate␣all␣2d␣basis␣functions␣for␣kinetic␣energy␣in␣z␣at␣all␣integration␣points=",t2-t1

t1=time()
################
tmpmat=dot(D2.T,d*drhofi_atyi).T
kmat_rho=dot(dot(D2.T,drhofi_atyi),tmpmat)
t2=time()
print "time␣to␣evaluate␣product␣of␣three␣matrices␣giving␣kmat_rho=",t2-t1
t1=time()
################
tmpmat=dot(D2.T,d*dzfi_atyi).T
kmat_z=dot(dot(D2.T,dzfi_atyi),tmpmat)
t2=time()
print "time␣to␣evaluate␣product␣of␣three␣matrices␣giving␣kmat_z=",t2-t1
kmattot=kmat_rho+kmat_z
print "time␣to␣calculate␣matrix␣for␣kinetic␣energy=",t2-t1
hmat=kmattot+vmat
neig=hmat.shape[0]
print "neig=",neig
t1=time()
evals,evecs=eigh(hmat, umat)
```

```
t2=time()
print "time␣to␣solve␣eigenvalue␣problem=",t2-t1
evals.sort()
print evals[:4]
WF="""
# now plot wavefunction of GS
ev0=evecs[:,0]
def waveFun(rho,z):
    fiofrz=zeros(ndof21)
    for k in range(ndof21):
        fiofrz[k]=f(rho,z,k)
    fi1=dot(D2.T,fiofrz)
    return dot(ev0,fi1)
pf=open("plot.dat","w")
rhoi=linspace(0.0,2.0,41)
zi=linspace(0.,2.,41)
for rho in rhoi:
    for z in zi:
        #anaf=1.0/pi**0.75*exp(-0.5*(rho*rho+z*z))
        numf=waveFun(rho,z)
        print >> pf,z,rho,numf*sqrt(wf(rho,z))#,numf-anaf
    print >> pf
pf.close()
zi1d=linspace(0.,2,1001)
pf1=open("plot1.dat","w")
for z in zi1d:
    print >> pf1,z,waveFun(0.0,z)
pf1.close()"""
```

6.5 Least Square Fits for H_2^+

LeastSquareFitH2.py

```
#!/usr/bin/env python3
from numpy import *
from  scipy import optimize
import pylab
lsq=optimize.least_squares
import sys
###

# Here you need to insert the code that reads the file and calculates the arrays xi and yi
E0=-2.2052684289898
from numpy import *
import pylab
import sys
FitFile=sys.argv[1]
data=loadtxt(FitFile)
#Delta E= a*(1/n)^b
#log(Delta E)= log a+b*log(1/n)
xi=log(1./data[:,0])
yi=log(data[:,1]-E0)
xmin=xi[-1]-0.1
#xmax=xi[0]+0.1
#pylab.xlim(xmin,xmax)
#pylab.plot(xi,yi,"x")
#pylab.xlabel(r'$log(1/n)$')
#pylab.ylabel(r'$log(E_n-E_{\rm gs})$')
#pylab.savefig("PlotEofNu8.png",dpi=600)

# Here the definition of xi and yi includes the array
###
nx=len(xi)
def fun(x):
    r=zeros(nx)
    for i in range(nx):
        r[i]=yi[i]-(x[0]*xi[i]+x[1])
    return r
tmp=lsq(fun,[5.0,0.5][:])
a,b=tmp["x"]
print (a,b)
def f(x):
    return a*x+b
xmin=xi[-1]-0.1
xmax=xi[0]+0.1
pylab.xlim(xmin,xmax)
xp=linspace(xmin,xmax,401)
fp=f(xp)
pylab.plot(xp,fp,"-")
pylab.plot(xi,yi,"x")
pylab.xlabel(r'$\log(1/n)$')
pylab.ylabel(r'$\log␣\Delta␣E_n$')
pylab.savefig("PlotEofNfit8.png",dpi=600)

##########
#Here last 4 lines of your python code with the logarithmus must go
#########
```