Solution of Schrödinger equation for Morse potential - computation of eigenvalues and eigenvectors in coordinate and momentum representation

Alessandro Cuoghi, Matteo Quinzi, Giacomo Rizzi

June 9, 2021

1 Summary

The computation of eigenvalues and eigenvectors of the mono dimensional Schrödinger equation in presence of the Morse potential is performed both in coordinate and momentum representation.

In the first case, the program works with mainly three parameters: the interval length L, the number M of wanted eigenvalues and the number N of sampling points. Hamiltonian diagonalization is then performed through DSTEVR routine from LAPACK library.

In the second case, an additional parameter d represents the basis dimension of the plane waves, which are linearly combined to build the wanted eigenvectors. FFTW library is used to compute the Fourier transform of the Morse potential and ZHEEVX routine from LAPACK library evaluates the Hamiltonian spectrum.

2 Introduction

The aim of this project is to find eigenvalues and eigenvectors of a quantum particle subject to a monodimensional Morse potential.
By applying operator methods Schrödinger equation for Morse potential can be solved, i.e, eigenvalues and eigenvectos can be found. However, since it is not trivial, we will assume that eigenenegies are unknown throughout the computation. Therefore, this program allows us to analyse the stability of computational results through convergence tests, but not their validity.

Morse potential is an interatomic interaction model that simulates the potential energy of a diatomic molecule. This model is a far better approximation than the harmonic one because it accounts for **bond breaking** with unbound states that are allowed to exist and it includes the effects of **anharmonicity** of real interatomic bonds.
Moreover, Morse potential yields vibrational eigenstates with the property of having a not zero transition dipole moment for transitions whose difference between vibrational state index is $\Delta\nu = \pm 1, \pm 2, \pm 3$ *etc*. Hence, there is a not zero probability transition between vibrational ground states ($\nu = 0$) and the second excited states ($\nu = 2$), which is an experimentally observed overtone.

Historically, Morse potential has been generalized to the Morse/Long-Range (MLR), which contains theoretical corrections to the long range form. MLR has heavily contributed to represents and predict experimental results in spectroscopy of several diatomical molecules, such as N_2 and Ca_2.

Finally computational methods used in the study of a Morse potential can be easily generalized to anharmonic potentials which are proportional to some power of the independent variable, since they affect the values on the Hamiltonian diagonal without changing its properties.

3 System definition and algorithms description

Considering the Morse potential the system's Hamiltonian in 1D coordinates representation is given by

$$\mathcal{H} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + D_e(1 - e^{-\alpha(x-x_0)})^2 - D_e, \tag{1}$$

where D_e is the absolute value of the energy in the minimum of the Morse potential, i.e, the well depth, known as dissociacion energy, while α is responsible for the width of the well. By expanding Morse potential around the equilibrium bond distance x_0 to the second derivative is possible to show the relationship between D_e and a, which are connected through the force constant (stiffness) k_e:

$$\alpha = \sqrt{\frac{k_e}{2D_e}}. \tag{2}$$

The atomic scale is chosen in order to ensure a proper computational implementation with better precision. In particular:

$$\begin{aligned} \hbar &\leftarrow 1 \\ m &\leftarrow 1 \\ De &\leftarrow \frac{De}{E_h} \\ \alpha &\leftarrow \frac{\alpha}{a_0} \end{aligned}$$

Where E_h is the Hartee energy[1] and a_0 is the bohr radius. Therefore stationary time-independent Schrödinger equation can be rewritten as follows:

$$\mathcal{H}\phi(x) = E\phi(x) \tag{3}$$

$$-\frac{1}{2}\frac{d^2\phi(x)}{dx^2} + [D_e(1 - e^{-\alpha(x-x_0)})^2 - D_e]\phi(x) = E\phi(x). \tag{4}$$

3.1 Coordinate representation

Even though eq.(4) can not be solved analytically, a numerical derivation of the eigenvalues and eigenvectors is possible by implementing a computational method that rewrites this equation using matrices.

In order for the wavefunctions to account for bond breaking, they must be defined on the whole real axis and to be normalized they must approach zero when $x \to \pm\infty$. To implement a computational method **a limited interval** $[0, L]$ **must be chosen**, forcing the wavefunctions to zero at

[1] $E_h = \frac{\hbar}{m_e a_0^2} = 27.21138$ eV.

interval endpoints. The physical analog of this situation is to have infinite potential barriers at the interval boundaries. The more the wavefunctions approach zero on the boundaries, the better this approximation works.

The chosen interval is then **discretized in N inner points**, obtaining N+1 subintervals of width h. Boundary points are excluded as it is assumed that each wavefunction is null on the boundaries.

$$x_j = +hj \quad , \quad j = 1, N \tag{5}$$

This produces also a discretization of the wavefunctions as well as the Morse potential V, as both are sampled on the $\{x_j\}$ points:

$$\phi_j := \phi(x_j) \quad , \quad j = 1, N. \tag{6}$$
$$V_j := V(x_j) \quad , \quad j = 1, N. \tag{7}$$

Finally the second derivative action on ϕ is approximated by the **finite difference** formula:

$$\frac{d^2\phi(x)}{dx^2}\Big|_{x=x_j} \to -\frac{\phi_{j-1} - 2\phi_j + \phi_{j+1}}{h^2}. \tag{8}$$

Applying the discretization on (4), a **linear system of N eigenvalues equations** is obtained with unknowns $\{\phi_j\}_j$:

$$-\frac{1}{h^2}\phi_{j-1} + \Big(\frac{2}{h^2} + V_j\Big)\phi_j - \frac{1}{h^2}\phi_{j+1} = 2E\phi_j \quad , \quad j = 1, N \tag{9}$$

As ϕ_j is linked to ϕ_{j-1} and ϕ_{j+1} for every j, the system of N equations can be rewritten in matrix format, where the Hamiltonian (1) is described by a tridiagonal $(N \times N)$ matrix $\hat{H}$ with elements d_j on the main diagonal and e_k on the secondary diagonals, while ϕ become a column vector.

$$\begin{aligned} d_j &= \frac{2}{h^2} + V_j^2 \quad , \quad j = 1, N. \\ e_k &= -\frac{1}{h^2} \quad , \quad k = 1, N-1. \end{aligned} \tag{10}$$

$$\phi = (\phi_1, \ldots, \phi_N)^T. \tag{11}$$

Thus, (9) becomes

$$\hat{H}\phi = 2E\phi \tag{12}$$

Having obtained an eigenvalues equation in matrix format, eigenvalues and eigenstates can be calculated through **matrix $\hat{H}$ diagonalization**. Care must be taken with the resulting eigenvalues, as they are twice the eigenenergies E_j of the harmonic oscillator and the discretization of the interval prevents the computation of more than N eigenvalues and eigenstates.

The matrix $\hat{H}$ is:

- real

- symmetric
- tridiagonal

The diagonalization proceeds using algorithms and routines provided by the LAPACK (Linear Algebra PACKage) library.

To reduce computational costs and improve accuracy, DSTEVR routine is chosen, as this routine is optimized to diagonalize **symmetric, tridiagonal** matrices whose elements are **double precision floating point numbers**. Whenever possible this routine relies on a Relatively Robust Representation to compute the eigenspectrum and eigenvectors, avoiding unnecessary use of Gram-Schmidt representation.

Moreover DSTEVR allows to compute only selected eigenvalues, enabling to fix the number M of wanted eigenvalues and to increase the number of points on the grid, obtaining more accuracy without prohibitive costs.

3.2 Reciprocal space

In general, any functions that is either periodical or bounded can be decomposed in Fourier's series. Therefore, since the eigenfunctions $\phi(x)$ approach zero as the $x \to 0$ and $x \to \infty$, they can be properly represented on an appropriate basis $\{u(x)\}$, which satifies both orthonormality and completeness conditions. Thus,

$$\phi(x) = \sum_k c_k u_k(x), \tag{13}$$

where c_k are the weighting coefficient, that are given by:

$$c_k = \int_\Omega u_k^*(x)\phi(x)dx, \tag{14}$$

where Ω represents the integration interval. For numerical purposes $\Omega = [0, L]$ is a bounded interval $[0, L]$ and $\phi(x)$ is decomposed in a basis of finite dimension d. If the condition of completeness is met, Parseval identity must hold, hence,

$$\int_\Omega |\phi^2(x)|dx = \sum_k |c_k^2|. \tag{15}$$

In quantum mechanics the preservation of the norm is guaranteed if the Hamiltonian is hermitian. Therefore, the squared coefficient summation shall be conserved throughout the computation. This condition provides reciprocal space with an invariant. In particular, since we are dealing with probability density wavefunctions, the sum of the squared coefficients has to be equal to one.

Rewriting (13) through (4) and multiplying by

$$\int_\Omega u_{k'}^*$$

it is easily obtained:

$$\sum_k c_k \Big[\int_\Omega u_{k'}^*(x) \Big(-\frac{d^2\phi(x)}{dx^2} \Big) u_k(x)dx + \int_\Omega u_{k'}^*(x)V(x)u_k(x)dx - 2E\int_\Omega u_{k'}^*(x)u_k(x)dx \Big] = 0. \tag{16}$$

The first and the last term of this equation can be rewritten introducing the Dirac's delta distribution, which follows from the orthogonality property of the basis functions, whereas the second integral can be efficiently computed by using the plane wave basis,

$$u_k(x) = \frac{e^{ikx}}{\sqrt{L}},$$

where k is the wave vector, defined as:

$$k = \frac{2j\pi}{L} \quad , \quad j = 1, d \tag{17}$$

By doing so the second integral becomes the Fourier's transform of the potential $V(x)$, denoted as $v(k-k')$, which can be computed through FFTW library (version 3.3.9). On account of its speed and portability on both C and Fortran interfaces, the plane waves basis is effective and convenient in terms of computational costs.

Hence,

$$\sum_k c_k \left[k^2 \delta_{k,k'} + v(k'-k) - 2E\delta k, k' \right] = 0 \tag{18}$$

which can be addressed with matrix formulation:

$$\hat{H} \cdot \hat{c} = 2E\hat{c}, \tag{19}$$

where $\hat{c}$ is the coefficient column vector and $\hat{H}$ is the system's Hamiltonian that is defined as follows:

$$\hat{H} \equiv \begin{cases} v(k-k') & \text{if } k > k' \\ k^2 + v(0) & \text{if } k = k' \\ v(k'-k) & \text{if } k < k' \end{cases} \tag{20}$$

Eigenvalues equation (19) can be solved numerically by diagonalizing the Hamiltonian through *LAPACK* library routines and algorithms. The matrix $\hat{H}$ is:

- complex
- hermitian
- sparse

Eventually, the routine ZHEEVX is chosen. Indeed, it is optimized to compute *selected* eigenvalues and eigenvectors of matrices that are in posses of the above described features and whose components are pairs of double precision floating point numbers.

4 Code

The program is fully written in Fortran (1990), although the graphics is performed through Python library "Matplotlib". The program is based on a main code, which is called `Morse`, responsible for calling other five modules. First of all, the main program calls the module `read_var` that reads and

allocates the input parameters from the file "data_sheet.dat". This file also contains information about the representation the program has to work with through the parameter **task**, which can be either *a* or *b*. In the first case, the eigenproblem is solved in coordinate representation by using the module `task_a`. In the second case, in momentum representation by calling `task_b`.

This program is capable of comparing the solution of eigenvalue equation in presence of the Morse potential with the more approximated case of the harmonic potential. Therefore, an additional parameter is given to the main program in order to invoke the module `potential`, that generates the potential array. Three different potentials are available: Morse, harmonic oscillator with the parabola's vertex in the origin and harmonic oscillator shifted horizontally and downward by respectively x_0 and D_e.

A further module, `error` is arbitrary used to conduct numerical experiment that analyse the eugenvalues convergences as a function of their index and the width of the interval **L**.

In the following sections we shall introduce separately each of the part of the program.

4.1 Morse

Initially, the five modules are called.

```
PROGRAM main

USE read_var     ! read from "data_sheet.dat" and allocate the input paramaters
USE potential    ! defines the potential function
USE task_a       ! perform the computation in the coordinates space
USE task_b       ! perform the computation in the reciprocal space
USE error        ! conduct a series of convergence tests to identify an efficient set
    of input parameters.
```

Listing 1: `Morse` program calling modules.

As mentioned above, variables are allocated by calling the module `read_var`, except for the arrays **x** and **V**, which are declared in `Morse`. x is a partition of evenly spaced points and V is one of the three potentials provided by the module `potential`, depending on the value of the input parameter **pot**. In addition, three other parameters useful in convergence analysis, **toll**, **maxIndex** and **maxPotPerc**, are declared in `Morse`.

```
REAL(KIND=8), ALLOCATABLE :: x(:)    ! Interval [a,b] equapartition
REAL(KIND=8), ALLOCATABLE  :: V(:)    ! Potential V(x)
REAL(KIND=8) :: toll ! Tolerance of the convergence tests
INTEGER :: maxInd, maxPotPerc  !Error analysis parameters
```

Listing 2: x, V and convergence analysis parameters declaration in `Morse`.

```
ALLOCATE(x(N), V(N))
x(:) = (/ (a + (b-a) * (i*1d0) / (N-1), i = 0, N-1) /)   ! discrete interval [a, b]
IF (V_str == 'M') V = MORSE(x,N)                ! Morse potential V(x)
IF (V_str == 'H') V = HARMONIC(x,N)             ! harmonic potential V(x)
IF (V_str == 'X') V = SHIFTED_HARMONIC(x,N)     ! harmonic potential shifted to overlap
    with Morse potential
```

Listing 3: Block of commands in `Morse` that gives value to x and V.

At this point, either the subruotine SOLVE_EIGH_X from task_a or SOLVE_EIGH_K from task_b are called, to respectively perform the computation in real or reciprocal space.

Optionally, the parameters toll, maxIndex and maxPotPerc are given value and the routine ERROR_ANALYSIS, which perform the convergence analysis, is called:

```
CALL ERROR_ANALYSIS(toll,"X",maxPotPerc,maxInd)
```

The first parameter is the tolerance of the convergence, the second is a character that can be either "X" or "K", depending on whether the analysis has to be performed in real or reciprocal space. MaxPotPerc is an integer that defines the interval length $\bar{L}$ that satifies the condition:

$$\bar{L} \ : V(\bar{L}) = -MaxPotPerc * D_e$$

Finally, maxInd is the highest eigenvalue index for which the convergence analysis is performed.

4.2 Potential and read_var

These modules do not have particular technical aspects that deserve to be reported in depths, thus, they are synthetically treated together. `potential` is a collection of three functions that returns the three different potentials, whereas `read_var` opens the file "data_sheet.dat" and read the value of the parameters listed below:

- **a,b**: bounds of the partition interval.
- **N**: number of sampling points of the partition.
- **V _str**: if $"M"$ then V is the Morse potential, if $"H"$ is the Harmonic, else if $"X"$ is the shifted Harmonic potential.
- **k_el, De, alpha**: elastic constant, well depth and force constant of the Morse potential.
- **d**: number of plane wave that are used to represent the eigenfunction in truncated Fourier's series.
- **task**: if $"a"$ then the computation is executed in coordinate representation, else if $"b"$ in momentum representation.
- **jobz, range**: input parameters of LAPACK libraries. If $job = "N"$, only eigenvalues computed, else if $job = "V"$ then both eigenvalues and eigenvectors are computed. *range* establish the intervals of the needed eigenvalues. If range is $"A"$ all eigevalues are computed. If it is $"V"$ all eigenvalues in the half-open interval (VL,VU] are found else if it is $"I"$ the IL-th through IU-th eigenvalues are found.
- **IL,IU,VL,VU**: input parameters of LAPACK routines. They define the interval of interest of eigenvalues.
- **store**: logical parameter; T (true) if eigenvalues and eigenvectors has to be stored in general text file, F (false) otherwise.

4.3 Task_a

In this module eigenvalues and, optionally, eigenvectors, are found in real space by solving Eq.(12) through matrix diagonalization. The diagonal and off-diagonal vectors that define this Hamiltonian of eq.(10) are denoted dg and e and are computed (see listing: 4) by the subroutine HAMILTONIAN.

```
h = DABS(x(2) - x(1))
dg(:) = 1d0/h**2 + V(:)
e(:) = -1d0/(2d0 * h**2)
```

Listing 4: computation of diagonal and subdiagonal elements of the Hamiltonian in real space by subroutine HAMILTONIAN.

The core subroutine of the module that calls the Lapack's routine DSTEVR is SOLVE_EIGH_X.

```
SUBROUTINE SOLVE_EIGH_X(x, V, N, err_eva, index)
```

Listing 5: Call to SOLVE_EIGH_X in module `task_a`.

Within this subroutine HAMILTONIAN is called to compute the diagonal and sub-diagonal vectors, which are then passed to DSTEVR:

```

CALL DSTEVR(JOBZ, RANGE, N, dg, e, VL, VU, IL, IU, ABSTOL, M, &
                W, Z, N, ISUPPZ, WORK, 20*N, IWORK, 10*N, INFO)
```

Listing 6: Call to Lapack's routine DSTEVR.

ABSTOL is the absolute error tolerance for the eigenvalues. Eigenvalues are computed more accurately when ABSTOL is set to twice the underflow threshold, i.e, 2*DLAMCH('S') instead of zero. **M** is the number of eigenvalues and eigenvectors that has to be returned, which are respectively stored in the arrays **W** and the columns of the matrix **Z**. In addition to that, the CPU time needed to execute DSTEVR is measured and printed out. Eventually, if **STORE** is "true", the results of the computation are stored in general text files.

Parameters **err_eva** and **index** are optional and needed in case of convergence tests; if index is present, the routine calls DSTEVR and computes err_eva with IL and IU both identical to index.

Subroutine DSTEVR computes by default normalized eigenfunctions, thus, their physical meaning is preserved. The program prints out the norm of the first three eigenfunctions to ensure that the computation has worked well.

4.4 Task_b

This module is the analogue of `task_a` in momentum representation; it has the same structure of the previous one, except for an additional function, called FFT, which computes the Fourier's Transform potential of V(x) to give v(k).

`read_var` is a dependency of this module, so that all the parameters are globally shared.

As previously mentioned, the Fourier's Transform is implemented via the library FFTW. This library is written in C language, and in general it is not possible to call C functions directly from Fortran. Nonetheless, FFTW libraries include special **wrapper** functions that allow this operation, so that an efficient and fast algorithms can be used to compute the Fourier's Tranform.

Firstly, the function FFT include the FFTW interface definition by using the command:

```
include 'fftw3.f'
```

Some important variables are then declared. In particolar, **FFT** and **IFT** are respectively the direct Fourier's transform and the inverse Fourier's transform. Even though IFT is not strictly necessary

for the computation, it is written on a general text file, so that it can be plotted with the original potential V to check whether they are equal. This prevents from numerical error that could be caused by an incorrect use the FFTW library. For this purpose only the real part is necessary, thus, IFT does not need a to be a complex type.

```
COMPLEX(KIND=8):: FFT(N/2+1) ! Fourier transfrom of V
COMPLEX(KIND=8) :: FFT2(N/2+1)
INTEGER(KIND=8) :: plan
REAL(KIND=8) :: k(N/2+1), IFT(N)
REAL(KIND=8) :: L
```

Listing 7: Variable declaration in FFT function.

In this framework the wave vector **k** is defined as follows:

$$k \leftarrow k' - k \tag{21}$$

where the notation of Eq.(16) has been used, so that k is the *difference* of two wave vectors. In addition to that, an integer called **plan**, which is intended to be an object that contains all the data that FFTW needs to compute the FFT, is declared. Indeed, FFTW library does not have a stardard way to solve a problem, but instead it creates a specifical plan, depending on the type and dimensions of the arrays it has to deal with. Each arrays with the same type and dimension is executed with same plan.

```
CALL DFFTW_PLAN_DFT_R2C_1D(plan, N, V, FFT, FFTW_ESTIMATE)
CALL DFFTW_EXECUTE_DFT_R2C(plan, V, FFT)
CALL DFFTW_DESTROY_PLAN(plan)
```

Listing 8: Core block of commands of function FFT

The first command create the appropriate plan for a DFT (Discrete Fourier Transform) of a real array into a complex one. The flag FFTW_ESTIMATE specifies that, instead of actual measurements of different algorithms, a simple heuristic is used to pick a (probably sub-optimal) plan quickly. With this flag, the input/output arrays are not overwritten during planning.

The second line executes the previously created plan, so that FFT is overwritten with the DFT of the potential. Eventually, the plan is destroyed and a new one can be created to compute IFT.

Care must be taken with the result of the computation: in order FFT to be normalized it has to be divided by the number of sampling points N.

FFT2 is then overwritten with the FFT, in order to ensure the preservation of the output array of the function. It is necessary to create a new plat to antitraform FFT2 into IFT, i.e, from a complex vector to a real one.

Eventually, the wave vector arrays is computed and results are printed out.

```
L = DABS(x(N) - x(1))
k(:) = (/ ((2d0 * i) * pi / L, i=0,N/2) /)
```

Listing 9: Interval length and wave vector definition in FFT, a function of task_b.

A subroutine called HAMILTONIAN, in analogy with the *task_a* builds up the Hamiltonian consistently with the definition (20):

```
H(:,:) = 0.d0
    DO i = 1,d
        H(i,i) = V(1) + 0.5 * k(i)**2
        H(i,i+1:) = V(2:d+1-i)
    END DO
```

Listing 10: Blocks of command to build up the Hamiltonian in reciprocal space.

Finally, the subroutine SOLVE_EIGH_K, which is the analogous of SOLVE_EIGH_X in reciprocal space, is executed. However, in this case the Hamiltonian of the system is sparse, complex and hermitian, thus, the routine ZHEEVX is chosen.

Eq.(15) states that the squared sum of Fourier's coefficient of the eigenfunctions has to be equal to one, regardless of the computation parameters. Therefore, the program prints out the squared sum of the weighting coefficient of the first three eigenfunctions.

4.5 Error

The aim of this module is to perform the convergence analysis as a function of both eigenvalue index and the width of the interval L. In the following sections we will denote $\bar{d}$ and $\bar{N}$ as the number of plane waves and sampling points that satisfies the convergence condition defined in Eq.(24).

First of all, three parameters are initialised:

- **x_step**: number of points increase per iteration.
- **k_step**: number of plane waves increase per iteration.
- **maxit**: maximum number of iteration.

The core routine of the module is N_d_conv, which iterates over N and d. Here is the declaration:

```
SUBROUTINE N_d_CONV(toll, L, index, space, N_out)
```

Listing 11: Call to N_ d_conv subroutine.

The first four parameters represent: the tolerance of the convergence analysis, the length of the interval, the eigenvalue index, and eventually the space the subroutine has to perform the computation, i.e, whether it has to compute the $\bar{d}$ (space = "K") or the $\bar{N}$ (space = "X") that satisfies convergence condition. $\bar{N}$ or $\bar{d}$ is eventually assigned the output parameter **N_out**.

In listing (12) we report the while loop that performs the convergence analysis in coordinate representation; in each iteration the integer "counter" is increased of a factor one, multiplied by x_step and assigned to N_points. The function SINGLE_EVA calls SOLVE_EIGH_X and returns the single eigenvalue computed with the couple of parameters $\{L, N\_points\}$. When the condition

$$toll > abs\_diff$$

is met, N_points is stored in the output variable N_out.

```
DO WHILE (toll<=abs_diff)
    counter = counter + 1
    IF (counter=maxit) THEN
```

```
        N = -1
        RETURN
    END IF
    N_points = x_step*counter
    eva_1 = eva_2
    eva_2 = SINGLE_EVA(N_points, L, index, "X")
                    abs_diff = abs(eva_1-eva_2)
END DO
N = N_points
```

Listing 12: While loop that performs convergence analysis in real space.

A similar while loop is performed to compute the $\bar{d}$ that satisfies convergence condition, although k_step is used instead of x_step.

Since the Morse potential is generally a steep curve around its minimum, it is reasonable to assume that taking an interval that covers an large percentage of the potential, the eigenvalues will not be negatively affected.

Therefore, we impose:

$$D_e(1 - e^{-\alpha(x-x_0)})^2 - D_e = -\frac{per\_cent}{100} D_e \tag{22}$$

where *per_cent* is the percentage of the potential that is truncated. Thus, through trivial algebra steps, we get that:

$$L = x = x_0 - \frac{1}{\alpha} ln(1 - \sqrt{1 - \frac{per\_cent}{100}}). \tag{23}$$

The subroutine WIDTH_CONVERGENCE takes in input the percentage per_cent, computes L and, eventually, it returns the $\bar{N}$ or $\bar{d}$ by calling the routine L_d_conv.

```
IF (mansion == "P") THEN
    L = x0 - (1/alpha)*DLOG(1.d0-sqrt(1-percent/100.d0))
ELSEIF (mansion == "D") THEN
    L = b
END IF
IF (space == "X") THEN
    CALL N_d_CONV(toll, L, index, N, "X")
ELSE IF (space =="K") THEN
    CALL N_d_CONV(toll, L, index, N, "K")
END IF
```

The parameter **mansion** determines whether L as to be computed from eq.(23) or overwritten with the value of b. This is useful in numerical experiments to evaluate the gain that is obtained with "smart" values of L in comparison with randomly chosen values. The parameter space is passed to N_d_conv and selects the representation where convergence analysis is carried out.

Finally, the routine ERROR_ANALYSIS calls width_convergence for different parameters: in both real and reciprocal space the convergence is computed as a function of the eigenvalue index as well as the percentage of the truncated potential that defines L.

5 Numerical experiments

5.1 Convergence analysis

Since we have assumed that true eigenvalues are unknown, numerically computed eigenvalues can only be compared among themselves through convergence tests. Input paramaters of the program can be classified in two different groups, regardless of the chosen representation. The first group defines the shape of the Morse potential, hence, the physical properties of the diatomic molecule bonding, and is composed of the well depths D_e, the strength constant a and the equilibrium position x_0. On the other hand, the second group is about the quantity of information that is given to the program. More specifically, the number of points N, the number d of plane waves and eventually the width of the interval L are part of this group.

For each set of first group parameters, the aim of the convergence test is to provide values of L that minimise the value of $\bar{N}$ and $\bar{d}$ that reach the following condition:

$$toll \geq |eigenvalue(\bar{N}, \bar{d}, L) - eigenvalue(N, d, L)|, \tag{24}$$

where $\{\bar{N}, \bar{d}\}$ are the smallest values of sampling points and plane waves that satisfies convergence condition for a certain L.

At fixed parameters of first group, the higher is the index of the eigenvalue, the higher are $\bar{N}$ and $\bar{d}$. A proof of this statement can be easily carried out by keeping the interval length L fixed, while computing different eigenvalues with increasing N or d up to convergence condition. This numerical experiment is performed with following input parameters:

- $D_e = 50$
- $\alpha = 0.5$
- $x_0 = 2.5$
- tol = 10^{-7}

In parallel with the input value of $L = 20$, the numerical cost for convergence is computed with L that realises the condition: V(L) = - 10%D_e. From Eq.(23) L = 8.439.

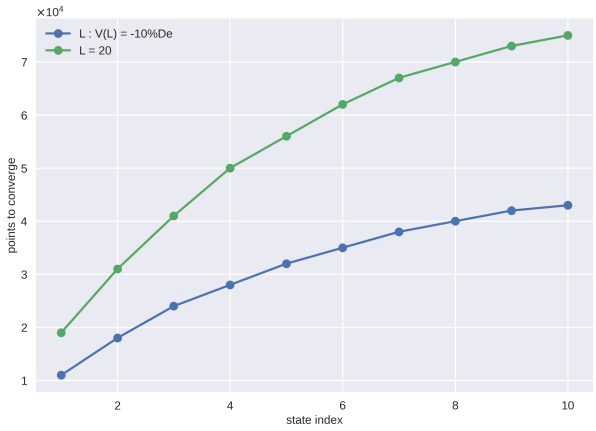

Figure 1: Convergence analysis in real space. The green curve refers to $L = 20$, while the blue to $L = 8.439$

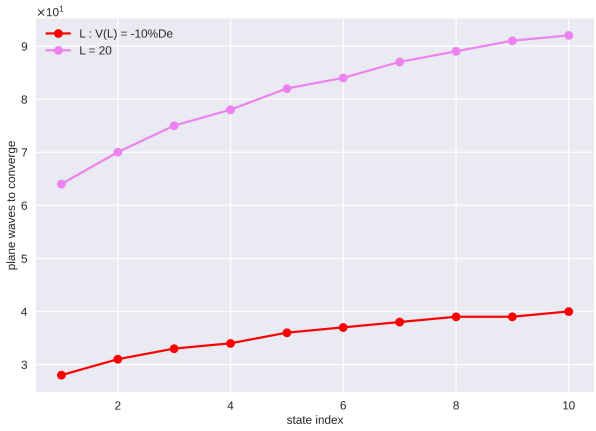

Figure 2: Convergence analysis in reciprocal space. The purple curve refers to $L = 20$, while the red to $L = 8.439$

This experiment proves that the highest index eigenvalue requires the highest number of $\bar{N}$ and $\bar{d}$. This phenomenon can be interpreted as a consequence of the half-bounded shape that is assumed by the Morse potential when approaching to zero. Indeed, in this region of low negative energy, the limited interval [0,L] forces the eigenfunctions to zero in a non negligible way, due to the asymmetry of the Morse potential.

Furthermore, this experiment highlights that $\bar{L} = 8.349$, guarantees the convergence condition with far less points and plane waves than it does $L = 20$. With regard to real space, this result establishes that if L is smaller, the density of sampling points will be higher in a meaningful region on space, whereas, if L is greater, sampling points around the minimum will be "moved" to the region where the Morse potential approaches steadily to zero, providing few information. Reduction of computational cost in reciprocal space is linked to the definition of the wave vector k, which becomes larger for smaller value of L, thus, the plane waves are more peaked around the minimum region and are more effective in decomposing the eigenfunctions.

In order to analyse in depths the convergence around a meaningful interval L, we select a set of truncated potential percentage, thus, a set of values of L that satisfies the Eq.(23), and we perform a numerical experiment with the same first group parameter of the previous one.

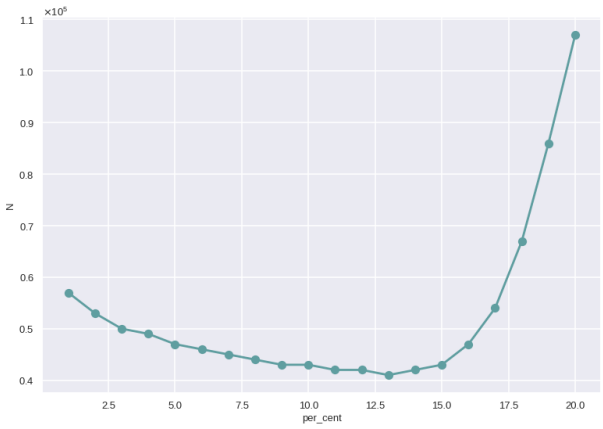

Figure 3: Points to converge in real space as a function of the chosen percentage of the potential that satisfies the condition defined in Eq.(23).

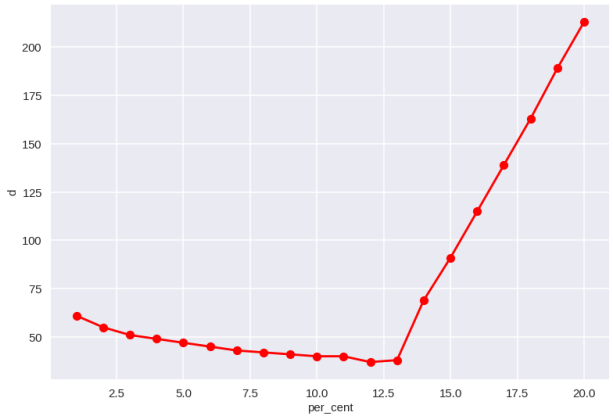

Figure 4: Plane waves to converge as a function of the percentage of the potential that satifies the condition defined in Eq.(23).

The result of the experiments remarks the importance of chosing an appropriate value of L to get stable results. For this particular input parameters there is a minimum, both in real and reciprocal space, given by the value of L obtained through a value of $per\_cent = 13\%$ in the first case and $per\_cent = 12\%$ in the second one. For greater value of percentage, i.e, smaller value of L, $\bar{N}$ and $\bar{d}$ tend to diverge because too much information are lost due to the truncation.

5.2 CPU time

Module "error" provides $\bar{N}$ and $\bar{d}$ that meet convergence condition. Computational costs to execute Lapack's routines with these values of $\bar{N}$ and $\bar{d}$ can be compared as they are associated with the same tolerance. In Fig.(3) and Fig.(4) we have pointed out an interesting behaviour of $\bar{N}$ and $\bar{d}$ as a function of L. We proceed further by analysing the CPU time to execute Lapack's libraries

as a function of L. For each value of L, the computation is performed with the $\bar{N}$ and $\bar{d}$ associated with interval length. The same parameters of the first experiment are used. Results are shown in Fig.(5). We first take a look at the curve trends: reasonably, they follow those of Fig.(3) and Fig.(4). Secondly, the CPU time needed to execute DSTEVR is much higher than that to execute ZHEEVX. At best, they differ of two orders of magnitude. This result implies that the execution of the complex matrix ZHEEVX is much quicker than the execution of DSTEVR, i.e, it is quicker to use a sparse low-dimensional matrix than a tridiagonal high-dimensional matrix. It is worth noting that this result does not imply that operating in reciprocal space is more efficient than it is in real space; generally, even though ZHEEVX is quicker, the routine (SOLVE_EIGH_K) is not, due to the computation of the discrete Fourier's Transform [2].

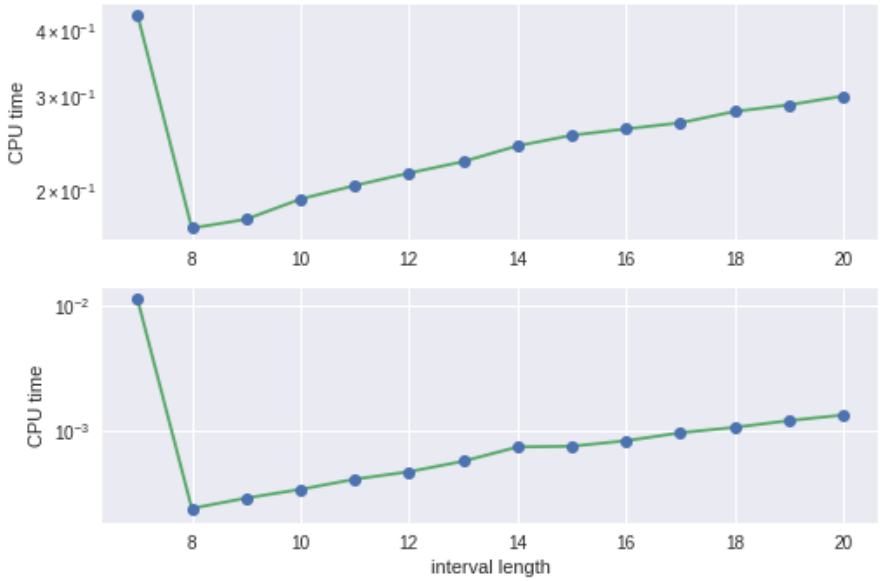

Figure 5: In the upper graph it is shown the CPU needed to execute DSTEVR as a function of L. In the lower graph the CPU needed to execute ZHEEVX. Each value of time was obtained by averaging 30 measurements of time.

5.3 Basis of harmonic oscillator eigenfunctions

Despite being very general and effective (thanks to the FFTW), plane waves basis is not the most appropriate one for this particular physical system. Therefore, in this numerical experiment we aim at testing the basis of Schrödinger equation eigenfunctions in presence of harmonic potential.

The program that performs the following experiment is reported in appendix(A). Owing to its resemblance to a shifted harmonic potential around the minimum, Morse potential can be expressed through Taylor's series truncated to the second order term. By doing so, Morse potential is approximated to the following harmonic potential:

$$V(x) = D_e\alpha^2(x - x_0)^2 - D_e, \tag{25}$$

Schrödinger equation can be readily rewritten through matrix diagonalization by proceeding as in Eq.(9) and eigenvalues and eigenvectors are found by Lapack's routine DSTEVR.

[2]The numerical experiment was conducted with a AMD® Ryzen 7 4700u processor- 8Gb RAM.

In Fig.(6) eigenfunctions in presence of Morse or harmonic potential are compared. While the seconds are evenly spaced, the firsts are not; approaching negatively to zero energy, they become more and more denser in order to continuously transform into free states. In addition to that, the asymmetric shape of the Morse potential results in a rigid shift toward right of all the eigenfunctions.

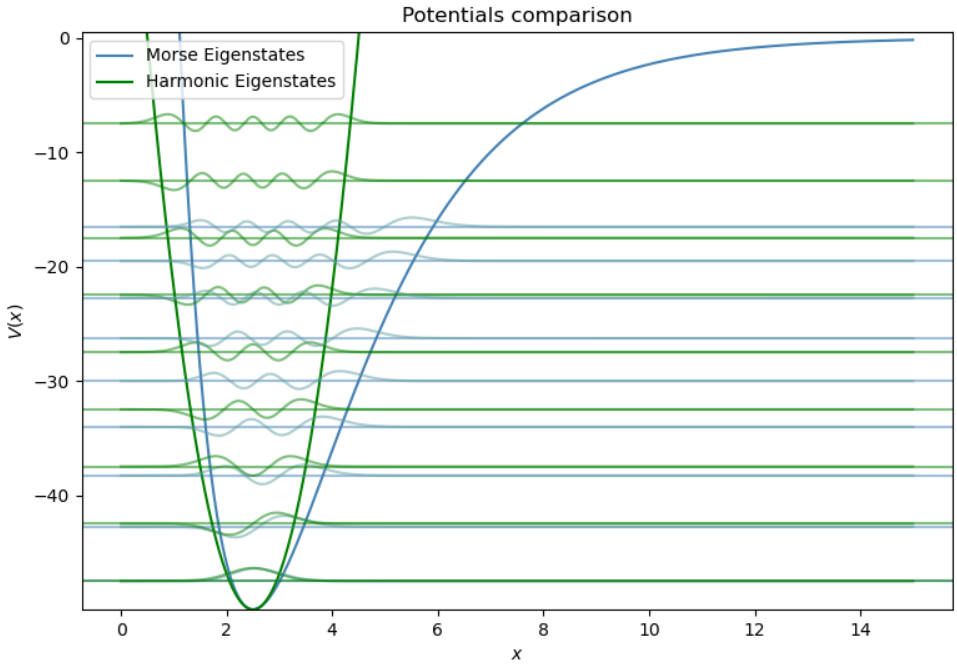

Figure 6: Eigenfunctions of the Schrödinger in presence of harmonic oscillator potential (green curve) or Morse potential (blue curve).

At this point, it is necessary to solve Eq.(16) by using eigenstates of harmonic potential defined in Eq.(25) instead of the plane waves.

In order to simplify the computation of the matrix elements of the Hamiltonian, we add and remove the quantum harmonic potential, whose eigenvalues are analytically known. Therefore, we define an effective potential V_{eff} by subtracting potential (25) to the Morse one.

The effective potential, which is plotted in Fig.(7), is a repulsive potential up to the equilibrium points, where it is changes the curvature and tends to zero. After the equilibrium point, it becomes rapidly attractive.

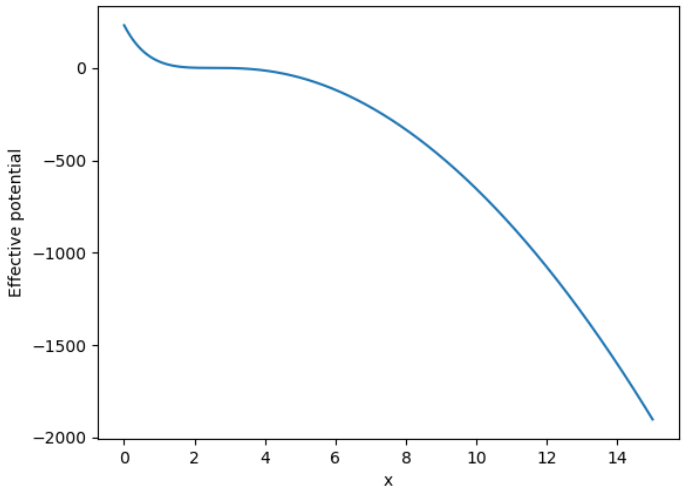

Figure 7: V_{eff} as a function of the x position.

In order to compute the matrix element of $V_{eff} - V_{harmonic}$ we implement a numerical integration through trapezoidal formula.

Finally, Schrödinger Eq.(1) is solved through matrix diagonalization and eigenvalues are found. They are compared to those obtained through real space strategy. Furthermore, the eigenenergies of the quantum harmonic oscillator are appended to Fig.(8) in order to highlight the effect of the effective potential V_{eff}. This computation was performed with the following set of parameters:

- $D_e = 50$
- $\alpha = 0.5$
- $x_0 = 2.5$
- $L = 15$
- $N = 5000$
- $M = 10$

Where M is the dimension of the harmonic oscillator basis. The results shows that, despite the small dimension of the basis, the eigenvalues are comparable with those calculated through direct space procedure. In addition, the two firsts eigenvalues are almost equal to those calculated for the Quantum Harmonic oscillator. For higher states, the Morse eigenvalues tend to flatten under the effect of the attractive effective potential.

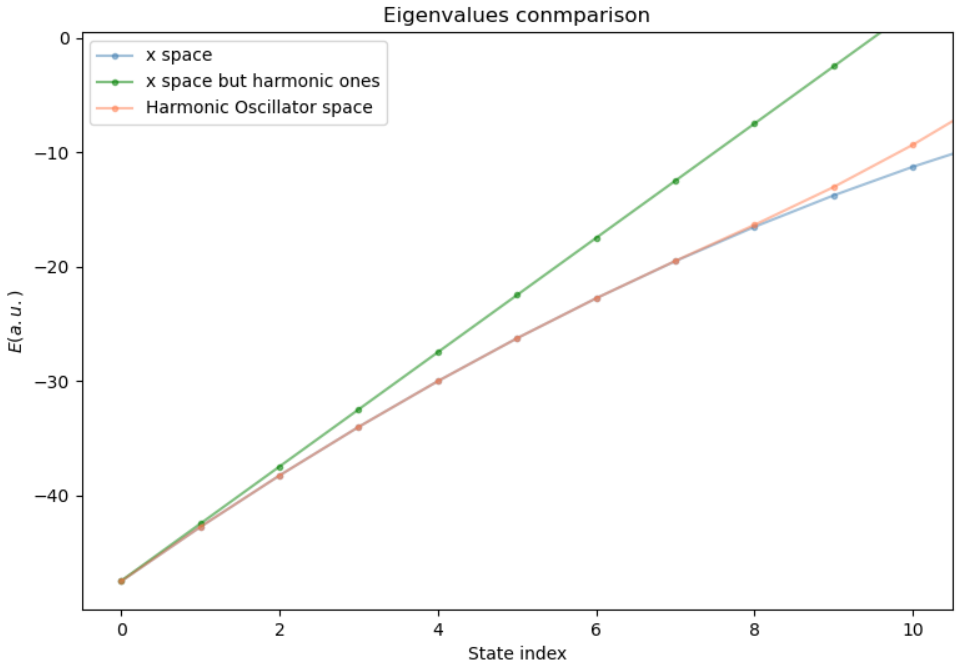

Figure 8: The curve labelled x-space represents the eigenvalues of the Schrödinger equation in presence of the Morse potential computed through the diagonalization of the matrix in real space. The curve labelled Harmonic Oscillator space represents the eigenvalues obtained through decomposition of the eigenfunctions on the basis of the harmonic oscillator eigenfunctions. Finally, the green curve represents the evenly spaced eigenvalues of the Quantum Harmonic Oscillator.

6 Conclusion

We have defined the Morse potential shifted downward by the disassociation energy D_e in order to obtain negative eigenvalues (bound states). Indeed, for positive eigenvalues, the energy levels become denser and less stably to be computed, due to the fact that their eigenfunctions extends over ∞ and the error associated with the truncation of the interval becomes relevant. Therefore, numerical experiments have been performed with values of $D_e = 50 = 1360,57\ eV$ that are far beyond real values of disassociation energy for real molecules (see Table.1) but necessary to compute multiple negative eigenvalues.

A comparison of the computation conducted in coordinate and momentum representation shows that eigenvalues of the same index are very similar, in particular for the most bounded states, as it can be readily seen in Fig.(9).

Bond-dissociation energy at 298 K	
Diatomic Molecule	Disassociation energy (eV/bond)
C-C	3.6-3.9
H-H	4.52
O-H (in water)	5.15
O-H (in methanol)	4.5
C ≡ O	11.16
N ≡ N	9.79

Table 1: Disassociation energy for some molecules.

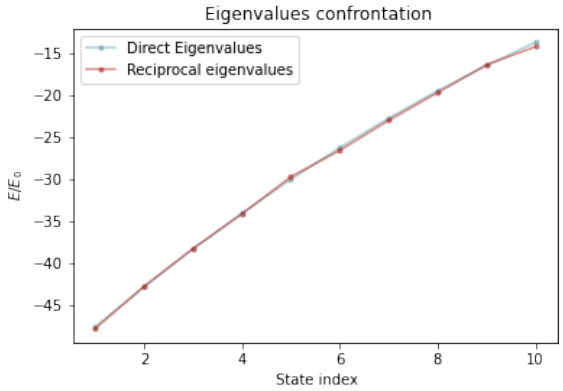

Figure 9: Eigenvalues comparison in real and reciprocal space.

In the section dedicated to numerical experiments(5.2) it has been shown that, generally, the CPU time necessary to execute ZHEEVX is much less than to execute DSTEVR. This result remarks the importance of the choice of a plane wave basis, which result in the Fourier's decomposition of the potential through the FFTW library. In addition to be a very efficient way to solve Schrödinger eq.(1), it is also extremely **general**. Indeed, any potential V that admits a Fourier's decomposition can be treated similarly.

Care must be taken with the interval length adopted both in real and reciprocal space computation. Indeed, too small values of L, i.e, values that truncate too much part of the Morse potential, can lead to unstable computation, whose result are strongly affected by the number of points N or the plane waves d.

For a given set of first group parameter, it is possible to find a L that minimises the computational cost to converge in terms of sampling points, i.e, in terms of Lapack's routine execution time. Indeed, a comparison between Fig.(3), Fig.(4) and Fig.(5) highlights that the the *cputime* is proportional to $\bar{N}$ and $\bar{d}$.

Finally, the basis of the harmonic oscillator eigenfunctions is extremely effective in converging to the eigenvalues of the eigenvalues of Schrödinger equation for Morse potential.

A APPENDIX

A.1 Basis of harmonic oscillator eigenfunctions

The program that solves Eq.(1) by decomposition of eigenfunction in the basis of the harmonic oscillator basis is explained in this section. The program is organized into a main program, called `waves`, which executes several routines that are provided by a module, called `eigen`. Although the aim of the program is to perform calculation in harmonic oscillator basis, it also computes both Morse potential and harmonic oscillator eigevalues and eigenfunctions to compare the results.

At first, x-partition, the Morse potential and the potential defined in Eq.(25) are created by calling three routines from `eigen`.

```
call linspace(0.d0, L, N, x)
call Morse(x, x_0, D_e, alpha, V_morse)
call m_to_h(x, x_0, D_e, alpha, V_harmonic)
```

Listing 13: Creation of x-partition, Morse and harmonic potential in `waves`.

At this point, two Hamiltonians are build up by the `eigen` routine Hamiltonian_x; one for the Morse and the other for the harmonic one. For both of them eigenvalues and eigenvectors are computed by the Lapack's library DSTEVR and rescaled, as this library does not compute orthonormal eigenvectors. Results are printed out for graphical purposes.

As eigenfunctions should preserve their physical meaning, the norm is computed and printed out through the call to the subroutine quadrature_1d, which implements the quadrature of a one dimensional array with trapezoidal formula.

As eigenfunctions of the harmonic oscillator are available, the matrix elements of the Hamiltonian are computed with the following commands:

```
M = size(eigenstates(1,:))
N = size(eigenstates(:,1))
allocate(temp_intg(N))
do i = 1,M
    do j=i,M
        ! define integral as <i| V_eff |j>
        temp_intg = (/ ( eigenstates(k,i) * V_eff(k) * eigenstates(k,j) , k=1,N)   /)
        call quadrature_1d(dx, temp_intg, H(i,j))
    end do
end do
! adding values on the diagonal
do i = 1,M
    H(i,i) = H(i,i) + eigenvalues(i)
end do
```

Listing 14: Computing the matrix elements of the hamiltonian in the basis of harmonic oscillator.

In the inner loop the integral, i.e, the matrix element $< i|H|j >$ is performed by the routine quadrature_1d. Subsequently, an element of the array eigenvalue is added to account of the harmonic potential combined with the kinetic energy term.

Finally, eigenenergies and eigenvectors of this hamiltonian are found by calling solve_hamiltonian_real, a subroutine of the module `eigen` that makes use of the lapack's

routine DSYEV, which computes all eigenvalues and, optionally, orthonormal eigenvectors of a real symmetric matrix.