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

Alessandro Cuoghi, Matteo Quinzi, Giacomo Rizzi

May 24, 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, while 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.
As it is not possible to solve Schrödinger equation in presence of the Morse potential analytically, true eigenvalues and eigenvectors are unknown. 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 an 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

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

where D_e represents the energy in the minimum of the Morse potential, i.e, the well depth, known as dissociacion energy, while a 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:

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

The atomic scale is chosen in order to ensure a proper computational implementation with better precision. Therefore stationary time-independent Schrödinger equation can be rewritten as follows:

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

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

3.1 Coordinate representation

As 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 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.

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

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

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. Thus,

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

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

$$c_k = \int_{-\infty}^{\infty} u_k^*(x)\phi(x)dx. \tag{14}$$

For numerical purposes, $\phi(x)$ is decomposed on a basis of finite dimension d. If the condition of completeness is met, Parseval identity must hold, hence,

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

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

$$\int_{-\infty}^{\infty} u_{k'}^*$$

it is easily obtained:

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

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:

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

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, thee 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:

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

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 The developed 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 *read_var* that reads and allocates the input parameters from the file "data_sheet.dat". In addition to that, this file 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 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 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; x is a partition of evenly spaced points and V is given value by calling one of the three functions of *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 parameter 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. label

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 in integer value that defines the interval length that satifies the condition:

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

, i.e, the percentage of potential that is truncated. 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* is deputed to access the file "data_sheet.dat" and read the value of the parameters, which are 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). 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 in $HAMILTONIAN$.

The core subroutine of the module that calls the routine $DSTEVR$ is $SOLVE\_EIGH\_X$.

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

Listing 5: input paramters of $SOLVE\_EIGH\_X$.

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: input paramters of $SOLVE\_EIGH\_X$.

Some of the routine parameters are already known from previous descriptions. *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 necessary for convergence tests; if *index* is present, the routine calls $DSTEVR$ and computes *err_eva* with IL and IU both identical to *index*.

Subroutine DSTEVR computed 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 is possible to use an efficient and fast algorithms to compute the Fourier's Tranform, making the choise of a plane wave basis very effective and general.

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 and compared with the potential V. 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. Here is the core block of commands of function FFT:

```
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)
```

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) /)
```

A subroutine called *HAMILTONIAN* builds up the Hamiltonian following the definition of 20 following the instructions:

```
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
```

Finally, the subroutine *SOLVE_EIGH_K* is the analogous of *SOLVE_EIGH_X* in reciprocal space, thus, its details are not reported. It must be noticed, though, that in this case the Hamiltonian of the system is neither tridiagonal nor real. Therefore, the routine ZHEEVX is appropriate.

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, as it was for *task_a*, 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 width of the interval *L*.

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 its declaration:

```
SUBROUTINE N_d_CONV(toll, L, index, mansion, N)
```

The first four parameters represent the tolerance of the convergence analysis, the length of the interval, the eigenvalue index, and eventually the mansion of the subroutine, i.e, whether it has to compute the number of plane waves d (mansion = "K") or the number of points N(mansion = "X") that satisfies convergence condition. The value of N or d is eventually assigned the output parameter N.

We report the while loop that performs the convergence analysis in coordinate representation; In each iteration a 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 L and N_points. When the condition $toll > abs\_diff$ is met, N_points is stored in the output variable N.

```
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
```

A similar while loop is performed to compute the number of plane waves 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 high 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 number of points (or plane waves) to converge 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 b. This is useful in numerical experiments to evaluate the gain that is obtained with "smart" values of L in comparison with randomly large values. The parameter space is passed to *N_d_conv* and selects the representation that perform the computation.

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

As it has already been mentioned, the Schrödinger equation (4) can not be analytically solved. This implies that the true eigenvalues are not known, thus, 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 and 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 is to provide values of L that reduce the numerical costs to reach the following condition:

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

where N, d, L and $\{\bar{N}, \bar{d}, \bar{L}\}$ are in general different.

At fixed parameters of first group, the higher is the index of the eigenvalue, the higher is the numerical cost to reach convergence condition. A proof of this statement can be easily carried out by keeping the interval length L fixed, while computing different eigenvalues with increasing sampling points N or basis dimension d up to convergence condition. We perform a numerical experimenti with following input parameters:

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

In parellel with the input value of $L = b = 20$, the numerical cost for convergence is computed with $\bar{L}$ that realises the condition: V($\bar{L}$) = - 10%D_e. From eq.(23), $L = 8.439$.

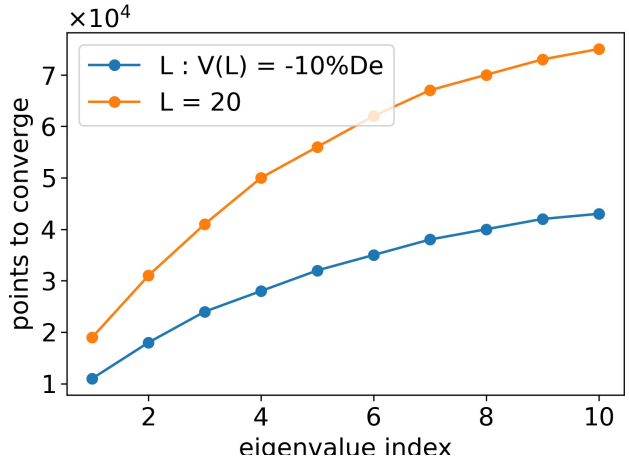

Figure 1: Convergence analysis in real space. The orange curve refers to $L = 20$, while the blue to $L = 8.439$

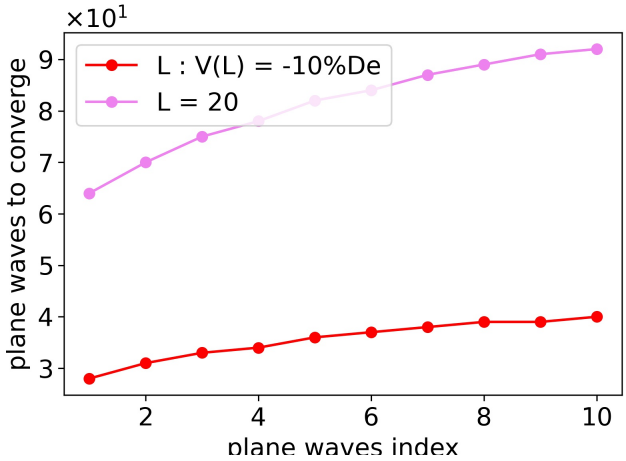

Figure 2: Convergence analysis in reciprocal space. The orange curve refers to $L = 20$, while the blue to $L = 8.439$

This experiment proves that the eigenvalue of highest index requires the highest number of N and d to converge. 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, eigenfunctions become denser and their probability densities tend to overlap, making more difficult for eigenvalues to be stably computed and reach convergence condition.

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 sampling a "smart" interval that covers the greatest part of the potential (90%) is more efficient than sampling a larger interval. In other terms, if L is smaller, the density of sampling points is higher in a meaningful region on space, whereas, if L is greater, sampling points around the minimum are "moved" to the region where the Morse potential approaches steadily to zero, providing few information. Reduction of computational cost in reciprocal space also evident and is linked to the definition of the wave vector k, which becomes larger for smaller value of L, thus, the plane 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 L that satisfies the eq. (23), and we perform a numerical experiment with the same first group parameter of the previous one.

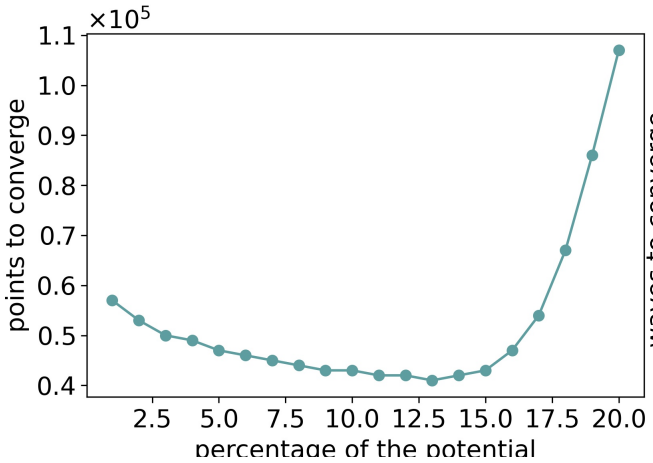

Figure 3: Points to converge in real space as a function of the chosen percentage of the potential that satisfies the condition.

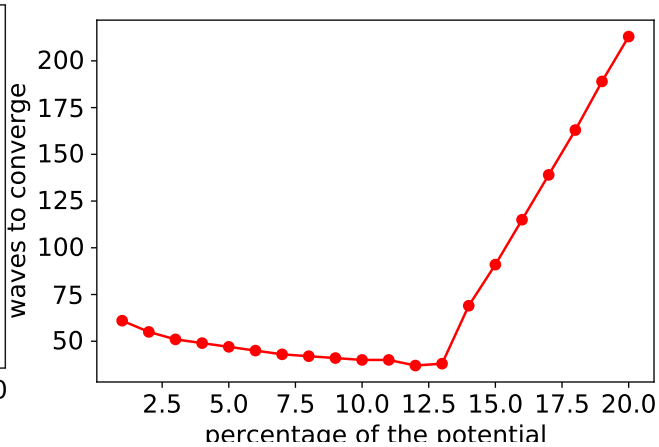

Figure 4: Plane waves to converge as a function of the percentage of

The result of the experiments remarks the importance of chosing an appropriate value of L to get stable results. For this particular input parameter there is a minimum, both in real and reciprocal space, given by the value of L that covers the 87.5 % of the potential. For greater value of percentage, i.e, smaller value of L, the number or points and the number of plane waves tend to diverge because too much information are lost due to the truncation.

5.2 CPU time

The results of the first numerical experiment provide the number of sampling points N and the number of plane waves d that satifies convergence condition for $L = 20$ and for $L = 8.349$. For these values of N and d we known that the program perform equally in terms of convergence, thus, it is meaningful to compare the CPU that is required to execute the computation in coordinate and momentum representation through the lapack libraries, respectively $DSTEVR$ and $ZHEEVX$ [1]

Results are synthetically reported in Tab.(1) and Tab.(2) and show that the $ZHEEVX$ requires much less time than $DSTEVR$. Indeed, even though the first deals with a sparse matrix, the matrix dimension is smaller than real space matrix. The behaviour is not isolated to this particular set of parameters, but instead a more general phenomenon, thus, computation in reciprocal space is more efficient than in real space.

CPU time: real space		
Interval length	Sampling points	CPU time
20	75000	0.239 ± 0.001
15	63000	0.136 ± 0.001
8.349	43000	0.1364 ± 0.0004

Table 1: My Table

[1] The numerical experiment has been executed with a AMD® Ryzen 7 4700u processor- 8Gb RAM.

CPU time: reciprocal space		
Interval length	Plane waves	CPU time
20	92	0.00113 ± 0.00001
15	70	0.00054 ± 0.00001
8.349	40	0.0002 ± 0.00001

Table 2: My Table

5.3 Basis of harmonic oscillator eigenfunctions

6 Conclusion

We have defined the Morse potential shifted downward by the disassociation energy D_e in order to obtain negative eigenvalues that reflect 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 $D_e = 50eV$ that, although it is far beyond real values of disassociation energy for real molecules (see Table.3), is necessary to compute multiple negative eigenvalues.

Molecule binding energy	
Molecule	Bond energy (eV)
C-C	6.3
H-H	4.5
O-H	4.4
Na-Cl	4.3
Fe-O	4.0

Table 3: My Table

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.(5).

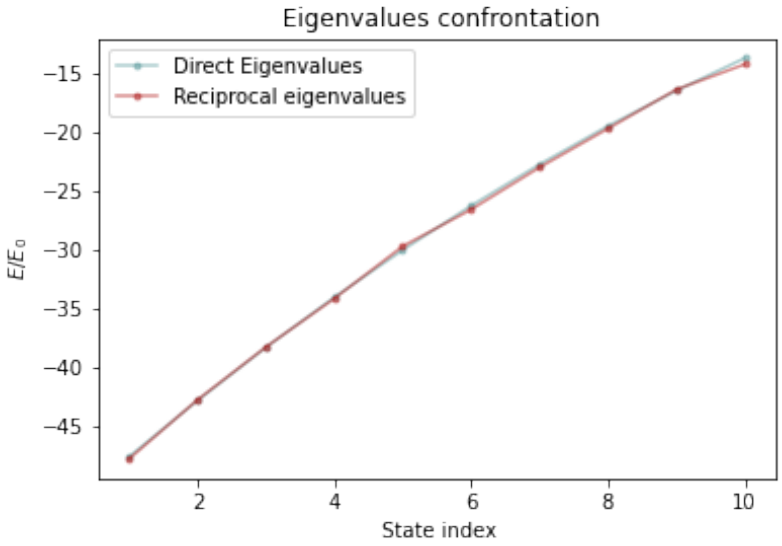

Figure 5: Caption

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.

-convergenza strettamente legata ad L, in particolare per valori di L troppo piccoli i risultati divergono e diventano molto poco stabili

A APPENDIX

B CODES