**Lecture notes**

# Numerical Methods in Quantum Mechanics

**Corso di Laurea Magistrale in Fisica**

**Interateneo Trieste – Udine**

**Anno accademico 2012/2013**

**Paolo Giannozzi**
*University of Udine*

Contains software and material written by

**Furio Ercolessi**[1] **and Stefano de Gironcoli**[2]
[1]*Formerly at University of Udine*
[2]*SISSA - Trieste*

Last modified May 23, 2013

# Contents

# Introduction

The aim of these lecture notes is to provide an introduction to methods and techniques used in the numerical solution of simple (non-relativistic) quantum-mechanical problems, with special emphasis on atomic and condensed-matter physics. The practical sessions are meant to be a sort of “computational laboratory”, introducing the basic ingredients used in the calculation of materials properties at a much larger scale. The latter is a very important field of today’s computational physics, due to its technological interest and potential applications.

The codes provided during the course are little more than templates. Students are expected to analyze them, to run them under various conditions, to examine their behavior as a function of input data, and most important, to interpret their output from a physical point of view. The students will be asked to extend or modify those codes, by adding or modifying some functionalities.

For further insight on the theory of Quantum Mechanics, many excellent textbooks are available (e.g. Griffiths, Schiff, or the ever-green Dirac and Landau). For further insight on the properly computational aspects of this course, we refer to the specialized texts quotes in the Bibliography section, and in particular to the book of Thijssen.

## 0.1 About Software

This course assumes some basic knowledge of how to write and execute simple programs, and how to plot their results. All that is needed is a fortran or C compiler and some visualization software. The target machine is a PC running Linux, but other operating systems can be used as well (including Mac OS-X and Windows), as long as the mentioned software is installed and working, and if you know how to use it in oractise.

### 0.1.1 Compilers

In order to run a code written in any programming language, we must first translate it into machine language, i.e. a language that the computer can understand. The translation is done by an *interpreter* or by a *compiler*: the former translates and immediately executes each instruction, the latter takes the file, produces the so-called *object code* that together with other object codes and with libraries is finally assembled into an *executable* file. Python, Java (or at an higher level, Matlab, Mathematica) are examples of “interpreted” language. Fortran, C, C++ are “compiled” languages.

Our codes are written in Fortran 90. This is a sophisticated and complex language offering dynamical memory management, arrays operations (e.g. matrix-vector products), modular and object-based structure. Fortran 90 however can be as efficient as Fortran 77 and maintains a wide compatibility with existing Fortran 77 codes. It is worth mentioning that the first applications of computers to physics go back to well before the birth of modern computer languages like C++, python, or even C: there is still a large number of codes and libraries written in Fortran 77 (or even Fortran 66!) widely used in physics.

Fortran 90 (or even Fortran 77, in this respect) is not a well known language. There are however many available resources (see for instance the web page mentioned in the bibliography section) and the codes themselves are very simple and make little usage of advanced language features. In any case, there are no objections if a student prefers to use a more widespread language like C. A version of all codes in C is also available (no warranty about the quality of the C code in terms of elegance and good coding practice).

In all cases, we need a C or Fortran 90 compiler. In PCs running Linux, the C compiler `gcc` is basically part of the operating system and is always present. Recent versions of `gcc` also include a Fortran compiler, called `gfortran`. If this is absent, or it is not easy to install it, one can download the free and rather reliable compiler, `g95`[1]. It is possible to install on Mac OS-X and on Windows either `gcc` with `gfortran` or `g95`.

### 0.1.2 Visualization Tools

Visualization of data produced by the codes (wave functions, charge densities, various other quantities) has a central role in the analysis and understanding of the results. Code `gnuplot` can be used to make two-dimensional or three-dimensional plots of data or of analytical expressions. `gnuplot` is open-source software, available for all operating systems and usually found pre-installed on Linux PCs. An introduction to `gnuplot`, with many links to more resources, can be found here: `http://www.gnuplot.info/help.html`.

Another software that can be used is `xmgrace`[2]. This is also open-source and highly portable, has a graphical user interface and thus it is easier to use than `gnuplot`, whose syntax is not always easy to remember.

### 0.1.3 Mathematical Libraries

The usage of efficient mathematical libraries is crucial in “serious” calculations. Some of the codes use routines from the BLAS[3] (Basic Linear Algebra Subprograms) library and from LAPACK[4] (Linear Algebra PACKage). The latter

[1]http://www.g95.org
[2]http://plasma-gate.weizmann.ac.il/Grace
[3]http://www.netlib.org/blas
[4]http://www.netlib.org/lapack

is an important and well-known library for all kinds of linear algebra operations: solution of linear systems, eigenvalue problems, etc. LAPACK calls BLAS routines for all CPU-intensive calculations. The latter are available in highly optimized form for many different architectures.

BLAS and LAPACK routines are written in Fortran 77. BLAS and LAPACK are often available in many operating systems and can be linked directly by the compiler by adding `-llapack -lblas`. In the case of C compiler, it may be needed to add an underscore (_) in the calling program, as in: `dsyev_`, `dgemm_`. This is due to different C-Fortran conventions for the naming of "symbols" (i.e. compiled routines). Note that the C compiler may also need `-lm` to link general mathematical libraries (i.e. operations like the square root).

### 0.1.4 Pitfalls in C-Fortran interlanguage calls

In addition to the above-mentioned potential mismatches between C and Fortran naming conventions, there are a few more pitfalls one has to be aware of when Fortran 77 routines are called by C (or vice versa).

- Fortran passes *pointers* to subroutines and functions; C passes *values*. In order to call a Fortran routine from C, all C variables appearing in the call must be either pointers or arrays.

- Indices of vectors and arrays start from 0 in C, from 1 in Fortran (unless differently specified in array declaration or allocation).

- Matrices in C are stored in memory row-wise, that is: `a[i][j+1]` follows `a[i][j]` in memory. In Fortran, they are stored column-wise (the other way round!): `a(i+1,j)` follows `a(i,j)` in memory.

An additional problem is that C does not provide run-time allocatable matrices like Fortran does, but only fixed-dimension matrices and arrays of pointers. The former are impractical, the latter are not usable as arguments to pass to Fortran. It would be possible, using either non-standard C syntax, or using C++ and the `new` command, to define dynamically allocated matrices similar to those used in Fortran. We have preferred for our simple C codes to "simulate" Fortran-style matrices (i.e. stored in memory column-wise) by mapping them onto one-dimensional C vectors.

We remark that Fortran 90 has a more advanced way of passing arrays to subroutines using "array descriptors". The codes used in this course however do not make use of this possibility but use the old-style Fortran 77 way of passing arrays via pointers.

## 0.2 Bibliography

J. M. Thijssen, *Computational Physics*, Cambridge University Press, Cambridge, 1999. A second edition has appeared:
`http://www.cambridge.org/gb/knowledge/isbn/item1171410`.

F. J. Vesely, *Computational Physics - An Introduction: Second Edition*, Kluwer, 2001. Also see the author's web page:
`http://www.ap.univie.ac.at/users/Franz.Vesely/cp0102/serious.html`, containing parts of the accompanying material.

S. E. Koonin e D. C. Meredith, *Computational physics - Fortran Version*, Addison-Wesley, 1990. See also Dawn Meredith's web page:
`http://pubpages.unh.edu/%7Edawnm/`.

Timothy H. Kaiser, A quick introduction to advanced Fortran-90:
`http://www.sdsc.edu/%7Etkaiser/f90.html`.

# Chapter 1

# One-dimensional Schrödinger equation

In this chapter we will start from the harmonic oscillator to introduce a general numerical methodology to solve the one-dimensional, time-independent Schrödinger equation. The analytical solution of the harmonic oscillator will be first derived and described. A specific integration algorithm (Numerov) will be used. The extension of the numerical methodology to other, more general types of potentials does not present any special difficulty.

For a particle of mass $m$ under a potential $V(x)$, the one-dimensional, time-independent Schrödinger equation is given by:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x), \tag{1.1}$$

where $\psi(x)$ is the wave function, in general complex, and $\hbar$ is the Planck constant $h$ divided by $2\pi$. In the following we are focusing on the *discrete spectrum*: the set of isolated energy values for which Eq.(1.1) has normalizable solutions, localized in space.

## 1.1 The harmonic oscillator

The harmonic oscillator is a fundamental problem in classical dynamics as well as in quantum mechanics. It represents the simplest model system in which attractive forces are present and is an important paradigm for all kinds of vibrational phenomena. For instance, the vibrations around equilibrium positions of a system of interacting particles may be described, via an appropriate coordinate transformation, in terms of independent harmonic oscillators known as *normal vibrational modes.* The same holds in quantum mechanics. The study of the quantum oscillator allows a deeper understanding of quantization and of its effects and of wave functions of bound states.

In this chapter we will first remind the main results of the theory of the harmonic oscillator, then we will show how to set up a computer code that allows to numerically solve the Schrödinger equation for the harmonic oscillator. The resulting code can be easily modified and adapted to a different (not simply quadratic) interaction potential. This will allow to study problems that, unlike the harmonic oscillator, do not have a simple analytical solution.

### 1.1.1 Units

The Schrödinger equation for a one-dimensional harmonic oscillator is, in usual notations:

$$\frac{d^2\psi}{dx^2} = -\frac{2m}{\hbar^2}\left(E - \frac{1}{2}Kx^2\right)\psi(x) \tag{1.2}$$

where $K$ the force constant (the force on the mass being $F = -Kx$, proportional to the displacement $x$ and directed towards the origin). Classically such an oscillator has a frequency (angular frequency)

$$\omega = \sqrt{\frac{K}{m}}. \tag{1.3}$$

It is convenient to work in adimensional units. These are the units that will be used by the codes presented at the end of this chapter. Let us introduce adimensional variables $\xi$, defined as

$$\xi = \left(\frac{mK}{\hbar^2}\right)^{1/4} x = \left(\frac{m\omega}{\hbar}\right)^{1/2} x \tag{1.4}$$

(using Eq.(1.3) for $\omega$), and $\epsilon$, defined as

$$\varepsilon = \frac{E}{\hbar\omega}. \tag{1.5}$$

By inserting these variables into the Schrödinger equation, we find

$$\frac{d^2\psi}{d\xi^2} = -2\left(\varepsilon - \frac{\xi^2}{2}\right)\psi(\xi) \tag{1.6}$$

which is written in adimensional units.

### 1.1.2 Exact solution

One can easily verify that for large $\xi$ (such that $\varepsilon$ can be neglected) the solutions of Eq.(1.6) must have an asymptotic behavior like

$$\psi(\xi) \sim \xi^n e^{\pm\xi^2/2} \tag{1.7}$$

where $n$ is any finite value. The + sign in the exponent must however be discarded: it would give raise to diverging, non-physical solutions (in which the particle would tend to leave the $\xi = 0$ point, instead of being attracted towards it by the elastic force). It is thus convenient to extract the asymptotic behavior and assume

$$\psi(\xi) = H(\xi)e^{-\xi^2/2} \tag{1.8}$$

where $H(\xi)$ is a well-behaved function for large $\xi$ (i.e. the asymptotic behavior is determined by the second factor $e^{-\xi^2/2}$). In particular, $H(\xi)$ must not grow like $e^{\xi^2}$, or else we fall back into a undesirable non-physical solution.

Under the assumption of Eq.(1.8), Eq.(1.6) becomes an equation for $H(\xi)$:

$$H''(\xi) - 2\xi H'(\xi) + (2\varepsilon - 1)H(\xi) = 0. \tag{1.9}$$

It is immediate to notice that $\varepsilon_0 = 1/2$, $H_0(\xi) = 1$ is the simplest solution. This is the *ground state*, i.e. the lowest-energy solution, as will soon be clear.

In order to find all solutions, we expand $H(\xi)$ into a series (in principle an infinite one):

$$H(\xi) = \sum_{n=0}^{\infty} A_n \xi^n, \tag{1.10}$$

we derive the series to find $H'$ and $H''$, plug the results into Eq.(1.9) and regroup terms with the same power of $\xi$. We find an equation

$$\sum_{n=0}^{\infty} \left[(n+2)(n+1)A_{n+2} + (2\varepsilon - 2n - 1)A_n\right] \xi^n = 0 \tag{1.11}$$

that can be satisfied for any value of $\xi$ only if the coefficients of all the orders are zero:

$$(n+2)(n+1)A_{n+2} + (2\varepsilon - 2n - 1)A_n = 0. \tag{1.12}$$

Thus, once $A_0$ and $A_1$ are given, Eq.(1.12) allows to determine by recursion the solution under the form of a power series.

Let us assume that the series contain an infinite number of terms. For large $n$, the coefficient of the series behave like

$$\frac{A_{n+2}}{A_n} \to \frac{2}{n}, \quad \text{that is:} \quad A_{n+2} \sim \frac{1}{(n/2)!}. \tag{1.13}$$

Remembering that $\exp(\xi^2) = \sum_n \xi^{2n}/n!$, whose coefficient also behave as in Eq.(1.13), we see that recursion relation Eq.(1.12) between coefficients produces a function $H(\xi)$ that grows like $\exp(\xi^2)$, that is, produces unphysical diverging solutions.

The only way to prevent this from happening is to have in Eq.(1.12) all coefficients beyond a given $n$ vanish, so that the infinite series reduces to a finite-degree polynomial. This happens if and only if

$$\varepsilon = n + \frac{1}{2} \tag{1.14}$$

where $n$ is a non-negative integer.

Allowed energies for the harmonic oscillator are thus *quantized*:

$$E_n = \left(n + \frac{1}{2}\right)\hbar\omega \quad n = 0, 1, 2, \ldots \tag{1.15}$$

The corresponding polynomials $H_n(\xi)$ are known as *Hermite polynomials*. $H_n(\xi)$ is of degree $n$ in $\xi$, has $n$ nodes, is even $[H_n(-\xi) = H_n(\xi)]$ for even $n$, odd $[H_n(-\xi) = -H_n(\xi)]$ for odd $n$. Since $\mathrm{e}^{-\xi^2/2}$ is node-less and even, the complete wave function corresponding to the energy $E_n$:

$$\psi_n(\xi) = H_n(\xi)\mathrm{e}^{-\xi^2/2} \tag{1.16}$$

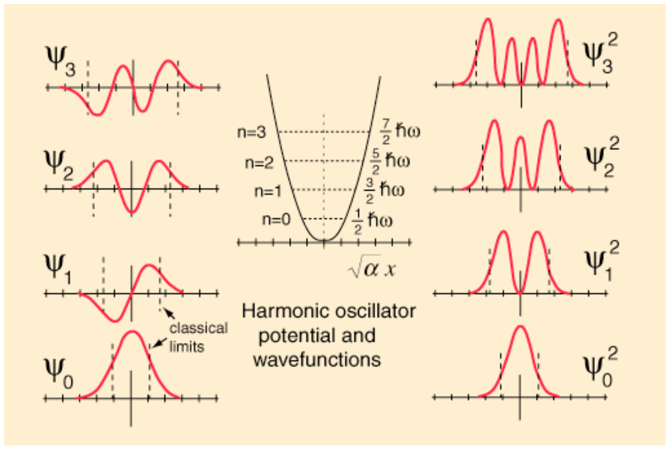

Figure 1.1: Wave functions and probability density for the quantum harmonic oscillator.

has $n$ nodes and the same parity as $n$. The fact that all solutions of the Schrödinger equation are either odd or even functions is a consequence of the symmetry of the potential: $V(-x) = V(x)$.

The lowest-order Hermite polynomials are

$$H_0(\xi) = 1, \quad H_1(\xi) = 2\xi, \quad H_2(\xi) = 4\xi^2 - 2, \quad H_3(\xi) = 8\xi^3 - 12\xi. \tag{1.17}$$

A graph of the corresponding wave functions and probability density is shown in fig. 1.1.

### 1.1.3 Comparison with classical probability density

The probability density for wave functions $\psi_n(x)$ of the harmonic oscillator have in general $n+1$ peaks, whose height increases while approaching the corresponding classical inversion points (i.e. points where $V(x) = E$).

These probability density can be compared to that of the classical harmonic oscillator, in which the mass moves according to $x(t) = x_0 \sin(\omega t)$. The probability $\rho(x)dx$ to find the mass between $x$ and $x+dx$ is proportional to the time needed to cross such a region, i.e. it is inversely proportional to the speed as a function of $x$:

$$\rho(x)dx \propto \frac{dx}{v(x)}. \tag{1.18}$$

Since $v(t) = x_0\omega\cos(\omega t) = \omega\sqrt{x_0^2 - x_0^2\sin^2(\omega t)}$, we have

$$\rho(x) \propto \frac{1}{\sqrt{x_0^2 - x^2}}. \tag{1.19}$$

This probability density has a minimum for $x = 0$, diverges at inversion points, is zero beyond inversion points.

The quantum probability density for the ground state is completely different: has a maximum for $x = 0$, decreases for increasing $x$. At the classical inversion point its value is still $\sim 60\%$ of the maximum value: the particle has a high probability to be in the classically forbidden region (for which $V(x) > E$).

In the limit of large quantum numbers (i.e. large values of the index $n$), the quantum density tends however to look similar to the quantum one, but it still displays the oscillatory behavior in the allowed region, typical for quantum systems.

## 1.2 Quantum mechanics and numerical codes: some observations

### 1.2.1 Quantization

A first aspect to be considered in the numerical solution of quantum problems is the presence of *quantization* of energy levels for bound states, such as for instance Eq.(1.15) for the harmonic oscillator. The acceptable energy values $E_n$ are not in general known *a priori*. Thus in the Schrödinger equation (1.1) the unknown is not just $\psi(x)$ but also $E$. For each allowed energy level, or *eigenvalue*, $E_n$, there will be a corresponding wave function, or *eigenfunction*, $\psi_n(x)$.

What happens if we try to solve the Schrödinger equation for an energy $E$ that does not correspond to an eigenvalue? In fact, a "solution" exists for *any* value of $E$. We have however seen while studying the harmonic oscillator that the quantization of energy originates from boundary conditions, requiring no unphysical divergence of the wave function in the forbidden regions. Thus, if $E$ is not an eigenvalue, we will observe a divergence of $\psi(x)$. Numerical codes searching for allowed energies must be able to recognize when the energy is not correct and search for a better energy, until it coincides – within numerical or predetermined accuracy – with an eigenvalue. The first code presented at the end of this chapter implements such a strategy.

### 1.2.2 A pitfall: pathological asymptotic behavior

An important aspect of quantum mechanics is the existence of "negative" kinetic energies: i.e., the wave function can be non zero (and thus the probability to find a particle can be finite) in regions for which $V(x) > E$, forbidden according to classical mechanics. Based on (1.1) and assuming the simple case in which $V$ is (or can be considered) constant, this means

$$\frac{d^2\psi}{dx^2} = k^2\psi(x) \tag{1.20}$$

where $k^2$ is a positive quantity. This in turns implies an exponential behavior, with both $\psi(x) \simeq \exp(kx)$ and $\psi(x) \simeq \exp(-kx)$ satisfying (1.20). As a rule only one of these two possibilities has a physical meaning: the one that gives raise to a wave function that *decreases* exponentially at large $|x|$.

It is very easy to distinguish between the "good" and the "bad" solution for a human. Numerical codes however are less good for such task: by their very nature, they accept both solutions, as long as they fulfill the equations. If even a tiny amount of the "bad" solution (due to numerical noise, for instance) is present, the integration algorithm will inexorably make it grow in the classically forbidden region. As the integration goes on, the "bad" solution will sooner or later dominate the "good" one and eventually produce crazy numbers (or crazy NaN's: Not a Number). Thus a nice-looking wave function in the classically allowed region, smoothly decaying in the classically forbidden region, may suddenly start to diverge beyond some limit, unless some wise strategy is employed to prevent it. The second code presented at the end of this chapter implements such a strategy.

## 1.3 Numerov's method

Let us consider now the numerical solution of the (time-independent) Schrödinger equation in one dimension. The basic assumption is that the equation can be *discretized*, i.e. written on a suitable finite grid of points, and *integrated*, i.e. solved, the solution being also given on the grid of points.

There are many big thick books on this subject, describing old and new methods, from the very simple to the very sophisticated, for all kinds of differential equations and all kinds of discretizations and integration algorithms. In the following, we will consider *Numerov's method* (named after Russian astronomer Boris Vasilyevich Numerov) as an example of a simple yet powerful and accurate algorithm. Numerov's method is useful to integrate second-order differential equations of the general form

$$\frac{d^2y}{dx^2} = -g(x)y(x) + s(x) \tag{1.21}$$

where $g(x)$ and $s(x)$ are known functions. Initial conditions for second-order differential equations are typically given as

$$y(x_0) = y_0, \quad y'(x_0) = y'_0. \tag{1.22}$$

The Schrödinger equation (1.1) has this form, with $g(x) \equiv \frac{2m}{\hbar^2}[E - V(x)]$ and $s(x) = 0$. We will see in the next chapter that also the radial Schrödinger equations in three dimensions for systems having spherical symmetry belongs to such class. Another important equation falling into this category is Poisson's equation of electromagnetism,

$$\frac{d^2\phi}{dx^2} = -4\pi\rho(x) \tag{1.23}$$

where $\rho(x)$ is the charge density. In this case $g(x) = 0$ and $s(x) = -4\pi\rho(x)$.

Let us consider a finite box containing the system: for instance, $-x_{\max} \le x \le x_{\max}$, with $x_{\max}$ large enough for our solutions to decay to negligibly small values. Let us divide our finite box into $N$ small intervals of equal size, $\Delta x$ wide. We call $x_i$ the points of the grid so obtained, $y_i = y(x_i)$ the values of the unknown function $y(x)$ on grid points. In the same way we indicate by $g_i$ and $s_i$ the values of the (known) functions $g(x)$ and $s(x)$ in the same grid points. In order to obtain a discretized version of the differential equation (i.e. to obtain an equation involving finite differences), we expand $y(x)$ into a Taylor series around a point $x_n$, up to fifth order:

$$\begin{aligned} y_{n-1} = \; & y_n - y'_n\Delta x + \tfrac{1}{2}y''_n(\Delta x)^2 - \tfrac{1}{6}y'''_n(\Delta x)^3 + \tfrac{1}{24}y''''_n(\Delta x)^4 - \tfrac{1}{120}y'''''_n(\Delta x)^5 \\ & + O[(\Delta x)^6] \\ y_{n+1} = \; & y_n + y'_n\Delta x + \tfrac{1}{2}y''_n(\Delta x)^2 + \tfrac{1}{6}y'''_n(\Delta x)^3 + \tfrac{1}{24}y''''_n(\Delta x)^4 + \tfrac{1}{120}y'''''_n(\Delta x)^5 \\ & + O[(\Delta x)^6]. \end{aligned} \tag{1.24}$$

If we sum the two equations, we obtain:

$$y_{n+1} + y_{n-1} = 2y_n + y''_n(\Delta x)^2 + \frac{1}{12}y''''_n(\Delta x)^4 + O[(\Delta x)^6]. \tag{1.25}$$

Eq.(1.21) tells us that

$$y''_n = -g_n y_n + s_n \equiv z_n. \tag{1.26}$$

The quantity $z_n$ above is introduced to simplify the notations. The following relation holds:

$$z_{n+1} + z_{n-1} = 2z_n + z''_n(\Delta x)^2 + O[(\Delta x)^4] \tag{1.27}$$

(this is the simple formula for discretized second derivative, that can be obtained in a straightforward way by Taylor expansion up to third order) and thus

$$y''''_n \equiv z''_n = \frac{z_{n+1} + z_{n-1} - 2z_n}{(\Delta x)^2} + O[(\Delta x)^2]. \tag{1.28}$$

By inserting back these results into Eq.(1.25) one finds

$$\begin{aligned} y_{n+1} \; & = 2y_n - y_{n-1} + (-g_n y_n + s_n)(\Delta x)^2 \\ & + \tfrac{1}{12}(-g_{n+1}y_{n+1} + s_{n+1} - g_{n-1}y_{n-1} + s_{n-1} + 2g_n y_n - 2s_n)(\Delta x)^2 \\ & + O[(\Delta x)^6] \end{aligned} \tag{1.29}$$

and finally the *Numerov's formula*

$$\begin{aligned} y_{n+1}\left[1 + g_{n+1}\tfrac{(\Delta x)^2}{12}\right] \; & = 2y_n\left[1 - 5g_n\tfrac{(\Delta x)^2}{12}\right] - y_{n-1}\left[1 + g_{n-1}\tfrac{(\Delta x)^2}{12}\right] \\ & + (s_{n+1} + 10s_n + s_{n-1})\tfrac{(\Delta x)^2}{12} + O[(\Delta x)^6] \end{aligned} \tag{1.30}$$

that allows to obtain $y_{n+1}$ starting from $y_n$ and $y_{n-1}$, and recursively the function in the entire box, as long as the value of the function is known in the first two points (note the difference with "traditional" initial conditions, Eq.(1.22), in which the value at one point and the derivative in the same point is specified). It is of course possible to integrate both in the direction of positive $x$ and in the direction of negative $x$. In the presence of inversion symmetry, it will be sufficient to integrate in just one direction.

In our case—Schrödinger equation—the $s_n$ terms are absent. It is convenient to introduce an auxiliary array $f_n$, defined as

$$f_n \equiv 1 + g_n\frac{(\Delta x)^2}{12}, \qquad \text{where} \qquad g_n = \frac{2m}{\hbar^2}[E - V(x_n)]. \tag{1.31}$$

Within such assumption Numerov's formula can be written as

$$y_{n+1} = \frac{(12 - 10f_n)y_n - f_{n-1}y_{n-1}}{f_{n+1}}. \tag{1.32}$$

### 1.3.1 Code: harmonic0

Code `harmonic0.f90`[1] (or `harmonic0.c`[2]) solves the Schrödinger equation for the quantum harmonic oscillator, using the Numerov's algorithm above described for integration, and searching eigenvalues using the "shooting method". The code uses the adimensional units introduced in (1.4).

The shooting method is quite similar to the bisection procedure for the search of the zero of a function. The code looks for a solution $\psi_n(x)$ with a pre-determined number $n$ of nodes, at an energy $E$ equal to the mid-point of the energy range $[E_{\min}, E_{\max}]$, i.e. $E = (E_{\max} + E_{\min})/2$. The energy range must contain the desired eigenvalue $E_n$. The wave function is integrated starting from $x = 0$ in the direction of positive $x$; tt the same time, the number of nodes (i.e. of changes of sign of the function) is counted. If the number of nodes is larger than $n$, $E$ is too high; if the number of nodes is smaller than $n$, $E$ is too low. We then choose the lower half-interval $[E_{\min}, E_{\max} = E]$, or the upper half-interval $[E_{\min} = E, E_{\max}]$, respectively, select a new trial eigenvalue $E$ in the mid-point of the new interval, iterate the procedure. When the energy interval is smaller than a pre-determined threshold, we assume that convergence has been reached.

For negative $x$ the function is constructed using symmetry, since $\psi_n(-x) = (-1)^n\psi_n(x)$. This is of course possible only because $V(-x) = V(x)$, otherwise integration would have been performed on the entire interval. The parity of the wave function determines the choice of the starting points for the recursion. For $n$ odd, the two first points can be chosen as $y_0 = 0$ and an arbitrary finite value for $y_1$. For $n$ even, $y_0$ is arbitrary and finite, $y_1$ is determined by Numerov's formula, Eq.(1.32), with $f_1 = f_{-1}$ and $y_1 = y_{-1}$:

$$y_1 = \frac{(12 - 10f_0)y_0}{2f_1}. \tag{1.33}$$

The code prompts for some input data:

- the limit $x_{\max}$ for integration (typical values: $5 \div 10$);
- the number $N$ of grid points (typical values range from hundreds to a few thousand); note that the grid point index actually runs from 0 to $N$, so that $\Delta x = x_{\max}/N$;
- the name of the file where output data is written;
- the required number $n$ of nodes (the code will stop if you give a negative number).

Finally the code prompts for a trial energy. You should answer 0 in order to search for an eigenvalue with $n$ nodes. The code will start iterating on the energy, printing on standard output (i.e. at the terminal): iteration number, number of nodes found (on the positive $x$ axis only), the current energy eigenvalue estimate. It is however possible to specify an energy (not necessarily an

[1] http://www.fisica.uniud.it/%7Egiannozz/Corsi/MQ/Software/F90/harmonic0.f90
[2] http://www.fisica.uniud.it/%7Egiannozz/Corsi/MQ/Software/C/harmonic0.c

eigenvalue) to force the code to perform an integration at fixed energy and see the resulting wave function. It is useful for testing purposes and to better understand how the eigenvalue search works (or doesn't work). Note that in this case the required number of nodes will not be honored; however the integration will be different for odd or even number of nodes, because the parity of $n$ determines how the first two grid points are chosen.

The output file contains five columns: respectively, $x$, $\psi(x)$, $|\psi(x)|^2$, $\rho_{\rm cl}(x)$ and $V(x)$. $\rho_{\rm cl}(x)$ is the classical probability density (normalized to 1) of the harmonic oscillator, given in Eq.(1.19). All these quantities can be plotted as a function of $x$ using any plotting program, such as `gnuplot`, shortly described in the introduction. Note that the code will prompt for a new value of the number of nodes after each calculation of the wave function: answer -1 to stop the code. If you perform more than one calculation, the output file will contain the result for all of them in sequence. Also note that the wave function are written for the entire box, from $-x_{\rm max}$ to $x_{\rm max}$.

It will become quickly evident that the code "sort of" works: the results look good in the region where the wave function is not vanishingly small, but invariably, the pathological behavior described in Sec.(1.2.2) sets up and wave functions diverge at large $|x|$. As a consequence, it is impossible to normalize the $\psi(x)$. The code definitely needs to be improved. The proper way to deal with such difficulty is to find an inherently stable algorithm.

### 1.3.2 Code: harmonic1

Code `harmonic1.f90`[3] (or `harmonic1.c`[4]) is the improved version of `harmonic0` that does not suffer from the problem of divergence at large $x$.

Two integrations are performed: a forward recursion, starting from $x = 0$, and a backward one, starting from $x_{\rm max}$. The eigenvalue is fixed by the condition that the two parts of the function match with continuous first derivative (as required for a physical wave function, if the potential is finite). The matching point is chosen in correspondence of the classical inversion point, $x_{\rm cl}$, i.e. where $V(x_{\rm cl}) = E$. Such point depends upon the trial energy $E$. For a function defined on a finite grid, the matching point is defined with an accuracy that is limited by the interval between grid points. In practice, one finds the index `icl` of the first grid point $x_c = \mathtt{icl}\Delta x$ such that $V(x_c) > E$; the classical inversion point will be located somewhere between $x_c - \Delta x$ and $x_c$.

The outward integration is performed until grid point `icl`, yielding a function $\psi_L(x)$ defined in $[0, x_c]$; the number $n$ of changes of sign is counted in the same way as in `harmonic0`. If $n$ is not correct the energy is adjusted (lowered if $n$ too high, raised if $n$ too low) as in `harmonic0`. We note that it is not needed to look for changes of sign beyond $x_c$: in fact we know *a priori* that in the classically forbidden region there cannot be any nodes (no oscillations, just decaying solution).

If the number of nodes is the expected one, the code starts to integrate inward from the rightmost points. Note the statement `y(mesh) = dx`: its only

[3]http://www.fisica.uniud.it/%7Egiannozz/Corsi/MQ/Software/F90/harmonic1.f90
[4]http://www.fisica.uniud.it/%7Egiannozz/Corsi/MQ/Software/C/harmonic1.c