Chapter 3

The quantum one-body problem

3.1 The time-independent 1D Schrödinger equation

We start the numerical solution of quantum problems with the time-independent one-dimensional Schrödinger equation for a particle with mass m in a Potential $V(x)$. In one dimension the Schrödinger equation is just an ordinary differential equation

$$-\frac{1}{2}\frac{\partial^2\psi(x)}{\partial x^2} + V(x)\psi(x) = E\psi(x), \tag{3.1}$$

where we have taken units in which $\hbar = m = 1$.

In order to efficiently solve this problem on a computer, in the great majority of numerical approaches it is necessary to introduce some *discretization* of the problem. The simplest discretization we can perform consists in discretizing the space. We then consider $x \in [x_0, x_p]$ living in a finite interval, and a discretization of the space into a mesh of uniform spacing δ. The points on the mesh are such that

$$x_n = n\delta + x_0, \tag{3.2}$$

where x_0 is the starting point of the interval and the wave function at these points is denoted by

$$\psi_n = \psi(x_n). \tag{3.3}$$

In the following we will assume that the mesh contains a total of $p + 1$ points, thus implying that the last point in the interval is

$$x_p = p\delta + x_0. \tag{3.4}$$

3.1.1 Discretizing the Hamiltonian

The previous discussion has introduced a simple strategy to represent the wave function as a finite-size vector of components ψ_n. In order to solve Schroedinger's equation, it is now necessary also to have finite-sized description of the Hamiltonian. This must be a matrix acting on the same vector space of the discretized state ψ_n. The potential energy

term in the hamiltonian is diagonal in the position basis and it is easily discretized into a diagonal matrix with diagonal entries V_n:

$$V(x) \quad \rightarrow \quad V_n \equiv V(x_n). \tag{3.5}$$

The kinetic energy term is off-diagonal in the position basis and we can also expect that its matrix representation is off diagonal. In order to find an explicit representation, we can use a finite-difference approximation of the derivative:

$$\frac{\partial^2}{\partial x^2} f(x_n) \simeq \frac{1}{\delta^2} \left(f(x_{n-1}) - 2f(x_n) + f(x_{n+1}) \right) + \mathcal{O}(\delta^2). \tag{3.6}$$

Using this discretization, the action of the hamiltonian on the wave function is written as

$$\langle x_n | \hat{H} | \psi \rangle = -\frac{1}{2\delta^2} \left(\psi_{n-1} - 2\psi_n + \psi_{n+1} \right) + V_n \psi_n + \mathcal{O}(\delta^2). \tag{3.7}$$

The right-hand side of this equation contains only linear combinations of the vector ψ_n and it is thus linear operator that corresponds to the discretization of the original Hamiltonian. It is easy to see that this linear operator is a matrix :

$$\hat{H}_\delta = \begin{pmatrix} \dots & -\frac{1}{2\delta^2} & 0 & 0 & \dots \\ -\frac{1}{2\delta^2} & V_{n-1} + \frac{1}{\delta^2} & -\frac{1}{2\delta^2} & 0 & \dots \\ 0 & -\frac{1}{2\delta^2} & V_n + \frac{1}{\delta^2} & -\frac{1}{2\frac{1}{2\delta^2}} & \dots \\ 0 & 0 & -\frac{1}{2\delta^2} & V_{n+1} + \frac{1}{\delta^2} & \dots \\ \dots & 0 & 0 & \dots & \dots \end{pmatrix} \tag{3.8}$$

Notice that we explicitly omitted, for the moment, the value of the matrix on the boundary. This is because the second-order finite difference scheme we have chosen would act also on $\psi(x_0 - \delta) \equiv \psi_{-1}$ and on $\psi(x_p + \delta) \equiv \psi_{p+1}$, but these are beyond the discretized region we have chosen for the vector ψ_n.

In the following we will concentrate on the conceptually easy (and practically relevant) case in which we have a bound state, thus the wave-function goes to zero at infinity. In this case, we can always choose an interval $[x_0, x_p]$ large enough such that, to good approximation $\psi_{-1} = \psi_{p+1} = 0$. In this case then the matrix above is exactly tridiagonal, since it is safe to just ignore the extra-boundary points ψ_{-1}, ψ_{p+1}.

For bound states then, finding solutions to the time-independent Schroedinger equation is a simple as diagonalizing the finite-dimensional matrix $\hat{H}_\delta$, and find the eigenvectors and eigen-energies

$$\hat{H}_\delta |\psi_k\rangle \quad = \quad E_k |\psi_k\rangle.$$

An interesting property of the matrix $\hat{H}_\delta$ is that it is tridiagonal, and it can be very efficiently diagonalized, as you will see more in detail in the exercises.

3.2 The time-independent Schrödinger equation in higher dimensions

In higher dimensions, in most common cases it is possible to reduce the problem to a one-dimensional problem. This happens if the problem, because of symmetries, factorizes.

3.2.1 Factorization along coordinate axis

A first example is a three-dimensional Schrödinger equation in a cubic box with potential $V(\vec{r}) = V(x) + V(y) + V(z)$ with $\vec{r} = (x, y, z)$. Using the product ansatz

$$\psi(\vec{r}) = \psi_x(x)\psi_y(y)\psi_z(z) \tag{3.9}$$

the Schroedinger's equation factorizes into three one-dimensional equations which can be solved as above.

3.2.2 Potential with spherical symmetry

Another famous trick is possible for spherically symmetric potentials with $V(\vec{r}) = V(|\vec{r}|)$ where an ansatz using spherical harmonics

$$\psi_{l,m}(\vec{r}) = \psi_{l,m}(r, \theta, \phi) = \frac{u(r)}{r} Y_{lm}(\theta, \phi) \tag{3.10}$$

can be used to reduce the three-dimensional Schrödinger equation to a one-dimensional one for the radial wave function $u(r)$:

$$\left[-\frac{\hbar^2}{2\mu} \frac{d^2}{dr^2} + \frac{\hbar^2 l(l+1)}{2\mu r^2} + V(r) \right] u(r) = E u(r) \tag{3.11}$$

where we have called the particle mass μ (to avoid confusion with magnetic quantum number m in the spherical harmonics). This is again a one-dimensional Schrödinger equation, with a modified effective potential

$$V_l(r) = V(r) + \frac{\hbar^2}{2\mu} \frac{l(l+1)}{r^2} \tag{3.12}$$

and with the radial wave-function defined in the interval $[0, \infty[$. In practice, for regular potentials we always have $u(0) = u(x_i) = 0$ and it is always possible to find a point $x_p \gg 1$ such that, with good approximation, $u(x_p) = 0$.

3.2.3 Finite difference methods in higher dimension

In higher dimension we can still discretize the space on a regular grid (for example, on a square grid in two dimensions). By doing so, we obtain once more a matrix representation of the kinetic energy. The Laplacian in two dimensions for example takes this form

$$\begin{aligned} \nabla^2 \psi(x_n, y_n) &= \frac{1}{\delta^2} \left[\psi(x_{n+1}, y_n) - 2\psi(x_n, y_n) + \psi(x_{n-1}, y_n) \right] + \\ &\quad + \frac{1}{\delta^2} \left[\psi(x_n, y_{n+1}) - 2\psi(x_n, y_n) + \psi(x_n, y_{n-1}) \right] \end{aligned} \tag{3.13}$$

While the resulting discretized Hamiltonian in general will not be of tridiagonal form as in the 1d case, it is essential to realize that the matrices produced by the discretization of the Schrödinger equation are still very sparse, meaning that only a small fraction of

the matrix entries are non zero. For these sparse systems of equations, optimized iterative numerical algorithms exist[1] and are implemented in numerical libraries such as in the EIGEN library (C++) [2] or in SciPy (Python) [3]. To calculate bound states, an eigenvalue problem has to be solved. For small problems, where the full matrix can be stored in memory, Mathematica or the `dsyev` eigensolver in the LAPACK library can be used. For bigger systems, sparse solvers such as the Lanczos algorithm (which will be discussed in detail in the following lectures) are best. Again there exist efficient implementations of iterative algorithms for sparse matrices.

3.3 The time-dependent Schrödinger equation

We now move to the problem of solving the time-dependent Schrödinger equation

$$i\frac{\partial}{\partial t}\psi(x,t) = -\frac{1}{2}\frac{\partial^2\psi(x,t)}{\partial x^2} + V(x)\psi(x,t), \tag{3.14}$$

with given initial condition $\psi(x,t_0)$.

3.3.1 Spectral methods

By introducing a basis and solving for the complete spectrum of energy eigenstates we can directly solve the time-dependent problem in the case of a stationary (time-independent) Hamiltonian. This is a consequence of the linearity of the Schrödinger equation.

To calculate the time evolution of a state $|\psi(t_0)\rangle$ from time t_0 to t we first solve the stationary eigenvalue problem $\hat{H}|\phi\rangle = E|\phi\rangle$ and calculate the eigenvectors $|\phi_n\rangle$ and eigenvalues ϵ_n. Next we represent the initial wave function $|\psi\rangle$ by a spectral decomposition

$$|\psi(t_0)\rangle = \sum_n c_n|\phi_n\rangle. \tag{3.15}$$

Since each of the $|\phi_n\rangle$ is an eigenvector of $\hat{H}$, the time evolution $e^{-i\hat{H}(t-t_0)}$ is trivial and we obtain at time t:

$$|\psi(t)\rangle = \sum_n c_n e^{-i\epsilon_n(t-t_0)}|\phi_n\rangle. \tag{3.16}$$

This approach is, however, only useful for very small problems since the effort of diagonalizing the matrix is huge. A better method is direct numerical integration, discussed in the next subsections.

[1] R. Barret, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* (SIAM, 1993)

[2] https://eigen.tuxfamily.org/

[3] https://docs.scipy.org/, the relevant routines are contained in *scipy.sparse.linalg*

3.3.2 Direct numerical integration

If the number of basis states is too large to perform a complete diagonalization of the Hamiltonian, or if the Hamiltonian changes over time, instead of the spectral method it is more convenient to perform a direct integration of the Schrödinger equation. Like other initial value problems of partial differential equations the Schrödinger equation can be solved by the method of lines. After choosing a set of basis functions or discretizing the spatial derivatives we obtain a set of coupled ordinary differential equations which can be evolved for each point along the time line (hence the name) by standard ODE solvers.

In the remainder of this chapter we use the symbol $\hat{H}_\delta$ to refer the representation of the Hamiltonian in the chosen finite basis. A simple ODE integration scheme is the forward Euler scheme

$$|\psi(t_{n+1})\rangle = |\psi(t_n)\rangle - i\delta_t \hat{H}_\delta |\psi(t_n)\rangle. \tag{3.17}$$

However, this method is not only numerically unstable, but it also violates the conservation of the norm of the wave function $\langle\psi(t)|\psi(t)\rangle = 1$, since

$$(1 - i\delta_t \hat{H}_\delta)\left(1 - i\delta_t \hat{H}_\delta\right)^\dagger \;=\; 1 + \delta_t^2 \hat{H}^2 \neq 1.$$

The exact quantum evolution

$$|\psi(t+\delta_t)\rangle = e^{-i\hat{H}_\delta \delta_t}|\psi(t)\rangle \tag{3.18}$$

is however clearly unitary and thus conserves the norm, we therefore want to look for a unitary approximation as integrator. Instead of using the forward Euler method (3.17) which is just a first order Taylor expansion of the exact time evolution

$$e^{-iH\delta_t} = 1 - i\delta_t \hat{H}_\delta + \mathrm{O}(\delta_t^2), \tag{3.19}$$

we reformulate the time evolution operator as

$$e^{-iH\delta_t} = \left(e^{iH\delta_t/2}\right)^{-1} e^{-iH\delta_t/2} = \left(1 + \frac{i\delta_t}{2}\hat{H}_\delta\right)^{-1}\left(1 - \frac{i\delta_t}{2}\hat{H}_\delta\right) + \mathrm{O}(\delta_t^3), \tag{3.20}$$

therefore

$$|\psi(t+\delta_t)\rangle = \left(1 + \frac{i\delta_t}{2}\hat{H}_\delta\right)^{-1}\left(1 - \frac{i\delta_t}{2}\hat{H}_\delta\right)|\psi(t)\rangle, \tag{3.21}$$

It is possible to check that the propagation scheme defined above is unitary (for example, showing that the two terms appearing commute with one another), thus we have a small time-step approach that is both unitary and second-order in δ_t. Equivalently, we can write it as

$$\left(1 + \frac{i\delta_t}{2}\hat{H}_\delta\right)|\psi(t+\delta_t)\rangle = \left(1 - \frac{i\delta_t}{2}\hat{H}_\delta\right)|\psi(t)\rangle, \tag{3.22}$$

which shows more explicitly that, unfortunately this is an implicit integrator, since the value of $|\psi(t+\delta_t)\rangle$ is not a simple linear combination of the wave-function values at the previous time-steps.

3.3.2.1 Practical implementations of the implicit scheme

Despite its implicit nature, this integrator can be still be used efficiently. Concentrating again on the 1d case, we have seen previously that if we discretize our problem on a mesh $x_n = n\delta + x_0$, that the Hamiltonian becomes a simple tridiagonal matrix. The implicit equation 3.22 then becomes a linear system of the form $\hat{A}y = b$, where the matrix $\hat{A} = \left(1 + \frac{i\delta_t}{2}\hat{H}_\delta\right)$, the right hand side is $b_n = \left(1 - \frac{i\delta_t}{2}\right)\psi_n(t)$ and the unknown vector $y_n = \psi_n(t + \delta_t)$.

At each time step, one can therefore solve this linear system of equations and find explicitly $\psi_n(t + \delta_t)$. Because of the tridiagonal structure, very efficient tridiagonal solver can be used.

In higher dimensions the matrix H will no longer be simply tridiagonal but still very sparse and we can use iterative algorithms, similar to the Lanczos algorithm for the eigenvalue problem. For details about these algorithms we refer to the nice summary at `http://mathworld.wolfram.com/topics/Templates.html` and especially the biconjugate gradient (BiCG) algorithm. Implementations of this algorithm are available, e.g. in the EIGEN Library C++, or in SciPy, for a Python version.

3.4 Appendix: The split operator method

An alternative to the unitary, implicit method described in the main text exists, and we discuss it as an optional argument for the interested reader. An explicit and unitary method is possible for a quantum particle in the real space picture with the "standard" Schrödinger equation for non-relativistic particles in continuous space. Writing the Hamiltonian operator as

$$H = \hat{T} + \hat{V} \tag{3.23}$$

with

$$\hat{T} = \frac{1}{2m}\hat{p}^2 \tag{3.24}$$

$$\hat{V} = V(\vec{x}) \tag{3.25}$$

it is easy to see that $\hat{V}$ is diagonal in position space while $\hat{T}$ is diagonal in momentum space.

Indeed if we consider a d-dimensional particle, its wave-function in momentum space is obtained through the Fourier transform:

$$\tilde{\psi}(\vec{k}) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \int_{-\infty}^{\infty} \psi(\vec{x}) e^{-i\vec{k}\cdot x} d\vec{x} \tag{3.26}$$

and the inverse Fourier transform yields

$$\psi(\vec{x}) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \int_{-\infty}^{\infty} \tilde{\psi}(\vec{k}) e^{i\vec{k}\cdot x} d\vec{k}. \tag{3.27}$$

It is then easy to check that $\tilde{\psi}(\vec{k})$ is an eigenstate of the kinetic operator T, and that $T\tilde{\psi}(\vec{k}) = ||\vec{k}||^2\tilde{\psi}(\vec{k})/2$.

If we split the time evolution as

$$e^{-i\Delta_t H/\hbar} = e^{-i\Delta_t \hat{V}/2\hbar} e^{-i\Delta_t \hat{T}/\hbar} e^{-i\Delta_t \hat{V}/2\hbar} + \mathrm{O}(\Delta_t^3) \tag{3.28}$$

we can perform the individual time evolutions $e^{-i\Delta_t \hat{V}/2\hbar}$ and $e^{-i\Delta_t \hat{T}/\hbar}$ exactly:

$$\left[e^{-i\Delta_t \hat{V}/2\hbar}|\psi\rangle\right](\vec{x}) = e^{-i\Delta_t V(\vec{x})/2\hbar}\psi(\vec{x}) \tag{3.29}$$

$$\left[e^{-i\Delta_t \hat{T}/\hbar}|\psi\rangle\right](\vec{k}) = e^{-i\Delta_t \hbar||\vec{k}||^2/2m}\psi(\vec{k}) \tag{3.30}$$

in real space for the first term and momentum space for the second term.

Propagating for a time $t = N\Delta_t$, two consecutive applications of $e^{-i\Delta_t \hat{V}/2\hbar}$ can easily be combined into a propagation by a full time step $e^{-i\Delta_t \hat{V}/\hbar}$, resulting in the propagation:

$$\begin{aligned} e^{-iHt/\hbar} &= \left(e^{-i\Delta_t \hat{V}/2\hbar} e^{-i\Delta_t \hat{T}/\hbar} e^{-i\Delta_t \hat{V}/2\hbar}\right)^N + \mathrm{O}(\Delta_t^2) \\ &= e^{-i\Delta_t \hat{V}/2\hbar}\left[e^{-i\Delta_t \hat{T}/\hbar} e^{-i\Delta_t \hat{V}/\hbar}\right]^{N-1} e^{-i\Delta_t \hat{T}/\hbar} e^{-i\Delta_t \hat{V}/2\hbar} \end{aligned} \tag{3.31}$$

In practice, in order to obtain efficient representations of the wave-functions both in real and momentum space we still need to discretize the real space with a suitable mesh of size Δx, for a total of P points per spatial direction. As a consequence of this discretization, the continuous Fourier transform becomes a discrete Fourier transform defined on the discrete set of wave-vectors $k_n = \frac{2\pi}{n}P$, for each spatial direction, with $n = 0, 1, \ldots P-1$. Changing from real space to momentum space then requires the application of the discrete Fourier transform and of its inverse when going back from momentum space to real space. This can be efficiently accomplished numerically thanks to the Fast Fourier Transform (FFT) algorithm, which performs the discrete Fourier transform in only $\mathcal{O}(P\log(P))$ operations.

The discretized algorithm then starts as

$$\psi_1(\vec{x}) = e^{-i\Delta_t V(\vec{x})/2\hbar}\psi_0(\vec{x}) \tag{3.32}$$

$$\psi_1(\vec{k}) = \mathcal{F}\psi_1(\vec{x}) \tag{3.33}$$

where $\mathcal{F}$ denotes the Fourier transform and $\mathcal{F}^{-1}$ will denote the inverse Fourier transform. Next we propagate in time using full time steps:

$$\psi_{2n}(\vec{k}) = e^{-i\Delta_t \hbar||\vec{k}||^2/2m}\psi_{2n-1}(\vec{k}) \tag{3.34}$$

$$\psi_{2n}(\vec{x}) = \mathcal{F}^{-1}\psi_{2n}(\vec{k}) \tag{3.35}$$

$$\psi_{2n+1}(\vec{x}) = e^{-i\Delta_t V(\vec{x})/\hbar}\psi_{2n}(\vec{x}) \tag{3.36}$$

$$\psi_{2n+1}(\vec{k}) = \mathcal{F}\psi_{2n+1}(\vec{x}) \tag{3.37}$$

except that in the last step we finish with another half time step in real space:

$$\psi_{2N+1}(\vec{x}) = e^{-i\Delta_t V(\vec{x})/2\hbar}\psi_{2N}(\vec{x}) \tag{3.38}$$

This is a fast and unitary integrator for the Schrödinger equation in real space. It could be improved by replacing the locally third order splitting (3.28) by a fifth-order version involving five instead of three terms.

Chapter 4

Exact diagonalization of many-body problems

4.1 Quantum spin models

After learning how to solve the 1-body Schrödinger equation, let us next generalize to more particles. If a single body quantum problem is described by a Hilbert space $\mathcal{H}$ of dimension $\dim\mathcal{H} = d$ then N *distinguishable* quantum particles are described by the tensor product of N Hilbert spaces

$$\mathcal{H}^{(N)} \equiv \mathcal{H}^{\otimes N} \equiv \bigotimes_{i=1}^{N} \mathcal{H} \tag{4.1}$$

with dimension d^N.

In this Chapter we will specifically focus on quantum spin-1/2 particles. A single spin-1/2 has a Hilbert space $\mathcal{H} = \mathbb{C}^2$ of dimension 2, but N spin-1/2 have a Hilbert space $\mathcal{H}^{(N)} = \mathbb{C}^{2^N}$ of dimension 2^N. This exponential scaling of the Hilbert space dimension with the number of particles is a big challenge. The basis for $N = 30$ spins is already of size $2^{30} \approx 10^9$. A single complex vector needs 16 GByte of memory and may just barely fit into the memory of your personal computer.

For small and moderately sized systems of up to about 30 spin-1/2 we can calculate exactly the ground state, low-lying spectrum, and time evolution by direct calculations. For more than 30 spins we cannot apply exact diagonalization techniques anymore, and this will be the subject of several methods we will study in the next chapters.

4.1.1 Hamiltonian Matrix

As we have seen already in the previous Chapter, to perform exact diagonalization to find eigenstates of a given Hamiltonian, $\hat{H}$, or study its dynamics, it is important to come up with a concrete representation of the Hamiltonian matrix that can be efficiently manipulated on a computer. One common feature of many-body quantum models is that the matrix representation of their hamiltonian is *sparse.* For example, taking again the case of quantum spins, one can see that the total number of non-zero elements in the matrix representation of the Hamiltonian is at most $k \times 2^N$, where k is

typically a small value (in most cases, $k \sim N$). This is to be contrasted to a general, full matrix, that instead contains $\mathcal{O}(2^N \times 2^N)$ elements. Sparsity–a generalization of the simple pattern of non-zero elements seen in tridiagonal matrices in the previous Chapter– can be exploited by exact diagonalization methods in different ways, both to find eigenvalues and eigenvectors of the Hamiltonian and to study its dynamics. Before seeing how sparsity can be exploited, we will first analyze a few prototypical spin models, in order to better understand where the sparse nature of the Hamiltonian matrix comes from. In all cases we will analyze in this Chapter we will consider the simple, and widely adopted, basis of eigenstates of $\hat{\sigma}^z$. Specifically, each many-spin state is written as a linear combination of 2^N basis states:

$$|\Psi\rangle = \sum_{s_1 s_2 \ldots s_N} c_{s_1 s_2 \ldots s_N} |s_1 s_2 \ldots s_N\rangle, \tag{4.2}$$

where

$$|s_1 s_2 \ldots s_N\rangle = |s_1\rangle \otimes |s_2\rangle \otimes \ldots |s_N\rangle \tag{4.3}$$

are eigen-kets of the $\hat{\sigma}^z$ Pauli matrix:

$$\hat{\sigma}_i^z |s_1 s_2 \ldots s_N\rangle = s_i |s_1 s_2 \ldots s_N\rangle, \tag{4.4}$$

for $s_i = \pm 1$.

4.1.2 Example: the transverse-field Ising model

The simplest quantum spin model is probably the quantum transverse field Ising model (TFIM), which adds a magnetic field in the x direction to a lattice of spin-1/2 particles coupled by an Ising interaction:

$$\hat{H} = \sum_{\langle i,j \rangle} J_{ij} \hat{\sigma}_i^z \hat{\sigma}_j^z - \Gamma \sum_i \hat{\sigma}_i^x. \tag{4.5}$$

Here the symbol $\langle i, j \rangle$ denotes a sum over all bonds in the lattice. In the absence of the second term, the first term is nothing but a classical Ising model and can be solved by your favorite method of simulating the Ising model. The second term does not exist in classical Ising models, where the spins point only in the z direction. Considering that the Pauli $\hat{\sigma}^x$ matrix is

$$\hat{\sigma}^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{4.6}$$

we see that this term flips an $\uparrow$ spin to a $\downarrow$ spin, and thus introduces quantum fluctuations to the classical Ising model.

The way of writing the hamiltonian as above is nothing but a short-hand for the more laborious (but more precise) notation with tensor products, that in this case would imply for example that a spin operator in the direction $\alpha = (x, y, z)$ and acting on spin i is in reality the following $2^N \times 2^N$ matrix:

$$\hat{\sigma}_i^\alpha \equiv \underbrace{\hat{I} \otimes \hat{I} \otimes \ldots \hat{I}}_{i-1 \text{ times}} \otimes \hat{\sigma}^\alpha \otimes \underbrace{\hat{I} \otimes \cdots \otimes \hat{I}}_{N-i \text{ times}} \tag{4.7}$$

$$= \hat{I}(2^{i-1}) \otimes \hat{\sigma}^\alpha \otimes \hat{I}(2^{N-i}), \tag{4.8}$$

where $\hat{I}(n)$ are identity matrices of dimension n, and the $\otimes$ product here denotes Kronecker product between matrices.

We can readily verify that this Hamiltonian is sparse. For example, let's start computing the diagonal matrix elements in the basis specified above:

$$\langle s_1 s_2 \dots s_N | \hat{H} | s_1 s_2 \dots s_N \rangle = \sum_{\langle i,j \rangle} J_{ij} s_i s_j, \tag{4.9}$$

which is the familiar classical Ising interaction term. Thus we have found the first 2^N (in general) non-zero matrix elements, corresponding to the diagonal of $\hat{H}$. The off-diagonal terms can be readily found noticing that the action of the $\hat{\sigma}_i^x$ operator is just to flip a spin:

$$\hat{\sigma}_i^x | s_1 \dots s_i \dots s_N \rangle = | s_1 \cdots - s_i \dots s_N \rangle, \tag{4.10}$$

thus there is only one non-zero matrix element per $\hat{\sigma}_i^x$ term:

$$\langle s'_1 s'_2 \dots s'_N | \hat{\sigma}_i^x | s_1 s_2 \dots s_N \rangle = \delta_{s'_1, s_1} \dots \delta_{s'_i - s_i} \dots \delta_{s'_N, s_N} \tag{4.11}$$

implying that, at fixed $|s_1 s_2 \dots s_N\rangle$, there is a total of N non zero matrix elements for the Hamiltonian. In total, therefore, we have that the TFI Hamiltonian contains "only" $(N+1) \times 2^N$ non-zero elements.

4.2 Finding Ground States

We start with the problem of finding the lowest eigenvector (and its energy) of the Hamiltonian, the so-called ground state. This task is realized by using an iterative matrix eigensolver. These solvers exploit the fact that computing the product of the Hamiltonian matrix with an arbitrary vector can be done efficiently. While for a generic matrix of size $M \times M$ a product $\hat{A}|v\rangle$ can be computed in $\mathcal{O}(M^2)$ time, for a matrix $M \times M$, in the case of Hamiltonians we are considering here this product is computable in only $\mathcal{O}(M) = \mathcal{O}(N^\alpha \times 2^N)$, where α is in general a small exponent ($\alpha = 1$, for the TFIM, as seen before).

4.2.1 Power Method

The power method is the simplest iterative solver we can use to find ground-states of many-body Hamiltonians that exploits sparseness. This method generates a sequence of P vectors $k = 1, \dots P$ by repeated application of the Hamiltonian:

$$|u_{k+1}\rangle = \left(\Lambda \hat{I} - \hat{H}\right) |u_k\rangle, \tag{4.12}$$

where Λ is a suitable constant, and the initial state $|u_0\rangle$ is given as starting condition for the algorithm. This sequence of vectors converges to the ground-state of the Hamiltonian under reasonable assumptions. To see this, let us formally expand the initial vector in terms of the eigen-states of the Hamiltonian:

$$|u_0\rangle = \sum_l c_l |E_l\rangle, \tag{4.13}$$

with $E_0 \leq E_1 \leq \ldots E_M$, thus the last state is

$$|u_P\rangle = \left(\Lambda \hat{I} - \hat{H}\right)^P |u_0\rangle, \tag{4.14}$$

$$= \sum_l (\Lambda - E_l)^P c_l |E_l\rangle, \tag{4.15}$$

and the overlap with the ground state is

$$\langle E_0|u_P\rangle = (\Lambda - E_0)^P c_0. \tag{4.16}$$

We notice however that the state $|u_P\rangle$ is not normalized in general, thus the probability amplitude of being in the ground-state after k iterations is

$$\frac{|\langle E_0|u_P\rangle|^2}{\langle u_P|u_P\rangle} = \frac{(\Lambda - E_0)^{2P} |c_0|^2}{(\Lambda - E_0)^{2P} |c_0|^2 + (\Lambda - E_1)^{2P} |c_1|^2 + \ldots} \tag{4.17}$$

$$= \frac{1}{1 + \frac{(\Lambda - E_1)^{2P}}{(\Lambda - E_0)^{2P}} \frac{|c_1|^2}{|c_0|^2} + \ldots}. \tag{4.18}$$

From this expression we can see that a suitable choice of Λ can force the state $|u_P\rangle$ have a probability amplitude of being in the ground state that is *exponentially* close to 1. Specifically, if we impose $\Lambda > E_M$, we have that

$$\lim_{P \to \infty} \frac{(\Lambda - E_l)^{2P}}{(\Lambda - E_0)^{2P}} = 0, \tag{4.19}$$

for any excited state l such that $E_l > E_0$. In the limit of large P we therefore have that

$$\frac{|\langle E_0|u_P\rangle|^2}{\langle u_P|u_P\rangle} \simeq 1 - \frac{(\Lambda - E_1)^{2P}}{(\Lambda - E_0)^{2P}} \frac{|c_1|^2}{|c_0|^2}, \tag{4.20}$$

and the correction can be made arbitrarily (and exponentially) small by increasing the number of steps P. We also see that for the exponential convergence to be true we need to have that the initial state has some finite overlap with the exact ground state, namely $|c_0|^2 \neq 0$. This can be achieved, for example, starting the iterations from a random vector.

The power method is therefore a very simple, yet exponentially converging method, to find the ground state of the Hamiltonian. If, for example, the Hamiltonian is stored in memory as a sparse matrix, then by simple iterative applications one can find the ground state. In practice, it is convenient to keep the state $|u_k\rangle$ normalized at each step, to avoid an exponential increase of the coefficients appearing in the vector $|u_k\rangle$.

4.2.2 The Lanczos Method

The Lanczos algorithm is an important improvement over the power method, that allows to reconstruct not only the ground state wave function, but also excited states. The Lanczos algorithm builds an orthogonal basis $\{v_1, v_2, \ldots, v_P\}$ for the Krylov-subspace

$K_P = \text{span}\{u_1, u_2, \ldots, u_P\}$, which is constructed by P iterations of the power method. This is achieved by the following iterations:

$$\beta_{n+1}|v_{n+1}\rangle = \hat{H}|v_n\rangle - \alpha_n|v_n\rangle - \beta_n|v_{n-1}\rangle, \tag{4.21}$$

where

$$\alpha_n = \langle v_n|\hat{H}|v_n\rangle, \qquad \beta_n = |\langle v_n|\hat{H}|v_{n-1}\rangle|. \tag{4.22}$$

Since the orthogonality condition

$$\langle v_i|v_j\rangle = \delta_{ij} \tag{4.23}$$

does not determine the phases of the basis vectors, the β_i can be chosen to be real and positive. It can be shown that we only need to keep three vectors of size M in memory, which makes the Lanczos algorithm very efficient, when compared to dense matrix eigen-solvers which require storage of order M^2 (see Table 4.1 for a summary of the complexity of matrix operations).

In the Krylov basis the matrix $\hat{H}$ is approximated by the following tridiagonal matrix

$$\hat{T}^{(n)} \doteq \begin{bmatrix} \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ \beta_2 & \alpha_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_n \\ 0 & \cdots & 0 & \beta_n & \alpha_n \end{bmatrix}, \tag{4.24}$$

and it can also been shown that the eigenvalues $\{\tau_1, \ldots, \tau_M\}$ of $\hat{T}$ are good approximations of the eigenvalues of $\hat{H}$. Moreover, the extreme eigenvalues converge very fast. Thus $P \ll M$ iterations are sufficient to obtain the extreme eigenvalues. Since the Lanczos matrix is tridiagonal, we can use all the efficient computational approaches discussed in the previous Chapter to find both its eigenvalues and eigenvectors.

In practice, the Lanczos method can be already found implemented in all linear algebra solvers for sparse matrices, for example in scipy. For Python users, we strongly suggest to use SciPy (in particular scipy.sparse.linalg) which performs Lanczos/Arnoldi calling an efficient, C-coded backend. These routines allow to diagonalize directly sparse matrices defined within scipy. Alternatively, and in order to avoid storing the sparse matrix, one can also define its own Matrix-Vector multiplication using scipy.sparse.linalg.LinearOperator, and then obtain the eigenvalues and eigenvectors with a call to scipy.sparse.linalg.eigsh.

A more detailed discussion of the Lanczos algorithm and the issue of ghost eigenvalues can be found in Appendix 4.5.

4.2.3 Implementation

From the practical implementation point of view, the main requirement to use the simple power method or the more refined Lanczos algorithm is to provide a function that computes the product of the Hamiltonian with an arbitrary vector $|v\rangle$:

$$\hat{H}|v\rangle \quad = \quad |v'\rangle. \tag{4.25}$$

There are two main approaches to implement this efficiently. One one hand, we can form and store the hamiltonian $\hat{H}$ as a sparse matrix. This approach is very elegant and can be readily implemented, for example, in Python with scipy. The only requirement, for spin hamiltonians, is to explicitly use and form the Kronecker products for spin operators, as seen before:

$$\hat{\sigma}_i^\alpha = \hat{I}(2^{i-1}) \otimes \hat{\sigma}^\alpha \otimes \hat{I}(2^{N-i}), \tag{4.26}$$

and then construct interactions terms as simple products of these matrices. For example, spin-spin interaction terms $\hat{\sigma}_i^\alpha \hat{\sigma}_j^\beta$ can be readily obtained as a sparse matrix-matrix multiplication.

In addition of being very elegant and compactly implemented, this approach has also the advantage that computing products of sparse matrices with vectors is a typically highly optimized operation in dedicated software libraries, thus the resulting scheme will be automatically highly efficient. The main drawback however is the memory requirements, since we need to store all the non-zero matrix elements of the Hamiltonian, and there are at least as many as $N \times 2^N$ of them, as we have seen before. This memory requirement is added to the requirements due to the necessity of storing (at least) the vectors $|v\rangle$ and $|v'\rangle$, yielding an additional 2×2^N coefficients to be stored.

The main alternative approach is to never store the matrix $\hat{H}$ and provide instead a function that computes the matrix-vector product "on the fly". This allows to drastically reduce the memory consumption to the bare minimum, namely to 2×2^N, at the expenses of, typically, a larger computational time. The latter approach is especially suited for applications where reaching to the largest possible value of N is crucial, and need specialized low-level implementations. In the exercises we will mostly focus on the first approach.

4.3 Quantum Dynamics

In the previous discussion we have seen how to explicitly construct sparse representations of the Hamiltonian of quantum spin systems, and how to use them to obtain the ground-state wave-function. We now focus on the problem of solving the time-dependent Schrödinger equation for the many-spin system, a task which requires specific techniques. For simplicity, we will analyze here the specific case of time-independent Hamiltonians, and leave the straightforward extension to time-dependent Hamiltonians as an exercise.

4.3.1 Taylor Expansion

To implement the time evolution we have to devise an efficient way to numerically compute the matrix exponential $\exp(-i\hat{H}t)$, since for a static Hamiltonian the time-evolved state that satisfies Schrödinger equation reads :

$$|\Psi(t)\rangle = e^{-i\hat{H}t} |\Psi(0)\rangle . \tag{4.27}$$

The most straightforward way to do so is to take a small time step Δ_t and consider a truncated Taylor expansion of the exponential, such that

$$|\Psi(t+\Delta_t)\rangle = \left(1 - i\Delta_t \hat{H} - \frac{\Delta_t^2}{2}\hat{H}^2 - i\frac{\Delta_t^3}{6}\hat{H}^3 + \dots\right)|\Psi(t)\rangle\,. \tag{4.28}$$

Taking the first s orders in the Taylor expansion guarantees a scheme locally of order $\mathcal{O}(\Delta_t^s)$. This scheme can be efficiently implemented recalling that the Hamiltonian $\hat{H}$ is sparse, and that we can efficiently compute products of $\hat{H}$ with a given vector:

$$|\Psi'\rangle = \hat{H}\,|\Psi\rangle\,. \tag{4.29}$$

A simple iterative scheme that realizes the Taylor expansion numerically is given by the following recursion formula:

$$|\Gamma_k\rangle = \frac{-i\Delta_t}{k}\hat{H}\,|\Gamma_{k-1}\rangle \tag{4.30}$$

$$|\Delta_k\rangle = |\Delta_{k-1}\rangle + |\Gamma_k\rangle\,, \tag{4.31}$$

for $k = 1, 2, \dots s$, up to the maximum truncation order chosen, and with zero-order conditions $|\Gamma_0\rangle = |\Delta_0\rangle = |\Psi(t)\rangle$. Then we have

$$|\Psi(t+\Delta_t)\rangle = |\Delta_s\rangle. \tag{4.32}$$

This scheme is particularly memory friendly, because it needs to store at most two vectors: $|\Delta_k\rangle$ and $|\Gamma_k\rangle$.

4.4 The Trotter-Suzuki decomposition

We now present an alternative numerical scheme which, at variance with the previous Taylor series, explicitly preserves the unitary character of the Hamiltonian evolution. To derive this scheme, we introduce one of the most important tools in computational quantum physics: the Trotter-Suzuki decomposition.[1]

To do this we split the Hamiltonian into a sum of K non-commuting terms $\hat{H} = \sum_{k=1}^{K}\hat{h}_k$. The splitting is done in such a way that the exponential of the individual terms, $e^{-i\hat{h}_k\Delta_t}$, can be easily computed. The time evolution operator $\exp(-i\hat{H}\Delta_t)$ for a small time step Δ_t is then decomposed into multiple products of the non-commuting terms in the Hamiltonian. To first order, the Trotter-Suzuki decomposition for a small time step Δ_t is

$$\exp(-i\hat{H}\Delta_t) = e^{-i\hat{h}_1\Delta_t}\dots e^{-i\hat{h}_K\Delta_t} + \mathcal{O}(\Delta_t^2). \tag{4.33}$$

The second order version of this formula reads

$$\exp(-i\hat{H}\Delta_t) = e^{-i\hat{h}_1\frac{\Delta_t}{2}}\dots e^{-i\hat{h}_K\frac{\Delta_t}{2}}e^{-i\hat{h}_K\frac{\Delta_t}{2}}\dots e^{-i\hat{h}_1\frac{\Delta_t}{2}} + \mathcal{O}(\Delta_t^3). \tag{4.34}$$

For the special case with $K = 2$ terms this expression simplifies to

$$\exp(-i\hat{H}\Delta_t) = e^{-i\hat{h}_1\Delta_t/2}e^{-i\hat{h}_2\Delta_t}e^{-i\hat{h}_1\Delta_t/2} \tag{4.35}$$

[1] H. F. Trotter, *On the product of semi-groups of operators*, Proc. Amer. Math. Soc. **10**, 545 (1959); M. Suzuki, *Generalized Trotter's formula and systematic approximants of exponential operators and inner derivations with applications to many-body problems*, Commun. Math. Phys. **51**, 183 (1976).

by combining the two terms $e^{-i\hat{h}_2\Delta_t/2}e^{-i\hat{h}_2\Delta_t/2}$ into $e^{-i\hat{h}_2\Delta_t}$. By similarly combining the terms $e^{-i\hat{h}_1\Delta_t/2}e^{-i\hat{h}_1\Delta_t/2}$ arising from two adjacent time steps into $e^{-i\hat{h}_1\Delta_t}$ one ultimately needs only one single additional terms for the full time evolution, when compared to the first order approximation. At second order, the full time evolution for $K = 2$ indeed reads

$$\begin{aligned} \exp(-i\hat{H}t) &\simeq e^{-i\hat{h}_1\Delta_t/2}e^{-i\hat{h}_2\Delta_t}e^{-i\hat{h}_1\Delta_t/2} \times e^{-i\hat{h}_1\Delta_t/2}e^{-i\hat{h}_2\Delta_t}e^{-i\hat{h}_1\Delta_t/2}\dots \quad (4.36) \\ &\simeq e^{-i\hat{h}_1\Delta_t/2}\dots e^{-i\hat{h}_2\Delta_t}e^{-i\hat{h}_1\Delta_t}\dots e^{-i\hat{h}_2\Delta_t}e^{-i\hat{h}_1\Delta_t/2}. \quad (4.37) \end{aligned}$$

4.4.1 Time evolution for the transverse field Ising model

To implement time evolution in the transverse field Ising model we split the Hamiltonian into $K = 2$ non-commuting terms. The first one is the the transverse field term

$$\hat{H}_x = -\Gamma \sum_l \hat{\sigma}_l^x, \quad (4.38)$$

and the second one is the Ising term

$$\hat{H}_z = \sum_{\langle l,m\rangle} J_{lm}\hat{\sigma}_l^z\hat{\sigma}_m^z. \quad (4.39)$$

We will now see that each of these terms can be easily exponentiated.

The transverse field term splits into N commuting terms for each of the spins:

$$e^{-i\hat{H}_x\Delta_t} = e^{i\Delta_t\Gamma\sum_l\hat{\sigma}_l^x} = \prod_l e^{i\Delta_t\Gamma\hat{\sigma}_l^x}. \quad (4.40)$$

Each of the terms in the product above can be written explicitly in Kronecker product form:

$$e^{i\Delta_t\Gamma\hat{\sigma}_l^x} = \hat{I}(2^{l-1}) \otimes \Big(\cos(\Delta_t\Gamma)\hat{I}(2) + i\sin(\Delta_t\Gamma)\hat{\sigma}^x\Big) \otimes \hat{I}(2^{N-l}). \quad (4.41)$$

The Ising term instead is diagonal, and the exponentiation is particularly simple, yielding a diagonal matrix:

$$e^{-i\hat{H}_z\Delta_t}|s_1 s_2 \dots s_N\rangle = \prod_{\langle l,m\rangle} e^{-i\Delta_t J_{lm}s_l s_m}|s_1 s_2 \dots s_N\rangle. \quad (4.42)$$

We can further write this as a sum of Kronecker products, noticing that (the proof is left as an exercise)

$$e^{-i\theta\hat{\sigma}_l^z\hat{\sigma}_m^z} = \cos\theta\hat{I}(2^N) - i\sin\theta\hat{\sigma}_l^z\hat{\sigma}_m^z, \quad (4.43)$$

thus each term in the product $\prod_{\langle l,m\rangle} e^{-i\Delta_t J_{lm}s_l s_m}$ can be easily applied recalling the explicit Kronecker product form of $\hat{\sigma}_l^z$ and $\hat{\sigma}_m^z$.

Overall, then both the diagonal and the off diagonal terms can easily be applied to a wave function in a similar way as we did for the multiplication with the Hamiltonian $\hat{H}$. We will implement time evolution for the TFIM in the exercises.

Table 4.1: Time and memory complexity for operations on sparse and dense $M \times M$ matrices

operation	time	memory
storage		
dense matrix	—	M^2
sparse matrix	—	$\mathrm{O}(M)$
matrix-vector multiplication		
dense matrix	$\mathrm{O}(M^2)$	$\mathrm{O}(M^2)$
sparse matrix	$\mathrm{O}(M)$	$\mathrm{O}(M)$
matrix-matrix multiplication		
dense matrix	$\mathrm{O}(M^{\ln 7/\ln 2})$	$\mathrm{O}(N^2)$
sparse matrix	$\mathrm{O}(M) \ldots \mathrm{O}(M^2)$	$\mathrm{O}(M) \ldots \mathrm{O}(M^2)$
all eigen values and vectors		
dense matrix	$\mathrm{O}(M^3)$	$\mathrm{O}(M^2)$
sparse matrix (iterative)	$\mathrm{O}(M^2)$	$\mathrm{O}(M^2)$
some eigen values and vectors		
dense matrix (iterative)	$\mathrm{O}(M^2)$	$\mathrm{O}(M^2)$
sparse matrix (iterative)	$\mathrm{O}(M)$	$\mathrm{O}(M)$

4.5 Appendix: The Lanczos algorithm

Sparse matrices with only $\mathcal{O}(M)$ non-zero elements are very common in scientific simulations. We have seen in this Chapter that many-body quantum Hamiltonians belong to the class of sparse matrices, and that for typical spin models one has $M \sim 2^N N^\alpha$, for some small power α.

The importance of sparsity becomes obvious when considering the cost of matrix operations as listed in table 4.1. For large M the sparsity leads to memory and time savings of several orders of magnitude.

Here we will discuss the iterative calculation of a few of the extreme eigenvalues of a matrix by the Lanczos algorithm. Similar methods can be used to solve sparse linear systems of equations.

4.5.1 Finding eigenvectors

While finding the eigenvectors of the tridiagonal Lanczos matrix $\hat{T}$ is a relatively easy computational task, however these are not directly the eigenvectors of the original matrix $\hat{H}$, since they are given in the (much smaller) Krylov basis $\{v_1, v_2, \ldots, v_P\}$. To obtain the eigenvectors in the original basis we need to perform a basis transformation.

Due to memory constraints we usually do not store all the v_i, but only the last three vectors. To transform the eigenvector to the original basis we have to do the Lanczos iterations a second time. Starting from the same initial vector v_1 we construct the vectors v_i iteratively and perform the basis transformation as we go along.

4.5.2 Roundoff errors and ghosts in the Lanczos algorithm

In exact arithmetic the vectors $\{v_i\}$ are orthogonal and the Lanczos iterations stop after at most $M-1$ steps. The eigenvalues of $\hat{T}$ are then the exact eigenvalues of $\hat{H}$.

Roundoff errors in finite precision however cause a loss of orthogonality. There are two ways to deal with that:

- Re-orthogonalization of the vectors after every step. This requires storing all of the vectors $\{v_i\}$ and is memory intensive.
- Control of the effects of roundoff.

We will discuss the second solution as it is faster and needs less memory. The main effect of roundoff errors is that the matrix $\hat{T}$ contains extra spurious eigenvalues, called "ghosts". These ghosts are not real eigenvalues of $\hat{A}$. However they converge towards real eigenvalues of $\hat{A}$ over time and increase their multiplicities.

A simple criterion distinguishes ghosts from real eigenvalues. Ghosts are caused by roundoff errors. Thus they do not depend on on the starting vector v_1. As a consequence these ghosts are also eigenvalues of the matrix $\hat{Q}^{(n)}$, which can be obtained from $\hat{T}$ by deleting the first row and column:

$$\hat{Q}^{(n)} := \begin{bmatrix} \alpha_2 & \beta_3 & 0 & \cdots & 0 \\ \beta_3 & \alpha_3 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_n \\ 0 & \cdots & 0 & \beta_n & \alpha_n \end{bmatrix}. \tag{4.44}$$

From these arguments we derive the following heuristic criterion to distinguish ghosts from real eigenvalues:

- All multiple eigenvalues are real, but their multiplicities might be too large.
- All single eigenvalues of T which are *not* eigenvalues of $\tilde{T}$ are also real.

4.5.3 Open-source implementations

Numerically stable and efficient implementations of the Lanczos algorithm can be obtained as part of open-source packages.

For Python users, we strongly suggest to use SciPy (in particular scipy.sparse.linalg) which performs Lanczos/Arnoldi calling an efficient, C-coded backend. These routines allow to diagonalize directly sparse matrices defined within scipy. Alternatively, and in order to avoid storing the sparse matrix, one can also define its own Matrix-Vector multiplication using scipy.sparse.linalg.LinearOperator, and then obtain the eigenvalues and eigenvectors with a call to scipy.sparse.linalg.eigsh.

For C++ users, we strongly suggest the use of the EIGEN library[2] in conjunction with SPECTRA[3]. Both libraries are header-only, require almost no installation effort

[2] http://eigen.tuxfamily.org/
[3] http://yixuan.cos.name/spectra/

(apart from downloading it), and are very efficient. SPECTRA handles the Lanczos/Arnoldi algorithm, and just needs the user to implement a function implementing the Matrix-Vector multiplication, with minor modifications with respect to the one previously discussed in the Lecture.