Quantum Information

perina
from the notes of Giovanni Ferrari and Andrea Boido

August 2019

Contents

1 Some linear algebra

Linear algebra is the basis of most of our computations. It is then interesting to analyze the costs of some common linear algebra operations:

Solving a linear system of equations

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = y_1 \\ \vdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n = y_n \end{cases} \longleftrightarrow A\vec{x} = \vec{y} \quad \left(\hat{H}\left|\psi\right\rangle = \left|\phi\right\rangle\right)$$

This can be solved by means of gaussian elimination, which transforms A into a triangular matrix R,

$$R = \begin{pmatrix} & \star \\ 0 & \end{pmatrix}$$

The equations can then be solved from the bottom, in a recursive way:

$$x_N = \frac{y_N}{R_{NN}}, \quad x_{N-1} = \frac{y_{N-1} - R_{N-1,N} x_N}{R_{N-1,N-1}}, \quad \dots \tag{1}$$

and so on. The triangularization requires $\mathcal{O}(N^3)$ operations: we can think of it as setting $N(N-1)/2 \sim N^2$ elements, each requiring N operations. The cost of the recursive solution is comparatively negligible, being linear in N. The fact that triangularization is $\mathcal{O}(N^3)$ bounds all linear algebra, as it is a rather ubiquitous procedure.

Inverting a matrix Given a matrix A, its inverse (if it exists) is the matrix A^{-1} such that

$$AA^{-1} = \mathbb{I}$$

The inversion can be done by performing linear operations on A until it becomes equal to $\mathbb{I}$, and then performing those same operations on $\mathbb{I}$ itself, so that at the end it becomes equal to A^{-1}. This happens because the matrix A^{-1} is composed of unknown entries, and computing them is equivalent to solving a system of systems of equations,

$$AA^{-1} = \mathbb{I} \longleftrightarrow A\vec{x_i} = \vec{y_i} = e_i \quad \forall i$$

using gaussian elimination. In practice, finding the inverse matrix amounts to find the N column vectors $\vec{x}_i$ which solve the system

$$A\vec{x}_i = \vec{y}_i, \tag{2}$$

where the $\vec{x}_i$ compose A^{-1} and the $\vec{y}_i$ compose the identity matrix.

LU reduction Given some matrix A, the LU reduction creates two matrices L and U, lower triangular and upper triangular respectively, such that

$$A = LU$$

U is the matrix R from before, while L is a matrix composed of those coefficients that were used in the gaussian elimination. For instance, the first column of L is simply

$$L_{1j} = \frac{A_{1j}}{A_{11}} \tag{3}$$

but the subsequent values are more complicated. Once LU reduction has been performed, solving the system

$$A\vec{x} = \vec{y}$$

becomes equivalent to solving

$$L\underbrace{U\vec{x}}_{\vec{z}} = \vec{y}$$

$$L\vec{z} = \vec{y}$$

which can be done in $\mathcal{O}(n^2)$ operations. However, the LU decomposition itself requires $\mathcal{O}(n^3)$ operations, while computing $\vec{z}$ takes $\mathcal{O}(n^2)$, so that, in total, the algorithm requires $\sim \mathcal{O}(n^3) + 2\mathcal{O}(n^2)$. As a consequence, using LU decomposition for solving systems is only advisable when there are many systems with the same A but different $\vec{x}$ and $\vec{y}$.

1.1 Algorithms for the eigenvalue problem

We now turn our attention to the eigenvalue problem. One way to try to solve an eigenvalue problem is the power method. We will assume the involved matrices to be hermitian[1]. Power method

We want to solve

$$A\vec{x}_\alpha = \lambda_\alpha \vec{x}_\alpha \longleftrightarrow \hat{H}\left|\psi_\alpha\right\rangle = E_\alpha \left|\psi_\alpha\right\rangle$$

with

$$\vec{x}_\alpha \cdot \vec{x}_\beta = \delta_{\alpha\beta}$$

and, for simplicity, we will assume no degeneracy and $|\lambda_n| > |\lambda_{n-1}| > \cdots > |\lambda_1|$.

Let us start from a random vector, $\vec{y}$. If we assume the existence of an orthogonal basis, as we just did, calling $\vec{x}_i$ the eigenvectors of matrix A, we can write

$$\vec{y} = \sum_i \beta_i \vec{x}_i,$$

but we only know the matrix A and not the β_i. Applying A,

$$A\vec{y} = \sum_i \beta_i \lambda_i \vec{x}_i,$$

and repeating k times,

$$\vec{y}_k = A^k\vec{y} = \sum_i \beta_i \lambda_i^k \vec{x}_i$$

The projection of the vector $\vec{y}$ on the eigenspaces is rescaled at each step in such a way that

$$\vec{y}_k = A^k\vec{y} \xrightarrow[k\to\infty]{} \vec{x}_n$$

where $\vec{x}_n$ is the eigenvector corresponding to the maximum eigenvalue. This is because k subsequent applications of the matrix have the effect of raising each eigenvalue to the k-th power; as a consequence, the larger one will have a

[1] A *hermitian matrix* is a complex square matrix that is equal to its own conjugate transpose

dominant contribution when $k \to \infty$. As a consequence, the β_i become less and less significant, and

$$\frac{|\vec{y}_{k+1}|}{|\vec{y}_k|} \xrightarrow[k\to\infty]{} \lambda_n.$$

The convergence speed is governed by the ratio

$$\frac{\lambda_n}{\lambda_{n-1}}, \tag{4}$$

which, if close to 1, can prevent the algorithm to converge in a meaningful time. Unfortunately, this is not uncommon; for example, phase transitions involve largest and second-largest eigenvalues to get very similar. On an unrelated note, from a mathematical point of view, if we picked the initial vector $\vec{y}$ to lie orthogonally with respect to the eigenspace of the largest eigenvalue, then we would completely miss the largest eigenvalue. However, in practice, finite machine precision makes this a non-issue.

Inverse power method

We can then try to use the inverse power method, i.e., the method applied to the matrix A^{-1}:

$$A\vec{x}_\alpha = \lambda_\alpha \vec{x}_\alpha \iff A^{-1}\vec{x}_\alpha = \lambda_\alpha^{-1}\vec{x}_\alpha \tag{5}$$

and, as we did before,

$$\vec{y}_{k+1} = A^{-1}\vec{y}_k \xrightarrow{k\to\infty} \vec{x_1}$$

which is the eigenvector corresponding to the smallest eigenvalue. Now, the convergence is governed by the ratio

$$\frac{\lambda_1^{-1}}{\lambda_2^{-1}} = \frac{\lambda_2}{\lambda_1}, \tag{6}$$

which is an advantage, as the extreme eigenvalue is now at the denominator (unlike before, when it was at the denominator). Instead of having to calculate A^{-1}, we can write

$$A\vec{y}_{k+1} = \vec{y}_k \tag{7}$$

and solve for y_{k+1}, as, at each step, y_k is given by the previous computation. Through LU decomposition,

$$LU\vec{y}_{k+1} = \vec{y}_k;$$

since we have to apply A k times, this LU decomposition is advantageous, because we can simply solve the system for $\vec{y}_{k+1}$. Then,

$$\frac{|\vec{y}_k|}{|\vec{y}_{k+1}|} \xrightarrow[k\to\infty]{} \frac{1}{\lambda_1}, \quad \vec{y}_k \xrightarrow[k\to\infty]{} \vec{x}_1$$

where λ_1 is the smallest eigenvalue and $\vec{x}_1$ is the corresponding eigenvector. This procedure is often useful because it is usually possible to compute λ_1 analitically (or at least giving it an estimate: think for example of an unperturbed system).

Let us call q the best guess for the value of λ_1. Now, we perform the following change:

$$A \to A - q\mathbb{1},$$

so that

$$A'\vec{x} = (A - q\mathbb{1})\vec{x} = (\lambda - q)\vec{x} = \lambda'\vec{x} \tag{8}$$

Now the first eigenvalue of this matrix is almost 0, and as a consequence the procedure is much faster, because

$$\frac{\lambda_2 - q}{\lambda_1 - q} \gg 1$$

Using these two methods, we managed to find an estimate for the two extreme eigenvalues. In order to get the other eigenvalues, we can do the following. We can effectively remove the $\vec{x}_n$ component from $\vec{y}$, by computing

$$\vec{y}' = \vec{y} - (\vec{y} \cdot \vec{x}_n)\vec{x}_n,$$

and repeat the procedure with

$$\vec{y}' = \sum_{i=1}^{n-1} \beta_i \vec{x}_i$$

to obtain $\vec{x}_{n-1}$ and so on. Equvalently, we can go with x_1, and remove components from there upwards. However, the problem in finding the next eigenvector is that when one tries to eliminate the $\vec{x}_1$ component from $\vec{y}$, errors can occur due to the finite precision, and the result is often $\vec{x}_1$ once again.

The various costs are summed up in the following table.

Operation	**Cost**	
Matrix-vector multiplication	$\mathcal{O}(n^2)$	
Matrix-matrix multiplication	$\mathcal{O}(n^3)$	
Gaussian elimination	$\mathcal{O}(n^3)$	
Inverse matrix computation	$\mathcal{O}(n^3)$	
LU decomposition	$\mathcal{O}(n^3)$	
Eigenvalue (power method)	$M \cdot \mathcal{O}(n^2)$	
Eigenvalue (inverse power method)	$\mathcal{O}(n^3) + m \cdot \mathcal{O}(n^2)$	(with $m \ll M$)
Determinant	$\mathcal{O}(n!)$ with Laplace, $\mathcal{O}(n^3)$ with LU decomposition	

Table 1: Summary of some common linear algebra operations and their cost.

1.2 Monster matrices and tridiagonalization

Other important objects are tridiagonal matrices: Tridiagonal matrices

$$T = \begin{pmatrix} a_1 & b_1 & & & 0 \\ b_1 & a_2 & & & \\ & & \ddots & & \\ & & & a_{n-1} & b_{n-1} \\ 0 & & & b_{n-1} & a_n \end{pmatrix}$$

It holds:

$$\det(T_n - \lambda \mathbb{I}) = (a_n - \lambda)\det(T_{n-1} - \lambda \mathbb{I}) - b_{n-1}^2 \det(T_{n-2} - \lambda \mathbb{I})$$

(just use Laplace's formula starting from the bottom right corner). Then, expressing it as a function of λ, we get

$$f_n(\lambda) = (a_n - \lambda) f_{n-1}(\lambda) - b_{n-1}^2 f_{n-2}(\lambda)$$

which is a recursive relation that can be easily solved.

Tridiagonal matrices appear naturally during the solutions of some problems, but we should also be able to transform any given matrix into a tridiagonal one; to do so, we can employ a change of coordinates. Since we will often have to deal with *monster matrices* (matrices so large that they do not fit in the RAM), we have to find a way to tridiagonalize matrices without explicitly writing the change of coordinates. This is possible thanks to the following algorithm, which is due to Lanczos (and is known as Lanczos algorithm). Lanczos algorithm

We start from a random normalized vector $\vec{y}_1$. Since we are dealing with a 'monster sized' matrices/vectors, $\vec{y}_1$ can be written with some effort; a vector of vectors (i.e., a matrix) can not. Let us consider a hermitian monster matrix M. We can then compute

$$M\vec{y}_1 = a_1 \vec{y}_1 + b_1 \vec{y}_2,$$

with $\vec{y}_1 \perp \vec{y}_2$. Then,

$$a_1 = (M\vec{y}_1) \cdot \vec{y}_1,$$

and we then define the remaining (not normalized) part as

$$\vec{z} = M\vec{y}_1 - a_1 \vec{y}_1$$

To normalize it, we simply write

$$\vec{y}_2 = \frac{\vec{z}}{b_1}$$

where $\vec{z} \cdot \vec{z} = b_1^2$. As a result, $|\vec{y}_2| = 1$. It is always possible to decompose the vector $M\vec{y}_1$ as above, in such a way that $\vec{y}_2$ is normalized. We can now repeat the procedure with $\vec{y}_2$. Since M is hermitian, it holds

$$\langle \vec{y}_2 | M\vec{y}_1 \rangle = b_1 \iff \langle M\vec{y}_2 | \vec{y}_1 \rangle = b_1$$

and so the coefficient of the $\vec{y}_1$ component of $M\vec{y}_2$ is precisely b_1; for the other coefficients, we once again denote them as a and b:

$$M\vec{y}_2 = b_1\vec{y}_1 + a_2\vec{y}_2 + b_2\vec{y}_3, \quad \vec{y}_1 \perp \vec{y}_2, \vec{y}_2 \perp \vec{y}_3, \vec{y}_1 \perp \vec{y}_3,$$

or, in a more 'constructive' definition (recall that we are progressively obtaining new $\vec{y}_i$),

$$\vec{z}' = M\vec{y}_2 - b_1\vec{y}_1 - a_2\vec{y}_2 = b_2\vec{y}_3,$$

and so

$$\vec{z}'^2 = b_2^2 \longrightarrow \vec{y}_3 = \frac{\vec{z}'}{b_2}$$

Once again, we can use the hermitian property in order to compute the value of the coefficient of the $\vec{y}_2$ component. Note that the vectors $\vec{y}_i$ must be normalized to 1; this can be assumed without loss of generality, as the norm can be absorbed inside the coefficients themselves. Applying M to $\vec{y}_3$ yields:

$$M\vec{y}_3 = \alpha\vec{y}_1 + b_2\vec{y}_3 + a_3\vec{y}_2 + b_3\vec{y}_4$$

with all the $\vec{y}$s being orthogonal. However, the value of α is actually 0, as can be shown by direct calculation using the orthogonality relations:

$$\alpha = \vec{y}_1 \cdot M\vec{y}_3 = \vec{y}_1 \cdot \frac{M\vec{y}_2 - b_1\vec{y}_1 - a_2\vec{y}_2}{b_2} = \vec{y}_1 \cdot \frac{(b_1\vec{y}_1 + a_2\vec{y}_2 + b_2\vec{y}_3) - b_1 - 0}{b_2} = \frac{b_1 - b_1}{b_2} = 0$$

As a consequence,

$$M\vec{y}_3 = b_2\vec{y}_2 + a_3\vec{y}_3 + b_3\vec{y}_4$$

Through this procedure, we get

$$M\vec{y}_k = b_{k-1}\vec{y}_{k-1} + a_k\vec{y}_k + b_k\vec{y}_{k+1}$$

So, in the $\vec{y}_k$ basis, the matrix M becomes

$$T = \begin{pmatrix} a_1 & b_1 & & & 0 \\ b_1 & a_2 & & & \\ & & \ddots & & \\ & & & a_{n-1} & b_{n-1} \\ 0 & & & b_{n-1} & a_n \end{pmatrix} = P^\dagger M P$$

where

$$P = \begin{pmatrix} \vdots & & \vdots \\ y_1 & \cdots & y_n \\ \vdots & & \vdots \end{pmatrix}$$

By using this algorithm, we do not ever need to explicitly calculate P or $P^\dagger MP$, because only the coefficients matter, and they are precisely what we get by applying the algorithm.

In practice, storing the tridiagonalized matrix T requires two vectors, while at most three $\vec{y}_i$ need to be kept in memory at any time. As a consequence, this algorithm only requires about 5 'monster vectors' in the RAM to run.

However, how can we compute $M\vec{y}_i$ without storing the whole M? The idea is that we do not actually need the whole matrix; in general, the action of a matrix on a vector can be expressed without a direct product. Take the following as an example.

Let $\mathcal{H}$ be a hamiltonian over a chain of spins-$\frac{1}{2}$. Its expression is

$$\mathcal{H} = \sum_{i=1}^{N-1} \hat{\sigma}_i^x \hat{\sigma}_{i+1}^x,$$

where $\hat{\sigma}_i^x$ is the operator which projects the (i)-th spin on the x direction. These operators act on a Hilbert space of dimension 2; if we employ the basis of the (normalized) eigenvectors of the z component of the spin (which, by convention, is the usual choice), the operator $\hat{\sigma}_i^x$ takes on the following form:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Let us now consider the case for $N = 2$; in this case, the 'spin chain' is simply composed of one interaction term:

$$\mathcal{H} = \hat{\sigma}_1^x \hat{\sigma}_2^x,$$

and we face two possible courses of action:

1. simply write the hamiltonian operator using the tensor product; this is a two body system, so

   $$\mathcal{H} = \hat{\sigma}_1^x \otimes \hat{\sigma}_2^x,$$

   and so, using the tensor product of the two bases, we can represent this operator as

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

   Note that applying this operator to any wavefunction will result in a flipping of all spins. Saving such a matrix representation requires a total of $(dN)^2$ complex numbers;

2. we can otherwise directly write the actions of the operators:

   $$\hat{\sigma}_1^x \hat{\sigma}_2^x |\psi\rangle = \hat{\sigma}_1^x \hat{\sigma}_2^x (\psi_{00} |00\rangle + \psi_{01} |01\rangle + \psi_{10} |10\rangle + \psi_{11} |11\rangle)$$

   and, since $\hat{\sigma}_i^x$ acts only on the i-th spin (in a brutish way, 'on the i-th number in the ket), we get

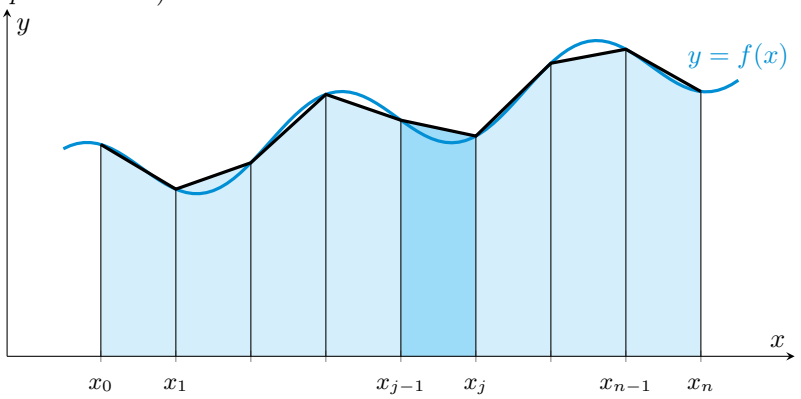
   $$\hat{\sigma}_1^x \hat{\sigma}_2^x |\psi\rangle = \psi_{00} |11\rangle + \psi_{01} |10\rangle + \psi_{10} |01\rangle + \psi_{11} |00\rangle ,$$

   which means we only need to store $(d^2 N)$ numbers.

It is then possible to have a more convenient way to store a monster matrix.

1.3 Integration

Another important problem is *integration*. Let us consider a function $f(x)$. In order to estimate the area laying under it, we can approximate it with trapezoids (*trapezoidal rule*). Trapezoid rule



So, the calculation will be (in the case of equally spaced points, so that $x_{i+1} - x_i = h, \quad \forall i$),

$$\int_{x_0}^{x_n} f(x)dx \simeq \sum_{i=0}^{n} A_i = \frac{1}{2}\sum_{i=0}^{n} \left[f(x_i) + f(x_{i+1})\right](x_{i+1} - x_i) =$$
$$\frac{1}{2}h\sum_{i=0}^{n} \left[f(x_i) + f(x_{i+1})\right] = h\left[\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n)\right] =$$
$$h\left[\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{n-1}) + \frac{1}{2}f(x_n)\right]$$

As this is not an exact result, it is important to estimate the error; using Taylor's expansion,

$$f(x) = f(0) + xf'(0) + \frac{1}{2}x^2 f''(0) + \cdots$$
$$\Longrightarrow \int_{-\frac{h}{2}}^{\frac{h}{2}} dx f(x) = hf(0) + \frac{h^3}{24}f''(0) + \cdots$$

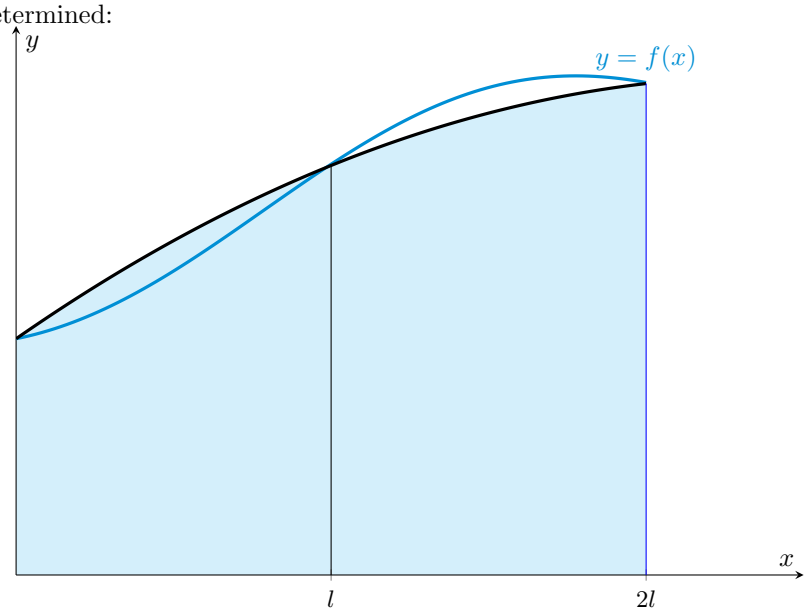
as only even terms survive the integration; also,

$$f\left(\pm\frac{h}{2}\right) = f(0) \pm \frac{h}{2}f'(0) + \frac{h^2}{8}f''(0) + \cdots$$
$$\Longrightarrow f(0) = \frac{1}{2}\left[f\left(\frac{h}{2}\right) + f\left(-\frac{h}{2}\right)\right] - \frac{h^2}{8}f''(0) + \cdots$$
$$\Longrightarrow \int_{-\frac{h}{2}}^{\frac{h}{2}} dx f(x) = \frac{h}{2}\left[f\left(\frac{h}{2}\right) + f\left(-\frac{h}{2}\right)\right] + \mathcal{O}(h^2)$$

What this last line is telling us is that the integration on the interval $\left[-\frac{h}{2}, \frac{h}{2}\right]$ is equal to the trapezoid approximation plus a remainder $\sim \mathcal{O}(h^2)$. The error is $\sim \mathcal{O}(h^2)$.

Simpson's rule

A possible improvement is the *Simpson's rule*, which consists in approximating each interval with a parabola. Each parabola needs three points to be determined:



In this case, we approximate as follows:

$$\int_0^{2l} dx f(x) = \frac{h}{3}\left[f_0 + 4f_l + f_{2l}\right]$$

and, in general,

$$\int_{x_0}^{x_n} dx f(x) = \sum_{i=0}^{n} A_i \simeq h\left[\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}f_2 + \frac{4}{3}f_3 + \cdots + \frac{4}{3}f_{n-1} + \frac{1}{3}f_n\right]$$

As can be seen from the formula, this method is not slower than the other one: only sums and multiplications are involved. However, it adds an order of precision:

$$\int_{-h}^{h} dx f(x) = 2hf(0) + \frac{h^3}{3}f''(0) + \frac{h^5}{60}f^{(IV)}(0) + \cdots$$

but also

$$f(\pm h) = f(0) \pm hf'(0) + \frac{h^2}{2}f''(0) + \cdots$$

so that, by summing $f(h)$ and $f(-h)$ we get that

$$h^2 f''(0) = f(h) - 2f(0) + f(-h) + \mathcal{O}(h^4) \Longrightarrow f''(0) = \frac{f(h) - 2f(0) + f(-h)}{h^2} + \mathcal{O}(h^2)$$

and substituting in the integral above, we get

$$\int_{-h}^{h} dx f(x) = 2hf(0) + \frac{h}{3}\left[f(h) - 2f(0) + f(-h)\right] + \mathcal{O}(h^5) = \frac{h}{3}\left[f(h) + 4f(0) + f(-h)\right] + \mathcal{O}(h^5)$$

The first term in the right hand side is exactly the result of the Simpson's rule; by reasoning as before, we conclude that the error is $\sim \mathcal{O}(h^5)$.

We can actually improve the estimate by changing the coefficient A_i:

$$\int_a^b dx f(x) \simeq \sum_{i=0}^{n} A_i f(x_i) = \sum_{i=0}^{n} A_i f_i,$$

where $x_i = a + ih$. There are some constraints on the coefficients A_i; indeed, we want to obtain exact results when the approximation is performed on a polynomial function, up to a certain order. Since

- $\int_a^b dx = b - a = \sum_{i=0}^{n} A_i f_i = \sum_{i=0}^{1} A_i$
- $\int_a^b x dx = \frac{b^2 - a^2}{2} = \sum_{i=0}^{n} A_i x_i$

  $\vdots$
- $\int_a^b x^m dx = \frac{b^{m+1} - a^{m+1}}{m+1} = \sum_{i=0}^{n} A_i x_i^m$

For $a = -\frac{h}{2}, b = \frac{h}{2}$ we want to impose, for a first order approximation,

$$\begin{cases} \sum_{i=0}^{1} A_i = \frac{h}{2} - \left(-\frac{h}{2}\right) \\ \sum_{i=0}^{1} A_i x_i = \frac{\left(\frac{h}{2}\right)^2 - \left(-\frac{h}{2}\right)^2}{2} \end{cases} \implies \begin{cases} A_1 + A_2 = h \\ -A_1 \frac{h}{2} + A_2 \frac{h}{2} = 0 \end{cases}$$

which is solved by

$$A_1 = A_2 = \frac{h}{2}$$

which leads to the trapezoid rule. With three points, $(-h, 0, h)$, and a second order polynomial approximation, we get

$$\begin{cases} \sum_{i=0}^{2} A_i = h - (-h) \\ \sum_{i=0}^{2} A_i x_i = \frac{h^2 - (-h)^2}{2} \\ \sum_{i=0}^{2} A_i x_i^2 = \frac{h^3 - (-h)^3}{3} \end{cases} \implies \begin{cases} A_1 + A_2 + A_3 = 2h \\ -A_1 h + A_3 h = 0 \\ A_1 h^2 + A_3 h^2 = \frac{2h^3}{3} \end{cases}$$

The solution of this system is

$$A_1 = A_3 = \frac{h}{3}, \qquad A_2 = \frac{4}{3}h$$

which is exactly Simpson's rule.

Another possible approach consists in letting the choice of the points x_i to be free, instead of choosing a fixed interval h for the discretization.

Legendre polynomials: Gaussian integration

By rescaling the integration variable, we can always have $a = -1, b = 1$. In this case, the previous constraints become

$$\sum_{i=0}^{n} A_i x_i^m = \begin{cases} 0 & \text{if } m \text{ is odd,} \\ \frac{2}{m+1} & \text{if } m \text{ is even} \end{cases}$$

We choose an orthogonal basis for the functions defined on the interval $[-1, 1]$: the *Legendre polynomials* $P_n(x)$. In this space, a scalar product is defined as follows:

$$\langle P_m | P_{m'} \rangle = \int_{-1}^{1} dx P_m(x) P_{m'}(x) = c_{mm'} \delta_{mm'}$$

where the constant $c_{mm'}$ is due to the fact that we do not require the $P_m(x)$ to be normalized.

We define

$$\begin{cases} P_0(x) = 1 \\ P_1(x) = x \end{cases}$$

so that

$$\langle P_0 | P_1 \rangle = \int_{-1}^{1} x dx = 0$$

In general,

$$P_n(x) = \sum_{k=0}^{n} a_{nk} x^k$$

Moreover,

$$x^m = \sum_{k=0}^{m} b_{mk} P_k(x)$$

and then

$$\langle P_n | x^m \rangle = 0 \qquad \forall \quad m < n$$

by using the orthogonality properties.

$P_n(x)$ has exactly n points y_i for which

$$P_n(y_i) = 0$$

Suppose now that we want to discretize an integral using n points. Choosing the coefficients A_i in a proper way, we can set

$$\begin{cases} E_0(\{x_i\}) = \sum_{i=1}^{n} A_i - \int_{-1}^{1} dx = 0 \\ E_1(\{x_i\}) = \sum_{i=1}^{n} A_i x_i - \int_{-1}^{1} x dx = 0 \\ \vdots \\ E_{n-1}(\{x_i\}) = \sum_{i=1}^{n} A_i x_i^{n-1} - \int_{-1}^{1} x^{n-1} dx = 0 \end{cases}$$

By also choosing the points x_i in a proper way, we can double our precision. For a certain integer $k < n$ we define

$$\begin{cases} E_n(\{x_i\}) = \sum_{i=1}^{n} A_i x_i^n - \int_{-1}^{1} x^n dx = Z_0(\{x_i\}) \\ \vdots \\ E_{n+k}(\{x_i\}) = \sum_{i=1}^{n} A_i x_i^{n+k} - \int_{-1}^{1} x^{n+k} dx = Z_k(\{x_i\}) \end{cases}$$

The purpose of the following method is to set $Z_0, \cdots, Z_k$ to 0. To do so, we write the following equations:

$$\begin{cases} a_{n,0} E_0(\{x_i\}) = 0 \\ \vdots \\ a_{n,n-1} E_{n-1}(\{x_i\}) = 0 \\ a_{n,n} E_n(\{x_i\}) = a_{n,n} Z_0(\{x_i\}) \end{cases}$$

and then we sum them up:

$$\begin{aligned} \sum_{m=0}^{n} a_{n,m} E_m(\{x_i\}) &= a_{n,n} Z_0(\{x_i\}) \\ \sum_{m=0}^{n} a_{n,m} \left[\sum_{i=1}^{n} A_i x_i^m - \int_{-1}^{1} x^m dx \right] &= a_{n,n} Z_0(\{x_i\}) \\ \sum_{i=1}^{n} A_i \sum_{m=0}^{n} a_{n,m} x_i^m - \sum_{m=0}^{n} \int_{-1}^{1} a_{n,m} x^m dx &= a_{n,n} Z_0(\{x_i\}) \\ \sum_{i=1}^{n} A_i P_n(x_i) - \int_{-1}^{1} P_n(x) dx &= a_{n,n} Z_0(\{x_i\}) \\ \sum_{i=1}^{n} A_i P_n(x_i) - \int_{-1}^{1} 1 \cdot P_n(x) dx &= a_{n,n} Z_0(\{x_i\}) \\ \sum_{i=1}^{n} A_i P_n(x_i) - \langle P_n | P_0 \rangle &= a_{n,n} Z_0(\{x_i\}) \\ \sum_{i=1}^{n} A_i P_n(x_i) &= a_{n,n} Z_0(\{x_i\}) \end{aligned}$$

Now, if

$$\begin{cases} (\{x_i\}) = (\{y_i\}), \\ P_n(y_i) = 0 \qquad \forall \quad i, \end{cases}$$

we have obtained

$$Z_0(\{x_i\}) = 0$$

Now, repeating the procedure, starting from E_1 up to E_{n+1}, we get

$$\begin{cases} a_{n,1}E_1(\{x_i\}) = 0 \\ \vdots \\ a_{n,n-1}E_n(\{x_i\}) = 0 \\ a_{n,n+1}E_{n+1}\{x_i\}) = a_{n,n}Z_1(\{x_i\}) \end{cases}$$

Once again, summing them all up:

$$\begin{aligned} \sum_{m=0}^{n} a_{n,m}E_{m+1} &= a_{n,n}Z_1(\{x_i\}) \\ \sum_{m=0}^{n} a_{n,m}\left[\sum_{i=1}^{n} A_i x_i^{m+1} - \int_{-1}^{1} x^{m+1}dx\right] &= a_{n,n}Z_1(\{x_i\}) \\ \sum_{i=1}^{n} A_i \sum_{m=0}^{n} a_{n,m}x_i^{m+1} - \sum_{m=0}^{n} a_{n,m}\int_{-1}^{1} x^{m+1}dx &= a_{n,n}Z_1(\{x_i\}) \\ \sum_{i=1}^{n} A_i \left(\sum_{m=0}^{n} a_{n,m}x_i^m\right) x_i - \int_{-1}^{1}\left(\sum_{m=0}^{n} a_{n,m}x^m\right) x dx &= a_{n,n}Z_1(\{x_i\}) \\ x_i \sum_{i=1}^{n} A_i P_n(x_i) - \int_{-1}^{1} P_n(x)x dx &= a_{n,n}Z_1(\{x_i\}) \\ x_i \sum_{i=1}^{n} A_i \underbrace{P_n(x_i)}_{0} - \underbrace{\langle P_n|P_1\rangle}_{0} &= a_{n,n}Z_1(\{x_i\}) \\ \implies Z_1(\{x_i\}) &= 0 \end{aligned}$$

We can continue up to $k = n - 1$. By choosing $(\{x_i\})$ as the n zeros of $P_n(x)$, we have reached a precision up to the $(2n - 1)$-th order.

1.4 Numerical solutions of differential equations

The next task is to solve a differential equation. Obviously, the most interesting one is Schrödinger's equation (in its time independent form):

$$-\frac{\hbar^2}{2m}\frac{\partial\psi}{\partial x^2} + V(x)\psi(x) = E\psi(x)$$

For discrete systems, we basically have to diagonalize a matrix, so that we return to the previous arguments. For continuous systems, on the other hand, we first have to perform an appropriate discretization.

Let us consider a basis of eigenstate $\{|i\rangle\}$, such that

$$|\psi\rangle = \sum_i \psi_i |i\rangle$$

and

$$\hat{H}\,|\psi\rangle = E\,|\psi\rangle$$
$$\implies \hat{H}\sum_i \psi_i\,|i\rangle = E\sum_i \psi_i\,|i\rangle$$

From now on, we will consider various algorithms for solving it. More in general, starting from an equation of the type

$$\psi''(x) = -k(x)\psi(x),$$

which is Schrödinger's equation if $k(x) = \frac{2m}{\hbar}(E - V(x))$, we can discretize our space in a homogeneous way:

$$\psi_n = \psi(x_n), \qquad x_n = n\Delta x$$

In our case, it holds

$$k(x) = \frac{2m}{\hbar^2}\left[E - V(x)\right]$$

Now, from a Taylor expansion,

$$\psi_{n\pm1} = \psi_n \pm \Delta x\psi'_n + \frac{\Delta x^2}{2}\psi''_n \pm \frac{\Delta x^3}{6}\psi'''_n + \frac{\Delta x^4}{24}\psi^{\mathrm{iv}}_n \pm \frac{\Delta x^5}{120}\psi^{\mathrm{v}}_n + \mathcal{O}(\Delta x^6)$$

so we can write

$$\psi_{n+1} + \psi_{n-1} = 2\psi_n + \Delta x^2\psi''_n + \frac{\Delta x^4}{12}\psi^{\mathrm{iv}}_n + \mathcal{O}(\Delta x^6)$$

Recall that the incremental ratio definition of the derivative still holds, so that

$$\psi^{\mathrm{iv}}_n = \frac{\psi''_{n+1} + \psi''_{n-1} - 2\psi''_n}{\Delta x^2}$$

therefore

$$\psi_{n+1} + \psi_{n-1} = 2\psi_n + \Delta x^2\psi''_n + \frac{\Delta x^2}{12}\left(\psi''_{n+1} + \psi''_{n-1} - 2\psi''_n\right) + \mathcal{O}(\Delta x^6)$$

and using the differential equation

$$\psi''_n = -k_n\psi_n \qquad (k_n = k(x_n))$$

we get, after substituting,

$$\psi_{n+1} + \psi_{n-1} = 2\psi_n - \Delta x^2 k_n\psi_n - \frac{\Delta x^2}{12}\left(k_{n_1}\psi_{n+1} + k_{n-1}\psi_{n-1} - 2k_n\psi_n\right) + \mathcal{O}(\Delta x^6)$$
$$\implies \left(1 + \frac{\Delta x^2}{12}k_{n+1}\right)\psi_{n+1} = \left(2 - \frac{5}{6}k_n\Delta x^2\right)\psi_n - \left(1 + \frac{\Delta x^2}{12}k_{n-1}\right)\psi_{n-1}$$

Hence, once two consecutive values of the wave function are known, the neighboring ones are also known, with a precision of order $\mathcal{O}(\Delta x^6)$. So, from two

known values, one can "expand" his knowledge of the wave, with some degree of error. However, there are two problems: in principle, we have to know E, and also a priori know two values of the wave function. However, since the Schrödinger equation is linear, the wave function can be arbitrarily renormalized. As a consequence, we only need to know the ratio between two consecutive values of the wave function. There are a few relevant cases where this is possible:

1. **Even potential**: $V(x) = V(-x)$
   We can simply choose a discretization grid, such that it is symmetric with respect to 0 and it does not contain it. Then, we know that
   $$\psi(x_{-1/2}) = \psi(x_{1/2})$$
   because when the potential $V(x)$ is symmetric, so has to be ψ.

2. **Odd potential**: $V(x) = -V(-x)$
   This is similar to the previous case, except that now
   $$\psi(x_{-1/2}) = -\psi(x_{1/2})$$

3. **Potential well**: $E < 0, \quad V(x) = 0$ outside a certain region.
   In this case, we know that, outside the well, the wave function must decay exponentially. We expect that
   $$\psi(x) \sim e^{-\frac{1}{\hbar}\sqrt{-2mE}x},$$
   and so
   $$\psi(x_{n+1}) = \psi(x_n)e^{-\frac{1}{\hbar}\sqrt{-2mE}\Delta x}$$
   from which we can get the ratio between two consecutive values. This approach is dependent on the energy we are dealing with. Initializing the algorithm with a too high E, the wave function will diverge at $+\infty$. Choosing it too low, instead, will make the function diverge at $-\infty$.

What should we do when $D > 1$? Even though this algorithm can be generalized to more than one dimension, one should always try to reduce the dimensionality of the problem at hand in order to simplify it. This can be done, for instance, by:

- taking advantage of symmetries;
- factorizing the potential when possible, for instance if $V(x, y, z) = V(x) + V(y) + V(z)$;
- changing coordinates.

When it is impossible to know the ratio between consecutive values of ψ, other methods should be employed. For instance, one alternative is the *finite difference method*. It needs a grid, which we will assume to be cubic. The wave

Finite difference method

function will be evaluated at the points of this lattice. Let us set $\hbar = 1$, and the lattice constant[2] to h. Then, the discretized version of the Laplacian is

$$\left(-\nabla^2 + e\right)\psi = -\frac{1}{h^2}\left[\psi(x_{n+1}, y_n, z_n) + \psi(x_n, y_{n+1}, z_n) + \psi(x_n, y_n, z_{n+1})\right] + \\ + \left[e(x_n, y_n, z_n) + \frac{6}{h^2}\right]\psi(x_n, y_n, z_n)$$

where

$$e(x_n, y_n, z_n) = V(x_n, y_n, z_n) - E$$

We then impose

$$H\vec{\psi} = 0,$$

with $H = \nabla^2 + e$.

This is an eigenvalue problem, and the dimension of $\vec{\psi}$ depends on how many points we choose to include in our grid. Usually, H is represented by a sparse matrix (i.e., mostly composed of null entries). For example, let us consider a free particle in 1D, using the discretized Laplacian as shown above:

$$\frac{1}{h^2}\left[\psi_{n+1} - \psi_{n-1}\right] + \frac{2}{h^2}\psi_n = E\psi_n$$

$$\implies H = \frac{1}{h^2}\begin{pmatrix} 2-h^2E & -1 & & & 0 \\ -1 & 2-h^2E & & & \\ & & \ddots & & \\ & & & 2-h^2E & -1 \\ 0 & & & -1 & 2-h^2E \end{pmatrix}$$

which is a tridiagonal matrix.

If we allow for more general grids, the algorithm is called *finite elements method*.

Another approach is the use of *variational methods* (one such method is due to Hartree-Fock). We will use a finite basis set $|u_i\rangle$ of functions, which are clearly not eigenfunctions of H. We can always write

Variational methods

$$|\phi\rangle = \sum_{i=1}^{n} a_i |u_i\rangle, \qquad E = \frac{\langle\phi|H|\phi\rangle}{\langle\phi|\phi\rangle}$$

The matrix elements of the hamiltonian are given by

$$H_{ij} = \langle u_i|H|u_j\rangle = \int d\vec{r} u_i^*(\vec{r})\left(-\frac{\hbar^2}{2m}\nabla^2 + V(\vec{r})\right)u_j(\vec{r})$$

One of the most expensive parts of the whole computation is this integral. A possible solution is for instance considering simple functions (e.g. gaussian

[2]Informally speaking, the 'step length' of the lattice.

functions), but this may force us into considering more functions (and so there is a sort of tradeoff).

We can define the correlation matrix as

$$S_{ij} = \langle u_i | u_j \rangle = \int d\vec{r} u_i^*(\vec{r}) u_j(\vec{r}),$$

which gives information about the overlap of functions (orthogonality of the basis is not required), and solve the generalized eigenvalue problem

$$H\vec{a} = SE\vec{a}$$

Obviously, for an orthonormal basis, we have

$$S = \mathbf{I},$$

and we have to solve the simple eigenvalue problem

$$H\vec{a} = E\vec{a}.$$

In general, this is not the case; one can however search for the matrix $\mathbf{U}$ such that

$$\mathbf{U}^T \mathbf{S} \mathbf{U} = \mathbf{I}$$

$\mathbf{U}$ is then the basis change matrix:

$$\vec{b} = \mathbf{U}^{-1}\vec{a} \Longrightarrow \vec{a} = \mathbf{U}\vec{b}$$

It is then possible to solve

$$\mathbf{U}^T H \mathbf{U} \vec{b} = E\mathbf{U}^T S \mathbf{U} \vec{b} \Longrightarrow H'\vec{b} = E\vec{b}$$

and then go back to $\vec{a}$. However, this change of coordinates may end up destroying some desirable properties of the hamiltonian.

Example: let us consider the anharmonic oscillator

$$H = \frac{1}{2}(\hat{p}^2]\omega^2\hat{q}^2) + \lambda\hat{q}^4$$

We can start by considering the solutions of the harmonic oscillator,

$$|u_n\rangle = |n\rangle\,,$$

and then rewrite the hamiltonian as follows:

$$\hat{H} = \frac{1}{2} + \hat{n} + \frac{\lambda}{4}(a^\dagger + a)^4$$

The first two terms are diagonal in the chosen basis. The last term is not diagonal, but contains powers of a and $a^\dagger$ up to the fourth order. As a consequence, its matrix has 9 non-vanishing diagonals,

$$\hat{H} =$$

and this simplifies the problem a lot from a computational point of view.

2 Time dependent Schrödinger equation

Let us now consider the time dependent Schrödinger equation. There are many methods to try and solve this equation.

2.1 Spectral method

Spectral method

The first one is the *spectral method*, based on the eigenfunctions computed accordingly to the time independent Schrödinger equation. We can always write

$$|\psi(t_0)\rangle = \sum_n c_n |\phi_n\rangle$$

and at arbitrary time t we will have

$$|\psi(t)\rangle = \sum_n c_n e^{-i\frac{E_n(t-t_0)}{\hbar}} |\phi_n\rangle$$

Unfortunately, this method can only be applied to simple systems, such as 1D continuous systems or higher dimensional discrete systems.

2.2 Direct integration

Direct numerical integration

Another tool we can use is *direct numerical integration*: we discretize time between t_0 and t and then we integrate ψ according to the Schrödinger equation. The first step is to choose a Δt that is small enough. We can estimate its size by using the indetermination principle:

$$\Delta t \approx \frac{1}{E}$$

Then, for each time step Δt, we have to compute

$$|\psi(x, t+\Delta t)\rangle = e^{-i\hat{H}\Delta t} |\psi(x,t)\rangle$$

The simplest way to compute the function of an operator (as the exponential in the expression above) is to diagonalize it, compute the function on its eigenvalues, and then go back to the former basis:

$$f(A) = f(P^{-1}DP) = P^{-1}f(D)P,$$

where

$$f(D) = \begin{pmatrix} f(\lambda_1) & & \mathbb{0} \\ & \ddots & \\ \mathbb{0} & & f(\lambda_n) \end{pmatrix}$$

However, this has a high computational cost. It is more convenient to just expand the exponential:

$$e^{-i\hat{H}\Delta t} = \mathbb{1} - i\hat{H}\Delta t + \mathcal{O}(\Delta t^2)$$

so that

$$|\psi(x,t+\Delta t)\rangle \approx |\psi(x,t)\rangle - i\Delta t\hat{H}\,|\psi(x,t)\rangle \quad \text{(Euler method)}$$

The problem of this approach is that now the time evolution operator is no longer unitary; as a consequence, the norm of the wave function is not conserved. A possible solution is to renormalize ψ at each time step. This is equivalent to a projection of $\psi(x,t+\Delta t)$, which does not belong to the manifold of unitary vectors, on the original manifold. Even though it may seem reasonable enough, there is no control over the approximation we introduce, and this procedure may in principle lead to completely wrong results.

2.3 Implicit method

A method which fixes this is the *implicit method*. We write

Implicit method

$$e^{-i\hat{H}\Delta t} = e^{-i\hat{H}\frac{\Delta t}{2}}e^{-i\hat{H}\frac{\Delta t}{2}} = \left(e^{i\hat{H}\frac{\Delta t}{2}}\right)^{-1}\left(e^{-i\hat{H}\frac{\Delta t}{2}}\right) \approx \left(\mathbb{1}+i\hat{H}\frac{\Delta t}{2}\right)^{-1}\left(\mathbb{1}-i\hat{H}\frac{\Delta t}{2}\right)$$

The operator we are left with is unitary, and there is no need for renormalization. However, in principle, calculating the inverse of an operator can be a tough challenge. Hence, we write

$$|\psi(x,t+\Delta t)\rangle = \left(\mathbb{1}+i\hat{H}\frac{\Delta t}{2}\right)^{-1}\left(\mathbb{1}-i\hat{H}\frac{\Delta t}{2}\right)|\psi(x,t)\rangle \quad \text{(Cranck-Nicholson)}$$

$$\left(\mathbb{1}+i\hat{H}\frac{\Delta t}{2}\right)|\psi(x,t+\Delta t)\rangle = \left(\mathbb{1}-i\hat{H}\frac{\Delta t}{2}\right)|\psi(x,t)\rangle$$

We notice that, on the left hand side of the equation, we have a known (and single) operator that acts on an unknown object; on the right hand side, instead, we have a known vector. Since this vector will change at each time step, while the operator will stay the same across iterations, it is convenient to LU decompose this operator, and then solve the equation like a linear system of equations.

2.4 Split operator method

Another useful method, based on the Fourier transform of the state, is the *split operator method*. Let us consider

Split operator method

$$\hat{H} = \frac{p^2}{2m} + V(x)$$

We have

$$e^{-i\hat{H}\Delta t} = e^{-i\left(\frac{p^2}{2m}+V(x)\right)\Delta t} =$$

$$= e^{-i\frac{\hat{V}\Delta t}{2}}e^{-i\hat{T}\Delta t}e^{-i\frac{\hat{V}\Delta t}{2}} + \mathcal{O}(\Delta t^3)$$

We know that $\hat{V}$ is diagonal in the x space, while $\hat{T}$ is diagonal in the p space: splitting $\hat{H}$ seems then reasonable enough. We then have

$$\begin{cases} e^{-i\frac{\hat{V}\Delta t}{2}}\left|\psi(x)\right\rangle = e^{-i\frac{V(x)\Delta t}{2}}\left|\psi(x)\right\rangle \\ \\ e^{-i\hat{T}\Delta t}\left|\psi(p)\right\rangle = e^{-i\left(\frac{p^2}{2m}\right)\Delta t}\left|\psi(p)\right\rangle \end{cases}$$

where $|\psi(p)\rangle$ is the Fourier transform of $\psi(x)$. Hence, we perform the following operation

$$|\psi(x,t+\Delta t)\rangle = e^{-i\frac{V(x)\Delta t}{2}}\mathcal{F}^{-1}e^{-i\frac{p^2}{2m}\Delta t}\mathcal{F}e^{-i\frac{V(x)\Delta t}{2}}\left|\psi(x,t)\right\rangle$$

This operation is trivial, except for the Fourier transform. There are however many excellent algorithms which allow its calculation in $\mathcal{O}(n\log n)$ time.

2.5 Runge-Kutta algorithm

Runge-Kutta algorithm

The last algorithm we are going to see is the *Runge-Kutta algorithm*. Let us start from the time dependent Schrödinger equation

$$i\dot{\psi}(t) = H\psi(t)$$

and, more in general, for an equation of the form

$$\frac{\partial\psi}{\partial t} = \mathcal{L}(\psi,t)$$

We integrate

$$|\psi(t+d\tau)\rangle - |\psi(t)\rangle = \int_t^{t+d\tau}\mathcal{L}(\psi(t'),t')dt'$$

In principle we can calculate this integral numerically, with an arbitrary choice of the rule (either trapezoid, Simpson's...)

$$\int_t^{t+d\tau}\mathcal{L}(\psi(t'),t')dt' \approx d\tau\left[a\mathcal{L}(\psi(t)) + b\mathcal{L}(\psi(t+d\tau))\right]$$

Now the equation can be written as

$$\hat{B}\psi(t+d\tau) = \hat{A}\psi(t)$$

with

$$\hat{B} = \mathbb{1} - b\mathcal{L}d\tau, \qquad \hat{A} = \mathbb{1} + a\mathcal{L}d\tau$$

For instance, for $a=1, b=0$, we find again the Euler method

$$\psi(t+d\tau) = \psi(t) + \mathcal{L}d\tau$$

If, instead, we choose $a = b = \frac{1}{2}$, we get back to the Cranck-Nicholson method, so we have to solve

$$\left(\mathbb{1} - \frac{1}{2}\mathcal{L}d\tau\right)\psi(t + d\tau) = \left(\mathbb{1} + \frac{1}{2}\mathcal{L}d\tau\right)\psi(t)$$

The problem is that now we are integrating $\mathcal{L}$, but we do not even know its value over the integration interval. We can estimate the value of the function via Taylor expansion:

$$\psi(t + d\tau) - \psi(t) = d\tau \sum_{k=0}^{K} \omega_k \mathcal{L}\left(\psi(t + \tau_k), t + \tau_k\right) + \mathcal{O}(d\tau^{2K})$$

The remainder is of order $2K$ if we choose the instants τ_k properly. We now define

$$Y_k = \psi(t + d\tau_k) - \psi(t), \qquad \Phi_k = \mathcal{L}\left(\psi(t + \tau_k), \tau_k\right)$$

Hence

$$\frac{\partial \psi}{\partial t} = \mathcal{L}(\psi, t) = \Phi$$

Example: for $K = 2$ (i.e., second order approximation):

$$Y_0 = 0, \qquad Y_1 = \psi(t + \tau_1) - \psi(t) = \psi(t + d\tau) - \psi(t) = d\tau R_{10}\Phi$$

The total time derivative of Φ is then

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \frac{\partial \Phi}{\partial t} + \frac{\partial \Phi}{\partial \psi}\frac{\partial \psi}{\partial t}$$

and so

$$\frac{\mathrm{d}^2\psi}{\mathrm{d}t^2} = \frac{\mathrm{d}\Phi}{\mathrm{d}t} = \frac{\partial \Phi}{\partial t} + \frac{\partial \Phi}{\partial \psi}\frac{\partial \psi}{\partial t} = \frac{\partial \Phi}{\partial t} + \Phi\frac{\partial \Phi}{\partial \psi}$$

Now we can write

$$\psi(t + d\tau) = \psi(t) + d\tau\left[\omega_0\Phi_0 + \omega_1\Phi_1\right]$$

with

$$\Phi_1 = \Phi_0 + \frac{\partial \Phi}{\partial \psi}\mathrm{d}\psi + \frac{\partial \Phi}{\partial t}\tau_1 = \Phi_0 + \frac{\partial \Phi}{\partial \psi}Y_1 + \frac{\partial \Phi}{\partial t}\tau_1$$

Finally, we get

$$\psi(t + d\tau) = \psi(t) + d\tau\left[\omega_0\Phi_0 + \omega_1\Phi_1\right] =$$
$$= \psi(t) + d\tau\left[\omega_0 + \omega_1\right]\Phi_0 + d\tau^2\omega_1\left[R_{10}\Phi_0\frac{\partial \Phi}{\partial \psi} + c_1\frac{\partial \Phi}{\partial t}\right]$$

However, we also know that

$$\psi(t + d\tau) = \psi(t) + d\tau\frac{\mathrm{d}\psi}{\mathrm{d}\tau} + \frac{d\tau^2}{2}\frac{\mathrm{d}^2\psi}{\mathrm{d}\tau^2}$$

Equating the last two expressions, we get

$$\begin{cases} \omega_0 + \omega_1 = 1 \\ \omega_1 R_{10} = \frac{1}{2} \\ \omega_1 c_1 = \frac{1}{2} \end{cases}$$

We can solve the system and find $\omega_0, \omega_1, R_{10}$ for an arbitrary value of c_1, i.e. the ratio between τ_1 and t_1.

3 Density matrices

Density matrices

We now introduce the formalism of *density matrices*. Wave functions describe closed systems, i.e. systems which do not interact with the external environment. This is not a very physical situation. In principle, one could always consider the wave function of the whole universe, but this is not a particularly useful approach.

Usually, systems interact with some external entity, like heat reservoirs. The time evolution of open systems is not unitary. A wave function can always be written as

$$|\psi\rangle = \sum_n \psi_n |\alpha_n\rangle , \qquad \{|\alpha_n\rangle\}_{n=1}^d$$

and an operator as

$$\hat{A} = \sum_{n,m} A_{nm} |a_m\rangle \langle a_n|$$

The object $\langle a_n|$ identifies the state on which the operator acts, and $|a_m\rangle$ is the resulting state. In a laboratory, we can only measure expected values,

$$\begin{aligned} \langle \hat{A} \rangle = \langle \psi | A | \psi \rangle = \sum_{m,n} \psi_m^* A_{nm} \psi_n = \\ = \sum_n \sum_m \psi_m^* \psi_n A_{nm} = \sum_{n,m} \rho_{nm} A_{nm} = \\ = \mathrm{Tr}\left(\hat{\rho}\hat{A}\right), \end{aligned}$$

where we defined

$$\hat{\rho} = \sum_{n,m} \rho_{nm} |\alpha_m\rangle \langle \alpha_n| = \sum_{n,m} \psi_m \psi_n^* |\alpha_m\rangle \langle \alpha_n|$$

to be the density matrix associated to ψ.
With the density matrix we can obtain all the results we could get by using the wave functions, but we can also do much more. Let us consider for example a qubit

$$|\psi\rangle = \psi_0 |0\rangle + \psi_1 |1\rangle$$

Its density matrix is

$$\rho = |\psi\rangle\langle\psi| = |\psi_0|^2 |0\rangle\langle 0| + |\psi_1|^2 |1\rangle\langle 1| + \psi_0^*\psi_1 |1\rangle\langle 0| + \psi_0\psi_1^* |0\rangle\langle 1|$$

It can also be represented as

$$\rho = \begin{pmatrix} |\psi_0|^2 & \psi_0\psi_1^* \\ \psi_0^*\psi_1 & |\psi_1|^2 \end{pmatrix}$$

The time derivative would then be

$$\begin{aligned} \dot{\rho} &= \frac{\mathrm{d}}{\mathrm{d}t} |\psi\rangle\langle\psi| = |\dot{\psi}\rangle\langle\psi| + |\psi\rangle\langle\dot{\psi}| = \\ &= -\frac{i}{\hbar} H |\psi\rangle\langle\psi| + \frac{i}{\hbar} |\psi\rangle\langle\psi| H = \\ &= \frac{i}{\hbar} (\rho H - H\rho) = \\ &= \frac{i}{\hbar} [\rho, H] \end{aligned}$$

This is known as the *Liouville equation*, and can be regarded as the quantum version of the Liouville's theorem. Density matrices have the following properties:

1. Hermitianity: $\rho = \rho^\dagger$
2. $\mathrm{Tr}(\rho) = \sum_n |\psi_n|^2 = 1$ [3]
3. Can be written in different basis:

   $$\rho' = \mathbf{U}^{-1}\rho\mathbf{U},$$

   with

   $$|n\rangle = \sum_\alpha U_{n\alpha} |\alpha\rangle$$

   Then

   $$\begin{aligned} \rho' = |\psi'\rangle\langle\psi'| &= \sum_{n,m} \psi'_n\psi'^*_m |n\rangle\langle m| = \\ &= \sum_{n,m} \psi'_n\psi'^*_m \sum_{\alpha,\beta} U_{n\alpha} |\alpha\rangle\langle\beta| U^*_{m\beta} = \\ &= \sum_{\alpha,\beta} |\alpha\rangle\langle\beta| \sum_{n,m} U_{n\alpha}\psi'_n U^*_{m\beta}\psi'^*_m = \\ &= \sum_{\alpha,\beta} \psi_\alpha\psi^*_\beta |\alpha\rangle\langle\beta| = \\ &= |\psi\rangle\langle\psi| = \rho \end{aligned}$$

[3] Properties 1 and 2 result in density matrices having $n^2 - 1$ free parameters; equivalently, we need these many parameters to characterize a system.

If $\rho = |\psi\rangle\langle\psi|$ for some $|\psi\rangle$, we can always choose $|\psi\rangle$ to be the first vector of some basis, and then complete it to an orthonormal basis. In this new basis, then, we have

$$\rho = \begin{pmatrix} 1 & & & \mathbb{0} \\ & 0 & & \\ & & \ddots & \\ \mathbb{0} & & & 0 \end{pmatrix}$$

Any system which can be brought to this form is equivalent to a system described by a wavefunction. If this is the case, we say that ρ represents a *pure state*. If, on the contrary, this is not possible, we say that ρ represents a *mixed state*. Two quick definitions:

- the set of the diagonal elements of ρ are called *population*;
- the off-diagonal elements of ρ are called *coherences*.

We can now add another property of density matrices:

4. The expectation value of the operator ρ is $\langle\rho\rangle = \mathrm{Tr}(\rho\rho) = \mathrm{Tr}\left(\rho^2\right) = \sum_m \rho_{mm}^2 \leq \left(\sum_m \rho_{mm}\right)^2 = \left(\mathrm{Tr}(\rho)\right)^2 = 1$ where, in the first passage, we computed the trace in a basis in which ρ is diagonal.

We can state that

$$\rho^2 = \rho \iff \mathrm{Tr}(\rho^2) = 1 \iff \rho \text{ is a pure state.}$$

This is useful because calculating the square of a matrix is much simpler than diagonalizing it. Another consequence is that mixed states have $\mathrm{Tr}(\rho^2) < 1$, and are defined as a probability mixture of pure states:

$$\sum_\alpha p_\alpha |\phi_\alpha\rangle\langle\phi_\alpha|$$

where p_α are classical probabilities.

Example: for a $\frac{1}{2}$-spin system,

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \Longrightarrow \rho_c = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

represents a pure state, while

$$\rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

represents a mixed state: there is no way to transform a scalar matrix[4] with a change of basis. We can now have two kinds of superpositions. *Coherent*

Coherent and decoherent superpositions

[4] Basically, a multiple of an identity matrix.

superposition occurs when, for instance, we have

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \Longrightarrow \rho_c = |\psi\rangle\langle\psi| = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

and so as in the first example above. We have instead *decoherent superposition* for example when

$$\rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

as in the second example. The only way to distinguish between ρ_c and ρ is to use an observable which mixes the states $|0\rangle$ and $|1\rangle$. If we are dealing with spins, and $|0\rangle$ and $|1\rangle$ are states of the spin along the x direction, we have for example to measure the spin along z. Indeed

$$\mathrm{Tr}(\sigma^2\rho_c) = 1 \neq \mathrm{Tr}(\sigma^2\rho) = 0$$

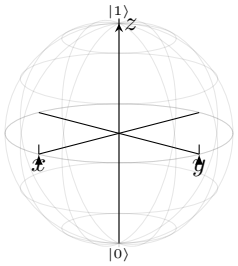
The most general density matrix for a qubit is an hermitian matrix with trace equal to 1:

$$\rho = \frac{1}{2}\left(\mathbb{1} + \vec{x}\cdot\vec{\sigma}\right) = \begin{pmatrix} 1+z & x+iz \\ x-iz & 1-z \end{pmatrix}$$

where $\vec{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$. As a consequence, a general density matrix for a qubit requires three parameters. This could be also deduced from the following argument: a 2×2 matrix is described by 8 real parameters, and the hermitianity condition leaves only 4 real parameters, and finally the condition $\mathrm{Tr}(\rho) = 1$ removes one more. On the contrary, a general qubit wave function can be written by using only two parameters:

$$|\psi\rangle = \cos\theta\,|0\rangle + \sin\theta e^{i\phi}\,|1\rangle$$

Indeed, in principle there are 4 free real parameters, but the normalization condition and the freedom of the overall phase eliminate two of them.
Two parameters describe the surface of a sphere. We can indeed represent qubits as points of the surface of a *Bloch sphere*.



Mixed states, described by a general density matrix and hence by three parameters, cannot live on the surface, and are indeed found inside the Bloch sphere.

4 Composite systems

Let us now consider the two Hilbert spaces $\mathcal{H}_A$ and $\mathcal{H}_B$. If we want to make the two corresponding systems interact, we have to consider the composition of the two Hilbert spaces via a tensor product; for the hamiltonians that is

$$\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$$

while for the states that is

$$\left\{ |\psi\rangle = \sum_{\alpha_1,\alpha_2} C_{\alpha_1,\alpha_2} |\alpha_1\rangle \otimes |\alpha_2\rangle \right\}$$

The states of the new system $\mathcal{H}$ can be written as a combination of states obtained as a tensor product of vectors of the basis of the two original systems. However, we can also change representation, choosing a new basis $|n\rangle$ for the new system and write, for a $|\psi\rangle \in \mathcal{H}$:

$$|\psi\rangle = \sum_n c_n |n\rangle$$

We simply turned the old matrix of coefficients C_{α_1,α_2} into a vector of coefficients c_n. For example, in the case of a qubit system, we perform the following change:

$$\sum_{\substack{i=0,1 \\ j=0,1}} C_{ij} |i\rangle \otimes |j\rangle \longleftrightarrow \sum_{n=0}^{3} c_n |n\rangle$$

where

$$\begin{cases} |0\rangle = |00\rangle = |0\rangle \otimes |0\rangle \\ |1\rangle = |01\rangle = |0\rangle \otimes |1\rangle \\ |2\rangle = |10\rangle = |1\rangle \otimes |0\rangle \\ |3\rangle = |11\rangle = |1\rangle \otimes |1\rangle \end{cases}$$

and

$$\begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix} \longleftrightarrow \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

where $c_0 = C_{00}, c_1 = C_{01}, c_2 = C_{10}, c_3 = C_{11}$. This way of turning a matrix into a vector is called a *Liouville representation.*

If we combine N qubits, we have 2^N coefficients $C_{\alpha_1 \cdots \alpha_N}$, and with just a few atoms it becomes impossible to simulate the system directly. Going back

to the two systems case, the density matrix of a state belonging to $\mathcal{H}$ is

$$\begin{aligned}\rho = |\psi\rangle\langle\psi| = \sum_{n,m} c_n c_m^* |n\rangle\langle m| = \\ = \sum_{\substack{\alpha_1,\alpha_2 \\ \alpha_1',\alpha_2'}} C_{\alpha_1\alpha_2} C_{\alpha_1'\alpha_2'} |\alpha_1\alpha_2\rangle\langle\alpha_1'\alpha_2'| = \\ = \sum_{\substack{\alpha_1,\alpha_2 \\ \alpha_1',\alpha_2'}} \rho_{\alpha_1\alpha_2}^{\alpha_1'\alpha_2'} |\alpha_1\alpha_2\rangle\langle\alpha_1'\alpha_2'|\end{aligned}$$

Since operators (matrices which act on vectors) can be represented as vectors, in the Liouville representation superoperators (tensors which act on matrices) are represented as simple matrices.

Now, we wrote the density matrix of a pure state, $|\psi\rangle$. However, the most general expression for a density matrix of $\mathcal{H}$ is

$$\rho = \sum_k \rho_k |\psi_k\rangle\langle\psi_k| = \sum_k \rho_k \sum_{\substack{\alpha_1,\alpha_2 \\ \alpha_1',\alpha_2'}} \rho_{\alpha_1\alpha_2}^{\alpha_1'\alpha_2'} |\alpha_1\alpha_2\rangle\langle\alpha_1'\alpha_2'|$$

Separable state

We say that, for a composed system $\mathcal{H} = \mathcal{H}_A + \mathcal{H}_B$, ρ is a *separable state* if one can write, for $p_k \geq 0$ and $\sum_k p_k = 1$,

$$\rho = \sum_k p_k \rho_A^k \otimes \rho_B^k, \qquad 0 \leq p_k \leq 1 \quad \forall k$$

where

$$\begin{aligned}\rho_A \otimes \rho_B = \left(\sum_{\alpha_1} c_{\alpha_1} |\alpha_1\rangle\langle\alpha_1|\right) \otimes \left(\sum_{\alpha_2} c_{\alpha_2} |\alpha_2\rangle\langle\alpha_2|\right) = \\ = \sum_{\alpha_1,\alpha_2} c_{\alpha_1} c_{\alpha_2} |\alpha_1\alpha_2\rangle\langle\alpha_1\alpha_2|\end{aligned}$$

Hence, a separable state is a state of a composed system which can be written as a combination (with coefficients given by a probability distribution) of states of the subsystems.

Separability and purity are independent. For example, the state

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

is pure but not separable, while the density matrices formalism allows for not pure but separable states. On the contrary, the state

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) = \frac{1}{\sqrt{2}}|0\rangle \otimes (|0\rangle + |1\rangle)$$

is pure and separable: its density matrix is

$$\rho = |\psi\rangle\langle\psi| = \left(\begin{array}{cc|c} \frac{1}{2} & \frac{1}{2} & \mathbf{0} \\ \frac{1}{2} & \frac{1}{2} & \\ \hline \mathbf{0} & & \mathbf{0} \end{array}\right) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

The probability of A being 0 is given by

$$1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = 1$$

while the probability of B being 0 is

$$\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 0 = \frac{1}{2}$$

which is equal to the probability of B being 1:

$$\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = \frac{1}{2}$$

Classical information theory

We can now switch to *classical information theory*. The main quantities of this theory are:

- *Entropy*: the amount of ignorance we have about a system;
- *Mutual information*: the amount of information random variables give about each other;
- *Relative entropy*: a distance (not in the mathematical sense) between probability density functions.

The objects of classical information theory are:

- *Random variables*;
- *Alphabets*: set of all possible values of a random variable;
- *Entropy (Shannon entropy)*:

$$S = -\sum_{x \in A} p(x) \log(x)$$

- *Expectation value*:

$$\mathbb{E}\left[\rho(x)\right] = \sum_{x \in A} p(x)\rho(x)$$

  Hence we can see that

$$S = \mathbb{E}\left[\log\left(\frac{1}{p(x)}\right)\right]$$

S has the following properties:

1. $S \geq 0$
2. $S \leq \mathbb{E}[\#Q] \leq S + 1$, where $\#Q$ is the minimum number of questions that need to be asked in order to guess the state of a system.

Let us see an example of this last proposition. Let us consider a random variable with the following alphabet:

$$A = \{a, b, c, d\}$$

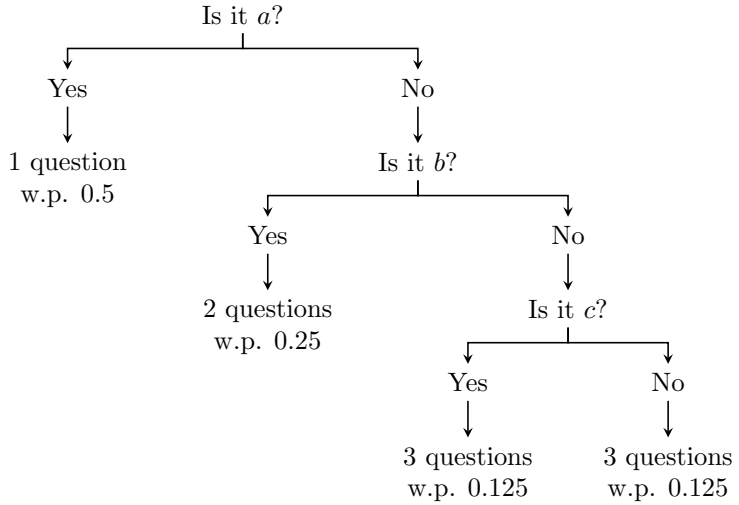
and the following distributions:

$$p(a) = \frac{1}{2} \qquad p(b) = \frac{1}{4} \qquad p(c) = p(d) = \frac{1}{8}$$

It is easy to compute the value of the entropy:

$$S \approx \frac{7}{4}$$

If we had to guess the value of the random variables using the the minimum number of questions, we would have to start asking if the system is in the state a, because it is the most probable value. Indeed, the most effective strategy is the following:



Hence

$$\mathbb{E}[\#Q] = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + 3 \cdot \frac{1}{8} = \frac{7}{4}$$

Intuitively, S is lower for more localized distributions. Let us consider

$$A = \{0, 1\}$$

with

$$p(0) = p, \qquad p(1) = 1 - p$$

Then

$$S = -\sum_{x=0,1} p(x) \log p(x) = -p \log p - (1-p) \log(1-p)$$

The plot of S is the following:

In the case of $p = 0$ or $p = 1$ the state is determined, and so the entropy (the ignorance about the state) is minimized. If, instead, $p = \frac{1}{2}$, the entropy is maximized because we are completely ignorant about the state of the system.

We can also define a *joint entropy* as

$$S(x,y) = -\sum_{x \in A} \sum_{y \in B} p(x,y) \log \left(p(x,y)\right)$$

and a *conditional entropy* as

$$\begin{aligned} S(y|x) &= -\sum_{x} p(x) \sum_{y} p(y|x) \log \left(p(y|x)\right) = \\ &= -\sum_{x,y} p(x) p(y|x) \log \left(p(y|x)\right) = \\ &= -\sum_{x,y} p(x,y) \log \left(p(y|x)\right) = \\ &= \mathbb{E}\left[\log\left(\frac{1}{p(y|x)}\right)\right] \end{aligned}$$

But also

$$\begin{aligned} S(x,y) &= -\sum_{x,y} p(x,y) \log \left(p(x,y)\right) = -\sum_{x,y} p(x,y) \log \left(p(x) p(y|x)\right) = \\ &= \sum_{x,y} p(x,y) \log \left(p(x)\right) - \sum_{x,y} p(x,y) \log \left(p(y|x)\right) \end{aligned}$$

The second term is just $S(y|x)$, and in the first there is no actual mention of y, as we are effectively marginalizing it: we then get

$$S(x,y) = -\sum_{x} p(x) \log \left(p(x)\right) + S(y|x) = S(x) + S(y|x)$$

and finally

$$S(y|x) = S(x,y) - S(x)$$

We can also define a *relative entropy* as

$$D(p||q) = \sum_{x} p(x) \log\left(\frac{p(x)}{q(x)}\right) = \mathbb{E}\left[\log\left(\frac{p(x)}{q(x)}\right)\right]$$

This new object has the intuitive meaning of a distance between the two distributions. Indeed, it is easy to see that

$$D(p||p) = 0$$

However, this is not a distance in strict mathematical terms; for instance,

$$D(p||q) \neq D(q||p)$$

Another quantity of interest is the *mutual information* between two variables:

$$I(x,y) = \sum_{x,y} p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

which can also be written as

$$I(x,y) = D(p(x,y)||p(x)p(y))$$

$I(x,y)$ quantifies the information that a random variable x contains about another random variable y. Indeed, if x and y are totally uncorrelated, then

$$p(x,y) = p(x)p(y)$$

and so

$$\log\left(\frac{p(x,y)}{p(x)p(y)}\right) = \log\left(\frac{p(x)p(y)}{p(x)p(y)}\right) = \log(1) = 0$$

We can also see that

$$\begin{aligned}
I(x,y) &= \sum_{x,y} p(x,y) \log\left(\frac{p(x,y)}{p(x)p(y)}\right) = \sum_{x,y} p(x,y) \log\left(\frac{p(x|y)p(y)}{p(x)p(y)}\right) = \sum_{x,y} p(x,y) \log\left(\frac{p(x|y)}{p(x)}\right) = \\
&= -\sum_{x,y} p(x,y) \log\left(p(x)\right) - \left[-\sum_{x,y} p(x,y) \log\left(p(x|y)\right)\right] = \\
&= -\sum_{x} p(x) \log\left(p(x)\right) - \left[-\sum_{x,y} p(x,y) \log\left(p(x|y)\right)\right] = \\
&= S(x) - S(x|y) = S(x) - [S(x,y) - S(y)] = \\
&= S(x) + S(y) - S(x,y)
\end{aligned}$$

and so

$$S(x,y) = S(x) + S(y) - I(x,y)$$

This is a very reasonable result: the ignorance (entropy) we have on the joint distribution is equal to the sum of the ignorance of both distributions minus the information each of them provides on the other. One can prove that

$$I(x,y) \geq 0$$

Quantity	Uncorrelated classical variables	Correlated classical variables
$p(x,y)$	$= p(x)p(y)$	$\neq p(x)p(y)$
$I(x,y)$	$= 0$	> 0
$x\bar{y} - \bar{x}\bar{y}$	$= 0$	$\neq 0$
$S(x,y)$	$= S(x) + S(y)$	$< S(x) + S(y)$

and hence

$$S(x,y) \leq S(x) + S(y)$$

For classical random variables, there are different ways to quantify the correlation (or simply check whether two random variables are correlated or not). We summarize some of them in the following:

We can try to generalize some of them to the quantum framework. For instance, for a wave function ψ and a couple of observables (X, Y), we can compute

$$\langle XY \rangle_\psi - \langle X \rangle_\psi \langle Y \rangle_\psi = \langle \psi | \hat{X}\hat{Y} | \psi \rangle - \langle \psi | \hat{X} | \psi \rangle \langle \psi | \hat{Y} | \psi \rangle$$

More in general, let us consider a state ρ and an observable $\hat{A}$. We can always decompose $\hat{A}$ as

$$\hat{A} = \sum_i a_i \hat{P}_i = \sum_i a_i |a_i\rangle \langle a_i|$$

We can now extract the probability distribution of the outcomes of a measure of $\hat{A}$:

$$p_i = \mathrm{Tr}\left(\rho \hat{P}_i\right)$$

If the state is pure,

$$\rho = |\psi\rangle \langle \psi| ,$$

then

$$\begin{aligned} p_i = \mathrm{Tr}\left(\rho \hat{P}_i\right) = \mathrm{Tr}\left(|\psi\rangle \langle \psi| \, |a_i\rangle \langle a_i|\right) = \\ = \langle \psi | \psi \rangle \langle \psi | a_i \rangle \langle a_i | \psi \rangle = \\ = |\langle \psi | a_i \rangle|^2 \end{aligned}$$

Indeed, $|\langle \psi | a_i \rangle|^2$ is nothing but the square modulus of the coefficient of $|\psi\rangle$ corresponding to $|a_i\rangle$ in the basis of the eigenstates of A.

If the state is mixed, then

$$\rho = \sum_k |\lambda_k|^2 |\alpha_k\rangle \langle \alpha_k|$$

and in this case we have

$$\begin{aligned} p_i &= \sum_k |\lambda_k|^2 \mathrm{Tr}\left(|\alpha_k\rangle \langle\alpha_k| \hat{P}_i\right) = \\ &= \sum_{j,k} |\lambda_k|^2 \langle\psi_j| |\alpha_k\rangle \langle\alpha_k| |a_i\rangle \langle a_i| |\alpha_j\rangle = \\ &= \sum_k |\lambda_k|^2 \langle\alpha_k| |a_i\rangle \langle a_i| |\alpha_k\rangle = \\ &= \sum_{j,k} |\lambda_k|^2 |\langle\alpha_k|a_i\rangle|^2 = \\ &= \sum_k |\lambda_k|^2 \tilde{p}_k = p_i \end{aligned}$$

where the $\tilde{p}_k$ are the purely quantum probabilities, while the p_i are the resulting ones. Once we have computed the probability distribution of the possible values of $\hat{A}$, we can also compute the corresponding Shannon entropy:

$$S(\rho, A) = -\sum_i p_i \log p_i = -\sum_i p(a_i) \log\left(p(a_i)\right)$$

For another observable B, we will have

$$S(\rho, B) = -\sum_i p(b_i) \log\left(p(b_i)\right)$$

and

$$S(\rho, A, B) = -\sum_{i,j} p(a_i, b_j) \log\left(p(a_i, b_j)\right)$$

Since $p(a_i)$ and $p(b_j)$ are usual probability distributions, the results we previously obtained still hold:

$$S(\rho, A, B) \leq S(\rho, A) + S(\rho, B)$$

In particular, if

$$S(\rho, A, B) < S(\rho, A) + S(\rho, B)$$

we say that there is correlation between the two observables.

There are two main problems with this approach:

- it only quantifies the classical part of correlation;
- results depend on the chosen A.

Instead, we want a definition of entropy which depends only on ρ. One such possibility is the *Von Neumann entropy*:

$$S_N(\rho) = -\mathrm{Tr}(\rho \log \rho)$$

S_N has the following properties:

1. $S(\rho, A) = S_N(\rho) \Longleftrightarrow [A, \rho] = 0$

2. $S(\rho, A) \geq S_N(\rho)$

3. *(Araki-Lieb inequality)* Let us consider two systems A and B, the composed system $A \otimes B$ in a state ρ_{AB} and let us define

$$\rho_A = \mathrm{Tr}_B \rho \qquad \rho_B = \mathrm{Tr}_A \rho$$

Hence

$$S_N(\rho_A) + S_N(\rho_B) \geq S_N(\rho) \geq |S_N(\rho_A) - S_N(\rho_B)|$$

As a consequence, if

$$S_N(\rho) = 0,$$

then

$$S_N(\rho_A) = S_N(\rho_B)$$

4. If ρ is pure, then

$$S_N(\rho) = \mathrm{Tr}\,(\rho \log \rho) = 0$$

Recall that the trace is invariant under unitary transformations. Since the logarithm is analytical and the ρ matrix is hermitian, we can write

$$\begin{aligned}
\mathrm{Tr}\,(\rho \log \rho) &= \mathrm{Tr}\left(\mathbf{U}\rho \log \rho \mathbf{U}^\dagger\right) = \\
&= \mathrm{Tr}\left(\mathbf{U}\rho\mathbf{U}^\dagger \mathbf{U} \log \rho \mathbf{U}^\dagger\right) = \\
&= \mathrm{Tr}\left(\mathbf{U}\rho\mathbf{U}^\dagger \log\left(\mathbf{U}\rho\mathbf{U}^\dagger\right)\right) = \\
&= \mathrm{Tr}\left(\mathbf{U}\rho\mathbf{U}^\dagger \log\left(\mathbf{U}\rho\mathbf{U}^\dagger\right)\right) = \\
&= \mathrm{Tr}\left\{ \begin{pmatrix} 1 & & \mathbb{0} \\ & 0 & \\ & & \ddots \\ \mathbb{0} & & 0 \end{pmatrix} \log \left[\begin{pmatrix} 1 & & \mathbb{0} \\ & 0 & \\ & & \ddots \\ \mathbb{0} & & 0 \end{pmatrix} \right] \right\}
\end{aligned}$$

where we have chosen the basis which diagonalizes ρ. Now,

$$\mathrm{Tr}(\rho \log \rho) = \sum_i \rho_{ii} \log \rho_{ii} = 0$$

Thus, a pure state has vanishing Von Neumann entropy.

5. $S_N(\rho_A \otimes \rho_B) = S_N(\rho_A) + S_N(\rho_B)$

6. *(Concavity)* If

$$\rho = \sum_i p_i \rho_i,$$

then

$$S_N(\rho) \geq \sum_i p_i S_N(\rho_i)$$

We can now define the *Von Neumann mutual information*, as

$$I_N(\rho_A, \rho_B, \rho) = S_N(\rho_A) + S_N(\rho_B) - S_N(\rho)$$

and the *Von Neumann relative entropy* as

$$S_N(\sigma||\rho) = \mathrm{Tr}\left(\sigma \log\left(\frac{\sigma}{\rho}\right)\right)$$

We see that

$$I_N = S_N(\rho_{AB}, \rho_A \otimes \rho_B)$$

The mutual information has the following properties:

1. $S_N(\sigma||\rho) = S_N(\mathbf{U}\sigma\mathbf{U}^\dagger||\mathbf{U}\rho\mathbf{U}^\dagger)$

   This is due to the fact that changing the basis does not change the eigenvalues.

2. $S_N(\mathrm{Tr}_P\sigma||\mathrm{Tr}_P\rho) \leq S_N(\sigma||\rho)$

3. $S_N(\sigma_1 \otimes \sigma_2||\rho_1 \otimes \rho_2) = S_N(\sigma_1||\rho_1) + S_N(\sigma_2||\rho_2)$

A generalization of unitary transformations are *CPT maps (completely positive transformation)*. These are the most general transformations that map a density matrix into another density matrix. A CPT map Φ can be written as

$$\Phi(\sigma) = \sum_i V_i \sigma V_i$$

where the V_i satisfy

$$\sum_i V_i^\dagger V_i = \mathbb{1}$$

From this definition, unitary maps are CPT maps with only one V_i. It holds

$$S_N(\Phi(\sigma)||\Phi(\rho)) \leq S_N(\sigma||\rho)$$

We now want to quantify the entanglement of a quantum system. In this framework the adjective "entangled" has to be intended as "not separable". To start, we have to decide what are the characteristics of a good measure of entanglement.

A separable state for a system with two subsystems can be written as

$$\rho_{AB} = \sum_i p_i \rho_A^{(i)} \otimes \rho_B^{(i)}, \qquad 0 \leq p_i \leq 1 \quad \forall \quad i$$

These are the most general states that can be created using only *LOCC (local operations and classical communications)*. These include only those transformations which affect only one subsystem at a time and classical communication between the two subsystems. This means that A and B can "talk" to each other and decide transformations to be applied on their own subsystems, but

they cannot make their subsystems interact with each other. For instance, they can decide to prepare their corresponding subsystems in the state $|0\rangle$ or $|1\rangle$ with probabilities 1/2 independently. In this case, the composed state will be

$$\begin{aligned}\rho &= \left[\frac{1}{2}\left(|0\rangle\langle 0| + |1\rangle\langle 1|\right)\right] \otimes \left[\frac{1}{2}\left(|0\rangle\langle 0| + |1\rangle\langle 1|\right)\right] = \\ &= \frac{1}{4}\left(|00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 10| + |11\rangle\langle 11|\right) = \\ &= \begin{pmatrix} \frac{1}{4} & & & \mathbb{0} \\ & \frac{1}{4} & & \\ & & \frac{1}{4} & \\ \mathbb{0} & & & \frac{1}{4} \end{pmatrix}\end{aligned}$$

The state can be written as

$$\rho = \rho_A \otimes \rho_B$$

and it is thus separable. Indeed, it has been prepared with LOCC. Another possibility is to prepare the subsystems in the state $|0\rangle$ or $|1\rangle$ with probabilities 1/2, but using also classical communication in order to (classically) correlate the two subsystems. For example, they can choose to always prepare the subsystems in the same state. In this case, they will obtain

$$\rho = \frac{1}{2}|00\rangle\langle 00| + \frac{1}{2}|11\rangle\langle 11| = \begin{pmatrix} \frac{1}{2} & & & \mathbb{0} \\ & 0 & & \\ & & 0 & \\ \mathbb{0} & & & \frac{1}{2} \end{pmatrix}$$

In this case, ρ can be written as

$$\rho = p_1\left(\rho_{A,1} \otimes \rho_{B,1}\right) + p_2\left(\rho_{A,2} \otimes \rho_{B,2}\right)$$

and thus it is again a separable state.

Using LOCC, A and B will be able to correlate the two subsystems in a classical sense, but not in a quantum one. Hence, we want these two states to have null entanglement.

We can now summarize the the properties that a good entanglement measure should have:

1. $E(\rho) = 0$
2. $E(\rho)$ should be constant under local operations, i.e.

$$E(\mathbf{U}_A \otimes \mathbf{U}_B \rho \mathbf{U}_A^\dagger \otimes \mathbf{U}_B^\dagger) = E(\rho)$$

3. Local operations, classical communication, subselection (i.e., the selection of a subset of the results) should not increase E
4. If ρ is a pure state,

$$E(\rho) = S_N(\rho_A)$$

This last property implies that if the partial trace of a pure state is still pure, then the original state is separable and and hence has null entanglement.

A first definition of entanglement measure is given by formation entanglement:

$$E_F(\rho) = \min \sum_j p_j S_N(\rho_A^j)$$

with, as usual,

$$S_N(\rho_A^j) = -\mathrm{Tr}\left(\rho_A^j \log\left(\rho_A^j\right)\right)$$

The minimum is taken over all the possible representations of ρ,

$$\{\rho_{AB}^i\}, \qquad \rho_{AB}^i = \sum_j p_j^i \left|\psi_j^i\right\rangle \left\langle\psi_j^i\right|$$

and

$$\rho_A^i = \mathrm{Tr}_B\left(\rho_{AB}^i\right)$$

The wave functions $|\psi_j^i\rangle$ are pure states of the composed system. E_F can be rewritten as

$$E_F(\rho) = \min_i \sum_j p_j^i E\left(|\psi_j^i\rangle\right)$$

The *E-bit* is the fundamental unit of entanglement, and it is an entanglement singlet.

Another approach to entanglement measure is the *entanglement of distillation*, which tries to measure the entanglement of a system by counting how many Bell states can be generated by applying LOCC on copies of the initial state.

Yet another possible definition is given through the *relative entropy of entanglement*:

$$E(\sigma) = \min_{\rho \in D} S(\sigma||\rho)$$

where D is a set of separable states.

These definitions of entanglement are easy to state, but they are not operatively useful, because they can be applied in practical computations in only a handful of cases.

Let us consider a Bell state,

$$|\psi^B\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$$

Its density matrix is

$$\rho^B = |\psi^B\rangle \langle\psi^B| = \begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

Let us also consider a mixed state:

$$\rho^C = \frac{1}{2}|00\rangle\langle 00| + \frac{1}{2}|11\rangle\langle 11| = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

ρ^C is a classically correlated state: indeed, it is separable. On the contrary, ρ^B is not separable. The two corresponding matrices have all the elements on the diagonal in common, and they differ only for the off-diagonal terms. If we compute

$$\langle \sigma_A^z \sigma_B^z \rangle - \langle \sigma_A^z \rangle \langle \sigma_B^z \rangle$$

for the two states, we obtain the same result, as this computation only depends on the diagonal elements. Indeed,

$$\begin{aligned} \langle \sigma_A^z \sigma_B^z \rangle^B &= \mathrm{Tr}\left(\sigma_A^z \sigma_B^z \rho^B\right) = \\ &= \langle \psi^B | \sigma_A^z \sigma_B^z | \psi^B \rangle = \\ &= \frac{1}{2}\left(\langle 00| + \langle 11|\right)\left(\sigma_A^z \otimes \sigma_B^z\right)\left(|00\rangle + |11\rangle\right) = \frac{1}{2} \end{aligned}$$

and

$$\begin{aligned} \langle \sigma_A^z \rangle^B &= \mathrm{Tr}\left(\sigma_A^z \rho^B\right) = \\ &= \langle \psi^B | \sigma_A^z | \psi^B \rangle = \\ &= \frac{1}{2}\left(\langle 00| + \langle 11|\right)\left(\sigma_A^z \otimes \mathbb{1}\right)\left(|00\rangle + |11\rangle\right) = \frac{1}{2} \end{aligned}$$

and, similarly,

$$\langle \sigma_B^z \rangle^B = \frac{1}{2}$$

In the same way,

$$\langle \sigma_A^z \sigma_B^z \rangle^C = \mathrm{Tr}\left(\sigma_A^z \sigma_B^z \rho^C\right)$$

but

$$\sigma_A^z \sigma_B^z = |11\rangle\langle 11|$$

and then

$$\begin{aligned} \langle \sigma_A^z \sigma_B^z \rangle^C &= \mathrm{Tr}\left(|11\rangle\langle 11|\left(\frac{1}{2}|11\rangle\langle 11| + \frac{1}{2}|00\rangle\langle 00|\right)\right) = \\ &= \mathrm{Tr}\left(\frac{1}{2}|11\rangle\langle 11|\right) = \\ &= \frac{1}{2}\left(\langle 00|11\rangle\langle 11|00\rangle + \langle 01|11\rangle\langle 11|01\rangle + \langle 10|11\rangle\langle 11|10\rangle + \langle 11|11\rangle\langle 11|11\rangle\right) = \frac{1}{2} \end{aligned}$$

and finally

$$\langle \sigma_A^z \rangle = \mathrm{Tr}\left(\sigma_A^z \rho^C\right) = \mathrm{Tr}_A\left(\sigma_A^z \rho_A^C\right)$$

with

$$\rho_A^C = \langle 0| \left(\frac{1}{2} |00\rangle \langle 00| + \frac{1}{2} |11\rangle \langle 11| \right) |0\rangle + \langle 1| \left(\frac{1}{2} |00\rangle \langle 00| + \frac{1}{2} |11\rangle \langle 11| \right) |1\rangle =$$
$$= \frac{1}{2} |0\rangle \langle 0| + \frac{1}{2} |1\rangle \langle 1|$$

Then

$$\langle \sigma_A^z \rangle^C = \mathrm{Tr}_A \left(\sigma_A^z \rho_A^C \right) = \frac{1}{2}$$

Similarly,

$$\langle \sigma_B^z \rangle^C = \frac{1}{2}$$

Since all the expectation values involving σ^z are equal in both cases, the correlation computed over σ^z will be the same. We should consider other operators (such as σ_x and σ_y) if we wanted to notice differences. In particular, we have

$$\mathrm{Tr}_A \left(\rho^C \right) = \rho_A^C = \frac{1}{2} |0\rangle \langle 0| + \frac{1}{2} |1\rangle \langle 1| = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

and

$$\mathrm{Tr}_A \left(\rho^B \right) = \rho_A^B =$$
$$= \langle 0| \left(\frac{1}{2} |00\rangle \langle 00| + \frac{1}{2} |00\rangle \langle 11| + \frac{1}{2} |11\rangle \langle 00| + \frac{1}{2} |11\rangle \langle 11| \right) |0\rangle +$$
$$\langle 1| \left(\frac{1}{2} |00\rangle \langle 00| + \frac{1}{2} |00\rangle \langle 11| + \frac{1}{2} |11\rangle \langle 00| + \frac{1}{2} |11\rangle \langle 11| \right) |1\rangle =$$
$$= \frac{1}{2} |0\rangle \langle 0| + \frac{1}{2} |1\rangle \langle 1| = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

Such operations are not enough to distinguish between the two states.

Another useful quantity for this kind of problem is *concurrence*. Let us consider a density matrix ρ. We define

$$\tilde{\rho} = (\sigma_y \otimes \sigma_y) \, \rho \, (\sigma_y \otimes \sigma_y)$$

and

$$R = \rho \tilde{\rho}$$

If $\lambda_1, \lambda_2, \lambda_3$ and λ_4 are the (inversely ordered) eigenvalues of R, we define concurrence to be

$$\mathcal{C}(\rho) = \max(0, \lambda_1 - \lambda_2 - \lambda_3 - \lambda_4)$$

For example, we have

$$R^C = \begin{pmatrix} \frac{1}{4} & & & \mathbb{0} \\ & \frac{1}{4} & & \\ & & \frac{1}{4} & \\ \mathbb{0} & & & \frac{1}{4} \end{pmatrix} \Longrightarrow \mathcal{C}\left(\rho^C \right) = 0$$

while

$$R^B = \begin{pmatrix} \frac{1}{4} & & \\ & & \\ & & \end{pmatrix} \Longrightarrow \mathcal{C}\left(\rho^B\right) = 1$$

We now introduce the algorithms based on the real space renormalization group. The idea is to start from a system composed of N particles, double it, obtaining $2N$ particles, and then ???? in some way.

The idea is based on a hypothesis due to Wilson (1975): the ground state of a system is composed only of few low energy states of its two halves. More formally, the ground state of the whole system can be written as

$$|\psi_G\rangle_{2N} = \sum_{i,j} c_{ij} \, |E_A^i\rangle \otimes |E_B^j\rangle$$

then, Wilson's hypothesis consists in assuming that just a few c_{ij}s are non-null, and that only low-energy states enter in the sum. The explicit algorithm is based on the following steps.

1. We start considering a system with N sites and a hamiltonian

$$\hat{H}_1 : \mathbb{C}^d \longrightarrow \mathbb{C}^d$$

2. We form a composed system of size $2N$:

$$H_{2N} = H_N^A + H_N^B + H^{AB}$$

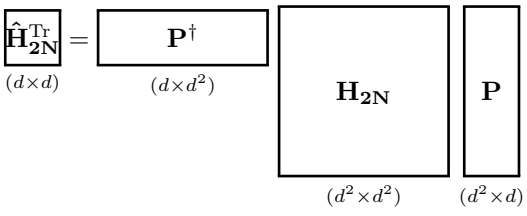
3. We diagonalize H_{2N} and we project it on the space of the first d lowest eigenstates:

$$\hat{H}_{2N} \underset{\hat{P}}{\longrightarrow} \hat{H}_{2N}^{\mathrm{Tr}} : \mathbb{C}^d \longrightarrow \mathbb{C}^d$$

where

$$\hat{H}_{2N}^{\mathrm{Tr}} = \mathbf{P}^\dagger H_{2N} \mathbf{P}$$

In the end,



4. Reiterate the procedure.

Since the scaling is exponential, in a few steps we can approximate a system with thousands of spins. This is called *Real-space Renormalization Group algorithm*. The problem is that the hypothesis, while it may seem intuitively true, is actually sometimes false.

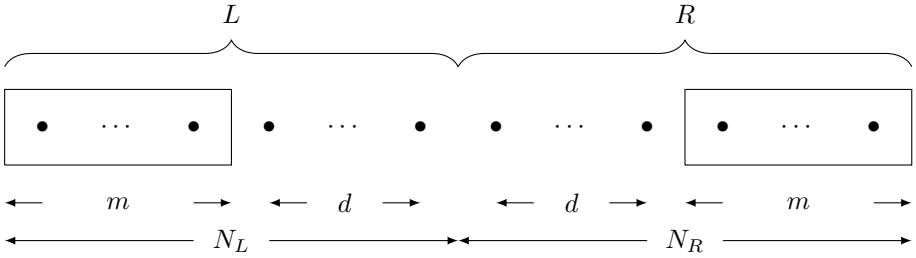
The simplest example in which this algorithm would fail is the particle in a box. At step 1, we have a discretization with d points of the box. The first eigenstates are the following:

At step 2, we double the system, obtaining a discretization with $2d$ points. The ground state is then

It is easy to see that, in order to write the ground state of the composed system we have to use an infinite sum of eigenstates of the single systems. Moreover, since the ground state of the composed system reaches its maximum at the boundaries of the box, the most important terms in the sum are those which have some maxima near this region, i.e., high energy eigenstates.

Stephen White (1992) proposed a new algorihtm, the *Density matrix Renormalization Group algorithm (DMRG)*. The first version is known as *Infinite Density matrix Renormalization Group algorithm (Infinite DMRG)*. The idea is to truncate the system, choosing the most probable state in a density matrix. The steps are the following:

1. We start from a system partitioned as follows:



   For example, it may be a system of 4 spins. The hamiltonian is

$$\hat{H} : \mathbb{C}^{(md)^2} \longrightarrow \mathbb{C}^{(md)^2}$$

   and the ground state can be written as

$$|\psi_G\rangle = \sum_{\alpha_1,\alpha_2,\alpha_3,\alpha_4} \psi_{\underbrace{\alpha_1\alpha_2}_{i}\underbrace{\alpha_3\alpha_4}_{j}} |\alpha_1\alpha_2\alpha_3\alpha_4\rangle = \sum_{i=0}^{N_L}\sum_{j=0}^{N_R} \psi_{ij} |ij\rangle$$

2. We write the density matrix of the ground state:

$$\rho = |\psi_G\rangle \langle\psi_G| = \sum_{i,i'}^{N_L}\sum_{j,j'}^{N_R} \psi_{ij}\psi^*_{i'j'} |ij\rangle \langle i'j'|$$

3. We trace ρ over the right system:

$$\rho_L = \mathrm{Tr}_R \rho$$

4. We diagonalize ρ_L:

$$\rho_L = \sum_{\alpha=1}^{N_L} \omega_\alpha \left|\omega_\alpha\right\rangle \left\langle\omega_\alpha\right|,$$

with

$$\sum_\alpha \omega_\alpha = 1, \qquad \omega_\alpha \geq 0 \quad \forall\alpha, \qquad \omega_1 \geq -2\omega_{N_L}$$

and we project it on the first m states. In this way, we choose the most probable states.

$$|i\rangle \xrightarrow{\mathbf{P}} |\omega_\alpha\rangle$$

5. We project the hamiltonian:

$$H \longrightarrow \mathbf{P}^\dagger H \mathbf{P}$$

At this point, we have a m-dimensional hamiltonian which describes (approximately) a system of initial size md.

6. We substitute this system to the old m-dimensional one. We add d spins and we double the system. We iterate the algorithm. At each step, we add d sites to that system, keeping the dimension of the matrix consistent.

Any expected value will be given by

$$\begin{aligned}
\langle\psi_G|\hat{O}_i|\psi_G\rangle = \mathrm{Tr}\left(\rho_l \hat{O}_i\right) = \\
&= \sum_{\alpha=1}^{md} \lambda_\alpha \langle\lambda_\alpha|\hat{O}_i|\lambda_\alpha\rangle \approx \\
&\approx \sum_{\alpha=1}^{m} \lambda_\alpha \langle\lambda_\alpha|\hat{O}_i|\lambda_\alpha\rangle \approx \\
&= \langle\hat{O}_i\rangle_{approx.}
\end{aligned}$$

The error we introduced is

$$\left|\langle\hat{O}_i\rangle - \langle\hat{O}_i\rangle_{approx.}\right| \leq \left|\sum_{\alpha=m}^{md} \lambda_\alpha\right| \cdot C$$

DMRG is the best approximation in terms of:

- Schmidt rank;
- Von Neumann entropy.

The *Schmidt rank* is introduced via the *Schur decomposition*, which is the following procedure.

Let us consider a pure state of a composed system:

$$|\psi_{AB}\rangle = \sum_{\alpha_1,\alpha_2} C_{\alpha_1\alpha_2} |\alpha_1\alpha_2\rangle$$

It can be proven that this can be rewritten as

$$|\psi_{AB}\rangle = \sum_n \lambda_n |\alpha'_n\alpha''_n\rangle$$

where $\{\alpha'_n\}$ and $\{\alpha''_n\}$ are orthonormal basis. Indeed, let us compute the density matrix

$$\rho_{AB} = |\psi_{AB}\rangle \langle\psi_{AB}| = \sum_{\alpha_1,\alpha_2} \sum_{\alpha'_1,\alpha'_2} C_{\alpha_1\alpha_2} C^*_{\alpha'_1\alpha'_2} |\alpha_1\alpha_2\rangle \langle\alpha'_1\alpha'_2|$$

We trace out the subsystem B:

$$\begin{aligned}
\rho_A = \mathrm{Tr}_B(\rho_{AB}) &= \sum_{\alpha_1,\alpha'_1,\beta} C_{\alpha_1\beta} C^*_{\alpha'_1\beta} |\alpha_1\rangle \langle\alpha'_1| = \\
&= \sum_{\alpha_1,\alpha'_1} \left[\sum_\beta C_{\alpha_1\beta} C^*_{\alpha'_1\beta} \right] |\alpha_1\rangle \langle\alpha'_1| = \\
&= \sum_{\alpha_1,\alpha'_1} (\rho_A)_{\alpha_1\alpha'_1} |\alpha_1\rangle \langle\alpha'_1|
\end{aligned}$$

We can now diagonalize ρ_A since it is hermitian, and rewrite it as

$$\rho_A = \sum_n |\lambda'|^2 |\lambda'\rangle \langle\lambda'|$$

If we go back to $|\psi_{AB}\rangle$ and rewrite it using the basis of the subsystem A which diagonalizes ρ_A we have:

$$\begin{aligned}
|\psi_{AB}\rangle &= \sum_{\alpha_1,\alpha_2} C_{\alpha_1\alpha_2} |\alpha_1\alpha_2\rangle = \\
&= \sum_{n,\alpha_2} C'_{n\alpha_2} |\lambda'_n\alpha_2\rangle = \\
&= \sum_n |\lambda'_n\rangle \sum_{\alpha_2} C'_{n\alpha_2} |\alpha_2\rangle = \\
&= \sum_n \lambda'_n |\lambda'_n\rangle \sum_{\alpha_2} \frac{C'_{n\alpha_2}}{\lambda'_n} |\alpha_2\rangle
\end{aligned}$$

Now, let us define

$$|\lambda''_n\rangle = \sum_{\alpha_2} \frac{C'_{n\alpha_2}}{\lambda'_n} |\alpha_2\rangle$$

We can show that $\{\lambda''_n\}$ is an orthonormal basis. Indeed,

$$\langle\lambda''_m|\lambda''_n\rangle = \sum_{\alpha_2,\alpha'_2} \frac{C'^*_{m\alpha'_2} C'_{n\alpha_2}}{\lambda'_n \lambda'^*_m} \langle\alpha'_2|\alpha_2\rangle = \sum_{\alpha_2} \frac{C'^*_{m\alpha_2} C'_{n\alpha_2}}{\lambda'_n \lambda'^*_m}$$

But it also holds, in the basis $\{|\lambda'_n\rangle\}$,

$$\mathrm{Tr}_B \left(|\psi_{AB}\rangle \langle\psi_{AB}|\right) = \sum_{\alpha_2} C'_{n\alpha_2} C'^*_{m\alpha_2} |\lambda'_n\rangle \langle\lambda'_m|$$

But we defined $\{|\lambda'_n\rangle\}$ as the basis which diagonalizes ρ_A, and have

$$\sum_{\alpha_2} C'_{n\alpha_2} C'^*_{m\alpha_2} = (\rho_A)_{nm} = \delta_{nm} |\lambda'_n|^2$$

and hence

$$\langle\lambda''_n|\lambda''_m\rangle = \delta_{nm} \frac{|\lambda'_n|^2}{\lambda'_n \lambda'^*_m} = \delta_{nm}$$

and is it is an orthonormal basis. So

$$|\psi_{AB}\rangle = \sum_n \lambda'_n |\lambda'_n\rangle |\lambda''_n\rangle$$

with $\{|\lambda'_n\rangle\}$ and $\{|\lambda''_n\rangle\}$ orthonormal. The density matrix of $|\psi_{AB}\rangle$ has become

$$\rho_{AB} = |\psi_{AB}\rangle \langle\psi_{AB}| = \sum_{n,m} \lambda'_n \lambda'^*_m |\lambda'_n \lambda''_m\rangle \langle\lambda'_n \lambda''_m|$$

and the partial traces are

$$\rho_A = \sum_n |\lambda'_n|^2 |\lambda'_n\rangle \langle\lambda'_n|, \qquad \rho_B = \sum_n |\lambda'_n|^2 |\lambda''_n\rangle \langle\lambda''_n|$$

The number of non null λ'_n coefficients is called Schmidt rank. The DMRG method keeps the bigger λ'_n; moreover, this method keeps as much entropy as possible, since, as we can see from the partial traces, entropy is related to the λ'_n coefficients. In this sense, we say that DMRG optimizes Schmidt rank and Von Neumann entropy.

An evolution of the infinite DMRG is the *finite DMRG*. The difference is that the system is not always split in two equal halves, but the partition is changed at each step. Starting from the following partition,

We iterate the procedure we already discussed, but we increase the size of, for instance, the left system, and decrease the size of the right one.

When the size of the right subsystem reaches 1, we we start the procedure in the other direction.

The algorithm is repeated until convergence.

We now introduce a new class of algorithms, called *Tensor network methods*, which are based on a specific formalism we will now introduce.

We represent operations among tensors as a connected graph, where tensors are represented as the vertices and the indexes as the edges. For example, a vector may be represented as

and a matrix as

A matrix-vector multiplication is given by

and a matrix-matrix multiplication as

We can see that contracted indexes are the edges that connect two vertices, while free indexes are the 'stumps' which are only connected to one vertex. With these conventions in mind, it is easy to calculate the computational cost of a tensorial operation: it is the product of $m_1, ..., m_N$, where N is the number of different indexes and m_i is the number of possible values of the i-th index, provided that contracted indexes are counted only one time. Tensor derivatives are also much simpler with this formalism. When we derive with respect to the coefficient of a tensor, we just remove that tensor from the network.

For example, is equivalent to

$$\frac{\partial}{\partial A_{\alpha_1\alpha_2\alpha_3}} \left[A_{\alpha_1\alpha_2\alpha_3} B_{\beta_1\beta_2\beta_3} C_{\beta_1\gamma_1} D_{\beta_3\gamma_1\delta_1}\right] = B_{\beta_1\beta_2\beta_3} C_{\beta_1\gamma_1} D_{\beta_3\gamma_1\delta_1}$$

When performing derivations, we have to keep in mind that A_α and A^*_α are independent, and hence which is equivalent to

$$\frac{\partial}{\partial A_\alpha} \left[A_\alpha A^*_\alpha\right] = A^*_\alpha$$

Sometimes, we have vertices with a lot of edges: This situation can, for example, arise when dealing with many bodies wave functions:

$$\langle\psi|\psi\rangle = \sum_{\alpha_1\cdots\alpha_N} \psi_{\alpha_1\cdots\alpha_N} \psi^*_{\alpha_1\cdots\alpha_N} \longleftrightarrow$$

with $\alpha_i = 1, ..., d$. We can obtain a single index starting from the N initial ones:

$$\longleftrightarrow p$$

where now the index α runs from 1 to d^N. This procedure is known as *fusion*. The reverse procedure is called *splitting*. The philosophy of tensor networks is to never explicitly write down high rank tensors, replacing them with networks of smaller ones.

5 Quantum Information Lab

Modern computers are based on the Von Neumann architecture; his proposal was to split hardware and software, by introducing a memory that could store not only data but also programs (instructions for the computer):

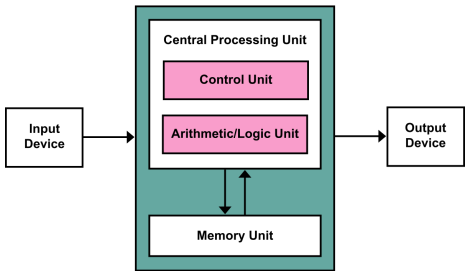


Figure 1: Von Neumann architecture.

In this architecture, we can identify three main subcomponents:

- Control unit (CU): a finite state machine (a cellular automata) which moves through those states following precise rules. The CU fetches data and instructions from the memory; it also sequentially coordinates all the operations.
- Arithmetic-logic unit (ALU): dedicated hardware for performing operations on register (based on clock cycles). Can also be coupled with coprocessors, for aiding in dealing with real numbers and their arithmetic.
- Memory: it contains both instructions (program) and data. This can be done because compilers turn source code into binaries, which can then be stored into the memory.

Numbers are processed by computers through binary code: one *bit* can either be a "1" or a "0", and a sequence of 8 bits is called a *byte*. Usually, the first bit is used for the sign of the number, so that, if we call s the value of the sign bit, the number will have its sign be equal to $(-1)^s$. Therefore, a byte can store a value from -128 up to 127. To subtract a number n, we can add instead $\bar{n}+1$, where $\bar{n}$ is the number obtained by substituting each 1 with a 0 and viceversa. In FORTRAN, there are different dimensions for integers: For real numbers, the exponential notation is particularly useful:

$$x = a \cdot 10^b,$$

where a is the *mantissa* and b is the *exponent*. We can employ 1 bit to store the sign, 1 byte to store the exponent and (3 bytes - 1 bit) to store the mantissa. In order to take into account negative exponents, the following convention is employed:

$$x = (-1)^s \cdot 2^{EXP-127} \cdot [1, F_{22} \cdots F_0]$$

Allocated memory	**Name**	**Range**
2 bytes	integer*2	$-2^{15} : 2^{15} - 1 \sim 3 \cdot 10^3$
4 bytes	integer*4	$-2^{31} : 2^{31} - 1 \sim 10^9$
8 bytes	integer*8	$-2^{63} : 2^{63} - 1 \sim 10^{17}$

Table 2:

By following this convention, we can get a precision of roughly 8 digits (in base 10) with 4 bytes. In a wide array of cases, this is not enough. With the *double precision* (8 bytes) we can achieve 15 digits of precision (in base 10), which is enough for physical applications (among the few exceptions are time measurements with atomic clocks and gravitational waves). As for the single digit case, one bit is reserved for the sign, while 11 bits are used for the exponent and the remaining 52 bits are used to store the mantissa.

The CPU is very fast: the bottleneck in computation is memory. Accessing and storing data are the slowest parts of the whole process. In order to improve this, a"memory hierarchy" has been created; the larger the storage capacity, the lower the speed.

The slowest is the *hard disk*, which serves as the long-term memory. Then the *RAM*, and then, next to the CPU, the *cache* (which itself consists in 3 layers, called L_1, L_2, L_3, with different speeds and dimensions). When a piece of memory is loaded in the cache, also nearby memory is loaded, as it is likely to be used in the immediate future. This is linked to the so called "principle of locality".

5.1 Brief summary of scientific computations

First of all, the goal is to describe nature up to a discrete precision. We can choose between floating point arithmetic, which is more flexible, and fixed point, which is more efficient. Indeed, in the latter, numbers are represented as

$$x = \Delta \times N,$$

where Δ can be thought of the smallest value which can be represented. Any other number can then be written as the product above. The main benefit of this approach is that we only have to care about N for all operations. Obviously,

$$x_{max} = \Delta \times N_{max}$$

As simple as it may seem, this can actually be troublesome: surpassing the maximum value x_{max} causes errors, one famous example of which was the Ariane 5 rocket incident.

A piece of code was responsible for controlling the acceleration of the rocket, and it used fixed point arithmetic. As it had performed flawlessly over the previous 20 years for the previous Ariane models, no one bothered to update it for the brand new Ariane 5. This rocket, however, was much more powerful

than its predecessors, and as a consequence its sensors experienced much higher values of acceleration than any other rocket before it. The code failed because the fixed point encoding was not capable of representing numbers as big as those that were experienced. The rocket exploded shortly after its takeoff. The moral of the story is that fixed point programs must be routinely checked and updated, more so than floating point ones.

We want our code to do research. As such, we are working within a specific framework:

- tasks push us to the limits (solving a single spin-$\frac{1}{2}$ system is not interesting, but solving a 55 spin-$\frac{1}{2}$ system would render Google's \$2 bn quantum computer obsolete);
- fast evolution (in a, say, 3 year span, most things become obsolete);
- goals are not necessarily well defined: on the contrary, they are often ill-defined, if not changing altogether. This means that we will face the dilemma of either thrashing everything each time or reuse what can be reused.

The requirements of scientific computation are:

R1 Change your program on the fly;

R2 Work with a lot of data;

R3 Solve hard problems (that is, either facing large scale problems or inefficient/not efficient enough algorithms).

Its solutions are:

S1 writing good, flexible, easy to debug/expand software: $\Rightarrow$ Software engineering;

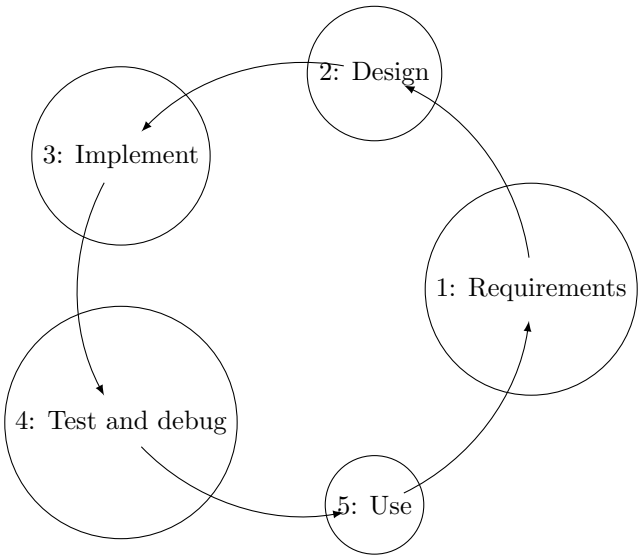
S2 automatize the work as much as possible, employing pre and post processing: $\Rightarrow$ Scripting;

S3 be as efficient as possible, and be prepared to use brute force: $\Rightarrow$ Optimization and HPC.

Why the distinction between "good" and "optimized" code? Good code can indeed be inefficient (a suboptimal algorithm may have been employed), but can be easily optimized by choosing a better algorithm. In that sense, the code is good.

When we write nontrivial code, it goes through a series of steps. We refer

to this as software lifecycle, and it goes like this:



The longer we go without having to rewrite everything, the better we have done the job. Eventually, however, rewriting becomes a necessity (think, for instance, of an OS like Windows; despite the many updates, eventually a brand new version pops out).

There are three main levels of programming, which refer to the three main different scopes of programming. Programming in the large requires:

a1 I/O specifications (Who is the user of our program? What should the program do? What performance should we achieve?);

a2 Data structures;

a3 Libraries and languages;

a4 Software portability;

a5 Coordination of software development.

Programming in the small requires:

b1 Avoiding premature optimization: "premature optimization is the root of all evil";

b2 Functions and variables having smart names; they should be meaningful while avoiding overdescription. When a program has more than 10 functions, each of them named 'f', 'g', 'h' and so on, errors become a given.

b3 Comments:

- comment on the interfaces (and not on the inner workings of subroutines/functions);
- declare loop invariants and conditions;
- warnings for unusual behavior;
- avoid redundant comments;
- keep comments updated.

Programming in the middle requires:

c1 Mathematical abstractions (for instance, if some data lends itself to matrix representation, use matrices);

c2 Information hiding (allowing for better interactions, reusability, modularity...)

c3 Flexibility (ability to easily implement additional features and/or extend use cases);

c4 Layered/modular software (BLAS, basic linear algebra subprograms);

c5 Interfaces;

c6 Data structures;

c7 Functions;

and so on.

Priorities are, in decreasing order of importance:

1. Correctness: $\rightarrow$ to be useful;

   This can be checked through the following:

   $\rightarrow$ compiler: checks for (mostly) syntax errors;

   $\rightarrow$ controlling memory allocation;

   $\rightarrow$ including pre and post conditions in every section of the program;

   $\rightarrow$ including checks for signs, norms, types, dimensions, and so on;

   $\rightarrow$ loop-invariants;

   $\rightarrow$ constants of motion/symmetries.

2. Numerical stability $\rightarrow$ to be trustful;

3. Accurate discretization: $\rightarrow$ to describe nature well;

4. Flexibility: $\rightarrow$ to be "used";

5. Efficiency: $\rightarrow$ to do a good job.

How can we reach these goals?

- Incremental testing: start from easy requirements during testing;
- Pre and post conditions: check that I/O is reasonable;
- Keep all the tests;
- Include errors & exceptions, in order to improve flexibility;
- Stress (large size problems) and random testing;
- Employ checkpoints in order to ease the debugging process.

It is important to be careful with all those operations which multiply the discretization error, especially during iterative procedures. For instance, let us define

$$I_k = \int_0^1 x^k e^{-x} dx$$

it is easy to check that

$$I_0 = 1 - e^{-1}$$

and

$$I_k = kI_{k-1} - e^{-1}$$

At each step, we multiply the initial error δ by k. So, after n steps, the error has become

$$1 \cdot 2 \cdots (n-1) \cdot n\delta = n!\delta$$

which is quickly increasing with n. A possible trick is to invert the formula:

$$I_{k-1} = \frac{I_k + e^{-1}}{k}$$

Now, the error gets divided by k at each step.

Every physical problem we choose to study by means of computational methods has to go through a finite discretization procedure. This discretization has to be accurate, i.e., it has to describe nature in the continuous limit. Let us consider a physical problem parameterized by time and space:

$$(x, t) \in [0, L] \times [0, T]$$

If dx and dt are the minimum space and time intervals respectively, the values of the system variables are given by

$$x_l = dx \cdot l, \qquad t_n = dt \cdot n$$

For example, if we consider a wave function in 1D, we get

$$\psi(x) = \psi_{dx} + e_{dx}$$

where ψ is the true (continuous) wave function, ψ_{dx} is the discretized version, stored in the computer's memory, and e_{dx} is the error. The requirement of *consistency* is summarized in the following limits:

$$\psi_{dx} \xrightarrow[dx\to 0]{} \psi \Longrightarrow e_{dx} \xrightarrow[dx\to 0]{} 0$$

As for the time, we can consider the temporal evolution operator $\mathbf{U}(t)$ such that

$$\psi(t) = \mathbf{U}(t)\psi(0)$$

and its discrete counterpart $\mathbf{U}_{dx}$. To get consistency, we require

$$\mathbf{U}_{dx} \xrightarrow[dx\to 0]{} \mathbf{U}$$

To check for consistency, we can check conserved quantities. Indeed, if the discretization is not consistent and the previous limits do not hold, some symmetries can be broken. For example, we can check the conservation of the energy or of the wave function's norm.

Another important parameter for a discretization procedure is the *accuracy*:

$$|\psi_{dx} - \psi| \sim dx^p$$

for a certain p. Different algorithms may have different accuracies. Let us consider two simple algorithms that compute the first derivative of a certain function $f(x)$. For the first one, we compute

$$f'(x) = \frac{f(x+dx) - f(x)}{dx}$$

while for the second one

$$f'(x) = \frac{f(x+dx) - f(x-dx)}{2dx}$$

Even if from an analytic point of view they are completely equivalent, and the results the yield in the limit $dx \to 0$ are the same, the second one is more accurate. Indeed,

$$\begin{aligned}\frac{f(x+dx) - f(x)}{dx} &= \frac{f(x) + f'(x)dx + \frac{1}{2}f''(x)dx^2 + \cdots - f(x)}{dx} = f'(x) + \frac{1}{2}f''(x)dx + \cdots = \\ &= f'(x) + \mathcal{O}(dx)\end{aligned}$$

while

$$\begin{aligned}\frac{f(x+dx) - f(x-dx)}{2dx} &= \frac{f(x) + f'(x)dx + \frac{1}{2}f''(x)dx^2 + \frac{1}{6}f'''(x)dx^3 + \cdots - f(x) + f'(x)dx - \frac{1}{2}f''(x)dx^2 + \frac{1}{6}f}{2dx} \\ &= f'(x) + \frac{1}{3}f'''(x)dx^2 + \cdots = \\ &= f'(x) + \mathcal{O}(dx^2)\end{aligned}$$

The intuitive reason is that the second algorithm computes the average between the right and the left derivatives. From a geometric point of view, this is even more clear:

To obtain flexible code, we should:

- think generally (not only about the problem at hand);
- if it is a specialized program, then build libraries.

There is always a tradeoff between concrete implementation and abstraction/generalization.

At the end, a program should also be efficient, i.e., we should try to reduce the complexity. One of the measures of complexity is the *algorithmic complexity*: if the input is of size n, then the algorithms spend a time equal to t, where

$$t \propto \mathcal{O}(f(n))$$

If $f(n)$ is polynomial, we say that it is an easy problem. If, on the other hand, $f(n)$ grows faster than a polynomial, we say that the problem is hard. We can also define other types of complexity; for example, *smooth complexity* measures how much time is required to get close to the exact solution, without reaching it. Here, "close" means that we can get closer than ϵ, for an arbitrary choice of ϵ. Some problems require a non polynomial time to be solved exactly, but just a polynomial time to get close to the correct solution.

In order to reduce the complexity of an algorithm we can:

- use different compilers;
- allocate memory;
- use parallelization;
- use ready-made code.

A Krylov subspaces

While using Lanczos algorithm, we are actually taking advantage of the properties of Krylov subspaces. A couple of definitions follow, for the more mathematically apt.

Krylov space Given a square matrix M and a vector $\vec{x}$, the sequence

$$\mathcal{K}_n(M) = \text{Span}\{\vec{x}, M\vec{x}, \ldots, M^n\vec{x}\}$$

is called a Krylov space of order n. A Krylov space $\mathcal{K}$ is N-invariant if, $\forall$ $\vec{x} \in \mathcal{K}$, the vectors $M\vec{x}, M^2\vec{x}, \ldots M^N\vec{x} \in \mathcal{K}$.