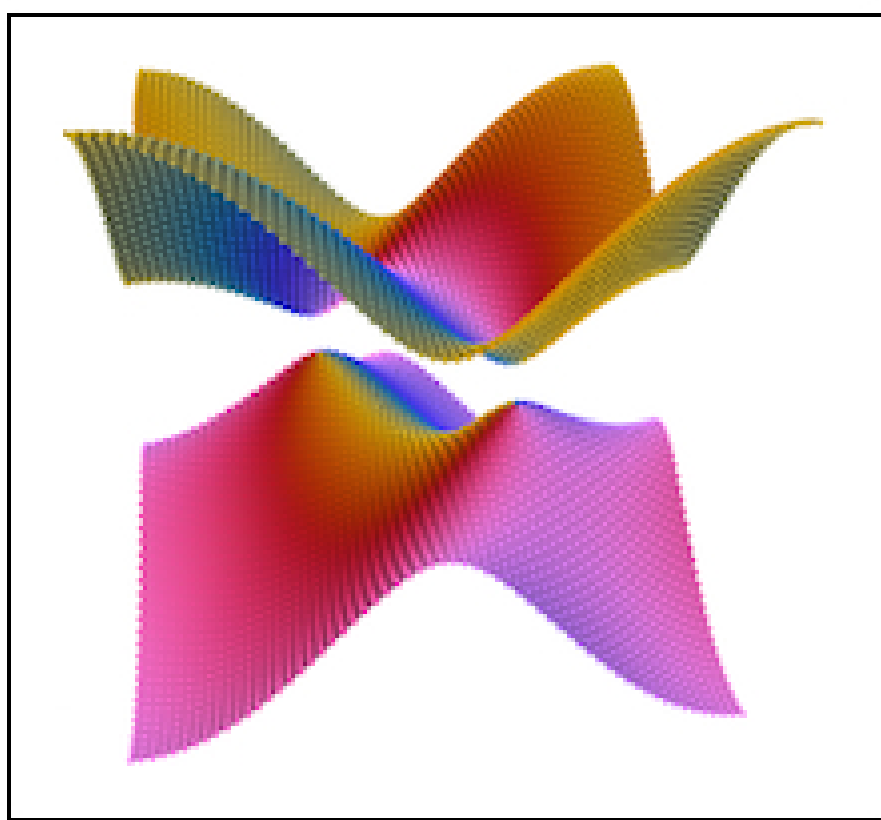


Journal notes on:
Introduction to numeric
methods for tight-binding
models in Quantica.jl

by Francisco Lobo, Pablo San-Jose



CONTENTS

Introduction	3
I Tutorial	4
I. Basics: from building the system to calculate its observables	4
A. Lattices	4
B. Models and Hamiltonian	5
C. Hamiltonians	7
D. Bandstructure	7
E. Greens functions	7
F. Observables	7
II. Advanced	7
A. Self-consistent mean-field problems	7
III. Appendix	7
A. Quantics Tensor cross interpolation	7
B. Numerical algorithms for multidimensional integration	9
References	11

INTRODUCTION

As a condensed matter theoretical physicist, one of the main goals of a work is to find fun, interesting and useful properties of a system at hand. Currently, the most efficient way to achieve such thing is to analysis the system's given observables numerically, thanks to the advancements of numerous super efficient numerical tools. These notes serve as an introduction to those necessary numerical methods.

The entirety of the code shown is written in the coding language *Julia*. The reason for this choice, say, for example, versus *Python*, a much more mature and wildy spread, is because Julia excels at ...[1]. **small paragraph why Julia is better for this type of computations.** We will *not* bother giving an introduction to the syntax's basics Julia as there are out there most better tutorials on the matter. If you are not familiar with Julia's basics I recommend checking out [2].

These notes, will, however, introduce the reader to the basics and inner works of the Julia package *Quantica.jl* developed by Pablo San-Jose. *Quantica.jl* main goal is to allow one to compute as efficiently as possible the observables of a system in tight-binding scheme. For this, there are basically a three-step program one follows. (1) For starters, define the lattice of your system and a tight-binding models with arbitrary parameter dependencies to build the a mono- or multiorbital Hamiltonian. (2) Having the Hamiltonian at hand you are able to compute the system's bandstructures or it's Green's functions (GFs). The GFs can be realized with *OpenHamiltonians*, i.e. Hamiltonians with attached self-energies from other Hamiltonians, such as leads. (3) From the GFs, we will be able to compute the system's obersables, such as spectral densities, current densities, local and nonlocal conductances, Josephson currents, critical currents, transmission probabilities, etc.

We closely following it's built-in tutorial [3] but provide more in-dept comments on the condensed matter theoretical and analytics theory behind it all, as well as some other details on the numerical front. Throughout the whole tutorial we will be using hexagonal boron nitride (hBN) as the main example, refereeing to the limiting case of graphene when it's interesting, with each step of the numerics introduced by the analytical counterpart. In the spirit of a noob-friendly tutorial, between the analytics and the numerics in *Quantica.jl*, there will be a section called "numerics for dummies", where I try to implement things in the most braindead manner possible. I do not recommend grownups to read this midsection as you will most certainly collapse from all the lovecraftian horrors you will try to make sense of.

The appendix section does not make part of the main tutorial as it is just me talking about miscellaneous numeric topics that I had to test or understand in a bit more detail for my own work.

Part I

Tutorial

I. BASICS: FROM BUILDING THE SYSTEM TO CALCULATE ITS OBSERVABLES

A. Lattices

Analytics Hexagonal boron nitride (hBN) is a 2D material composed of a simple layer of alternating boron and nitrogen atoms disposed in a planar honeycomb lattice, as shown in Fig.(2)(a). hBN shares a lot of similarities with graphene, also a 2D honeycomb structured material but instead composed of only carbon atoms. As we will see, the most relevant distinction is that graphene behaves as a semi-metal with a zero-gap at its Dirac points while hBN, due the different electrostatic environment in the boron and in the nitrogen atom, has an opening gap. Also, hBN has a slightly larger lattice constant than graphene (about 1.8%), being around $a_0 = 2.5\text{\AA}$ [Ish03].

The planar honeycomb lattice can be described as a triangular Bravais lattice generated by the real vectors basis

$$\mathbf{a}_1 = \frac{a_0}{2} \begin{bmatrix} 1 \\ \sqrt{3} \end{bmatrix} \text{ and } \mathbf{a}_2 = \frac{a_0}{2} \begin{bmatrix} -1 \\ \sqrt{3} \end{bmatrix}.$$

In each Wigner-Seitz cell, we have one boron atom and one nitride atom, which we designate as sub-lattices A and B respectively, being located at

$$\mathbf{s}_A = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ and } \mathbf{s}_B = \frac{a_0}{\sqrt{3}} \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

For each site A , the position of the nearest-neighbors (NN) in the sites B are given by

$$\boldsymbol{\delta}_1 = \frac{a_0}{2\sqrt{3}} \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \boldsymbol{\delta}_2 = \frac{a_0}{2\sqrt{3}} \begin{bmatrix} -\sqrt{3} \\ -1 \end{bmatrix} \text{ and } \boldsymbol{\delta}_3 = \frac{a_0}{2\sqrt{3}} \begin{bmatrix} \sqrt{3} \\ -1 \end{bmatrix}.$$

All these vectors are shown in Fig.(2)(a) within the real space lattice. Furthermore, from the real lattice basis vectors, in order to fulfill $\mathbf{a}_i \cdot \mathbf{b}_j = 2\pi\delta_{ij}$, the reciprocal lattice basis vectors follow as

$$\mathbf{b}_1 = \frac{2\pi}{\sqrt{3}a_0} \begin{bmatrix} \sqrt{3} \\ 1 \end{bmatrix} \text{ and } \mathbf{b}_2 = \frac{2\pi}{\sqrt{3}a_0} \begin{bmatrix} -\sqrt{3} \\ 1 \end{bmatrix}.$$

These are also shown in Fig.(2)(b) together with the first zone of Brillouin, formed by the area enclosed by the intersection of their bisectrices.

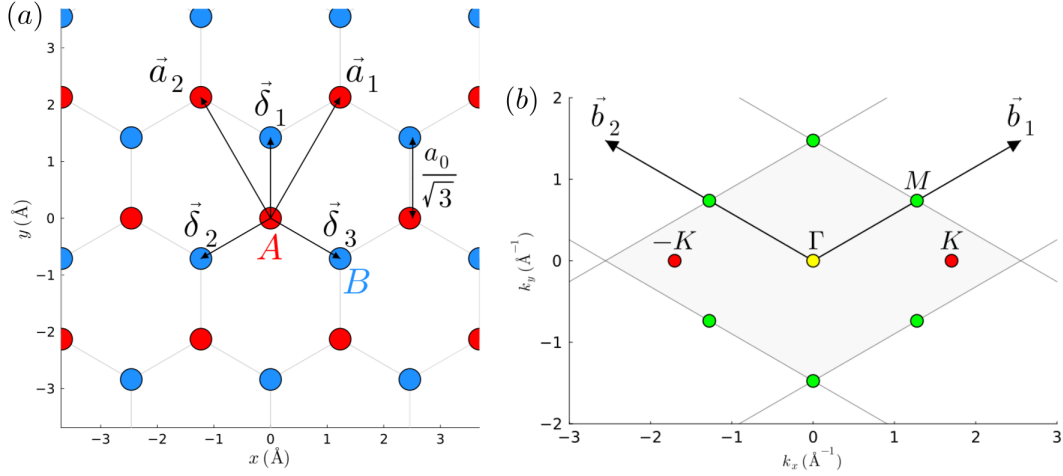


Figure 1.

Numerics for dummies aaaa
Numerics in Quantica.jl aaa

B. Models and Hamiltonian

In order to compute the excitonic energies using Eqs.(?) through (?), we first need to know the single-particle Bloch wave-functions $\phi_{\mathbf{k}\lambda}^s$. In this work we calculate them in a nearest-neighbors tight-binding model. For this, we write the system's single-particle NN tight-binding Hamiltonian in real space as

$$H_{\text{TB}}(\mathbf{R}) = \sum_i \epsilon_A a_{\mathbf{R}_i}^\dagger a_{\mathbf{R}_i} + \sum_i \epsilon_B b_{\mathbf{R}_i}^\dagger b_{\mathbf{R}_i} - t \sum_{\langle i,j \rangle} \left(a_{\mathbf{R}_i}^\dagger b_{\mathbf{R}_i + \delta_j} + b_{\mathbf{R}_j}^\dagger a_{\mathbf{R}_i - \delta_j} \right), \quad (1)$$

where the operators $a_{\mathbf{R}_i}^\dagger$ ($a_{\mathbf{R}_i}$) create (annihilate) an electron in the sub-lattice A in a given Bravais lattice site \mathbf{R}_i while the operators $b_{\mathbf{R}_i}^\dagger$ ($b_{\mathbf{R}_i}$) create (annihilate) an electron instead in the sub-lattice B (in a given Bravais lattice site \mathbf{R}_i). Therefore, the first two terms correspond to the isolated single-particles Hamiltonian of the site A and B , respectively, and the last term to the hybridization between neighboring sites i and j , describing the possible hoppings from site A to site B and vice-versa. We only assess hopping terms up to the first neighbors terms, which is denote by $\langle i, j \rangle$, and consider a static hopping term in either direction, i.e $t_{\mathbf{R}_i, \mathbf{R}_j} \rightarrow -t$. Notice that, contrarily to graphene, since the atoms on sites A and B are different the single-particle energies ϵ_A and ϵ_B are inherently different. We can represent the NN TB Hamiltonian in Eq.(1) in reciprocal space by expressing the creation/annihilation operators as their Fourier counterparts,

$$a_{\mathbf{R}_i} = \frac{1}{\sqrt{V}} \sum_{\mathbf{k}} e^{+i\mathbf{k} \cdot (\mathbf{R}_i + \mathbf{s}_A)} a_{\mathbf{k}} \quad \text{and} \quad b_{\mathbf{R}_i} = \frac{1}{\sqrt{V}} \sum_{\mathbf{k}} e^{+i\mathbf{k} \cdot (\mathbf{R}_i + \mathbf{s}_B)} b_{\mathbf{k}} \quad (2)$$

and rearranging the expression just that the identity $\delta(\mathbf{k} - \mathbf{k}') = 1/N \sum_i e^{-i\mathbf{R}_i \cdot (\mathbf{k} - \mathbf{k}')}$ is apparent. We obtain

$$H_{\text{TB}}(\mathbf{R}) = \sum_{\mathbf{k}} \epsilon_A a_{\mathbf{k}}^\dagger a_{\mathbf{k}} + \sum_{\mathbf{k}} \epsilon_B b_{\mathbf{k}}^\dagger b_{\mathbf{k}} - t \sum_{\mathbf{k}} \left(\gamma_{\mathbf{k}} a_{\mathbf{k}}^\dagger b_{\mathbf{k}} + \gamma_{\mathbf{k}}^\dagger b_{\mathbf{k}}^\dagger a_{\mathbf{k}} \right), \quad (3)$$

with the newly-defined γ complex number $\gamma_{\mathbf{k}} = \sum_{\langle j \rangle} \exp(+i\mathbf{k} \cdot \boldsymbol{\delta}_j)$. If we now define a row vector $c_{\mathbf{k}}^\dagger = [a_{\mathbf{k}}^\dagger \ b_{\mathbf{k}}^\dagger]$ we can rewrite the system's Hamiltonian as $H_{\mathbf{R}}^{\text{TB}} = \sum_{\mathbf{k}} c_{\mathbf{k}}^\dagger H_{\mathbf{k}}^{\text{TB}} c_{\mathbf{k}}$ with the hBN NN TB Hamiltonian matrix being

$$H_{\text{TB}}(\mathbf{k}) = \begin{bmatrix} \epsilon_A & -t\gamma_{\mathbf{k}} \\ -t\gamma_{\mathbf{k}}^\dagger & \epsilon_B \end{bmatrix}. \quad (4)$$

Within this simplified tight-binding model, the expression for the electronic two-band structure can easily be obtained analytically by diagonalizing the matrix in Eq.(4), yielding

$$E_{\text{TB}}^\pm(\mathbf{k}) = \pm \sqrt{\epsilon^2 + t^2 \left[3 + 2 \cos(a_0 k_x) + 4 \cos\left(\frac{a_0 \sqrt{3}}{2} k_y\right) \cos\left(\frac{a_0}{2} k_x\right) \right]}. \quad (5)$$

Here we defined the zero point energy at $(\epsilon_A + \epsilon_B)/2$ and defined $\epsilon \equiv (\epsilon_A - \epsilon_B)/2$ at the middle of the gap such that $\epsilon_A = \epsilon$ and $\epsilon_B = -\epsilon$. The valence band corresponds to the $E_{\text{TB}}^-(\mathbf{k})$ dispersion while the $E_{\text{TB}}^+(\mathbf{k})$ corresponds to the conduction band, as shown in Fig.(2)(c) which is accompanied by the density of states $\text{DoS}(E) = \sum_{\mathbf{k}} \delta(E - E(\mathbf{k}))$. Notice that, if $\epsilon_A = \epsilon_B$, as is the case for graphene, we obtain $\epsilon = 0$ and the band dispersion closes in a linear fashion at the so called Dirac points, $\mathbf{K}^\pm = (\pm 4\pi/(3a_0), 0)$. In hBN, the electronic band dispersion is also at its minimum near these points but has instead a parabolic shape. In either case, these points represent a fundamental symmetry of the system, called valley parity. To see why the dispersion is parabolic at these valley points, we Taylor series expand the exponential of $\gamma_{\mathbf{k}}$ in Eq.(?) near $\mathbf{k} \rightarrow \mathbf{K} + \mathbf{p}$ with $\mathbf{p} \rightarrow 0$. We obtain $\exp(+i\mathbf{p} \cdot \boldsymbol{\delta}_j) \approx 1 + i\mathbf{p} \cdot \boldsymbol{\delta}_j$. Now, since $\sum_{\langle j \rangle} \exp(+i\mathbf{K} \cdot \boldsymbol{\delta}_j) = 0$ we are left with $\gamma_{\mathbf{K}+\mathbf{p}} \simeq i\mathbf{p} \cdot \sum_{\langle j \rangle} \exp(+i\mathbf{K} \cdot \boldsymbol{\delta}_j) \boldsymbol{\delta}_j = -\sqrt{3}a_0/2 (p_x - ip_y)$. Invoking the Pauli matrices definitions, from Eq.(4) we can write the TB Hamiltonian $H_{\text{TB}}^{\mathbf{k}}$ in this low-energy regime as

$$H_{\text{TB}}(\mathbf{K} + \mathbf{p}) = \epsilon \sigma_z + t \frac{\sqrt{3}a_0}{2} (\mathbf{p} \cdot \boldsymbol{\sigma}), \quad (6)$$

which clearly resembles the 2D Dirac Hamiltonian, $H_{\text{Dirac}} = \sigma_z mc^2 + c(\mathbf{p} \cdot \boldsymbol{\sigma})$ with ϵ taking the role of the rest mass energy mc^2 and instead with a velocity $v_F = t\sqrt{3}a_0/2$, termed the *Fermi velocity*, as a replacement to the velocity of light c . Notice that, for the case of graphene, since $\epsilon = 0$, the electrons would behave as if they are massless. In this limit, the hBN low-energy dispersion can be written as the typical relativistic dispersion relation

$$E_{\text{TB}}(\mathbf{K} + \mathbf{p}) = \pm \sqrt{p^2 v_F^2 + m_{\text{eff}}^2 v_F^4}. \quad (7)$$

where m_{eff} is the effective mass of the electron at a given point near the valleys.

Although not captured in this simple tight-binding model, if one does some type of DFT to obtain a more complete electronic band structure, one could see that the hBN bands do actually cross between themselves (see, for example, Fig.(1) from [Fe19]). This appears to be troublesome to our two-band time-dependent Hartree-Fock mean-field theory since certain transitions could occur between bands that are not accounted for in our model. However, these other intersecting bands corresponds to electronic states that are orthogonal to the ones we use in our two-band model and thus, will not interfere (i.e, even if we accounted for this other bands in our model, the form factors in Eqs.(?)-(?) would always give zero for transitions between those bands). However, this only applies for the hBN case since, if one was dealing instead with TMDs, one would need to account for at least three bands [Li13]. Furthermore, following the context of TMDs, one should also need to account for the spin-orbit coupling (SOC) where the effective Hamiltonian for such a system could be obtained by adding to

Eq.(6) the term $H_{\text{SOC}} = t_s t s_z (\sigma_z - \mathbb{1})/2$ where t_s quantifies the spin-orbit coupling and $s_z = \pm 1$ labels the spin projection of the bands [Li13, Schl7].

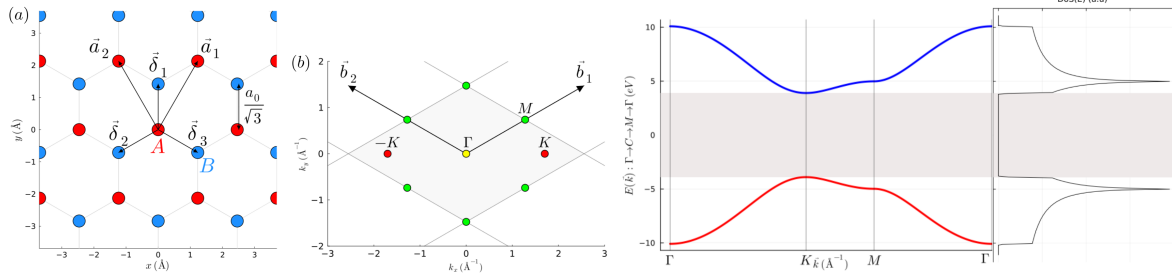


Figure 2. **(a)** hBN real space honeycomb lattice constructed from two superposed triangular sub-lattices of boron atoms (depicted in red), denoted as sub-lattice A , and of nitrogen atoms (depicted in blue), denoted as sub-lattice B . The vectors \mathbf{a}_1 and \mathbf{a}_2 are the lattice basis vectors and δ_1 , δ_2 and δ_3 are the nearest-neighbor vectors. **(b)** hBN reciprocal space lattice with \mathbf{b}_1 and \mathbf{b}_2 its basis vectors. The first Brillouin zone is emphasized in light gray while the remaining are only outlined. The red dots correspond to the Dirac points $\pm K$ and the green dots correspond to the 1BZ edges M points. **(c)** hBN electronic band structure from a nearest-neighbor tight-binding model accompanied by the density of the states. The dispersion goes along the symmetry path $\mathbf{k} : \Gamma \rightarrow K \rightarrow M \rightarrow \Gamma$ and was calculated using $\epsilon_g = 7.8\text{eV}$ for the energy gap, $t = 3.1\text{eV}$ for the hopping parameter and $a_0 = 1.42\sqrt{3}\text{\AA}$ for the honeycomb lattice length.

C. Hamiltonians

D. Bandstructure

E. Greens functions

F. Observables

II. ADVANCED

A. Self-consistent mean-field problems

III. APPENDIX

A. Quantics Tensor cross interpolation

Consider some complicated function of many variables, having its structure resolved in many different length scales, some long, some short. On one hand, long scales will demand large domains of definitions, on another hand, short scales will demand dense grids. Putting these two together means that any numerical operation with our function (for example, integrating it) requires HUGE amounts of memory costs. So, can this exponential cost of memory be avoided without loss of accuracy? Yes, if functions *compressible*! Recently, two promising strategies have emerged to account for this accurate resolution with parsimonious memory usage, *quantics* and *tensor cross interpolation* (TCI). The quantics representation expresses functions as multi-index tensors with each index representing one bit of a binary encoding of one variable and then the TCI, if applicable, yields parsimonious interpolation of those multi-index tensors.

Let us consider a many-body wavefunction of a $\mathcal{D} = 2$ dimensional tight-binding spin system. Let $x \rightarrow i = 1, \dots, \mathcal{N}$ and $y \rightarrow j = 1, 2, \dots, \mathcal{N}$ be the discretized grid site's nomenclature and σ_{ij} the system's spin degree of freedom, such that at a given site indexed by ij , the spin state $|\sigma_{ij}\rangle$ can either be spin-up $|\uparrow\rangle \equiv |1\rangle$ or spin-down $|\downarrow\rangle \equiv |0\rangle$. Let us consider the notation where $\boldsymbol{\sigma}$ encompasses all the system's variables, and that, instead of the 2D grid ij indexing, we consider instead the mapping into a 1D grid i , i.e $\{ij\} = \{1, \dots, \mathcal{N}\} \otimes \{1, \dots, \mathcal{N}\} = \{11, \dots, 1\mathcal{N}, 21, \dots, 2\mathcal{N}, \mathcal{N}1, \dots, \mathcal{N}\mathcal{N}\}$ to $\{\ell\} = \{1, \dots, \mathcal{L}\}$ with $\mathcal{L} = \mathcal{D} \times \mathcal{N} = 2\mathcal{N}$. We describe such wavefunction as

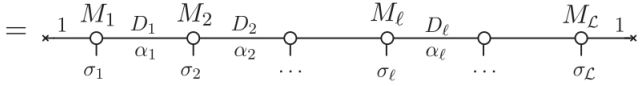
$$|\Psi(\mathbf{r})\rangle = \sum_{\boldsymbol{\sigma}} F_{\boldsymbol{\sigma}} |\boldsymbol{\sigma}\rangle \quad (8)$$

where the probability of the system being in a certain configuration state $|\boldsymbol{\sigma}\rangle \equiv |\sigma_1 \sigma_2 \dots \sigma_{\mathcal{N}}\rangle \equiv |\sigma_1\rangle \otimes \dots \otimes |\sigma_{\mathcal{L}}\rangle$ is dictated by $F_{\boldsymbol{\sigma}}$. So, for example, for a $\mathcal{N} \times \mathcal{N}$ grid with $\mathcal{N} = 2$ we have $\mathcal{L} = 4$ indexed sites/bits, meaning that the system can be in one of the $2^{\mathcal{N}} = 16$ configurations $\{|0000\rangle, |0001\rangle, |0010\rangle, \dots, |1111\rangle\}$ with probabilities $\{F_{0000}, F_{0001}, F_{0010}, \dots, F_{1111}\}$ respectively.

The main idea is that one can compress F as a product of matrix states (MPS), each encoding the information of a given site ℓ , yielding a factorized representation,

$$F_{\boldsymbol{\sigma}} = f(\sigma_1, \sigma_2, \dots, \sigma_{\mathcal{L}}) \approx \mathbf{M}_1(\sigma_1) \mathbf{M}_2(\sigma_2) \dots \mathbf{M}_{\mathcal{L}}(\sigma_{\mathcal{L}})$$

In a so-called *tensor trains* (TT) graphical notation one can represent this function as

$$F_{\boldsymbol{\sigma}} \approx [M_1]_{1\alpha_1}^{\sigma_1} [M_2]_{\alpha_1\alpha_2}^{\sigma_2} \dots [M_{\mathcal{L}}]_{\alpha_{\mathcal{L}-1}1}^{\sigma_{\mathcal{L}}}.$$


with the Einstein notation repeated dummy indices sum $\alpha_{\ell-1}, \alpha_{\ell}$ represented as the "train couplers" connecting the back and front "wagons", and the physical dependence on σ_{ℓ} of the matrices represented as the "cargo door" of each wagon. This compressed notation allows one to trivially calculate it's integral as a product of independent sums,

$$I^{(\mathcal{L})} = \int d\boldsymbol{\sigma} F_{\boldsymbol{\sigma}} \approx \sum_{\sigma_1} \mathbf{M}_1(\sigma_1) \sum_{\sigma_2} \mathbf{M}_2(\sigma_2) \dots \sum_{\sigma_{\mathcal{L}}} \mathbf{M}_{\mathcal{L}}(\sigma_{\mathcal{L}}).$$

At this point one might be thinking "Alright, this seems pretty neat, but how does one actually do this compression?" The standard toolbox for compressing tensors is via a repeated *singular value decomposition* (SVD) where each matrix is factorized as $\mathbf{M}_{\ell} \approx \mathbf{U}_{\ell} \mathbf{S}_{\ell} \mathbf{V}_{\ell}^{\dagger}$ with \mathbf{S}_{ℓ} a diagonal sparse matrix of dimension $\mathcal{L} \times \mathcal{L}$. See that the dimension of the "local" bond indices σ_{ℓ} is always $d = 2$ since σ_{ℓ} are basically function of position **ainda não percebi esta**. On the other hand, the dimension D_{ℓ} of the "virtual" bond indices α_{ℓ} does not need to always be \mathcal{L} as this would be an incredible waste of memory, so one truncates D_{ℓ} at a given D_{\max} , by discarding singular values smaller than a specified truncation threshold ε . In this sense, the SVD decomposition method is known as a rank-revealing decomposition. This is a reasonable approximation if $D_{\max} \ll 2^{\mathcal{L}/2}$, implying that $F_{\boldsymbol{\sigma}}$ has a noticeable internal structure that can be encoded with a small information content. However, even if $F_{\boldsymbol{\sigma}}$ is strongly compressible, the SVD unfolding strategy, although optimally accurate, is exponentially inefficient. In another words, although it uncovers structure in $F_{\boldsymbol{\sigma}}$, it does not exploit it as it is constructing it as it requires knowledge of the full tensor $F_{\boldsymbol{\sigma}}$. So, to resume, SVD provides great accuracy but has exponentially growing memory cost $\mathcal{O}(2^{\mathcal{L}/2})$ and exponentially long runtimes $\mathcal{O}(2^{\mathcal{L}})$, which is not, by any mean, ideal.

Hence comes the tensor cross interpolation (TCI), a exponentially more efficient algorithm, memory and runtime wise. It needs at most $\mathcal{O}(D_{\max}^2 \mathcal{L})$ function evaluations and a runtime of at most

$\mathcal{O}(D_{\max}^3 \mathcal{L})$, just with the small sacrifice of a tad bit of accuracy, requiring a slightly larger D_{\max} for a specified error tolerance ε .

The TCI algorithm serves as a black box that samples F_{σ} at some clever choices of σ and iteratively constructs the TT from the sampled values. TCI achieves the factorization needed for unfolding by employing matrix cross interpolation (MCI) rather than SVD. The MCI formula approximates it as

$$\mathbf{M}_{\ell} \approx \mathbf{C}_{\ell} \mathbf{P}_{\ell}^{-1} \mathbf{R}_{\ell} \quad (9)$$

$$\Leftrightarrow \left(\begin{array}{cccccccc} \bullet & \circ & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{array} \right) \approx \left(\begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) \left(\begin{array}{c} \circ \\ \circ \\ \circ \\ \circ \\ \circ \\ \circ \\ \circ \\ \circ \end{array} \right)^{-1} \left(\begin{array}{cccccccc} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{array} \right)$$

or $\begin{array}{|c|} \hline \square \\ \hline \end{array} \approx \begin{array}{|c|c|} \hline \square & \diamond \\ \hline \end{array} \begin{array}{|c|} \hline \square \\ \hline \end{array}$

The column, row and pivot matrices \mathbf{C} , \mathbf{R} , and \mathbf{P} , are all constructed from elements of \mathbf{M} : \mathbf{C} contains D columns (red), \mathbf{R} contains D rows (blue), and \mathbf{P} their intersections, the pivots (purple). The resulting $\tilde{\mathbf{M}}$ exactly reproduces all elements of \mathbf{M} contained in \mathbf{C} and \mathbf{R} ; the remaining (grey) elements are in effect interpolated from the “crosses” formed by these (hence “cross interpolation”). Understand that if the original matrix \mathbf{M} is completely random then the interpolation will fail miserably, meaning that non-smooth functions will have it’s issues. If $\mathbf{D} = \dim(\mathbf{M})$, one can obtain an exact representation of the full matrix, $\mathbf{M} = \tilde{\mathbf{M}}$.

It is important to note that MCI is not the numerical excellence peak of decomposition methods; as of today, that title belongs to *partial rank-revealing lower-upper* (prLU) decomposition. Although mathematically equivalent to MCI, it is numerically more stable as the pivot matrices are never constructed nor inverted explicitly. The standard LU decomposition factorizes a matrix as $\mathbf{M} = \mathbf{L}\mathbf{D}\mathbf{U}$, where \mathbf{L} is lower-triangular, \mathbf{D} diagonal and \mathbf{U} upper-triangular through the Gaussian elimination algorithm, known for inverting matrices or solving linear systems of equations. As a step up, the prLU decomposition has two additional features, it is both rank-revealing—the largest remaining element, found by pivoting on both rows and columns, is used for the next pivot—, and partial—Gaussian elimination is stopped after constructing the first $\tilde{\chi}$ columns of \mathbf{L} and rows of \mathbf{U} , such that $\mathbf{L}\mathbf{D}\mathbf{U}$ is a rank- $\tilde{\chi}$ factorization of \mathbf{M} .

B. Numerical algorithms for multidimensional integration

Here, we benchmark the efficiency of the different integration solvers in the eventuality of the functions having kinks and/or discontinuities. This will be very useful when deciding which of the solvers one should pick when integrating over the 1st Brillouin zone (1BZ) in order to obtain the density matrix of the system’s unbounded Hamiltonian at a given site at zero temperature,

$$\rho_{ii} = \frac{1}{2\pi} \int_{1\text{BZ}} d\mathbf{k} \sum_n^{\text{bw}} f(\omega_{\mathbf{k}n}, \omega_f) |\mathbf{k}n\rangle \langle \mathbf{k}n|,$$

where f is the Fermi-Dirac distribution, behaving as an Heaviside step-function at zero temperature at the Fermi energy ω_f , $\omega_{\mathbf{k}n}$ the eigen-energy at a given momentum point \mathbf{k} and band n on the bandstructure and $|\mathbf{k}n\rangle$ the respective eigen-state.

In general, performing this type of integration numerically is a very intensive task and several methods have been developed to make it as efficient as possible. Here we highlight the main contenders: the Gauss–Kronrod’s quadrature (QuadGK) [4], the Cubature integration [5] and the tensor cross interpolation (TCI) [6]. The main practical difference between QuadGK and Cubature is that Cubature

can, by itself, find the problematic points (be it kinks or discontinuities) and focus its attention around them, always avoiding evaluating exactly the point. QuadGK, on the other hand, is very much dependent on these problematic points, but one can actively subdivide the integrals into segments where the exact endpoints (chosen as the problematic points) are ignored. As for the TCI method, the main idea is to codify (in a highly efficient way, mind you) the function we want to integrate into a so-called tensor train, which we then operate on. In general, this method is very promising because one could codify something for single set of parameters and then, with little to no work, build that something for all other set of parameters. What this means in practicality is that one could compute a phase diagrams in record time. Integration wise there are some downturns because TCI does not deal well with discontinuities (which are unavoidable at zero temperature). If you wish to know how what exactly are in the inner works of these algorithms, as well as the theoretical details, check [7] for QuadGK, [8] for Cubature and [9][10] as well as the next section for TCI.

Since this is just a benchmark of the different algorithms, we will consider simpler functions $\int_{-1}^{+1} d\mathbf{r} f(\mathbf{r})$, namely a normal distribution, a cone and a cylinder as shown in Fig.(3), allowing us to test how the algorithms handle kinks and discontinuities respectively. We benchmark the integrations first for a 1D case, picking some linecut of fixed y , and then the full 2D integration. As we can see from the benchmark of this simple examples in Fig.(3), for smaller tolerances, the quadGK algorithm with the problematic points given a priori, wins hcubature every time, by, at least, a factor of 10. Because of this we take the hcubature method out of the race and focus only on the quadGK (with the problematic points given a priori) and on the QTCI. One could think that from this results QTCI seem to be even worse than hcubature, but this is only because we are not using it right and mostly because we aren't even in the regime when it excels at, which is high dimensional functions. So, for this, we examine in the next section how to really use the QTCI, and then compare it with quadGK.

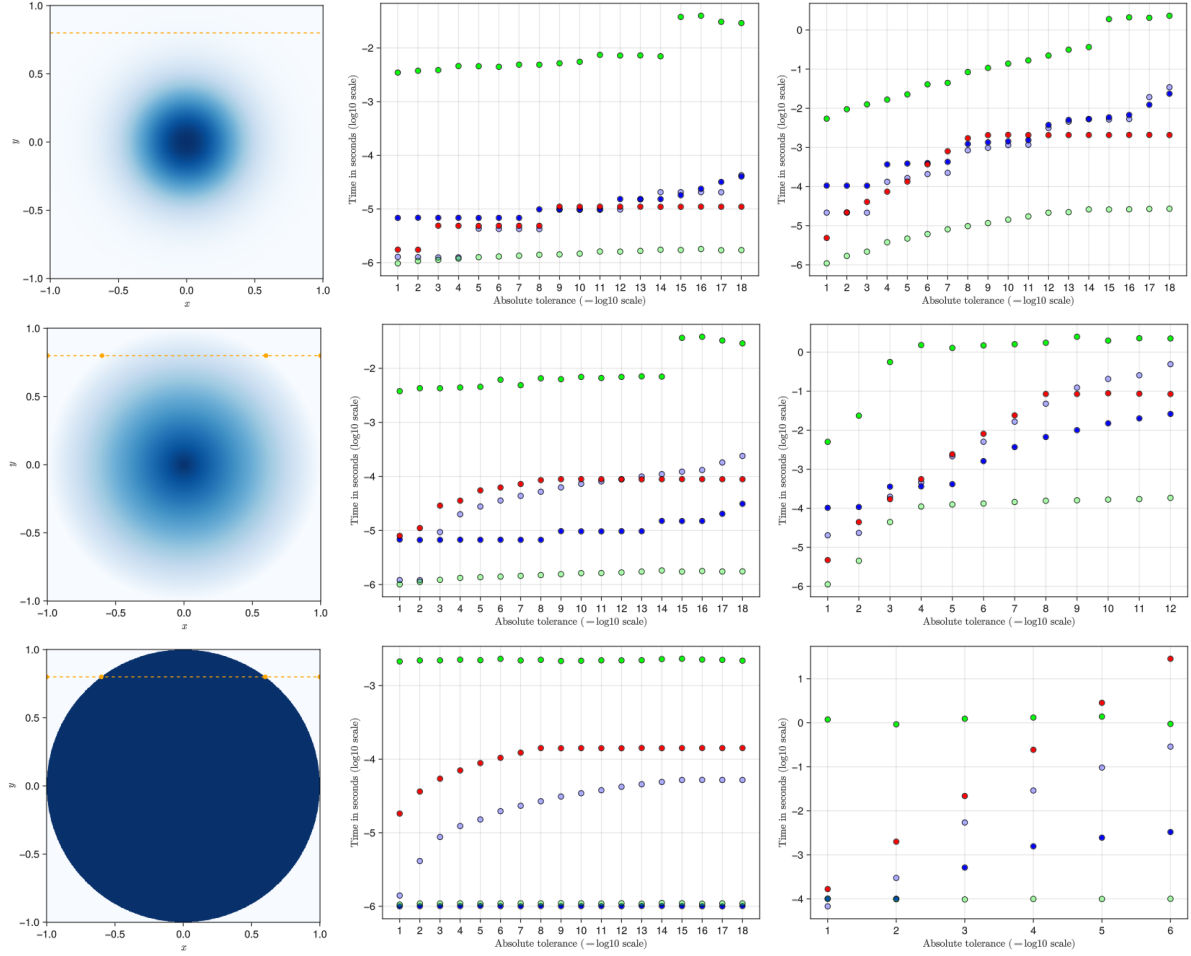


Figure 3. From left to right: **(top)** gaussian, **(center)** cone and **(bottom)** cylinder integration benchmark times in 1D (along the orange dashed linecut) and in 2D. As for the methods of integrations we have in (red) hcubature, (green) quantics TCI interpolation *plus* integration times, with transparent green jst the integration time, and (blue) quadGK, with transparent blue without feeding it the discontinuity/kink points

-
- [1] ??
 - [2] ??
 - [3] <https://pablosanjose.github.io/Quantica.jl/dev/tutorial/tutorial/>.
 - [4] <https://juliamath.github.io/QuadGK.jl/stable/gauss-kronrod/>.
 - [5] <https://github.com/JuliaMath/Cubature.jl>.
 - [6] <https://tensor4all.org/TensorCrossInterpolation.jl/dev/>.
 - [7] ??
 - [8] ??
 - [9] Phys. Rev. Lett. 132, 056501.
 - [10] ArXiv:2407.02454 [physics.comp-ph].