Journal notes on:

Miscellaneous topics on condensed matter physics

by Francisco Lobo

CONTENTS

I. MISCELLANEOUS TIGHT-BINDING MODELS

A. Anderson impurity model

B. Aubry-Andre model

C. Heisenberg model

D. Hubbard model

E. Ising model

F. Jaynes-Cummings model

II. MISCELLANEOUS NUMERICAL METHODS

A. Integration algorithms

1. *Schur's 1D integration method*

2. *Quantics tensor cross interpolation*

Quantics representation Consider some complicated function of many variables, having it's 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 $x \to i = 1, ..., \mathcal{N}$ and $y \to j = 1, 2, ..., \mathcal{N}$ 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 $|\uparrow\rangle \equiv |0\rangle$. 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 $\{ij\} = \{1, ..., \mathcal{N}\} \otimes \{1, ..., \mathcal{N}\} = \{11, ..., 1\mathcal{N}, 21, ..., 2\mathcal{N}, \mathcal{N}1, ..., \mathcal{N}\mathcal{N}\}$ to $\{\ell\} = \{1, ..., \mathcal{L}\}$ with $\mathcal{L} = \mathcal{D} \times \mathcal{N} = 2\mathcal{N}$. We describe such wavefunction as

$$|\Psi(\mathbf{r})\rangle = \sum_{\boldsymbol{\sigma}} f_{\boldsymbol{\sigma}} \, |\boldsymbol{\sigma}\rangle \tag{1}$$

where the probability of the system being in a certain configuration state $|\boldsymbol{\sigma}\rangle \equiv |\sigma_1 \sigma_2 ... \sigma_N\rangle \equiv |\sigma_1\rangle \otimes ... \otimes |\sigma_{\mathcal{L}}\rangle$ 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 $\{|0000\rangle, |0001\rangle, |0010\rangle, ..., |1111\rangle\}$ with probabilities $\{f_{0000}, f_{0001}, f_{0010}, ..., f_{1111}\}$ 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,

$$f_{\boldsymbol{\sigma}} \approx \mathbf{M}_1(\sigma_1)\mathbf{M}_2(\sigma_2)...\mathbf{M}_\ell(\sigma_\ell)...\mathbf{M}_{\mathcal{L}}(\sigma_{\mathcal{L}})$$

This compressed notation allows one to trivially calculate it's integral as a product of independent sums,

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

Tensor cross interpolation 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 $\mathbf{S}_\ell$ a diagonal sparse matrix of dimension $\mathcal{L} \times \mathcal{L}$. 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 $F_{\boldsymbol{\sigma}}$ at some clever choices of $\boldsymbol{\sigma}$ 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 $\mathbf{M}_\ell \approx \mathbf{C}_\ell \mathbf{P}_\ell^{-1} \mathbf{R}_\ell$. 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* (prrLU) 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 $\mathbf{M} = \mathbf{LDU}$, 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 prrLU 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 $\mathbf{LDU}$ is a rank-$\tilde{\chi}$ factorization of $\mathbf{M}$.

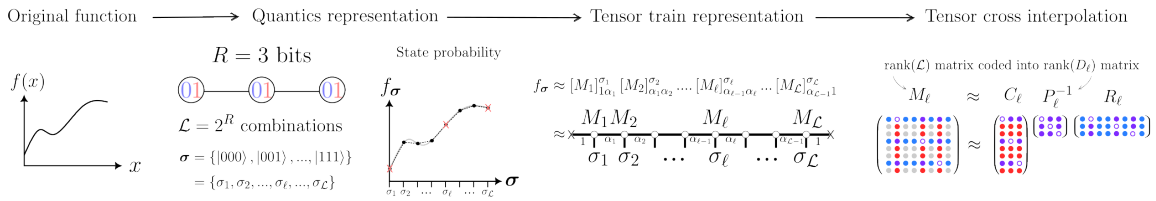

Figure 1. QTCI algorithm sketch. The red crosses on $f_{\boldsymbol{\sigma}}$ are to show that the TCI algorithm does not need to evaluate all points, only a selected few. For larger bit sizes $N_{TT} \lll \mathcal{L}$.

QTCI benchmarking for smooth, non-differentiable and discontinuous functions Here, we benchmark and test the QTCI codification algorithm (see a tutorial here [1]) for three different functions—smooth, non-differentiable and discontinuous functions—, in 1D and 2D. As we explained above, we know that the QTCI is very efficient to resolve structure in many different length scales. But how does it deal when this structure is not smooth? Will it find order in it's kinks and jumps? Or will it lose all efficiency? This become very relevant if one is, for example, dealing with systems at zero-temperature.

For the 2D tests we consider a normal distribution as our smooth function, a cone as our non-differentiable function and a cylinder as our discontinuous functions. For the 1D teste we consider the $y = 0.0$ linecuts of this functions.

Let us start with the 1D tests, which we expected better results since TT are a 1D dimensional object. In Fig.(2)(a) we show the

as shown in Fig.(2), 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.(2), 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.

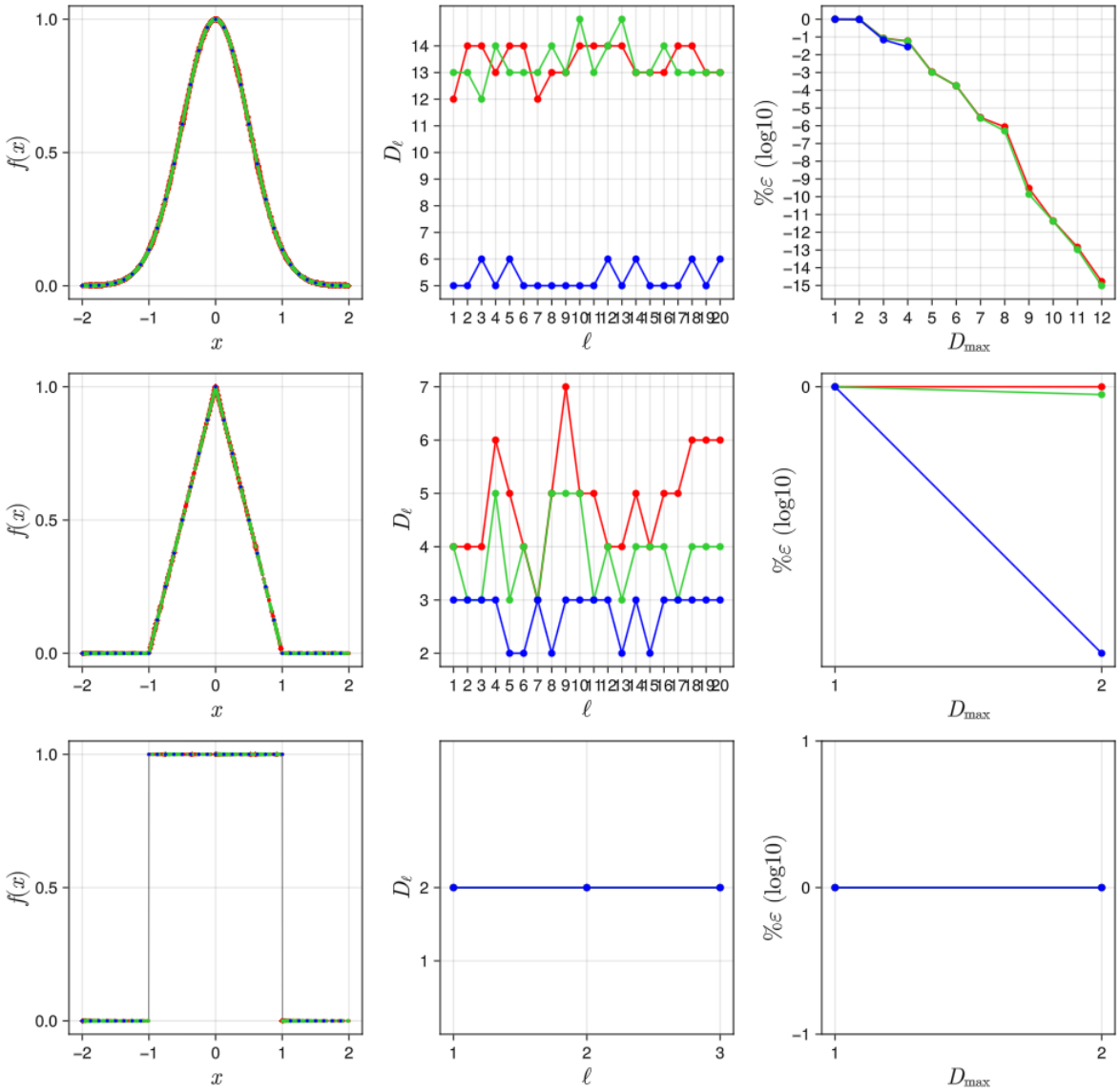

Figure 2. 1D QTCI benchmarking of a **(1)** Gaussian (smooth function), a **(2)** triangle (non-differentiable function), and a **(3)** square (discontinuous function). For the three cases, the panels **(a)** through **(c)** use a quantics grid of $R = 3$, 10 and 20 bits, at a total of $N_R = 2^R$ grid points, corresponding, respectively, to the QTCI reconstructed curves in red, green and blue. Moreover, the absolute tolerance for all curves is $\varepsilon_a = 10^{-10}$. In panels **(a)**, the scatter points in the different colors, at a total N_{QTT} for each color, are the points that the QTCI algorithm actually evaluated. For few bits $N_{QTT} \lessapprox N_R$ and for a lot of bits, ideally, $N_{QTT} \lll N_R$. The original function is shown in black. In panels **(b)** we plot the virtual bond dimensions D_ℓ of the QTT. In panels **(c)** we plot the relative error estimates as a function of $D_{\max} = \max(D_\ell)$, being capped at rank 20. In panels **(d)** we plot the time of QTCI as a function of the bits R and absolute tolerance ε_a tyo seek the

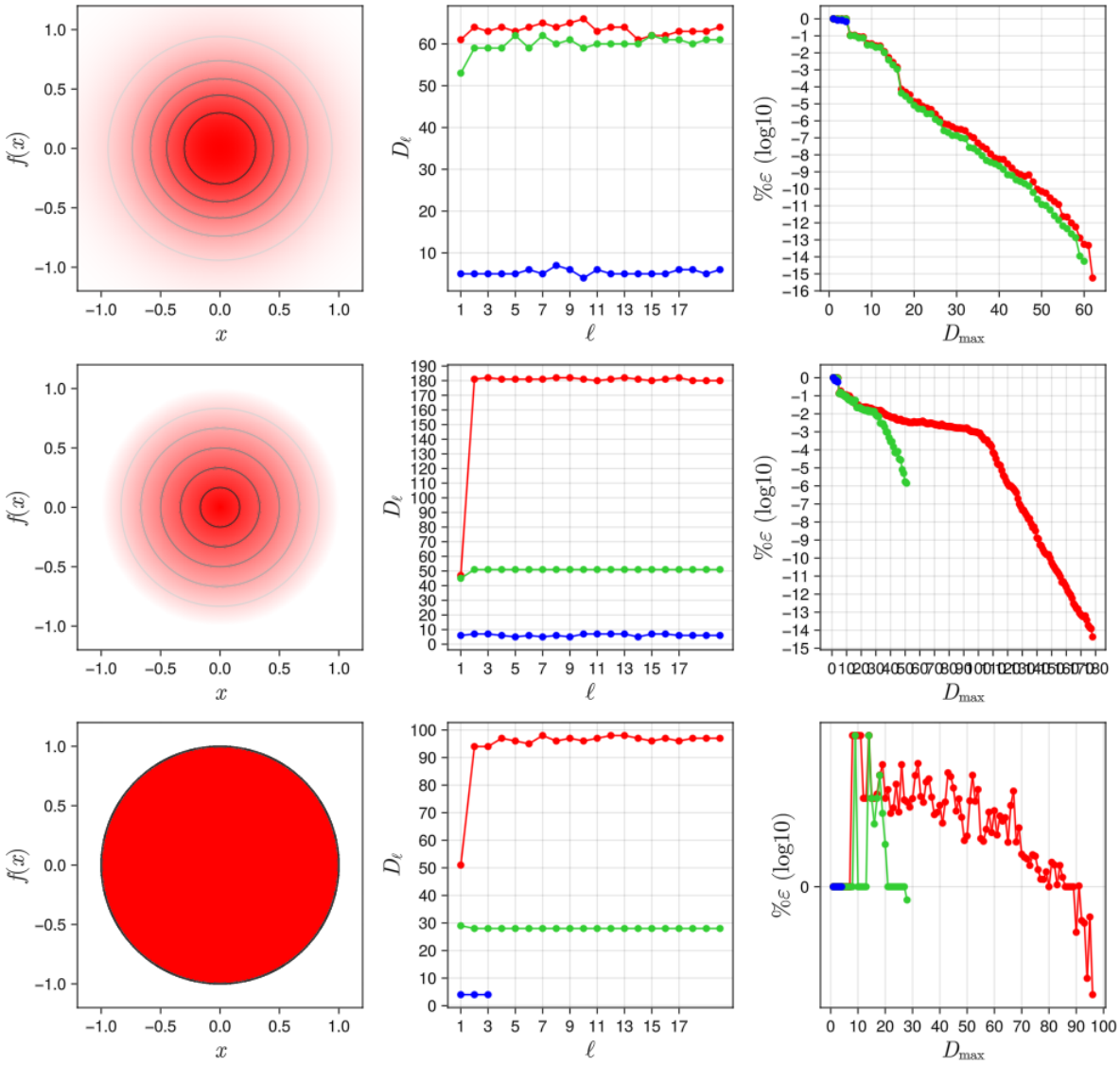

Figure 3. Same as Fig.(2) but in 2D and for $R = 3$, 6 and 9 bits with a a cap at rank 100.

3. *Benchmarking integration algorithms*

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,

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

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) [2], the Cubature integration [3] and the tensor cross interpolation (TCI) [4]. 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 [5] for QuadGK, [6] for Cubature and [7][8] 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.(4), 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.(4), 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.

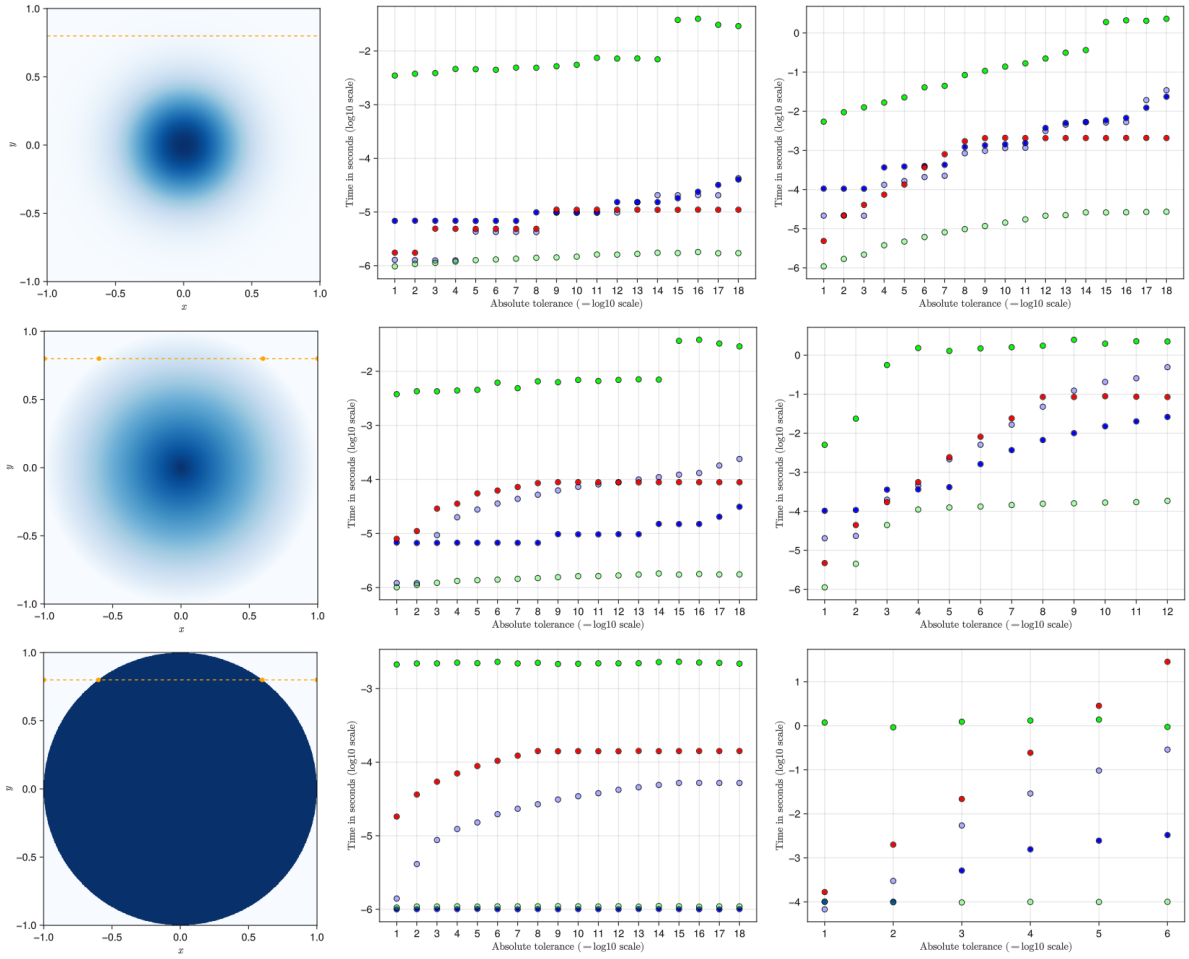

Figure 4. From left to right: **(top)** gaussian, **(center)** cone and **(bottom)** cylinder integration benchmark times in 1D (along the orange dashed linecut) and in 2D. As for the methods of integrations we have in (red) hcubature, (green) quantics TCI interpolation *plus* integration times, with transparent green jst the integration time, and (blue) quadGK, with transparent blue without feeding it the discontinuity/kink points

Buzzwords (in no particular order) that I should already know about or that seems interesting enough to have a closer look

1. Exciton Fractional
2. High-harmonic generation
3. Imaging techniques
4. Instanton
5. Kondo effect
6. Kondo insulators
7. Laudau levels
8. Linear response theory
9. Luttinger liquid
10. Luttinger-Kohn model
11. Magnon
12. Mott insulators
13. Semiconductor/Optical Bloch equations
14. Skyrmions

---

[1] Https://tensor4all.org/T4AJuliaTutorials/ipynbs/quantics1d.html#.
[2] Https://juliamath.github.io/QuadGK.jl/stable/gauss-kronrod/.
[3] Https://github.com/JuliaMath/Cubature.jl.
[4] Https://tensor4all.org/TensorCrossInterpolation.jl/dev/.
[5] ??
[6] ??
[7] Phys. Rev. Lett. 132, 056501.
[8] ArXiv:2407.02454 [physics.comp-ph].