Tensor network based integration methods

Author: Pau Torrente Badia
*Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona, Spain.**

Advisors: Stefano Carignano & Joan Soto Riera

Abstract: In this work we overview the Tensor Train Cross decomposition of large tensors and its applicability to high-dimensional integration. Furthermore, two different algorithms for building this decomposition are showcased and compared against a Monte Carlo method, both outperforming it in terms of resource efficiency. A python package is also presented, containing these two algorithms along other tools to leverage the power of the framework in a comprehensive and easy to use manner.

I. INTRODUCTION

Reliable numerical integration tools are essential, as analytical descriptions are not available in most cases. When working with very few variables, quadrature methods that evaluate the desired function in a discretized variable space are very useful. For systems with a large amount of degrees of freedom, though, the exponential increase in the amount of points to be evaluated makes these methods unusable. This is known as curse of dimensionality. In these cases where regular quadrature rules are not feasible, then, the standard tools to tackle numerical integration are Monte Carlo (MC) methods, which rely on sampling the function in the variable space randomly. Nonetheless, MC methods also have their drawbacks. First, they converge with the number of sampled points (N) only as $1/\sqrt{N}$. Secondly, when the integrated function rapidly oscillates and is not strictly positive, the random samples of opposite sign end up cancelling each other and not allow the method to converge. This type of integrals occur frequently e.g. in gauge theories with fermions, and this difficulty for reaching convergence is known as a sign problem.

In this work we study a tensor network approach to the same problem of highly-dimensional integration. Tensor networks have found success in condensed matter physics and applied mathematics by being able to reduce the correlation degrees of freedom of the studied system to just those that are necessary or that can be tracked. In the context of integration, the evaluation of the function in a grid is interpreted as a large tensor that is approximated with a tensor network with fewer elements. Neither its size nor the number of function samples needed to build it grow exponentially with the number of variables, and so regular quadrature rules can be used again. These methods not only have been shown to outperform MC methods in certain scenarios [1], but since they do not rely on a probabilistic process, theories which present sign problems can be treated without issues.

This paper, then, is organized as follows. In Sec. II, the formalism of tensor networks is briefly introduced. In Sec. III the tensor train approximation to the quadrature tensor is presented. In IV, two algorithms for building this approximation with a limited amount of samples are showcased. In Sec. V, both algorithms are tested against an integral of Ising type [2] and compared to a MC method. Lastly, conclusions are discussed in Sec VI.

*Electronic address: ptbadia@gmail.com

II. TENSOR NETWORKS

Tensor networks are at their core a diagrammatic description of multilinear equations. By representing tensors of rank N as objects with N legs coming out of them, large arrays of contracted tensors can be expressed in a visual way as a network interconnected via the legs that represent common indices. The indices that are not contracted, are left as open legs. Fig. 1 displays a very simple network for illustrative purposes.

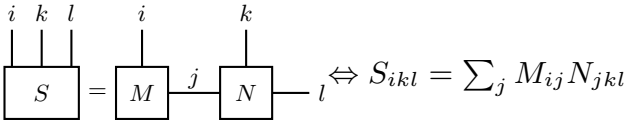

FIG. 1: Example of a very simple tensor network that decomposes a tensor S into two M, N, connected through index j.

Notice that the range of the bonded index j determines the number of entries in M and N, since the dimensions of the open legs can't change. We will refer to the dimensions of these bonded indices as bond dimensions.

Apart from simply representing systems of equations, tensor networks are also very powerful as a computational tool for dissecting larger systems into small parts and controlling the correlations between them. To perform these decompositions, the most commonly tool used is the singular value decomposition (SVD), which factorizes a matrix M into two isometries joined by a diagonal matrix of singular values as $M = USV^\dagger$. These singular values are a measure of the correlations between the two subspaces into which the matrix has been separated, and in the context of quantum physics, for example, are directly related to entanglement [3]. By discarding the smallest ones, this decomposition also guarantees the best low rank approximation to the original matrix.

The SVD then, allows us to go from full tensors to networks that contain fewer elements by just keeping the correlation degrees of freedom desired i.e, by adjusting the resulting bond dimensions. In the context of this work, it will be used in Sec. IV A to determine how correlated are two sets of variables.

III. THE TT-CROSS APPROXIMATION FOR INTEGRATION

Let $f : D_1 \times D_2 \times ... \times D_N \to \mathbb{C}$ with $D_k \subset \mathbb{R}$ be a multivariate scalar function. For generality purposes, we can suppose that it is complex valued. Its integral over the domain on which it is defined can be approached numerically using quadrature rules by evaluating it over all the points of a grid that discretizes the variables. We can encapsulate these evaluations in a tensor A of rank N whose entries are defined by:

$$\begin{aligned} A(i_1, \dots, i_N) &= f(x_{i_1}, \dots, x_{i_N}) \\ i_s &= 1, \dots, k_s, s = 1, \dots, N \end{aligned} \tag{1}$$

where by x_{i_s} we denote the i_s-th point in the grid in the s-th variable/direction. With this tensor then, the integral can be computed as:

$$\begin{aligned} I &= \int_{D_1} dx_1 \cdots \int_{D_N} dx_N f(x_1, \dots, x_N) \\ &\approx \sum_{i_1 \dots i_N} \prod_{s=1}^{N} \omega_s(i_s) A(i_1, \dots, i_N) \end{aligned} \tag{2}$$

where ω_s is a vector containing the quadrature weights for each of the points in the direction s of the grid.

Our goal then is to go from this N-legged tensor with an exponentially large number of entries to a tensor network approximation of the form:

$$\begin{aligned} A(i_1, \dots, i_N) \approx \sum_{b_1, \dots, b_{N-1}} & X^{(1)}(i_1, b_1) X^{(2)}(b_1, i_2, b_2) \cdots \times \\ & \times X^{(N-1)}(b_{N-2}, i_{N-1}, b_{N-1}) X^{(N)}(b_{N-1}, i_N) \end{aligned} \tag{3}$$

We will refer to this construction as a tensor train, because of its chain-like structure, which is depicted in Fig. 2. The number of elements in it scales as $\mathcal{O}(Nn\chi^2)$, with $n = \max_k |\{i_k\}|$ and $\chi = \max_k |\{b_k\}|$. If the size of the bond indices can be contained while maintaining the quality of the approximation, then we will have compressed greatly the information of the function from the original tensor containing n^N elements.

Starting from the initial tensor this decomposition would easily be reached by performing sequentially $N-1$ singular value decompositions [4], but that would require the evaluation of the function in the whole grid, which is precisely what we want to avoid. We need, therefore, a way of building this tensor train with a cleverer strategy, leveraging more efficiently the information given by our function of interest.

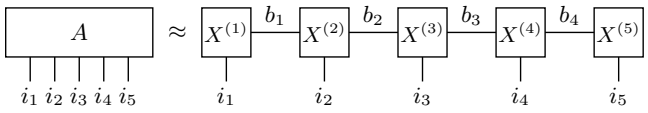

FIG. 2: Tensor network representing the decomposition of the initial tensor A from Eq. (5) for the case $N = 5$.

A. Cross decomposition of a matrix

Let A be a matrix of size $n \times m$ and rank r, this is, that it has r linearly independent rows and columns. Let $I = \{i_1, i_2, ..., i_k\}$ and $J = \{j_1, j_2, ..., j_k\}$ be sets of k row and column indices, respectively, and $A(I, J)$ the submatrix formed with the entries that lie at the intersection of rows I and J in A. We will refer to these sets as *crosses*. If $\text{rank}(A) = k$ and the sets I and J denote k linearly independent rows and columns on A, the elements of the matrix can be expressed as:

$$A(i, j) = \tilde{A}(i, j) := \sum_{s,l=1}^{k} A(i, j_l) \left[A(I, J)\right]^{-1}_{(i_s, j_l)} A(i_s, j) \tag{4}$$

or using Einstein notation to avoid writing the sums:

$$A(i, j) = \tilde{A}(i, j) = A(i, J) \left[A(I, J)\right]^{-1} A(I, j) \tag{5}$$

which is known as the skeleton or cross decomposition of A. If on the other hand $\text{rank}(A) > k$ the cross decomposition will be an approximation of A. In this case this approximation will be best when $A(I, J)$ is the submatrix of A of maximal volume. Despite the fact that finding this submatrix is an NP-Hard problem [5], the *Maxvol* algorithm introduced in Ref. [6] presents an iterative and heuristic way of obtaining a large volume submatrix of size $k \times k$ in a $r \times k$ matrix to construct the cross approximation with a bounded error. Regardless of if $\text{rank}(A) = k$ or not, one of the most important characteristics of the decomposition is the exact interpolating property:

$$\begin{aligned} A(i, j) &= A(i, J) \left[A(I, J)\right]^{-1} A(I, j) \\ &\text{if } i \in I \text{ or } j \in J, 1 \leq |I|, |J| \leq \text{rank}(A) \end{aligned} \tag{6}$$

This is, the entries of the decomposition in the crosses are equal to those in the original matrix, no matter the number of crosses used to build the approximation.

B. TT-Cross approximation

The matrix cross approximation can be extended into the tensor train format [4]. Let A be a rank N tensor with entries $A(i_1, \dots, i_N)$. This tensor can be reshaped into an arbitrary rank tensor by grouping indices $i_1, \dots, i_s$, for example, into a single multi-index $i_1 \dots i_s$. When these

grouped indices are ordered, we introduce the following notation shortcuts:

$$i_{\leq k} = i_1 \dots i_k, \quad i_{>k} = i_{k+1} \dots i_N \tag{7}$$

With this we can, for example, reshape our initial tensor to another one of rank 3 where its entries are given by:

$$B(i_{\leq k-1}, i_k, i_{>k}) = A(i_1, \dots, i_N) \tag{8}$$

For the following developments, we will omit the renaming to B and take the shape by the number of indices that are used to compute the entries of the tensor. We will also reference a tensor from just its entries, i.e. the tensor $A(i_{\leq k-1}, i_k, i_{>k})$.

With this, the tensor $A(i_1, \dots, i_N)$ can be reshaped into a matrix that isolates the first index from the rest and apply to it the cross decomposition, such that its entries are given by:

$$A(i_1, i_2 \dots i_N) \approx \sum_{t_1, s_1 = 1}^{r_1} A(i_1, \mathcal{J}_{t_1}^{(1)}) \times \\ \times \left[A(\mathcal{I}_{s_1}^{(1)}, \mathcal{J}_{t_1}^{(1)}) \right]^{-1} A(\mathcal{I}_{s_1}^{(1)}, i_2 \dots i_N), \tag{9}$$

where $\mathcal{I}^{(1)}, \mathcal{J}^{(1)}$ are sets of r_1 multi-indices $i_{\leq 1}$ and $j_{>1}$. If r_1 is smaller than the rank of $A(i_1, i_2 \dots i_N)$, the choice of indices will be crucial for the quality of the approximation.

If we proceed in the same way with the rightmost matrix in Eq. (9), reshaped into $A(\mathcal{I}_{f_\infty}{}^{(1)} i_2, \dots i_N)$, and repeat the same process iteratively for all the remaining indices, we arrive at the Tensor Train Cross decomposition of our initial rank N tensor A:

$$A(i_1, \dots, i_N) \approx \tilde{A}(i_1, \dots, i_N) = A(i_1, \mathcal{J}^{(1)}) \times \\ \times \left[A(\mathcal{I}^{(1)}, \mathcal{J}^{(1)}) \right]^{-1} A(\mathcal{I}^{(1)}, i_2, \mathcal{J}^{(2)}) \dots A(\mathcal{I}^{(N-1)}, i_N) \tag{10}$$

where Einstein notation for the index sums has been used, as in Eq. (5). In this expression $\tilde{A}$ is the TT-Cross decompostion of A and $\mathcal{I}^{(k)}, \mathcal{J}^{(k)}$ are sets of r_k multi-indices $i_{\leq k}$ and $j_{>k}$, respectively. We will refer to r_k as the TT-ranks of the approximation, which are a measure of the correlations between the variables at both sides of the chain. Notice about this construction that if at each step in the iterative matrix cross decomposition the number of index sets does not match the rank of the matrix, i.e. it is not a full rank decomposition, the choice of index sets will be crucial. This choice of $\mathcal{I}^{(k)}$, however, won't be from the sets of all possible multi-indices $\{i_{\leq k}\}$ in our iterative process, but just from $\mathcal{I}^{(k)} \otimes \{i_{\leq k-1}\}$. If this property is also enforced on the $\mathcal{J}^{(k)}$ index sets, such that the following condition is satisfied:

$$\mathcal{I}^{(k+1)} \subset \mathcal{I}^{(k)} \otimes \{i_{k+1}\} \ ; \ \mathcal{J}^{(k)} \subset \{i_k\} \otimes \mathcal{J}^{(k+1)} \tag{11}$$

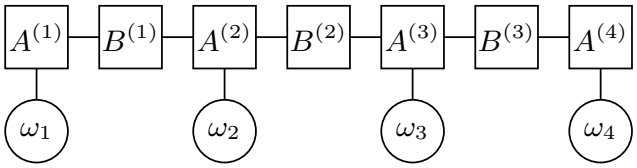

FIG. 3: Tensor diagram representing the approximated value of the integral as shown in Eq. (13) for a simple case with $N = 4$. In the figure, the 3-legged tensors labeled as $A^{(k)}$ refer to $A(\mathcal{I}^{(k-1)}, i_k, \mathcal{J}^{(k+1)})$, the 2-legged ones labeled as $B^{(k)}$ refer to $\left[A(\mathcal{I}^{(k)}, \mathcal{J}^{(k)})\right]^{-1}$ and lastly the circle tensors labeled as ω_k represent the quadrature weight vectors.

it can be proven that Eq. (10) also interpolates the entries of the tensor [7]:

$$A(\mathcal{I}^{(k-1)}, i_k, \mathcal{J}^{(k)}) = \tilde{A}(\mathcal{I}^{(k-1)}, i_k, \mathcal{J}^{(k)}) \tag{12}$$

We will refer to the index sets $\mathcal{I}^{(k)}$ ($\mathcal{J}^{(k)}$) that satisfy this condition as left (right) nested.

C. Integrating from the TT-Cross approximation

If instead of the whole tensor A we work with its TT-Cross approximation in Eq. (2), the integral can be expressed as:

$$I \approx \sum_{i_1 \dots i_N} w_1(i_1) A(i_1, \mathcal{J}^{(1)}) \left[A(\mathcal{I}^{(1)}, \mathcal{J}^{(1)}) \right]^{-1} w_2(i_2) \times \\ \times A(\mathcal{I}^{(1)}, i_2, \mathcal{J}^{(2)}) \dots w_N(i_N) A(\mathcal{I}^{(N-1)}, i_N), \tag{13}$$

which in diagrammatic terms corresponds to the network shown in Fig. 3. The calculation therefore results in just a product of vectors and matrices, comparatively small with respect to the size of A, which can be performed efficiently. Furthermore, once the approximation has been built, apart from computing integrals, we have a way of accessing in an approximate way an exponential number of points with a memory cost of $\mathcal{O}(Nn\chi^2)$, where $\chi = \max_k |\mathcal{I}^{(k)}|$ and $n = \max_k |\{i_k\}|$. The problem hence becomes finding which are the best sets $\{\mathcal{I}^{(k)}\}, \{\mathcal{J}^{(k)}\}$.

IV. ALGORITHMS FOR FINDING THE CROSSES.

In this section we give an overview of two algorithms whose objective is to find the sets $\{\mathcal{I}^{(k)}\}, \{\mathcal{J}^{(k)}\}$ that result in a good TT-Cross approximation. Starting from an initial guess of sets, both algorithms work sequentially with the matrices $A(\mathcal{I}_s^{(k-1)} i_k, i_{k+1} \mathcal{J}_l^{(k+1)})$ to find the best index sets $\mathcal{I}^{(k)}$, $\mathcal{J}^{(k)}$ for $k = 1, \dots, N-1$. For coherence purposes, we introduce the dummy index sets $\mathcal{I}^{(0)} = \emptyset$ and $\mathcal{J}^{(0)} = \emptyset$ to define these 2-site blocks in the ends of the network in the same way.

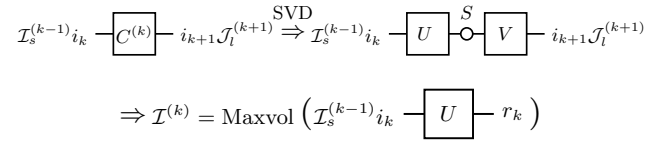

FIG. 4: Diagrammatic description of the two steps that play in an index update performed by the TTRC algorithm in a left-to-right sweep. The tensor labeled as $C^{(k)}$ refers to the matrix $A(\mathcal{I}_s^{(k-1)} i_k, i_{k+1}\mathcal{J}_l^{(k+1)})$.

A. The TTRC algorithm.

The first algorithm we present is the tensor train renormalization cross (TTRC) algorithm, first introduced in Ref. [8]. The algorithm starts off with an initial crude approximation to the sets and updates them doing sweeps from left to right and back. The update process of the index sets proceeds in two steps. After computing the matrix $A(\mathcal{I}_s^{(k-1)} i_k, i_{k+1}\mathcal{J}_l^{(k+1)})$ an SVD is performed. With this, we are able to capture the correlations between the multi-indices $i_{\leq k}$ and $i_{>k}$ and determine the TT-rank between sites k and $k+1$. At this step this bond dimension can also be truncated in a controlled way if needed with minimal error.

After this decomposition has been performed, the Maxvol algorithm is applied to the left (right) output block of the SVD in a left-to-right (right-to-left) sweep. With this, we obtain the best set of indices $\mathcal{I}^{(k)}$ ($\mathcal{J}^{(k)}$), of size r_k, out of the total $\mathcal{I}^{(k-1)} \otimes \{i_k\}$ ($\{i_{k+1}\} \otimes \mathcal{J}^{(k+1)}$) which formed the original matrix. Fig. 4 displays these two steps in diagrammatic notation for the sake of clarity. A more detailed description of the algorithm can be found in Algorithm 1 of [8].

B. The Greedy-Cross algorithm

The second algorithm we present is the Greedy-Cross (GC) algorithm, which follows a greedy approach for finding the best set of crosses $\{\mathcal{I}^{(k)}\}$, $\{\mathcal{J}^{(k)}\}$. Starting off from just a single multi-index in the index sets, the algorithm sweeps back and forth the tensor train adding a new element to $\mathcal{I}^{(k)}$ and $\mathcal{J}^{(k)}$ at each step improving the approximation. To do so, the matrix $A(\mathcal{I}_s^{(k-1)} i_k, i_{k+1}\mathcal{J}_l^{(k+1)})$ is computed as in the TTRC algorithm. We will denote this matrix by $\mathcal{A}$. On the other hand, the index sets $\mathcal{I}^{(k)}$,$\mathcal{J}^{(k)}$ give its matrix cross approximation. If we denote by S and K the positions of the elements of $\mathcal{I}^{(k)}$ and $\mathcal{J}^{(k)}$ in the ordered sets $\mathcal{I}^{(k-1)} \otimes \{i_k\}$, $\{i_{k+1}\} \otimes \mathcal{J}^{(k+1)}$ all the entries of $\mathcal{A}$ can be obtained from:

$$\mathcal{A}(l,r) \approx \tilde{\mathcal{A}}(l,r) = \mathcal{A}(l,K)\left[\mathcal{A}(S,K)\right]^{-1}\mathcal{A}(S,r) \quad (14)$$

Since we know the cross decomposition has the interpolating property, see Eq. (6), the error in the entries with $l \in S$ or $r \in K$ is zero. Consider now the multi-indices $i_{\leq k}$ and $i_{>k}$ associated with the row and column where the error $|\mathcal{A} - \tilde{\mathcal{A}}|$ is the largest. If we add these crosses to $\mathcal{I}^{(k)}$, $\mathcal{J}^{(k)}$, the respective row and column will now also be interpolated in an exact way. The approach then is greedy in the sense that it assumes that the previous crosses are good enough, and just adds the multi-indices that improve the approximation the most at the current step. More details about this algorithm, along with more sophisticated variants that focus on parallelization can be found in Ref. [7] and Ref. [1].

V. BENCHMARKING USING INTEGRALS OF ISING TYPE

To benchmark the TTRC and GC algorithms we use, as Ref. [7], the following integral of Ising type:

$$C_n = 2\int_0^1 \cdots \int_0^1 \frac{dt_2 \dots dt_n}{\left(1+\sum_{k=2}^n w_k\right)\left(1+\sum_{k=2}^n v_k\right)}$$
$$w_k := \prod_{i=2}^{k} t_i \;\; ; \;\; v_k := \prod_{i=k}^{n} t_i \quad (15)$$

The reasons for using this integral are three. First of all, it treats a function which is not separable, which means that the TT-Cross decomposition won't be trivial in terms of correlations. Secondly, the values of C_n have been computed to very high precision in Ref. [2]. And at last, it is of physical interest as it is involved in the study of 2-D spin lattices, as discussed in the same paper. We have tested the TTRC and GC algorithms against C_{64}, using a Gauss-Legendre quadrature rule with 5 points per variable, ensuring a high degree of precision with very few points. Their performance has been compared against a MC method computing the same integral as:

$$C_n \approx C_n^{MC} = \frac{1}{N_{eval}} \sum_{i=1}^{N_{eval}} f(x_1^i, \dots, x_n^i) \;\; ; \;\; x_k^i \in U[0,1] \quad (16)$$

The implementation itself of the algorithms in Python can be found in the py_ttcross package [9], which is the major outcome of this work.

Fig. 5 shows the error with respect to the number of calls to the integrated function. For the GC, the integral is computed after each site update. For the TTRC, the points shown correspond to running the algorithm until convergence for varying maximum TT-ranks. Since the cross sets are updated independently, depending on the sweep direction, they are not of equal size at each update and the integral cannot be computed at each step unlike GC, as $A(\mathcal{I}^{(k)}, \mathcal{J}^{(k)})$ cannot be inverted.

From the figure we can see how the TT-Cross based methods start off with a slow convergence, but once they pick up enough crosses, they converge very rapidly, much faster than the MC method. Values for these two methods are not shown for more function calls due to the implementation not working with high enough floating

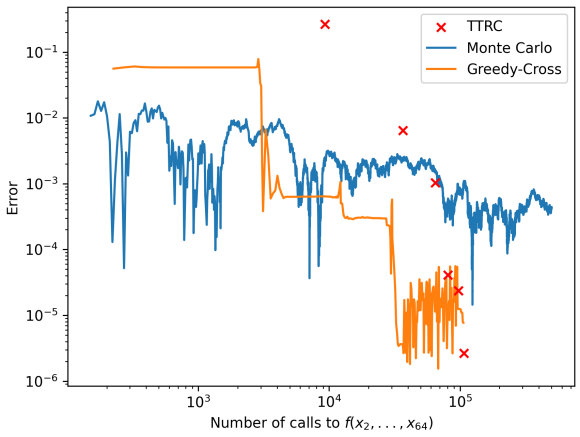

FIG. 5: Error committed by a MC method and the TTRC and GC interpolations on evaluating the integral C_{64} as a function of the number of calls to the integrated function. The GC data correspond to adding new crosses up to a TT-rank of 6. The TTRC data correspond to running the same interpolating process with a maximum TT-rank from 1 to 6.

point precision. Once the tensor train gets large enough, errors start to propagate rapidly, resulting in the algorithm reaching a plateau first, as can be seen in the GC data, and then diverging. This later behaviour is not shown in the figure. These results match with what is shown in Ref. [1], where a more in-depth analysis is performed varying the precision, showing that the TT-Cross framework is able to saturate it quickly and that by scaling it, the integral value is also able to reach many more correct decimals using a fraction of the resources used.

VI. CONCLUSIONS

In this work we have studied the TT-Cross framework as an alternative to MC techniques for highly-dimensional integration. From the results shown, it is clear that the TTRC and GC algorithms are able to produce a TT-Cross decomposition that results in an integral error that scales better with the number of function calls than the MC approach. This analysis could be extended in future work to theories with sign problems to obtain quantitative results out of reach for MC methods. Apart from this, the core part of this project has been the development of the py_ttcross Python package [9], which includes several interpolation and integration tools based on the two algorithms showcased in this paper. Although a performance oriented implementation of the GC algorithm is presented in Ref. [1], the py_ttcross package is, to our knowledge, the first package available that encompasses both interpolation and integration using multiple cross-finding algorithms. On top of that, the language choice, along with the large focus on ease of use and readability with which it has been built lower the entry barrier to these methods as much as possible. Nonetheless, taking into account the capabilities observed from the TT-Cross based algorithms and the limitations of the current implementation, a reasonable continuation of the project would be to translate the package to a more performance oriented language, such as Julia or Fortran for example, focusing on floating point precision, while keeping the code easy to understand and use.

Acknowledgments

First and foremost, I want to thank my supervisor Dr. Stefano Carignano for his support and guidance and Dr. Luca Tagliacozzo for his ideas and insights at the beginning of the project. I would also like to thank my colleague Adrià Blanco for our very fruitful conversations and my family for their support. Lastly, I want to thank Carla Gil for always being there.

[1] S. Dolgov and D. Savostyanov, "Parallel cross interpolation for high-precision calculation of high-dimensional integrals," *Computer Physics Communications*, vol. 246, p. 106869, 2020.

[2] D. H. Bailey, J. M. Borwein, and R. E. Crandall, "Integrals of the ising class," *Journal of Physics A: Mathematical and General*, vol. 39, p. 12271, sep 2006.

[3] R. Orús, "Tensor networks for complex quantum systems," *Nature Reviews Physics*, vol. 1, p. 538–550, Aug. 2019.

[4] I. Oseledets and E. Tyrtyshnikov, "Tt-cross approximation for multidimensional arrays," *Linear Algebra and its Applications*, vol. 432, no. 1, pp. 70–88, 2010.

[5] J. J. Bartholdi, "A good submatrix is hard to find," *Operations Research Letters*, vol. 1, no. 5, pp. 190–193, 1982.

[6] S. Goreinov, I. Oseledets, D. Savostyanov, E. Tyrtyshnikov, and N. Zamarashkin, "How to find a good submatrix," *Matrix Methods: Theory, Algorithms and Applications*, 04 2010.

[7] D. V. Savostyanov, "Quasioptimality of maximum-volume cross interpolation of tensors," *Linear Algebra and its Applications*, vol. 458, p. 217–244, Oct. 2014.

[8] D. Savostyanov and I. Oseledets, "Fast adaptive interpolation of multi-dimensional arrays in tensor train format," pp. 1–8, 2011.

[9] P. Torrente, "py_ttcross: A python package for tt-cross interpolation and integration." `https://github.com/pau-torrente/py_ttcross`.

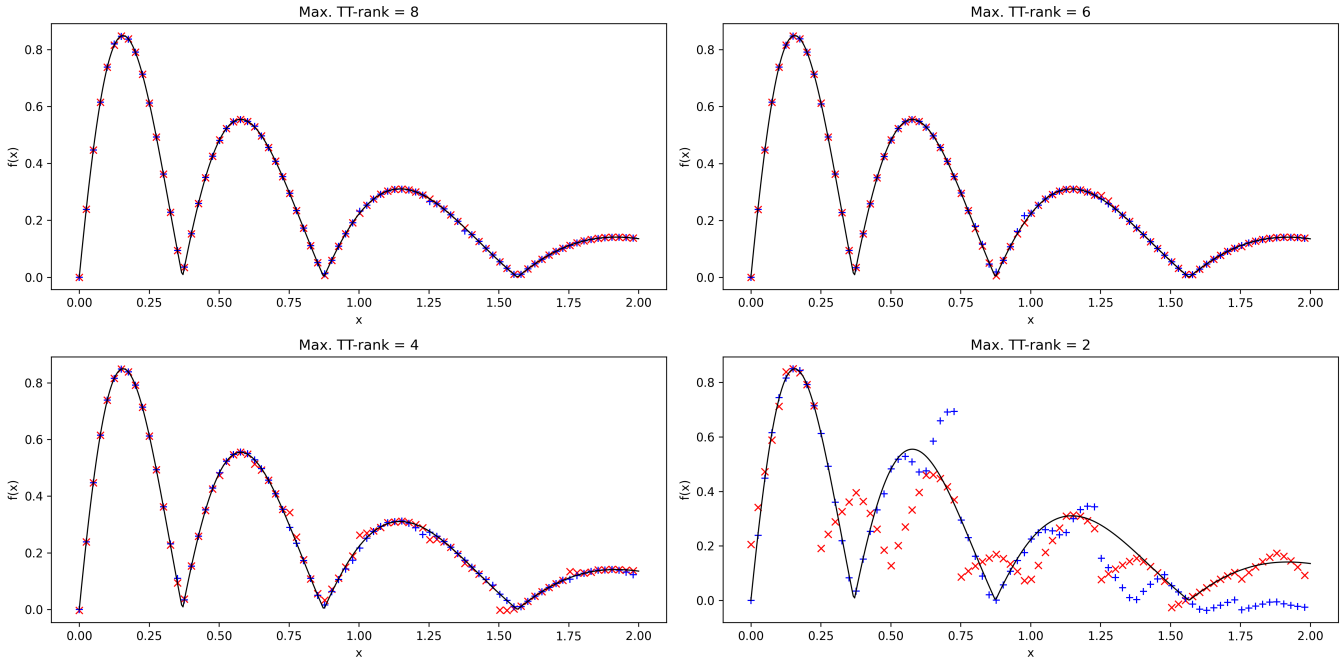

FIG. A1: TT-Cross interpolation of the function given in Eq. (A3) built by the TTRC (in blue) and the GC (in red) algorithms, for four different values of maximum TT-rank. The black curves depict the exact values of the function.

Appendix A: Interpolation of one dimensional functions

Here we briefly present the Quantics Tensor Train (QTT) interpolation of a function [8], as it is also a tool present in the py_ttcross package [9] that allows us to visualize the interpolating property. The main idea behind it is the transformation of a discretized variable $x_i = \frac{L}{2^d}\left(i + \frac{1}{2}\right)$, $i = 0, \ldots, 2^d - 1$ into a binary grid:

$$i \leftrightarrow (i_1, \ldots, i_d), \quad i = \sum_{p=1}^{d} i_p 2^{p-1}, i_p = 0, 1 \tag{A1}$$

With this, we can convert a single variate function into a d-variate one, for example, and apply the TT-Cross based interpolation algorithms showcased in this work, getting an exponentially fine discretization. Once the interpolation has been built, the function can be evaluated in the interval $[0, L]$ by converting a given point into binary in the interpolated interval and performing the contraction shown in Eq. (13) with weight vectors given by Eq. (A2).

$$\omega_p = \begin{cases} (1,0) \text{ if } i_p = 0 \\ (0,1) \text{ if } i_p = 1 \end{cases} \tag{A2}$$

As an example, we apply this procedure to the following function in the interval $[0, 2]$ with $d = 16$:

$$f(x) = |\sin(10 \log(x+1))|\, e^{-x} \tag{A3}$$

Fig. A1 shows its interpolation both from the GC and the TTRC algorithms for 4 different values of maximum TT-rank. In the top two plots, with $\max(r_k) = 6, 8$ we are able to match the behaviour of the studied function with almost no discrepancies. The case with $\max(r_k) = 4$ does start to show some deviations, and the one with $\max(r_k) = 2$ cannot follow the function's curve.

This is another clear proof, apart from the error metric of Fig. 5, that the algorithms are able to find the correct index sets, and that once the TT-Cross approximation is built with enough good crosses, it is capable of interpolating the studied function with great accuracy.