Arithmetic circuit tensor networks, multivariable function representation, and high-dimensional integration

Ruojing Peng ,[*] Johnnie Gray , and Garnet Kin-Lic Chan
Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena, California 91125, USA

(Received 1 September 2022; revised 30 January 2023; accepted 6 February 2023; published 28 February 2023)

Many computational problems can be formulated in terms of high-dimensional functions. Simple representations of such functions and resulting computations with them typically suffer from the "curse of dimensionality," an exponential cost dependence on dimension. Tensor networks provide a way to represent certain classes of high-dimensional functions with polynomial memory. This results in computations where the exponential cost is ameliorated or, in some cases, removed, if the tensor network representation can be obtained. Here, we introduce a direct mapping from the arithmetic circuit of a function to arithmetic circuit tensor networks, avoiding the need to perform any optimization or functional fit. We demonstrate the power of the circuit construction in examples of multivariable integration on the unit hypercube in up to 50 dimensions, where the complexity of integration can be understood from the circuit structure. We find very favorable cost scaling compared with quasi–Monte Carlo integration for these cases and further give an example where efficient quasi–Monte Carlo integration cannot be performed without knowledge of the underlying tensor network circuit structure.

DOI: 10.1103/PhysRevResearch.5.013156

I. INTRODUCTION

High-dimensional multivariable functions (henceforth, multivariable functions) and their integrals appear in a multitude of areas, ranging from statistical and quantum many-body physics [1–3] to applications in machine learning [4–9]. Simple representations of such functions, for example, on a product grid, require exponential storage, and subsequent manipulation of the functions, e.g., in integration by quadrature, then requires exponential cost in dimension. Many techniques have been introduced to bypass this exponential cost. For example, high-dimensional integration is often carried out by Monte Carlo or quasi–Monte Carlo methods, which sample the function at a set of random or preselected points [10–13], thereby exchanging the exponential dependence on dimension for weaker guarantees on error.

In the many-body physics community, tensor networks (TNs) such as matrix product states, projected entangled pair states, and the multiscale entanglement renormalization ansatz have long been used to represent multivariable physical quantities, such as quantum states [14–18] or Boltzmann densities [19–21]. Similar techniques (although mainly for more restricted classes of tensor networks, such as canonical decomposition (CANDECOMP) or parallel factor analysis (PARAFAC) [22–26], hierarchical Tucker decomposition [26–28], and tensor trains [28,29]) have appeared in the applied mathematics community as well, and have been used for high-dimensional function computation and approximation. In both cases, the idea is to represent an (often discretized) high-dimensional function as a connected network of low-dimensional tensors. This can reveal a nontrivial low-rank structure in the function, thereby achieving a great reduction in memory. Because TNs further come with a natural notion of approximation (from the compression of pairs of tensors via, e.g., the singular value decomposition), the use of such approximations can ameliorate, and in some cases remove, the exponential cost with respect to dimension, when computing with the TN.

For multivariable function computation with TNs, we must first obtain a TN representation of the function. The manner in which such representations are obtained usually involves problem-specific numerical computation. For instance, if the target function can be efficiently evaluated, then determination of the tensor parameters can be formulated as an optimization problem [24,25,30–35]. Another common scenario is when the function is implicitly defined from a minimization, in which case the tensor representation can be optimized by the variational principle [36–38]. When the function satisfies a differential equation with a known initial condition that is easily expressed as a TN, the TN representation can be propagated [39–44]. In all these cases, therefore, the determination of the tensor network representation of the function involves an associated, potentially large, computational cost. In addition, the tensor networks are generally chosen with a fixed network structure ahead of time (for example, a matrix product state or a tensor train) in order to make the determination of the representation feasible, even though such a structure may not be the most compact [38,45,46].

Here, we introduce an alternative way to construct the TN representation of a function from its arithmetic circuit. As both classical arithmetic circuits and quantum arithmetic circuits can be viewed as TNs, it is immediately clear that multivariable functions can be represented as TNs through these circuits. However, such circuits carry various disadvantages;

[*]Corresponding author: rppeng@caltech.edu

for example, classical logic gates lead to extremely sparse tensors where the number of indices is proportional to the number of bits of precision [47,48], while quantum arithmetic circuits are constrained to unitary tensors and may be suboptimal for classical calculations [49,50]. Consequently, we introduce a tensor network circuit representation that takes advantage of both the ability to store floating point numbers as entries in the tensors and the lack of unitary constraints. This leads to a concise construction that avoids the auxiliary computation involved in obtaining the TN representation itself.

A by-product of the arithmetic circuit TN form is that the structure of the tensor network is dictated by the circuit, rather than specified beforehand. Consequently, the TN structure that arises can have a general connectivity, and can in principle look quite different from those typically encountered in quantum many-body physics or applied mathematics settings. Recent developments in the exact and approximate contraction of tensor networks with more unstructured geometries are discussed, for example, in Refs. [51–55].

Using the arithmetic circuit TN representation, we carry out high-dimensional quadrature over the unit hypercube (in up to 50 dimensions) for multivariable polynomials and multivariable Gaussians. The use of tensor networks in conjunction with high-dimensional integration or summation is hardly new; it is one of the main applications of TNs (see, for example, Refs. [33,34,56,57]). However, the availability of the circuit structure of the function leads to new insights into this problem. For example, for the multivariable polynomials, we find exact compressibility of the TN circuit in certain limits of the polynomial parameters. This can then be decoded into an exact integration rule (see Ref. [58] for a related result). Furthermore, away from the exact point, we can relate the difficulty of approximation to the degree of nonlinearity (number of copy operations) in the circuit. In terms of practical performance, because Gaussian quadrature weights can be used in each dimension of the TN quadrature, we find that TN integration converges in accuracy orders of magnitudes more quickly than quasi–Monte Carlo (quasi-MC) integration for instances of the multivariable polynomial and Gaussian integrals. Finally, we finish with an artificial but instructive case where we construct a function (based on the multiscale entanglement renormalization ansatz [59,60]) that can be integrated efficiently when using knowledge of its internal TN circuit structure, but for which function evaluation is hard, thus making quasi-MC hard if the function is treated as a black box.

II. TENSOR NETWORK ARITHMETIC CIRCUITS

A. Function tensor representation and circuit composition of functions

We first introduce a tensor representation of individual single-variable and multivariable functions and show how to use the representation to compose complicated functions from simpler ones. We start with a single-variable function $f(x)$. Assuming for the time being that x is a continuous variable, we introduce the continuous function tensor (function matrix) representation F, where the function values are stored in the elements $F_{x\alpha}$, specifically,

$$F_{x0} = 1,$$
$$F_{x1} = f(x). \tag{1}$$

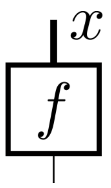


FIG. 1. Tensor representation F of the scalar function $f(x)$. The "control" index is represented by the thin leg on the bottom.

We refer to x as the variable and α as the control index or leg (the dimension of the control index is 2). We consider here scalar-valued functions, but vector-valued functions can be similarly defined. In numerical applications, x will typically be discretized, e.g., on a grid with G points. The control index is used to perform arithmetic and other gate operations on the functions.

It will be convenient to use the standard graphical notation of tensor networks. We show a diagram of F in Fig. 1. We use the convention that labeled lines represent indexed elements and unlabeled lines between two tensors are summed over for discrete indices and integrated over for continuous indices.

The control index can be thought of as a "qubit" index in the tensor product Hilbert space of functions, analogous to the qubit representation used in quantum mechanics; from Eq. (1), we see that $|1\rangle$ is associated with a basis function $f(x)$, and $|0\rangle$ is replaced by the scalar 1. Then given a set of functions $\{f_i(x_i)\}_{i=0\cdots N-1}$, we represent a monomial of functions as

$$F^{[1]}_{x_0\alpha_0}F^{[2]}_{x_1\alpha_1}F^{[N-1]}_{x_{N-1}\alpha_{N-1}} = f_0(x_0)^{\alpha_0} f_1(x_1)^{\alpha_1}\cdots f_{N-1}(x_{N-1})^{\alpha_{N-1}}. \tag{2}$$

A multivariable function $c(x_0, x_1, \ldots, x_{N-1})$ in the product Hilbert space $\prod_i\{1, f(x_i)\}$ takes the form

$$c(x_0, \ldots, x_{N-1}) = \sum_{\{\alpha_i\}}\prod_i C_{\alpha_0,\ldots,\alpha_{N-1}} F^{[i]}_{x_i\alpha_i}. \tag{3}$$

Using the qubit analogy, $C_{\alpha_0,\ldots,\alpha_{N-1}}$ may be regarded as a wavefunction amplitude in the computational basis. Note that the above constructs the Hilbert space using single-variable functions, but we could also use more complex building blocks, e.g., a two-variable function tensor, $F_{xy\alpha} \leftrightarrow f(x, y)$. We also note that the use of a product Hilbert space to represent multivariable functions has been considered in other contexts, for instance, for length scale separation in the solution of partial differential equations, or for efficient function parameter storage. For relevant discussions, see, e.g., Refs. [35,43,44].

We use a tensor network to build a circuit representation of the function $c(x_0, \ldots, x_{N-1})$ using contractions of the function tensors either with themselves or with other fixed tensors. The simplest example corresponds to performing classical arithmetic and logic to combine single-variable functions, using control tensors. The addition tensor $(+)$ is a three-index control tensor, with elements

$$(+)_{\alpha\beta\gamma} = \begin{cases} 1 & \alpha + \beta = \gamma \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Note that unlike the usual binary arithmetic, the addition is not modulo 2. The multiplication tensor $(\times)$ is also a three-index

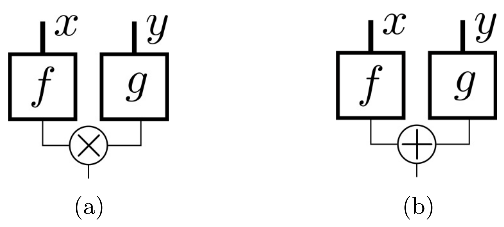


FIG. 2. Arithmetic circuit tensor network representation of (a) $f(x)g(y)$ and (b) $f(x)+g(y)$.

control tensor,

$$(\times)_{\alpha\beta\gamma} = \begin{cases} 1 & \alpha=\beta=\gamma \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

With such definitions we can perform classical arithmetic on functions. Contraction of two function tensors, F and G, with the multiplication or addition tensors yields the higher-dimensional objects (shown as a tensor network in Fig. 2) corresponding to functions of two variables, i.e.,

$$F_{x\alpha}G_{y\beta}(\times)_{\alpha\beta\gamma} \leftrightarrow f(x)g(y),$$
$$F_{x\alpha}G_{y\beta}(+)_{\alpha\beta\gamma} \leftrightarrow f(x)+g(y),$$

where we have assumed summation over repeated indices.

The above are simple examples of arithmetic circuit tensor networks. Within these arithmetic circuits, the 0 value of the control index is only needed to perform addition; if we know that $f(x)$ only enters the circuit via multiplication, we can always choose to fix the control index $\alpha = 1$ of the corresponding tensor F, and thus omit the control leg entirely, i.e., $F_x = F_{x\alpha}\delta_{\alpha,1}$. Similarly, scalars have no variable dependence and can thus be specified without their variable leg.

A second circuit structure involves contraction between the variable legs of the function tensors, which corresponds to integrating a common variable between two functions. For example, given $F_{x\alpha} \leftrightarrow f(x)$ and $G_{k,x} \leftrightarrow g(k,x)$ (we have dropped the control index on G since we are only using it for multiplication), then

$$F_{x\alpha}G_{kx} \leftrightarrow \int dx f(x)g(k,x), \tag{6}$$

where we have assumed integration over the repeated continuous index on the left.

Multivariable functions built from the above arithmetic operations will be multilinear in the underlying functions. To build nonlinearity, we use multiple functions of the same variable. To do so, we define a tensor COPY on the continuous variable legs, with entries

$$(\mathsf{COPY})_{xyz} = \begin{cases} 1 & x=y=z \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Then, using the COPY tensor we can multiply or add two functions of the same variable, as shown in Fig. 3, corresponding to the contractions

$$(\mathsf{COPY})_{xyz}F_{y\alpha}G_{z\beta}(\times)_{\alpha\beta\gamma} \leftrightarrow f(x)g(x),$$
$$(\mathsf{COPY})_{xyz}F_{y\alpha}G_{z\beta}(+)_{\alpha\beta\gamma} \leftrightarrow f(x)+g(x).$$

Note that the contraction of the COPY tensor is an example of a third type of circuit operation, namely, the contraction over the variable legs of functions with additional variable leg tensors; further examples of this kind are discussed in Sec. II B.

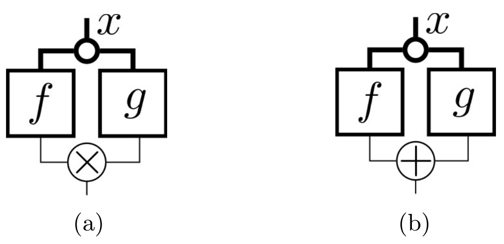


FIG. 3. Arithmetic circuit tensor network representation of (a) $f(x)g(x)$ and (b) $f(x)+g(x)$. The circle represents the COPY tensor.

Finally, we briefly mention that the control legs above are contracted with tensors that implement classical arithmetic or logic; contraction generates a single function output. However, one can also apply more general binary operations, such as quantum logic gates where a single set of control inputs may map to multiple control outputs. For example, we can create a circuit using a quantum CNOT gate (see Fig. 4), giving

$$F_{x\alpha}G_{y\beta}(\text{CNOT})_{\alpha\beta,\gamma\delta}(+)_{\gamma\delta\epsilon} \leftrightarrow f(x)+f(x)g(y). \tag{8}$$

B. Variable circuits, transformations of variables, and other representations

The COPY operation defined above generalizes to other transformations on the variable legs of the function tensors. For instance, Eq. (6) represents a transformation of variable $f(x) \to \tilde{f}(k)$ that can be written as an integral with a kernel $g(x,k)$. As a concrete example, consider the convolution of N-periodic sequences written as a TN contraction with kernel Z

$$(f_N * g_N)[n] = \sum_{m=0}^{N} f_N[m]g_N[n-m] \leftrightarrow F_m G_l Z_{mln}, \tag{9}$$

where

$$Z_{mln} = \begin{cases} 1 & l=n-m \\ 0 & \text{otherwise} \end{cases}$$

and the subtraction of indices is modulo N. Note that, as previously explained, we have dropped the control index ($F_m = F_{m\alpha}\delta_{\alpha,1}$) since the functions f_N and g_N enter only through multiplication. Similarly, discrete Fourier transforms (DFTs) can be written as

$$\mathcal{F}^{-1}(\mathcal{F}(f_N)\mathcal{F}(g_N)) \leftrightarrow (\mathcal{F}^{-1})_{n,k}\mathcal{F}_{k,m}F_m\mathcal{F}_{k,l}G_l, \tag{10}$$

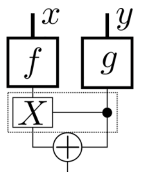
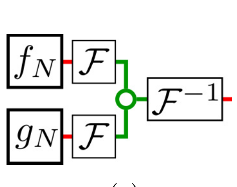


FIG. 4. Arithmetic circuit tensor network representation of Eq. (8). The CNOT gate is represented by the dashed box.



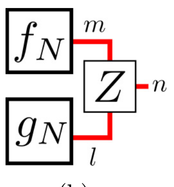


FIG. 5. Representation of the convolution theorem: (a) $\mathcal{F}^{-1}(\mathcal{F}(f_N)\mathcal{F}(g_N))$ and (b) $(f_N * g_N)[n]$. The red legs represent position space variables l, m, n, and the green legs represent momentum space variable k. The green circle represents the COPY tensor.

where

$$\mathcal{F}_{k,n} = e^{-2\pi ikn/N}, \quad (\mathcal{F}^{-1})_{n,k} = \frac{1}{N}e^{2\pi ikn/N}$$

are the kernels for the DFT and inverse DFT. The convolution theorem $(f_N * g_N)[n] = \mathcal{F}^{-1}(\mathcal{F}(f_N)\mathcal{F}(g_N))[n]$ can then be expressed as the diagram shown in Fig. 5.

In contrast to the representation introduced above, which stores function values in the tensor elements, we also want to mention an alternative "classical" representation of functions where function values are stored in the tensor indices. In this case, the function $y(x)$ can be thought of as represented by a function tensor with two continuous indices,

$$\underline{F}_{y'x} = 1 \quad \text{if } y' = y(x), \tag{11}$$

where we have used the underline to distinguish this classical representation from our previous representation. One can, e.g., perform arithmetic in this classical representation, using the addition and multiplication tensors

$$\underline{(+)}_{xyz} = 1 \quad \text{if } z = x + y,$$
$$\underline{(\times)}_{xyz} = 1 \quad \text{if } z = xy. \tag{12}$$

In principle, one can represent all operators and all functions from circuits built up this way. However, in a discrete computation, the continuous variables must be discretized in some manner. For instance, classical finite-precision binary arithmetic discretizes each variable as a binary string (e.g., of size N, so that each variable is represented by N tensor legs of dimension 2), in which case the $\underline{(+)}$ and $\underline{(\times)}$ tensors decompose into subnetworks of classical logic gates, e.g., $\underline{(\text{XOR})}$ and $\underline{(\text{AND})}$. The precision of the representation is then limited by the length of the binary string, i.e., the number of tensor legs, for each variable.

C. Arithmetic circuit tensor networks and integration

The composition of the above elements clearly allows us to construct a general multivariable function by operations on function tensors. This yields the arithmetic circuit tensor network representation of the multivariable function. A simple example for the function $\prod_{i=1}^{3}(f_i(x) + g_i(y))$ is shown in Fig. 6(a).

Given the TN representation, it is trivial to define integration over the input variables. Assuming a product quadrature for each variable $x_0, \ldots, x_{N-1}$, then each continuous variable is discretized $x_i[p] \leftrightarrow x_i$, where $[p]$ denotes the pth grid point.

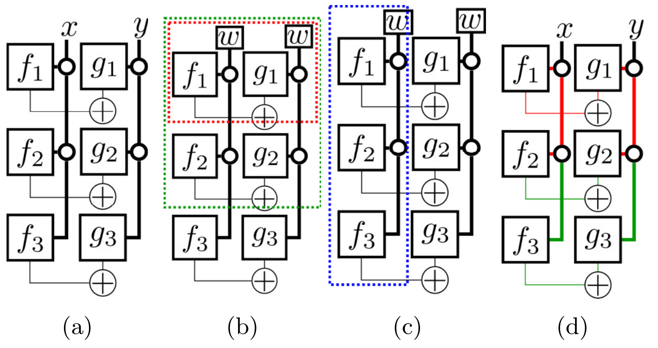


FIG. 6. Representation of the function $\prod_{i=1}^{3}(f_i(x) + g_i(y))$. (a) TN representation of the function. As discussed previously, since each factor $f_i(x) + g_i(y)$ enters the function through multiplication only, we omit the open control leg on the (+) tensors. (b) Integration of the function with one contraction scheme: The tensors in the red box are contracted first, then contracted with the tensors in the green box, and then contracted with the remaining tensors. (c) Integration of the function with another contraction scheme: The tensors in the blue box are contracted first and then contracted with the remaining tensors. (d) Representation of the function showing loops (red and green lines).

One can also introduce quadrature weights $w[p]$, giving

$$F_\alpha^\int = \sum_p w_p F_{p\alpha} \leftrightarrow \int dx f(x) \tag{13}$$

as shown in Figs. 6(b) and 6(c).

D. Contracting the arithmetic circuit

To obtain the value of the function from the arithmetic circuit, one needs to contract the tensors. There are many techniques for contracting tensor networks. Here we will view exact and approximate tensor network contraction as mainly black box algorithms; thus we do not discuss the details and only give a brief idea of the fundamentals. Further details of techniques for exact contraction are described in Refs. [51,61–69]. Additional information on approximate contraction can be found in Refs. [14–18,38,39,42,52–55,68,70,71].

Although a typical arithmetic circuit has a flow from input values to output values, the fact that the arithmetic circuit TN encodes all output values for all inputs simultaneously, removes the directionality of the circuit. In particular, the contractions in the circuit can be evaluated in any order [see Figs. 6(b) and 6(c)], interchanging the order of summations and products. This can lead to drastic changes in the complexity of evaluation. Heuristic techniques exist to search for and find good orders for exact contraction which have been applied to tensor network and quantum circuit tensor network contraction problems (see Refs. [51,72–78]).

Tensor networks without loops (i.e., trees) can be contracted exactly with a cost linear in the number of tensors. For arbitrary connectivity, e.g., with loops, the exact tensor network contraction scales exponentially with the number of tensors. However, tensor networks can also be contracted approximately with an approximation error. Such approximate contraction is widely used in tensor network applications in the simulation of quantum systems, and is closely related to low-rank matrix factorization [79–81]. During the contraction of the tensor network, tensors will be generated which share an increased number of legs with neighboring tensors [see Figs. 6(b) and 6(c)]; note the combined dimension of the legs D. In approximate contraction, we use projectors of dimension $D \times \chi$ to project the shared leg of dimension D down to a shared leg of dimension χ, thus controlling the cost of further operations. In practice, these isometric matrices are usually determined from a singular value decomposition (SVD).

A simple example of a compression algorithm is the "boundary" compression algorithm for a regular two-dimensional (2D) tensor network. Here, rows of tensors (so-called matrix product states and matrix product operators) are contracted together, to form matrix product states with shared bonds; then a series of SVDs are applied to reduce the bonds to dimension χ (see Refs. [14–18,38,39,42,55,68,71]). We will deploy the boundary contraction algorithm as the approximation contraction algorithm below. However, just as for exact contraction, the choice of order of when to contract and compress can greatly affect cost and accuracy, and similar to exact contraction, there are heuristics to choose an optimized order of contraction and compression. In this paper, we do not encounter sufficiently complicated network structures to use these more sophisticated strategies, but the interested reader is referred to Ref. [55].

Approximate contraction is critical for applications of arithmetic tensor networks, because it allows subclasses of arithmetic tensor networks to be executed for less than the brute-force exponential cost in the number of tensors. This is useful in defining classes of computational problems where the curse of dimensionality is circumvented, as we now examine in our application to multidimensional integration below.

III. APPLICATIONS TO INTEGRATION

We now apply the arithmetic tensor network formalism to the problem of multivariable integration. We start with some basic intuition about complexity and then proceed to progressively more complicated examples of multivariable polynomial and multidimensional Gaussian integration in the hypercube, illustrating the power of the method versus another high-dimensional technique, namely, quasi–Monte Carlo integration. We finish with a specific circuit function with a complexity-theoretic guarantee of hardness with respect to sampling its values (and thus integration by direct application of quasi–Monte Carlo methods) but which can be efficiently integrated if one uses its tensor network structure.

Intuition regarding complexity

It is well known that multivariable functions admitting a separation of variables are easy to integrate over a separable range. The simplest examples are

$$\int_{\Omega^2} dxdyf(x)g(y) = \left(\int_\Omega dxf(x)\right)\left(\int_\Omega dyg(y)\right), \tag{14}$$

$$\int_{\Omega^2} dxdy(f(x)+g(y)) = \Omega\int_\Omega dxf(x) + \Omega\int_\Omega dyg(y), \tag{15}$$

where Ω is the integration range in each variable. In TN language, the separability in the above equations corresponds to the tree structure in the TN diagrams shown in Fig. 2. As discussed, such loop-free tensor networks are easy to contract. In the case of integration, one integrates over the variable legs [numerically, one sums over the discrete variable index with grid weights as in Eq. (13)], and then one repeatedly contracts child tensors into their parents, never creating any shared legs. Note that loop-free structures constitute a larger class of functions than separable functions. For example, assuming all operations are addition and multiplication operations, then the function

$$[(f_1(x_1)f_2(x_2) + f_3(x_3))f_4(x_4) + f_5(x_5)]f_6(x_6)\cdots, \tag{16}$$

where the nested parentheses reflect a binary tree structure, is easily integrated.

In contrast, the multivariable integration

$$\int_{\Omega^2} dxdy(f_1(x)+g_1(y))(f_2(x)+g_2(y))(f_3(x)+g_3(y)), \tag{17}$$

illustrated in Fig. 6(d), does not generate a loop-free arithmetic tensor network. In the corresponding TN diagram the COPY tensor for each variable results in the red and green loops, arising from a nonlinear dependence of the function on a variable. The contraction of tensors that are part of two loops with their neighbors can lead to tensors with more legs or larger size. Similarly, the use of quantum gates can result in loops. Thus in our arithmetic TN circuits the use of COPY tensors and quantum gates makes the resulting tensor networks increasingly hard to contract and the resulting functions harder to integrate.

IV. MULTIVARIABLE POLYNOMIAL OF FUNCTIONS

As an instructive example, consider the integral

$$Z = \int_{\Omega^N} dx_1\cdots dx_N f(x_1,\dots x_N) \tag{18}$$

with f being a polynomial of the form

$$f(x_1,\dots,x_N) = \prod_{i=1}^{k} p_i(x_1,\dots,x_N) \tag{19}$$

with

$$p_i(x_1,\dots,x_N) = \sum_{j=1}^{N} q_{ji}(x_j) \tag{20}$$

and where $q_{ji}(x_j)$ are single-variable functions. This functional form was considered in Ref. [82], which showed that integration over the domain $[0,1]^N$ is NP-hard for arbitrary functions q_{ji}. We first construct an arithmetic circuit TN representation and then proceed to investigate the complexity of integration for different choices of the functions $q_{ji}(x_j)$.

A. Arithmetic tensor network representation

Since the total function is explicitly a product of factors, it is natural to construct a representation for each factor and then multiply them together. Consider a single factor $p_i(x_1,\dots,x_N)$. This is a sum of terms, and we can use a circuit [shown in Fig. 7(a)] to implement Eq. (20), where the

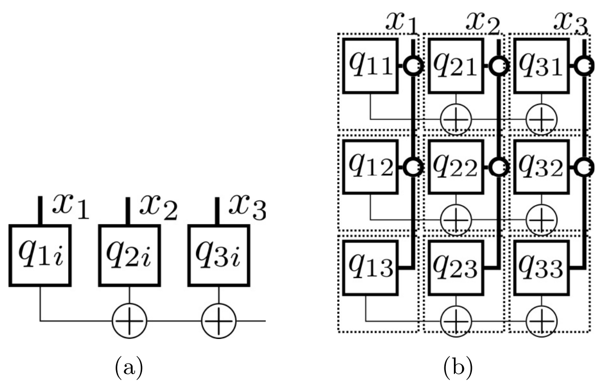


FIG. 7. Representation of (a) $p_i(x_1,\ldots,x_N)$ and (b) $f(x_1,\ldots,x_N)$ for $N=3$. As discussed in Sec. II, we have omitted the open control leg for each $p_i(x_1,\ldots,x_N)$ since they enter $f(x_1,\ldots,x_N)$ only through multiplication.

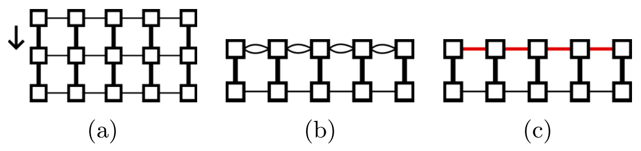


FIG. 8. Boundary row compression of a PEPS with three rows and five columns. (a) shows the PEPS, where each tensor $T_{i,j}$ (i and j index rows and columns, respectively) corresponds to a dashed box in Fig. 7(b). (b) For each column j, contract $T_{1,j}$ with $T_{2,j}$. (c) For each pair of tensors $T_{2,j}$, $T_{2,j+1}$ connected by multibonds, compress the multibonds to a maximum bond dimension χ (red).

horizontal bonds of dimension 2 are a manifestation of the low-rank structure of $p(x_1,\ldots,x_N)$. The arithmetic circuit for the full tensor network is shown in Fig. 7(b), where the TN for each factor $p_i(x_1,\ldots,x_N)$ becomes a row (with index i) and the set of q_{ji} connected by COPY tensors for a given variable x_j becomes a column (indexed by j). The COPY tensors make the entire network loopy; as each variable is copied k times, k is a measure of the loopiness of the network. If we first contract the tensors in each dashed box, the resulting circuit has a regular 2D structure, known as a projected entangled pair state (PEPS) [14] structure.

B. Exact compressibility and an identity

We first consider the case where each factor is identical, i.e., $p_i(x_1,\ldots,x_N)=p(x_1,\ldots,x_N)$. Then the total function takes the form

$$f(x_1,\ldots,x_N)=(q_1(x_1)+\cdots+q_N(x_N))^k. \tag{21}$$

To perform the integration, we first discretize each variable x_i on a grid of G points and allow each single-variable function on the grid to take random values between -1 and 1 (thus each q_i represents a discretized version of a function that is oscillating between -1 and 1). We use an equally weighted quadrature.

The contraction of the 2D tensor network corresponds to the contraction of the PEPS; thus we use an approximate tensor network contraction strategy commonly used in PEPS, compressing to a finite bond dimension χ. Within the PEPS structure, the horizontal bonds have dimension 2, while the vertical bonds associated with the grid points have dimension G. For $G\gg 2$, the cost of contracting along the vertical dimension is much cheaper than contracting along the horizontal dimension. Thus we perform the contraction by a boundary row contraction method, contracting rows into rows (see Fig. 8). To compress-contract the whole tensor network, one needs $O(Nk)$ tensor operations (contractions and SVDs) on adjacent tensor pairs. The contraction between a $\chi\times\chi\times G$ tensor and a $2\times 2\times G\times G$ tensor costs $O(\chi^2G^2)$ floating point operations (FLOPs). The SVD of a $2\chi\times 2\chi\times G$ tensor costs $O(\chi^3G^2)$ FLOPs. Hence the FLOP count scales as $O(Nk\chi^3G^2)$ from the leading SVD part.

After a row is contracted into a row, the dimension of the tensors along the boundary increases; the maximum bond dimension of the boundary tensors after k rows is 2^k. However, when performing compression, we immediately find that the boundary tensors are *exactly compressible* to $\chi<2^k$ after each row contraction. In fact, the entire tensor network can be exactly contracted with bond dimension $\chi=k$, as illustrated in Fig. 9(a), which shows the error in the computed integral (due to compressing to finite bond dimension χ); we see that the error of contraction drops to 0 for $\chi=k$. Overall this means that the integral can then be computed exactly with cost linear in the number of variables N and polynomial in the function nonlinearity k.

The ability to exactly contract the network with small χ implies the existence of an exact algebraic identity. Each compression corresponds to the insertion of isometries or projectors into the 2D TN. For each row $i=2,\ldots,k$ and column $j=1,\ldots,N$, we insert left and right projectors $P_L[i,j]$ [of dimension $i\times 2\times(i+1)$] and $P_R[i,j]$ [of dimension $(i+1)\times 2\times i$] between rows $i-1$ and i and between columns $j-1$ and j where

$$(P_L[i,j])_{ab,c}=\begin{cases}1 & a=c,\ b=0\\ 1 & a=j-1,\ c=j,\ b=1\\ 0 & \text{otherwise},\end{cases} \tag{22}$$

$$(P_R[i,j])_{c,ab}=\begin{cases}1 & a+b=c\\ 0 & \text{otherwise}.\end{cases} \tag{23}$$

This is shown step by step in Figs. 9(b)–9(e). For example, in Fig. 9(b), we insert a pair of projectors $P_L[2,1]$, $P_R[2,1]$ where $P_L[2,1]_{ab,c}$ takes two input vectors (the a,b indices) $[1,q_1]$, $[1,q_1]$ and maps them to an output vector (the c index) $[1,q_1,q_1^2]$; $P_R[2,1]_{c,ab}$ takes the input vector (c index) $[1,q_1,q_1^2]$ and maps it to two output vectors (a,b indices) $[1,q_1]$, $[1,q_1]$. Similarly, in Fig. 9(c), we insert the pair of projectors $P_L[2,2]$, $P_R[2,2]$, where $P_L[2,2]$ takes two input vectors $[1,q_1+q_2]$, $[1,q_1+q_2]$ and maps them to an output vector $[1,q_1+q_2,(q_1+q_2)^2]$; $P_R[2,2]$ takes the input vector $[1,q_1+q_2,(q_1+q_2)^2]$ and maps it to two output vectors $[1,q_1+q_2]$, $[1,q_1+q_2]$. In general,

$$P_L[i,j]:\begin{bmatrix}1\\ s_j\\ \cdots\\ s_j^{i-1}\end{bmatrix}\otimes\begin{bmatrix}1\\ s_j\end{bmatrix}\rightarrow\begin{bmatrix}1\\ s_j\\ \cdots\\ s_j^{i}\end{bmatrix}, \tag{24}$$

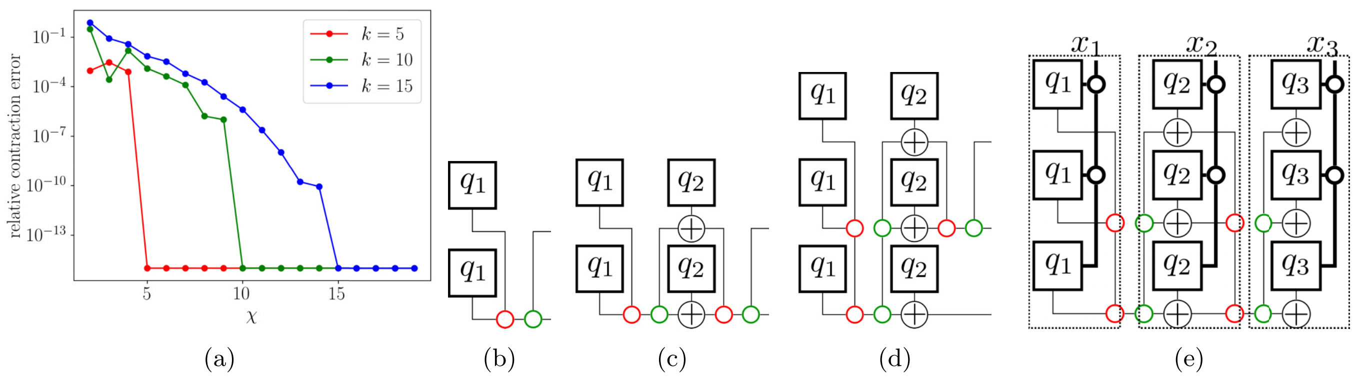


FIG. 9. (a) Accuracy of TN integration (relative contraction error) for the integral of the multivariable function polynomial in Eq. (21). This demonstrates exact compressibility via SVD for the 2D arithmetic circuit TN, with number of variables $N = 20$ and number of points per variable $G = 10$. The exact compressibility is equivalent to inserting projectors $P_L[i, j]$ (red) and $P_R[i, j]$ (green) as in diagrams (b)–(e). (b) inserts projector $P_L[2, 1]$ and $P_R[2, 1]$. (c) inserts $P_L[2, 2]$ and $P_R[2, 2]$. (d) inserts projectors $P_L[3, 1]$ and $P_R[3, 1]$. (e) Full TN for $N = 3$, $k = 3$ with projectors inserted. The result from each grouped column can be related to a recursive computation of the integral. See Sec. IV B for details.

$$P_R[i,j]:\begin{bmatrix}1\\ s_j\\ \cdots\\ s_j^i\end{bmatrix}\to\begin{bmatrix}1\\ s_j\\ \cdots\\ s_j^{i-1}\end{bmatrix}\otimes\begin{bmatrix}1\\ s_j\end{bmatrix}, \tag{25}$$

where $s_j = \sum_{j'=1}^{j} q_{j'}$ is the sum of single-variable functions up to the jth term. Thus we see that the role of the left projector is to retain only the nonredundant polynomials of the single-variable functions q_j, and the right projector redistributes nonredundant powers into a direct product.

The tensor network exact compression then corresponds to computing the integral through a recursive formula (see Ref. [58] for a related constructive approach). For example, we can recursively define a set of integrals over a subset of variables

$$I_1^{k_1} = \int dx_1 q_1(x_1)^{k_1}, \tag{26}$$

$$\begin{aligned}I_2^{k_2} &= \int dx_1 dx_2 (q_1(x_1) + q_2(x_2))^{k_2}\\ &= \int dx_1 dx_2 \sum_{k_1=0}^{k_2}\binom{k_2}{k_1} q_1(x_1)^{k_1} q_2(x_2)^{k_2-k_1}\\ &= \sum_{k_1=0}^{k_2}\binom{k_2}{k_1} I_1^{k_1}\int dx_2 q_2(x_2)^{k_2-k_1},\end{aligned} \tag{27}$$

$$\begin{aligned}I_3^{k_3} &= \int dx_1 dx_2 dx_3 (q_1(x_1) + q_2(x_2) + q_3(x_3))^{k_3}\\ &= \int dx_1 dx_2 dx_3 \Bigg[\sum_{k_2=0}^{k_3}\binom{k_3}{k_2}\\ &\quad \times (q_1(x_1)+q_2(x_2))^{k_2} q_3(x_3)^{k_3-k_2}\Bigg]\\ &= \sum_{k_2=0}^{k_3} I_2^{k_2}\int dx_3 q_3(x_3)^{k_3-k_2}\end{aligned} \tag{28}$$

and so on up to

$$\begin{aligned}I_N^k &= \int dx_1\cdots dx_N (q_1(x_1)+\cdots+q_N(x_N))^k\\ &= \int dx_1\cdots dx_N\Bigg[\sum_{k_{N-1}=0}^{k}\binom{k}{k_{N-1}}\\ &\quad\times (q_1(x_1)+\cdots+q_{N-1}(x_{N-1}))^{k_{N-1}} q_N(x_N)^{k-k_{N-1}}\Bigg]\\ &= \sum_{k_{N-1}=0}^{k}\binom{k}{k_{N-1}} I_{N-1}^{k_{N-1}}\int dx_N q_N(x_N)^{k-k_{N-1}},\end{aligned} \tag{29}$$

where each iterative step involves integration of single-variable integrals. The quantities $I_j^{k_j}$ can be related to the TN with the projectors inserted in each layer. Figure 9(e) shows a simple example for $N = 3$, $k = 3$. From left to right, the open legs of the first two dashed boxes correspond to $I_1^{k_1}$ and $I_2^{k_2}$, respectively, for $k_1, k_2 = 0,\ldots,3$, and the full contraction of the TN corresponds to I_3^k for $k = 3$.

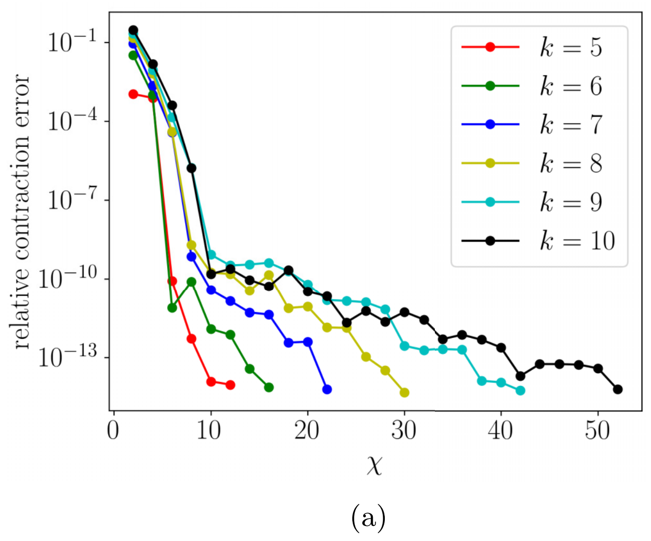
C. Perturbations away from exact compressibility

In the above, the compression of the circuit tensor network for the multivariable polynomial function allows us to identify an efficiently integrable case. However, when inserting more general polynomials in Eq. (19), we cannot expect the tensor network to be exactly compressible since we know the general case is NP-hard. However, we might expect that functions that are close to the efficiently integrable case remain efficiently integrable up to some accuracy.

We thus now consider polynomial functions in Eq. (19) that are obtained by perturbing away from the exactly compressible case. We do this by defining

$$q_{ji}(x_j) = q_{j1}(x_j) + \delta\cdot r_{ji}(x_j), \tag{30}$$

where the single-variable functions $q_{j1}(x_j)$, $r_{ji}(x_j)$ are random length-G vectors with values in $[-1, 1]$ (recall that G is the number of grid points). For $i = 1$, we set $\delta = 0$ for



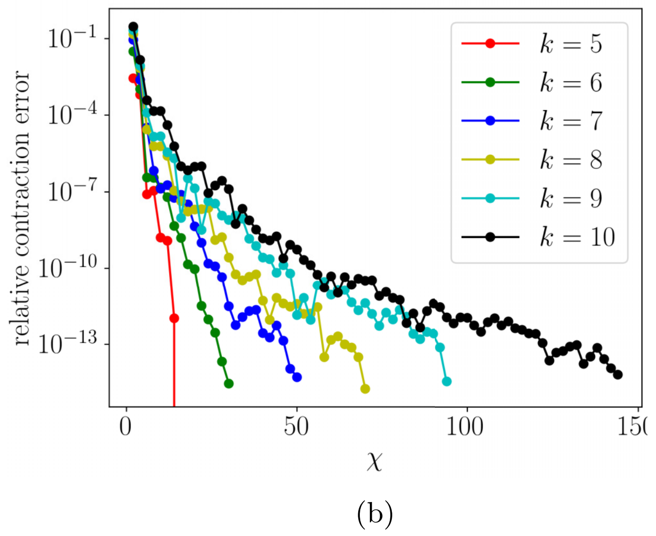


FIG. 10. Accuracy of TN integration for the integral of a multivariable function polynomial perturbed away from the exactly compressible point as a function of contraction bond dimension χ: (a) $\delta = 0.01$ and (b) $\delta = 0.1$. Magnitude of perturbation δ, number of variables $N = 20$, number of points per variable $G = 10$. k is a measure of the nonlinearity expressed in the TN. Note that there is fast convergence with χ up to a critical precision, which depends on δ and k. See Sec. IV C for details.

$j = 1, \dots, N$. Then for $i > 1$, the difference between $q_{ji}(x_j)$ and $q_{j1}(x_j)$ is controlled by δ. In Fig. 10 we use perturbations of magnitude $\delta = 0.01$ and $\delta = 0.1$ for $N = 20$, $G = 10$, and we monitor the error in the multivariable integral as a function of χ. In both cases we see that the integration error shows two regions of convergence; first there is a rapid convergence to some finite error (around 10^{-10} for $\delta = 0.01$, and 10^{-4} for $\delta = 0.1$), followed by a much slower convergence to smaller error. In the rapidly converging regime, the required bond dimension for a given relative error grows as $\chi \sim O(k)$; in the slow-convergence regime, the required bond dimension $\chi \sim \exp(k)$. In all cases, however, the cost to integrate is linear in the number of variables N, and this is independent of the desired precision (not shown for this case, but see next section). Exponential dependence of the complexity appears instead in the cost to improve the precision past a critical threshold.

D. General case

For the general polynomial in Eq. (19), we can expect the bond dimension for a given relative precision to scale $\sim \exp(k)$ as discussed; the tensor network is, in some sense, incompressible. However, in some cases, we may still be able to approximate the integral to good accuracy. This depends strongly on the integrand range, as illustrated in Fig. 11. In Fig. 11(a), we take $N = 20$, $G = 10$, and single-variable function values chosen randomly in the range $[-1, 1]$. Then, the corresponding 2D TN representation of the integral appears almost completely incompressible: Only when χ reaches the bond dimension of the exact contraction $\sim \exp(k)$ do we suddenly see a significant improvement in the integral error (although for rough estimates, e.g., to 10^{-1} precision, it is possible to take χ orders of magnitude below that required for exact contraction). We can make the single-variable functions less oscillatory by increasing the lower bound of the range of the single-variable functions, i.e., $[\lambda, 1]$, where λ is increased from -1 to 0. As we do so, the integration problems become easier, as can be seen from Figs. 11(a)–11(c). The decrease of relative error with increasing bond dimension is much faster as we raise the lower bound λ of the single-variable functions.

Monte Carlo integration can face difficulties with oscillatory functions with small or vanishing integrals. In the circuit tensor network representation, there is an exponential dependence for such oscillatory functions, but unlike in quadrature, the exponential cost is in the nonlinearity (parametrized by k), not in the number of variables N. Figure 11(d) plots the required bond dimension for a relative error 10^{-4} with $N = 20$, $G = 10$ as a function of the nonlinearity k for various single-variable function ranges $q_{ji} \in [\lambda, 1]$, where λ is increased from -1 to 0. We see that the required bond dimension increases exponentially in the nonlinearity. However, the exponent is much larger for oscillatory integrands ($\lambda = -1$) than for more positive integrands $\lambda > -1$.

We finally confirm that the error of compressed contraction is essentially independent of the number of variables using boundary contraction as can be seen from Figs. 11(e) and 11(f). Either for the general functional form, Eq. (19), with highly oscillatory values from random single-variable functions, or for the perturbation from the exact case, Eq. (30), with $\delta = 0.1$, the rate of decrease of relative error with increasing bond dimension is comparable as we change the number of variables N.

E. Comparison with quasi–Monte Carlo integration

To understand the concrete performance of integration using the arithmetic circuit TN, we now compare costs with those of quasi-MC. For this, we construct the polynomial in Eq. (19) with the multivariable function

$$q_{ji}(x_j) = \sin(2\pi(x_j + a_{ji})) + c \tag{31}$$

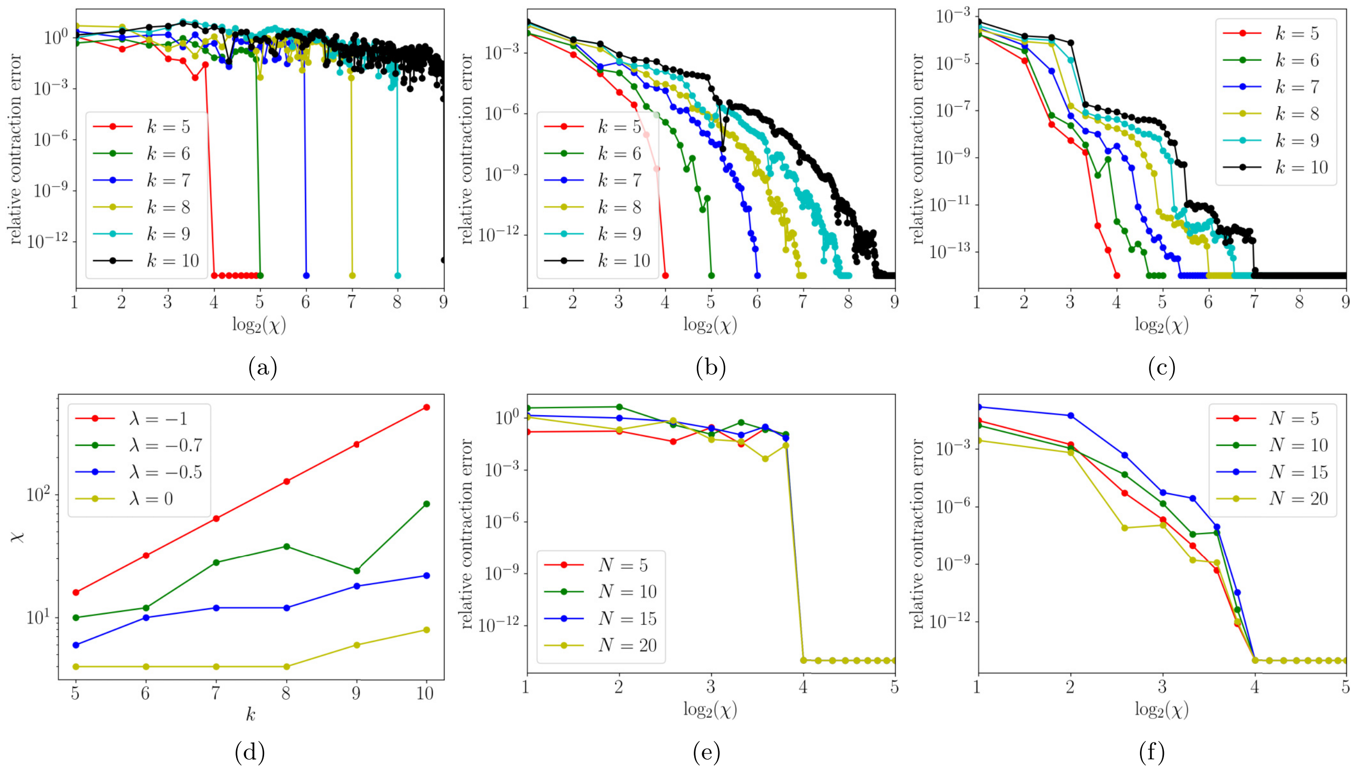


FIG. 11. Accuracy of TN integration of multivariable function polynomials. For (a)–(c), we plot accuracy vs χ for number of variables $N = 20$ and number of points per variable $G = 10$, with random single-variable functions q_{ji} with values $\in [\lambda, 1]$, for three different λ [(a) $\lambda = -1$, (b) $\lambda = -0.5$, and (c) $\lambda = 0$]; the difficulty of integration changes with the positivity of the integrand. (d) Bond dimension χ vs nonlinearity k to achieve a relative accuracy $\leqslant 10^{-4}$ for various λ for $N = 20$, $G = 10$. For (e) and (f), we plot relative accuracy vs χ for the general and perturbative ($\delta = 0.1$) function polynomial cases, respectively, with $k = 5$, $G = 10$, and single-variable functions $q_{ji}, r_{ji} \in [-1, 1]$, demonstrating the independence of accuracy with respect to the number of variables N. See Sec. IV D for details.

for random $a_{ji} \in [0, 1]$ and some constant c and integrate over $x_j \in [0, 1]$. We compare the results from TN contraction with quasi-MC integration in Fig. 12 (we report representative results for random a_{ji}) changing both the number of variables N and the polynomial power (nonlinearity) k. All calculations are done on Intel Xeon CPU E5-2697 v4 processors. For the quasi-MC calculations, the integrand in Eq. (19) was coded in PYTHON and then compiled by JAX [83], and the sample points were generated using the QMCPY [84] package with Sobol generating matrices using a batch size of 10^7, with the function values for each batch computed on a single core. The TN timings are reported as the runtime for contraction using the QUIMB [85] package on a single core or two cores, with the time normalized to a single core time. The reference exact result for the integration was taken as the converged TN result with respect to χ and G (specified in the caption of Fig. 12).

Figure 12(a) plots the convergence of the integral versus time for TN integration and quasi-MC for $N = 10$, $c = 0$, and various k. The TN data points represent increasing grid order G using Gauss-Legendre quadrature and bond dimension $\chi = 2^k$. For moderate k, quasi-MC already faces convergence difficulties as a result of the highly oscillatory function values around 0. On the other hand, the TN result converges quickly with G as shown in Fig. 12(b). Although details of implementation make it difficult to interpret small differences in absolute timing, the rapid convergence of the TN integration means that it is orders of magnitude more efficient than quasi-MC for high accuracies. However, the TN contraction cost becomes prohibitive at large k, due to using a $\chi = 2^k$ bond dimension.

For $c > 0$, the integrand is more positive, and we expect the TN to be more compressible, making it possible to integrate the function for larger k. In Fig. 12(c), we plot the convergence of the integral versus time for TN integration and quasi-MC for $N = 50$, $k = 30$, and various c. The TN data points correspond to $G = 12$ Gauss-Legendre quadrature for increasing bond dimension $\chi \in [60, 560]$. [To justify the choice of fixed $G = 12$, Fig. 12(d) shows the convergence of the TN contraction result as a function of G for various c at fixed χ (solid, dashed, and dotted lines correspond to $\chi = 180$, 120, and 60, respectively); at $G = 12$, the quadrature errors are below 10^{-6} for $c = 0.3$ and below 10^{-8} for $c = 0.4$ and $c = 0.5$, and these errors remain essentially constant as χ increases from 60 to 180.] As expected, for a fixed computational time, the integration error decreases for both methods with increasing c. Both methods display relatively quick convergence to a loose threshold, with slower convergence to tighter thresholds. The TN integration converges relatively smoothly to high accuracy, while it appears difficult to converge quasi-MC systematically to high accuracy, which again opens up a significant timing advantage for TN integration over quasi-MC.

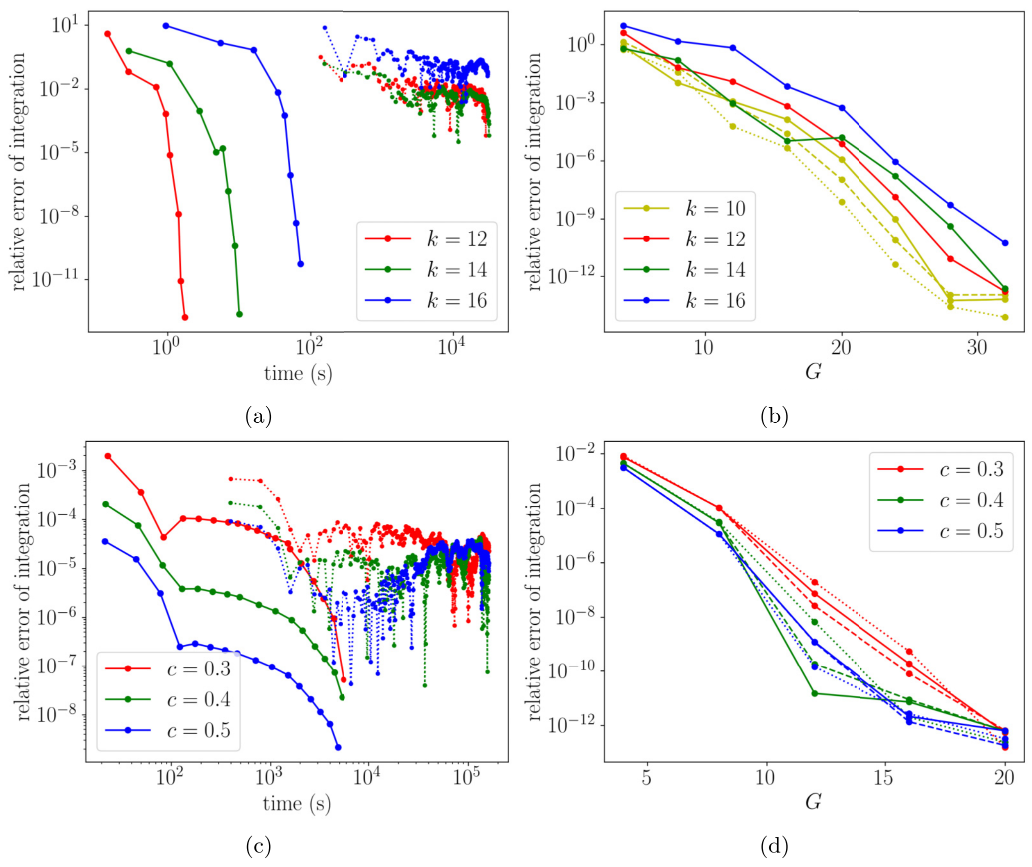


FIG. 12. Comparison of TN and quasi-MC integration for the multivariable function polynomial in Eqs. (19), (20), and (31). N is the number of variables, k is the number of factor powers in the function, and c is a function parameter that controls the positivity of the integrand. G is the number of quadrature points per variable in the TN integration. (a) TN (solid lines) and quasi-MC (dotted lines) integral convergence with runtime for various k at $N = 10$, $c = 0$. Exact result taken as TN integral at $G = 36$, bond dimension $\chi = 2^k$. (b) TN integral convergence with respect to G at $\chi = 2^k$ for various k at $c = 0$. Solid, dashed, and dotted lines are for $N = 10$, 20, and 40, respectively. (c) TN (solid lines) and quasi-MC (dotted lines) integral convergence with time for various c at $N = 50$, $k = 30$. Exact result taken as TN integral at $G = 12$, $\chi = 560$. (d) TN integral convergence with respect to G for various c at $N = 50$, $k = 30$. Solid, dashed, and dotted lines are for $\chi = 180$, 120, and 60, respectively. See Sec. IV E for further details.

V. MULTIVARIABLE GAUSSIAN INTEGRALS IN A HYPERCUBE

As another example, we consider the multivariable Gaussian integral over a finite hypercube

$$Z = \int_\Omega dx_1 \cdots dx_N \exp\left(- \sum_{ij} A_{ij} x_i x_j \right), \tag{32}$$

where A is an $N \times N$ matrix and $\Omega = [-1, 1]^N$. The expression directly corresponds to a tensor network contraction of tensors (T_i) and (T_{ij}) for all $i < j$ where

$$(T_i)_{x_i} = \exp\left(- A_{ii} x_i^2 \right), \tag{33}$$

$$(T_{ij})_{x_i, x_j} = \exp\left(- (A_{ij} + A_{ji}) x_i x_j \right). \tag{34}$$

Note that all tensors enter into the final function via multiplication only; thus there is no need for control legs as discussed in Sec. II, and they are omitted in the figures.

The structure of A plays an important role in the cost of approximability of the integral. In the following we consider band-diagonal A with width W, i.e., $A_{ij} \neq 0$ only if $|i - j| \leqslant W$ (dense A corresponds to $W = N - 1$). In Fig. 13, we show a systematic construction of the TN representation of the integral for $N = 5$, $W = 3$, which can easily be generalized to arbitrary N, W. We start with a TN consisting of only T_{12} [representing $\exp(-A_{12} x_1 x_2)$] as in Fig. 13(a). For each $j = 2, \dots, 1 + W$, we add to the TN the tensor T_{1j} and a COPY tensor to account for the additional occurrence of x_1 as in Figs. 13(b) and 13(c). For example, in Fig. 13(c), the four open legs represent the variables x_1, x_2, x_3, x_4, and the overall tensor network represents $\exp\left(- (A_{12} x_1 x_2 + A_{13} x_1 x_3 + A_{14} x_1 x_4) \right)$.

Next, for $i = 2$, and for each $j = i + 1, \dots, i + W$, we add to the TN the tensor T_{ij} and two COPY tensors; for example, in Figs. 13(d) and 13(e), the COPY tensors connected by the horizontal bonds correspond to copying the variable x_i, and the COPY tensors connected by the vertical bonds correspond to copying x_j for each $j = 3, 4, \dots$. Iterating the previous step for each $i = 3, \dots, N - 1$ as in Fig. 13(f), one adds all T_{ij} to the TN. Finally, we add T_i to the TN for each $i = 1, \dots, N$ as in Fig. 13(g). Contracting the tensors in each dashed box, one obtains the 2D TN shown in Fig. 13(h). The TN obtained from

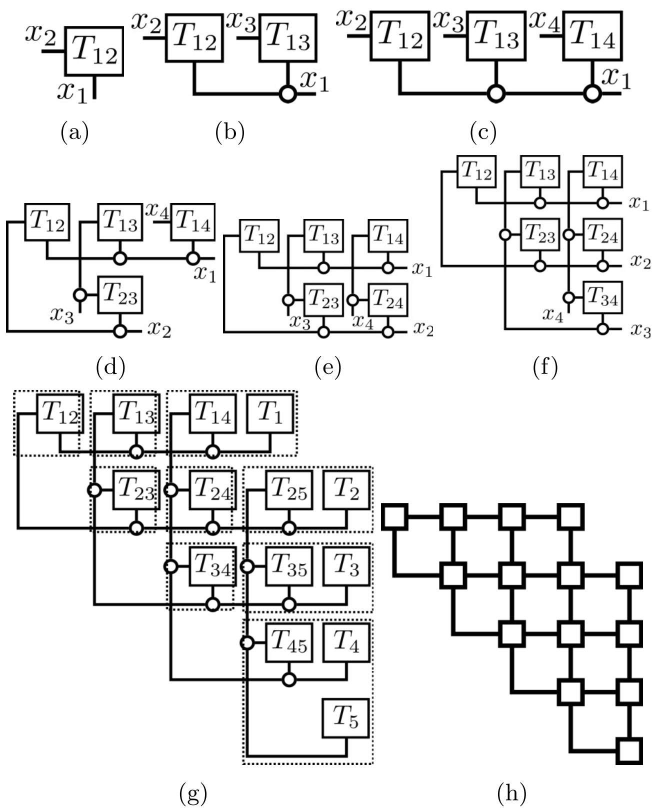


FIG. 13. (a)–(h) Steps for building the arithmetic circuit TN for a multivariable Gaussian with $N = 5$ variables and width $W = 3$. Circles represent COPY tensors.

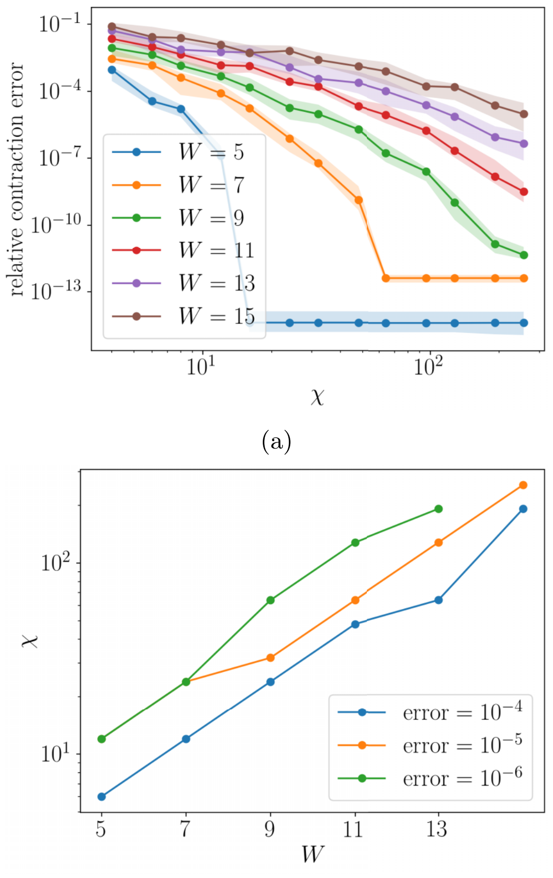


FIG. 15. TN integration for a multivariable Gaussian with exactly banded matrix A in the hypercube. Accuracy vs bond dimension χ for number of variables $N = 30$ and number of points per variable $G = 4$. (a) Median and interquartile range of error over 20 random instances of the Gaussian matrix A. (b) Bond dimension to achieve various median contraction errors. For details, see Sec. V A.

the procedure above clearly has a direct correspondence with the structure of A. If A is dense, then the graph is a triangular network, and if A is banded, then the network is also banded.

A. Fixed width compressibility

For banded A (fixed W with N), the TN has a quasi-1D structure along the diagonal direction. Thus it is always possible to contract the network *exactly* with cost linear in N and exponential in W along the diagonal direction. If W is large, it may still be too expensive to use exact contraction, but the regular structure lends itself to a variant of boundary contraction as shown in Fig. 14.

In this case we limit the maximum bond dimension χ during the approximate contraction, with χ expected to scale as $\sim e^W$ to achieve a fixed relative error. To compress-contract the whole tensor network, one needs to perform $O(NW)$ contractions and SVDs on adjacent tensor pairs. The contraction between a $\chi \times \chi \times G$ tensor and a $G \times G \times G \times G$ tensor costs $O(\chi^2 G^4)$ FLOPs, and the SVD of a $\chi G \times \chi G \times G$ tensor costs $O(\chi^3 G^4)$ FLOPs. Hence the total FLOP count scales as $O(NW\chi^3 G^4)$. We show the relative contraction error with respect to χ for various W in Fig. 15(a) and the bond dimension to achieve various median contraction error as a function of W in Fig. 15(b), for a 30×30 band-diagonal matrix A with width W and random nonzero elements in $[-1, 1]$. We see that the difficulty of compression indeed scales exponentially with W from the linear trend in the log-linear plot.

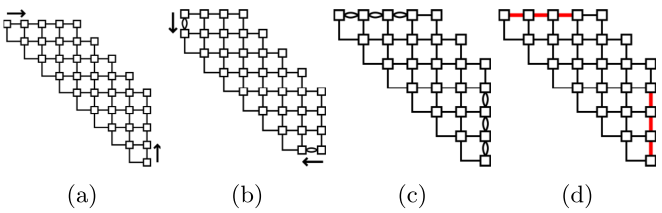


FIG. 14. (a)–(d) Variant of boundary contraction used to contract the arithmetic circuit TN for the multivariable Gaussian integral.

B. Approximately banded case

Above, we demonstrate that for moderate width W, the quasi-1D TN can be exactly contracted. As in the perturbed case for the polynomial example, we can also ask whether the TN for an approximately band-diagonal A remains compressible. We thus consider A of the form

$$A = A_W + \delta \cdot \tilde{A}_W, \tag{35}$$

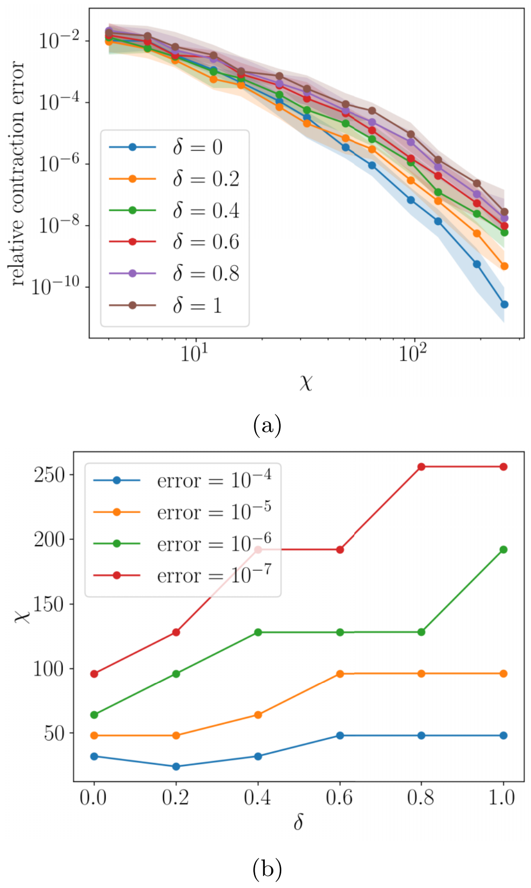


FIG. 16. TN integration for a multivariable Gaussian with approximately banded matrix A in the hypercube. Accuracy vs bond dimension χ for number of variables $N = 30$ and number of points per variable $G = 4$. (a) Median and interquartile range over 20 random instances. (b) Bond dimension to achieve various median contraction errors. For details, see Sec. V B.

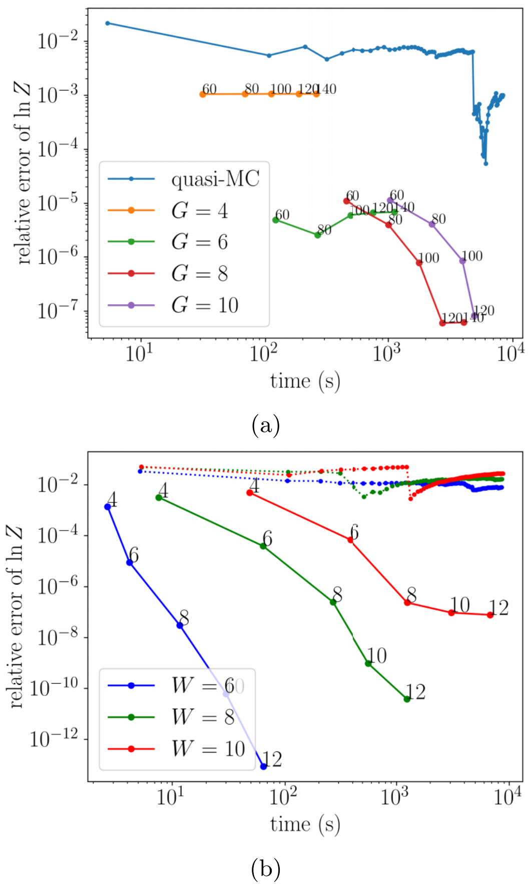


FIG. 17. Comparison of TN and quasi-MC integration errors for the multivariable Gaussian integral Z in the hypercube, plotted against runtime. (Error plotted is for $\ln Z$ to better distinguish the curves.) N is the number of variables, and G is the number of quadrature points per variable in the TN. (a) Exact result taken as TN partition function at $G = 10$ and bond dimension $\chi = 140$ (bond dimension error below 5×10^{-7}). The TN errors are plotted for $\chi = 60, 80, 100, 120, 140$ for each G, which are the labels next to the TN data points. (b) Exact result is taken at $G = 14$ for converged $\chi = 80$, 140, and 200 for $W = 6$, 8, and 10 with contraction error $\leqslant 10^{-15}$, 10^{-9}, and 10^{-6}, respectively. TN (quasi-MC) results are represented by solid (dotted) lines. The label next to each data point corresponds to its G value.

where A_W is an $N \times N$ band-diagonal matrix with width W such that $(A_W)_{ij} \neq 0$ only if $|i-j| \leqslant W$, $\tilde{A}_W$ is an $N \times N$ matrix such that $(\tilde{A}_W)_{ij} \neq 0$ only if $|i-j| > W$, and nonzero elements of $A_W, \tilde{A}_W$ take random values in $[-1, 1]$. Then the magnitude of the perturbation of A away from A_W is controlled by δ. Note that the existence of the perturbation makes the TN a full 2D triangle, rather than quasi-1D.

We show the relative contraction error with respect to χ for various δ in Fig. 16(a) and the bond dimension required to achieve various median errors as a function of δ in Fig. 16(b) for A of dimension 20×20. We require increasingly large χ to reach the same error as δ increases. However, Fig. 16(b) is a linear-linear plot, showing that the required χ appears to grow only linearly with the size of the perturbation δ.

C. Comparison with quasi–Monte Carlo integration

We now compare the efficiency of TN integration for Gaussian integrals with banded (or approximately banded) A with that of quasi–Monte Carlo integration. We consider two cases for A of the form in Eq. (35): In case (a), $N = 50$, $W = 5$, and $\delta = 0.1$ with random nonzero elements of $A_W, \tilde{A}_W$ in $[-1, 1]$, and in case (b), $N = 50$, $W = 6$, 8, or 10, and $\delta = 0$ with random nonzero elements of A_W in $[-1, 1]$.

In Fig. 17, we show the relative error of $\ln Z$ from TN and quasi-MC integration for cases (a) and (b). For the quasi-MC results, we directly computed the exponent $-\sum_{ij} A_{ij} x_i x_j$ for each sample point x with batched linear algebra code using PYTHON and JAX [83] and accumulated the log of the integral using the identity

$$\ln(e^a + e^b) = a + \ln(1 + e^{b-a}).$$

The sample points were generated using QMCPY [84] with Niederreiter generating matrices in batches of size 10^7 and plotted every 20 batches. The TN integration was performed using QUIMB [85] with Gauss-Legendre quadrature, and the exact value of the integral was estimated from TN integration using converged G and χ (indicated in the caption of Fig. 17).

All calculations are done on a single Intel Xeon CPU E5-2697 v4 processor.

For the TN integration data in Fig. 17(a) [corresponding to case (a)] each line labeled by G is a sequence of TN estimates of $\ln Z$ at the corresponding G with increasing $\chi = 60, 80, \ldots, 140$. For $G = 4$ and $G = 6$, the value of $\ln Z$ plateaus at large χ, corresponding to the intrinsic quadrature error for that G. For $G = 8$ and $G = 10$, the TN results are well converged with quadrature, and the small perturbation strength $\delta = 0.1$ allows for quick convergence with χ. In comparison, quasi-MC struggles with this integrand, and in fact shows little systematic convergence behavior over the $\sim$1600 $\times$ 10^7 samples.

For case (b), shown in Fig. 17(b), we plot the TN data points for increasing $G = 4, 6, \ldots, 12$ at converged χ for each W (solid lines). This allows us to examine the speed of convergence of the quadrature, which is very fast. Similar to case (a), quasi-MC struggles to reduce the relative error below 10^{-2} even with more than 1600 $\times$ 10^7 samples. Thus, in both cases, we see a substantial advantage of TN integration over quasi-MC for this class of integrands.

VI. A THEORETICAL EXAMPLE OF SPEEDUP VERSUS QUASI–MONTE CARLO INTEGRATION

Here we give an example of a class of integrals that can be computed easily with arithmetic TN methods but for which quasi-MC is hard, because there is a theoretical guarantee that the integrand value cannot be efficiently evaluated without exponential cost in the number of variables N.

Consider the general functional form

$$f(x_1, \ldots, x_N) = \prod_{n=1}^{N} g(x_n)^{i_n} (\mathrm{TN})_{i_1,\ldots,i_N}, \tag{36}$$

where $g(x_n)$ are single-variable functions and $(\mathrm{TN})_{i_1,\ldots,i_N}$ is some fixed tensor (possibly represented as a tensor network) for all $i_n = 0, 1$. We can choose the structure and the values of the $\mathrm{TN}_{i_1,\ldots,i_N}$ and properties of $g(x_n)$ to allow for the easy computation of certain quantities. In our case, we choose the TN to be a 2D multiscale entanglement renormalization ansatz (MERA) [59,60] (see Fig. 18). In this case, the TN in Eq. (36) is constructed from tensors satisfying certain unitary and isometric properties such that the evaluation of the trace $\sum_{\{i\}} |\mathrm{TN}_{i_1,\ldots,i_N}|^2 = 1$. The unitary and isometric constraints on the tensors and how they lead to the trivial trace are shown in Figs. 18(c) and 18(d). Note that it is known to be classically hard (as a function of N) to compute the value $\mathrm{TN}_{i_1,\ldots,i_N}$ for a MERA composed of arbitrary unitaries and isometries [86].

To extend these properties to the more general functional form, we impose a normalization condition on the single-variable functions

$$\int_{-1}^{1} dx |g(x)|^2 = 2, \tag{37}$$

$$\int_{-1}^{1} dx g(x) = 0. \tag{38}$$

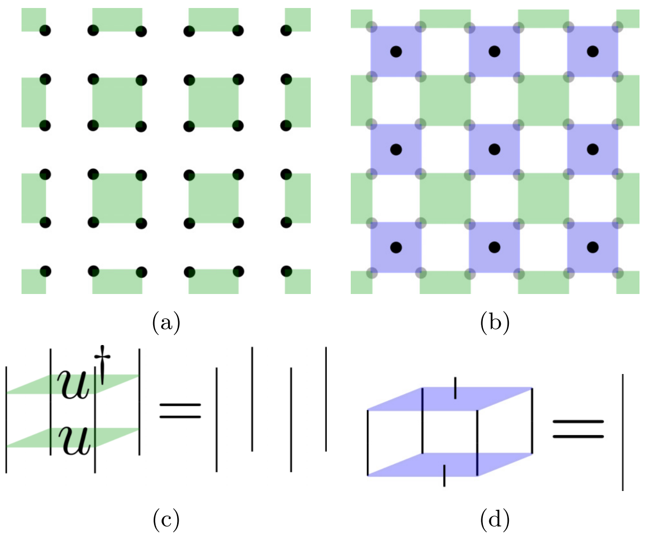


FIG. 18. $\mathrm{TN}_{i_1,\ldots,i_N}$ as a 2D MERA, where the legs $i_1, \ldots, i_N$ lie on a 2D lattice. (a) Part of the 2D lattice, where each solid dot represent a leg in $\{i_1, \ldots, i_N\}$. Each green square represent a unitary tensor. (b) After the layer of unitaries, a layer of isometries (blue squares) is applied to the legs $i_1, \ldots, i_N$. Note that after a layer of isometries is applied, the number of legs in the layer is reduced. Traversing upwards through the MERA, one reaches the top tensor where all upwards legs are eliminated, and the MERA represents a TN with legs $i_1, \ldots, i_N$ on the bottom. (c) Property of a unitary tensor. (d) Property of an isometric tensor.

Then, suppose we want to integrate the probability of the function over $\Omega = [-1, 1]^N$

$$I = \int_{\Omega} |f(\{x_n\})|^2 \prod dx_n. \tag{39}$$

Because of the additional constraints we have imposed on the single-variable functions, contraction of the tensors associated with I can be done efficiently for matching pairs, which then simplify to multiples of the identity, yielding 2^N as the value of the integral, even though sampling in the basis of $\{x_n\}$ is computationally hard. This result is clearly contrived because, in addition to requiring a restricted class of tensors and functions, we must also contract the tensor network in a certain order. Furthermore, the structure of the tensors permits sampling in a different basis than $\{x_n\}$ [86,87]. Nonetheless, this is a concrete example supported by complexity arguments where using the tensor network structure of the function circuit leads to an exponential improvement in cost versus quasi–Monte Carlo methods which assume that the function is a black box.

VII. CONCLUSIONS

In this paper we introduced an arithmetic circuit tensor network representation of multivariable functions and demonstrated its power for high-dimensional integration. Compared with existing techniques to represent functions with tensor networks, the ability to use the arithmetic circuit construction removes the need for extra computation to find the tensor representation, while the circuit structure suggests a tensor network connectivity natural to the function. In our examples of high-dimensional integration, we find that the tensor network representation allows us to circumvent the curse of dimensionality in many cases, exchanging the exponential cost dependence on dimension for a cost dependence on other circuit characteristics, for example, the number of nonlinear circuit operations. In practice, we find superior performance to quasi–Monte Carlo integration across a range of dimensionalities and accuracies.

While the work here focuses on integration as an example, we envision the arithmetic circuit tensor network construction to be powerful also in differential equation applications. Here, connections with existing tensor network techniques are intriguing, as are applications of these ideas to many other areas where tensor networks are currently employed, such as for many-body simulations.

ACKNOWLEDGMENTS

R.P. (arithmetic tensor network development) was partially supported by an Eddleman Graduate Fellowship and the US Department of Energy, Office of Science, via Award No. DE-SC0019390. Work by J.G. (Quimb development and support) was supported by the US Department of Energy, Office of Science, via Award No. DE-SC0018140. G.K.-L.C. acknowledges support from the Simons Investigator award.

[1] V. Popov, J. Niederle, and L. Hlavaty, *Functional Integrals in Quantum Field Theory and Statistical Physics*, Mathematical Physics and Applied Mathematics (Springer, Dordrecht, Netherlands, 2001).
[2] I. M. Gel'fand and A. M. Yaglom, Integration in functional spaces and its applications in quantum physics, J. Math. Phys. **1**, 48 (1960).
[3] P. Cartier and C. DeWitt-Morette, Functional integration, J. Math. Phys. **41**, 4154 (2000).
[4] D. D. Lewis and J. Catlett, Heterogeneous uncertainty sampling for supervised learning, in *Machine Learning Proceedings 1994*, edited by W. W. Cohen and H. Hirsh (Kaufmann, San Francisco, 1994), pp. 148–156.
[5] E. Liberty, K. Lang, and K. Shmakov, Stratified sampling meets machine learning, in *Proceedings of The 33rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, edited by M. F. Balcan and K. Q. Weinberger (PMLR, New York, 2016), Vol. 48, pp. 2320–2329.
[6] A. ElRafey and J. Wojtusiak, Recent advances in scaling-down sampling methods in machine learning, WIREs Comput. Stat. **9**, e1414 (2017).
[7] C. A. Ramezan, T. A. Warner, and A. E. Maxwell, Evaluation of sampling and cross-validation tuning strategies for regional-scale machine learning classification, Remote Sens. **11**, 185 (2019).
[8] S. Tyagi and S. Mittal, Sampling approaches for imbalanced data classification problem in machine learning, in *Proceedings of ICRIC 2019*, edited by P. K. Singh, A. K. Kar, Y. Singh, M. H. Kolekar, and S. Tanwar (Springer, Cham, Switzerland, 2020), pp. 209–221.
[9] C. Kaltenecker, A. Grebhahn, N. Siegmund, and S. Apel, The interplay of sampling and machine learning for software performance prediction, IEEE Software **37**, 58 (2020).
[10] W. J. Morokoff and R. E. Caflisch, Quasi-random sequences and their discrepancies, SIAM J. Sci. Comput. **15**, 1251 (1994).
[11] B. Tuffin, On the use of low discrepancy sequences in Monte Carlo methods, Monte Carlo Methods Appl. **2**, 295 (1996).
[12] F. James, J. Hoogland, and R. Kleiss, Multidimensional sampling for simulation and integration: Measures, discrepancies, and quasi-random numbers, Comput. Phys. Commun. **99**, 180 (1997).
[13] L. Kocis and W. J. Whiten, Computational investigations of low-discrepancy sequences, ACM Trans. Math. Software **23**, 266 (1997).
[14] R. Orús, A practical introduction to tensor networks: Matrix product states and projected entangled pair states, Ann. Phys. (Amsterdam) **349**, 117 (2014).
[15] F. Verstraete, V. Murg, and J. Cirac, Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems, Adv. Phys. **57**, 143 (2008).
[16] P. Silvi, F. Tschirsich, M. Gerster, J. Jünemann, D. Jaschke, M. Rizzi, and S. Montangero, The tensor networks anthology: Simulation techniques for many-body quantum lattice systems, SciPost Phys. Lect. Notes **2019**, 8 (2019).
[17] R. Orús, Tensor networks for complex quantum systems, Nat. Rev. Phys. **1**, 538 (2019).
[18] T. Felser, S. Notarnicola, and S. Montangero, Efficient Tensor Network *Ansatz* for High-Dimensional Quantum Many-Body Problems, Phys. Rev. Lett. **126**, 170603 (2021).
[19] J. Chen, S. Cheng, H. Xie, L. Wang, and T. Xiang, Equivalence of restricted Boltzmann machines and tensor network states, Phys. Rev. B **97**, 085104 (2018).
[20] L. Pastori, R. Kaubruegger, and J. C. Budich, Generalized transfer matrix states from artificial neural networks, Phys. Rev. B **99**, 165123 (2019).
[21] S. Li, F. Pan, P. Zhou, and P. Zhang, Boltzmann machines as two-dimensional tensor networks, Phys. Rev. B **104**, 075154 (2021).
[22] R. Harshman, *Foundations of the PARAFAC Procedure: Models and Conditions for an "Explanatory" Multi-modal Factor Analysis*, UCLA Working Papers in Phonetics (University of California, Los Angeles, 1970).
[23] J. D. Carroll and J.-J. Chang, Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition, Psychometrika **35**, 283 (1970).
[24] G. Beylkin and M. J. Mohlenkamp, Numerical operator calculus in higher dimensions, Proc. Natl. Acad. Sci. USA **99**, 10246 (2002).
[25] G. Beylkin and M. J. Mohlenkamp, Algorithms for numerical analysis in high dimensions, SIAM J. Sci. Comput. **26**, 2133 (2005).
[26] T. G. Kolda and B. W. Bader, Tensor decompositions and applications, SIAM Rev. **51**, 455 (2009).
[27] L. Grasedyck, Hierarchical singular value decomposition of tensors, SIAM J. Matrix Anal. Appl. **31**, 2029 (2010).
[28] L. Grasedyck and W. Hackbusch, An introduction to hierarchical (H-) rank and TT-rank of tensors with examples, Comput. Methods Appl. Math. **11**, 291 (2011).

[29] I. V. Oseledets, Tensor-train decomposition, SIAM J. Sci. Comput. **33**, 2295 (2011).

[30] S. Holtz, T. Rohwedder, and R. Schneider, The alternating linear scheme for tensor optimization in the tensor train format, SIAM J. Sci. Comput. **34**, A683 (2012).

[31] M. Espig, W. Hackbusch, S. Handschuh, and R. Schneider, Optimization problems in contracted tensor networks, Comput. Visualization Sci. **14**, 271 (2011).

[32] A. Novikov, M. Trofimov, and I. Oseledets, Exponential machines, Bulletin of the Polish Academy of Sciences **66**, 789 (2018).

[33] I. Oseledets and E. Tyrtyshnikov, TT-cross approximation for multidimensional arrays, Linear Algebra Its Appl. **432**, 70 (2010).

[34] S. Dolgov and D. Savostyanov, Parallel cross interpolation for high-precision calculation of high-dimensional integrals, Comput. Phys. Commun. **246**, 106869 (2020).

[35] J. J. García-Ripoll, Quantum-inspired algorithms for multivariate analysis: from interpolation to partial differential equations, Quantum **5**, 431 (2021).

[36] G. Evenbly and R. N. C. Pfeifer, Improving the efficiency of variational tensor network algorithms, Phys. Rev. B **89**, 245118 (2014).

[37] L. Vanderstraeten, J. Haegeman, P. Corboz, and F. Verstraete, Gradient methods for variational optimization of projected entangled-pair states, Phys. Rev. B **94**, 155123 (2016).

[38] R. Haghshenas, J. Gray, A. C. Potter, and G. K.-L. Chan, Variational Power of Quantum Circuit Tensor Networks, Phys. Rev. X **12**, 011047 (2022).

[39] A. J. Daley, C. Kollath, U. Schollwöck, and G. Vidal, Time-dependent density-matrix renormalization-group using adaptive effective Hilbert spaces, J. Stat. Mech.: Theory Exp. (2004) P04005.

[40] S. Iblisdir, R. Orús, and J. I. Latorre, Matrix product states algorithms and continuous systems, Phys. Rev. B **75**, 104305 (2007).

[41] H. N. Phien, I. P. McCulloch, and G. Vidal, Fast convergence of imaginary time evolution tensor network algorithms by recycling the environment, Phys. Rev. B **91**, 115137 (2015).

[42] S.-H. Lin, R. Dilip, A. G. Green, A. Smith, and F. Pollmann, Real- and imaginary-time evolution with compressed quantum circuits, PRX Quantum **2**, 010342 (2021).

[43] M. Lubasch, P. Moinier, and D. Jaksch, Multigrid renormalization, J. Comput. Phys. **372**, 587 (2018).

[44] N. Gourianov, M. Lubasch, S. Dolgov, Q. Y. van den Berg, H. Babaee, P. Givi, M. Kiffner, and D. Jaksch, A quantum-inspired approach to exploit turbulence structures, Nat. Comput. Sci. **2**, 30 (2022).

[45] I. Glasser, R. Sweke, N. Pancotti, J. Eisert, and J. I. Cirac, Expressive power of tensor-network factorizations for probabilistic modeling, in *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (Curran Associates, Red Hook, NY, 2019), pp. 1498–1510.

[46] Y. Levine, O. Sharir, N. Cohen, and A. Shashua, Quantum Entanglement in Deep Learning Architectures, Phys. Rev. Lett. **122**, 065301 (2019).

[47] A. K. Verma, P. Brisk, and P. Ienne, Variable latency speculative addition: A new paradigm for arithmetic circuit design, in *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '08* (Association for Computing Machinery, New York, 2008), pp. 1250–1255.

[48] M. Ciesielski, T. Su, A. Yasin, and C. Yu, Understanding algebraic rewriting for arithmetic circuit verification: A bit-flow model, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **39**, 1346 (2020).

[49] N. Wiebe and M. Roetteler, Quantum arithmetic and numerical analysis using repeat-until-success circuits, Quantum Inf. Comput. **16**, 134 (2016).

[50] T. Haener, M. Soeken, M. Roetteler, and K. M. Svore, Quantum circuits for floating-point arithmetic, in *Reversible Computation*, edited by J. Kari and I. Ulidowski (Springer, Cham, Switzerland, 2018), pp. 162–174.

[51] J. Gray and S. Kourtis, Hyper-optimized tensor network contraction, Quantum **5**, 410 (2021).

[52] F. Pan, P. Zhou, S. Li, and P. Zhang, Contracting Arbitrary Tensor Networks: General Approximate Algorithm and Applications in Graphical Models and Quantum Circuit Simulations, Phys. Rev. Lett. **125**, 060503 (2020).

[53] A. S. Jermyn, Automatic contraction of unstructured tensor networks, SciPost Phys. **8**, 005 (2020).

[54] C. T. Chubb, General tensor network decoding of 2D Pauli codes, arXiv:2101.04125.

[55] J. Gray and G. K.-L. Chan, Hyper-optimized compressed contraction of tensor networks with arbitrary geometry, arXiv:2206.07044.

[56] L. Vysotsky, A. Smirnov, and E. Tyrtyshnikov, Tensor-train numerical integration of multivariate functions with singularities, Lobachevskii J. Math. **42**, 1608 (2021).

[57] Y. Nuñez-Fernández, M. Jeannin, P. T. Dumitrescu, T. Kloss, J. Kaye, O. Parcollet, and X. Waintal, Learning Feynman Diagrams with Tensor Trains, Phys. Rev. X **12**, 041018 (2022).

[58] I. Oseledets, Constructive representation of functions in low-rank tensor formats, Constructive Approximation **37**, 1 (2013).

[59] G. Evenbly and G. Vidal, Algorithms for entanglement renormalization, Phys. Rev. B **79**, 144108 (2009).

[60] G. Evenbly and G. Vidal, Algorithms for entanglement renormalization: Boundaries, impurities and interfaces, J. Stat. Phys. **157**, 931 (2014).

[61] S. Bravyi, *Contraction of matchgate tensor networks on non-planar graphs, in Advances in Quantum Computation*, Contemporary Mathematics (American Mathematical Society, Providence, RI, 2009), Vol. 482, pp. 179–211.

[62] E. S. Fried, N. P. D. Sawaya, Y. Cao, I. D. Kivlichan, J. Romero, and A. Aspuru-Guzik, qTorch: The quantum tensor contraction handler, PLoS ONE **13**, 1 (2018).

[63] M. Levin and C. P. Nave, Tensor Renormalization Group Approach to Two-Dimensional Classical Lattice Models, Phys. Rev. Lett. **99**, 120601 (2007).

[64] G. Evenbly and G. Vidal, Tensor Network Renormalization, Phys. Rev. Lett. **115**, 180405 (2015).

[65] G. Evenbly, Algorithms for tensor network renormalization, Phys. Rev. B **95**, 045117 (2017).

[66] M. Bal, M. Mariën, J. Haegeman, and F. Verstraete, Renormalization Group Flows of Hamiltonians using Tensor Networks, Phys. Rev. Lett. **118**, 250602 (2017).

[67] S. Yang, Z.-C. Gu, and X.-G. Wen, Loop Optimization for Tensor Network Renormalization, Phys. Rev. Lett. **118**, 110504 (2017).

[68] S.-J. Ran, E. Tirrito, C. Peng, X. Chen, L. Tagliacozzo, G. Su, and M. Lewenstein, *Tensor Network Contractions*, Lecture Notes in Physics (Springer, Cham, Switzerland, 2020).

[69] F. Schindler and A. S. Jermyn, Algorithms for tensor network contraction ordering, Mach. Learn.: Sci. Technol. **1**, 035001 (2020).

[70] S. J. Denny, J. D. Biamonte, D. Jaksch, and S. R. Clark, Algebraically contractible topological tensor network states, J. Phys. A: Math. Theor. **45**, 015309 (2012).

[71] P. C. G. Vlaar and P. Corboz, Simulation of three-dimensional quantum systems with projected entangled-pair states, Phys. Rev. B **103**, 205137 (2021).

[72] V. Gogate and R. Dechter, A complete anytime algorithm for treewidth, arXiv:1207.4109.

[73] R. N. C. Pfeifer, J. Haegeman, and F. Verstraete, Faster identification of optimal contraction sequences for tensor networks, Phys. Rev. E **90**, 033315 (2014).

[74] B. Strasser, Computing tree decompositions with FlowCutter: PACE 2017 submission, arXiv:1709.08949.

[75] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, Engineering a direct *k*-way hypergraph partitioning algorithm, in *2017 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017), pp. 28–42.

[76] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, and H. Neven, Simulation of low-depth quantum circuits as complex undirected graphical models, arXiv:1712.05384.

[77] S. Kourtis, C. Chamon, E. R. Mucciolo, and A. E. Ruckenstein, Fast counting with tensor networks, SciPost Phys. **7**, 060 (2019).

[78] J. M. Dudek, L. Dueñas-Osorio, and M. Y. Vardi, Efficient contraction of large tensor networks for weighted model counting through graph decompositions, arXiv:1908.04381.

[79] L. Grasedyck, D. Kressner, and C. Tobler, A literature survey of low-rank tensor approximation techniques, arXiv:1302.7121.

[80] N. K. Kumar and J. Schneider, Literature survey on low rank approximation of matrices, Linear Multilinear Algebra **65**, 2212 (2017).

[81] B. N. Khoromskij, Tensors-structured numerical methods in scientific computing: Survey on recent advances, Chemom. Intell. Lab. Syst. **110**, 1 (2012).

[82] B. Fu, Multivariate polynomial integration and differentiation are polynomial time inapproximable unless P=NP, in *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, edited by J. Snoeyink, P. Lu, K. Su, and L. Wang (Springer, Berlin, 2012), pp. 182–191.

[83] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, JAX: Composable transformations of Python+NumPy programs (2018).

[84] S.-C. T. Choi, F. J. Hickernell, M. McCourt, and A. Sorokin, QMCPy: A quasi-Monte Carlo Python library, https://github.com/google/jax.

[85] J. Gray, quimb: A python library for quantum information and many-body calculations, J. Open Source Software **3**, 819 (2018).

[86] A. J. Ferris and G. Vidal, Variational Monte Carlo with the multiscale entanglement renormalization ansatz, Phys. Rev. B **85**, 165147 (2012).

[87] A. J. Ferris and G. Vidal, Perfect sampling with unitary tensor networks, Phys. Rev. B **85**, 165146 (2012).