YASTN: Yet another symmetric tensor networks; A Python library for Abelian symmetric tensor network calculations

Marek M. Rams[1⋆], Gabriela Wójtowicz[1,2†], Aritra Sinha[1,3‡] and Juraj Hasik[4,5○]

1 Jagiellonian University, Institute of Theoretical Physics, Łojasiewicza 11, 30-348 Kraków, Poland
2 Institut für Theoretische Physik und IQST, Albert-Einstein-Allee 11, Universität Ulm, D-89081 Ulm, Germany
3 Max Planck Institute for the Physics of Complex Systems, Nöthnitzer Strasse 38, Dresden 01187, Germany
4 Institute for Theoretical Physics and Delta Institute for Theoretical Physics, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands
5 Department of Physics, University of Zurich, Winterthurerstrasse 190, 8057 Zurich, Switzerland

⋆ marek.rams@uj.edu.pl , † gabriela.wojtowicz@uni-ulm.de , ‡ asinha@pks.mpg.de , ○ juraj.hasik@physik.uzh.ch

Abstract

We present an open-source tensor network Python library for quantum many-body simulations. At its core is an Abelian-symmetric tensor, implemented as a sparse block structure managed by a logical layer on top of a dense multidimensional array backend. This serves as the basis for higher-level tensor network algorithms operating on matrix product states and projected entangled pair states. An appropriate backend, such as PyTorch, gives direct access to automatic differentiation (AD) for cost-function gradient calculations and execution on GPU and other supported accelerators. We show the library performance in simulations with infinite projected entangled-pair states, such as finding the ground states with AD and simulating thermal states of the Hubbard model via imaginary time evolution. For these challenging examples, we identify and quantify sources of the numerical advantage exploited by the symmetric-tensor implementation.

Received 2024-06-27
Accepted 2025-02-03
Published 2025-02-26
doi:10.21468/SciPostPhysCodeb.52

Check for updates

This publication is part of a bundle: Please cite both the article and the release you used.

DOI	Type
doi:10.21468/SciPostPhysCodeb.52	Article
doi:10.21468/SciPostPhysCodeb.52-r1.2	Codebase release

Contents

1 Introduction

Full numerical treatment of quantum-mechanical systems is generally prohibitively expensive, due to the exponential growth of the Hilbert space size with the number of interacting degrees of freedom. *Tensor network* (TN) techniques allow efficient representation and manipulation of the states of such large quantum systems [1–3]. The *density matrix renormalization group* (DMRG) introduced by White [4, 5] and its modern reformulation in terms of *matrix product states* [6–10] (MPS), a one-dimensional (1D) tensor network, is a prime example of TN capabilities. Since their inception, MPS quickly became a reference method for addressing ground states in 1D, and were soon followed by extensions to excited states, time evolution, and open systems, forming a comprehensive framework.

The descriptive power of TNs generalizes to higher-dimensional models. The MPS, despite its intrinsic 1D geometry, can be readily applied to systems in higher dimensions by imposing linear ordering of lattice sites. Two-dimensional (2D) systems are often limited to finite cylinders that are mapped to the MPS Ansatz by imposing ordering that winds around the cylinder circumference; see Fig. 1(c). More natural TN geometry for 2D states is assured by the *projected entangled-pair states* (PEPS) [11, 12], see Fig. 1(e), and the similar for 3D states [13, 14].

The TN ansätze in Fig. 1 provide a state-of-the-art numerical approach to strongly correlated systems of condensed matter. The computational complexity of MPS typically scales as $\mathcal{O}(D^3)$, and PEPS algorithms often reach $\mathcal{O}(D^{12})$ scaling, where bond dimension D governs the size of the tensors and the overall precision of the TN approximations. Although the scaling of PEPS seems less favorable, it is important to note that the bond dimension encodes correlations between sites. Imposing winding, column-by-column ordering for the MPS on a cylinder stretches the correlations between columns and leads to long-range correlations across the MPS. The PEPS, on the other hand, already possesses natural geometry for nearest-neighbor correlations in 2D. As a result, the PEPS can reach comparable or better precision even at low bond dimensions once the cylinder width within the MPS approach exceeds a few sites.

The most effective way to mitigate computational complexity is to take advantage of the symmetries present in physical systems. Two principal types of symmetries to consider are

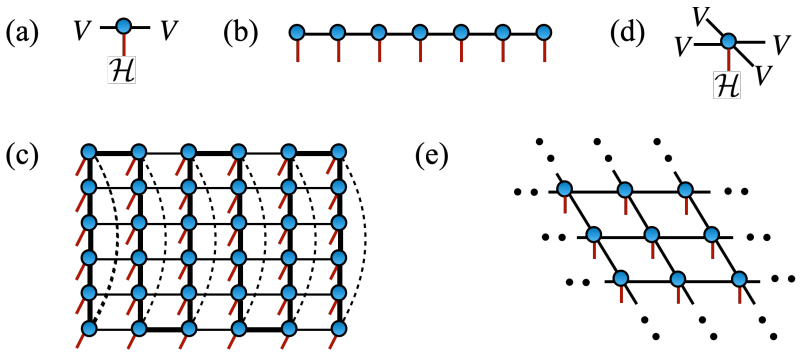

Figure 1: **Tensor networks.** Diagrams depict (a) a rank-3 tensor of the MPS with left and right virtual spaces V and physical space $\mathcal{H}$, (b) an MPS Ansatz, (c) a MPS winding on a finite width-6 cylinder, (d) a rank-5 tensor of the PEPS, and (e) an infinite-PEPS Ansatz.

spatial and internal symmetries. Tensor networks can be formulated directly in the thermodynamic limit by an infinitely repeated pattern (a unit cell) of tensors, hence realizing translation symmetry. These are infinite-MPS (iMPS), also known as uniform MPS [15] in 1D and infinite-PEPS (iPEPS) [16] in 2D. The computational complexity of the iMPS/iPEPS algorithms scales linearly with the size of the unit cell.

For internal symmetries $U|\Psi\rangle = |\Psi\rangle$, we consider their common form of global symmetries, i.e., when $U = \otimes_i u_i$ with the same unitaries u_i acting on each lattice site. These can be both Abelian (e.g., particle conservation) or non-Abelian (e.g., SU(2)-spin). Crucially, such global symmetries can be implemented in TNs locally by requiring individual tensors to transform covariantly under the action u of the symmetry group [17–21]. These symmetric tensors take block-spare form, with original dense virtual spaces V of bond dimension D split into a direct sum of blocks $V = \oplus_r V_r$ with dimensions $\{D_1, \ldots, D_r\}$ each associated with irreducible representation r of the symmetry group considered. Block sparsity substantially reduces computational complexity, permitting large D simulations, in particular, for (i)PEPS algorithms.

Here, we introduce the *Yet Another Symmetric Tensor Network* (YASTN) library [22]. YASTN is an open-source Python library with Abelian-symmetric tensor as a basic type and associated linear algebra operations on such tensors. The implementation enables *automatic differentiation* (AD) via appropriate dense linear algebra backends, allowing convenient variational optimization of TNs. This is particularly important for iPEPS [23–25], where no alternative direct energy minimization algorithms are known. This is in contrast to (i)MPS where the DMRG provides efficient and robust optimization. YASTN thus joins a continually growing collection of tensor network software with various degree of support for symmetries and automatic differentiation, such as ITensor [26], TenPy [27], Block2 [28], Quantum TEA [29], TensorNetwork [30], Cytnx [31], TeNes [32], TensorKit [33], Qspace [34], peps-torch [35], ad-peps [36], variPEPS [37], PEPSKit [38], TenNetLib [39].

In the following sections, we outline the design principles of YASTN and present a set of benchmarks demonstrating the computational speed-up from Abelian symmetries. We focus on variational optimization of iPEPS for SU(2)-symmetric spin-$\frac{1}{2}$ model, SU(3)-symmetric model, and observables of a Hubbard model at finite temperature simulated via imaginary-time evolution.

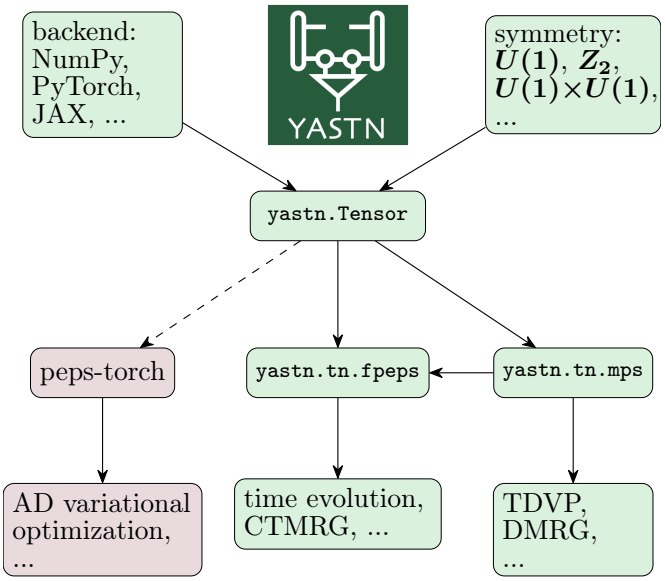

Figure 2: **Schematic design of `yastn` package.** The core element of the package is `yastn.Tensor`, which implements block-sparse tensor structure corresponding to a given Abelian symmetry on top of a dense linear algebra backend, such as NumPy [40] or PyTorch [41]. More complex tensor networks, built on top of symmetric tensor, include standard but versatile MPS toolbox and 2D PEPS implementations to simulate the time evolution or ground-state variational optimization. The latter can benefit from automatic differentiation supported by some underlying backends, such as PyTorch.

2 Design principles

In this section, we give an overview of the structure of YASTN, presented in Fig. 2, and comment on some aspects of implementation. The basic building block of the library is the `yastn.Tensor` which is defined by the symmetry structure and the backend. The symmetry structure determines a set of allowed blocks and how to manipulate them when performing tensor algebra. The backend handles the execution of dense linear algebra operations and storage of tensor elements. These two are independent of each other. Symmetric tensors are used to construct TN ansätze, such as MPS and PEPS, and to finally define high-level algorithms that are applied to specific TN. For a detailed description of the library and all its functionalities, see the documentation under Ref. [22].

2.1 Abelian-symmetric tensor

Tensors are multilinear maps from products of several vector spaces

$$T: \quad V^1 \otimes V^2 \otimes V^3 \otimes \cdots \to \mathbb{C}\,, \tag{1}$$

where V^i is a vector space and $\otimes$ is a tensor product. In a quantum-mechanical context, we work with Hilbert spaces $\mathcal{H}$ and their duals $\mathcal{H}^*$. By choosing some bases in each of these spaces the tensors can be written out in components

$$T = \sum_{abc\cdots ijk\cdots} T^{abc\cdots}_{ijk\cdots} |i\rangle|j\rangle|k\rangle \cdots \langle a|\langle b|\langle c| \cdots\,, \tag{2}$$

where $i, j, k, \ldots$ are indices of bases in $\mathcal{H}$ spaces, $a, b, c, \ldots$ in $\mathcal{H}^*$ spaces, and $T^{abc\cdots}_{ijk\cdots}$ is the corresponding tensor element. The action of the element g of Abelian group G on tensor

T can be represented in a proper basis by diagonal matrices $U^i(g)$ transforming the tensor elements

$$(g \circ T)^{ab\cdots}_{ij\ldots} = \sum_{a'b'\cdots i'j'\ldots} T^{a'b'\cdots}_{i'j'\ldots} [U^1(g)]^{i'}_i [U^2(g)]^{j'}_j \cdots [U^m(g)^*]^a_{a'} [U^{m+1}(g)^*]^b_{b'} \cdots . \tag{3}$$

The matrix elements of $U^i(g)$ are

$$[U^i(g)]^j_{j'} = \delta_{jj'} \exp(-i\theta_g t^{[i]}_j), \tag{4}$$

forming a diagonal matrix of complex phases defined by integer charges $t^{[i]}_j$, with angle $\theta_g \in [0, 2\pi)$ which depends on $g \in G$, and $\delta_{jj'}$ being Kronecker delta. Therefore, under the action of $g \in G$, each tensor element simply acquires a phase given by the sum of charges

$$(g \circ T)^{ab\cdots}_{ij\ldots} = T^{ab\cdots}_{ij\ldots} \exp[-i\theta_g (t^{[1]}_i + t^{[2]}_j + \ldots - t^{[m]}_a - t^{[m+1]}_b - \ldots)]. \tag{5}$$

This form of the transformation gives a simple selection rule, a charge conservation, on the elements of symmetric tensors

$$t^{[1]}_i + t^{[2]}_j + \ldots - t^{[m]}_a - t^{[m+1]}_b - \ldots = n. \tag{6}$$

The charge of each non-zero element $T^{ab\cdots}_{ij\ldots}$ of a symmetric tensor must be n. In the case of $n = 0$, the tensor is invariant (unchanged) under the action of the symmetry. Otherwise, it transforms covariantly as all its elements are altered by the same complex phase $\exp(-i\theta_g n)$. The charges $t^{[i]}_j$ and n and the precise form of their addition depend on the Abelian group. For elementary Abelian groups such as the cyclic group C_N (often denoted $\mathbb{Z}_N$) the individual charges $t^{[i]}_j$ are elements of $\mathbb{Z}_N$, since in this case the allowed angles θ_g are integer multiples of $2\pi/N$, while for the circle group $U(1)$ the charges are elements of $\mathbb{Z}$. For direct products of Abelian groups, charges become tuples $\vec{t}^{[i]}_j$ in the corresponding product of $\mathbb{Z}_N$'s and $\mathbb{Z}$'s.

By ordering the basis elements in each Hilbert space by their charge, the tensor $T^{ab\cdots}_{ij\ldots}$ naturally attains a block-sparse structure, which is central to the computational advantage offered by Abelian-symmetric tensor network algorithms.

At the core of YASTN is the implementation of the symmetric tensor `yastn.Tensor`, as outlined in Refs. [17–19]. It is defined jointly by symmetry (block) structure data and tensor elements of existing blocks. First, we define a vector space with a charge structure, a `yastn.Leg`, determined by a signature $s = \pm 1$ (distinguishing between $\mathcal{H}$ and dual $\mathcal{H}^*$), its charge sectors $\vec{t}$, and their corresponding dimensions $\vec{D}$, now decorated with arrows to denote tuples,

$$V(s, \vec{t}, \vec{D}) = \oplus_{\rho \in \vec{t}} \mathbb{C}^{\vec{D}_\rho}, \tag{7}$$

where ρ now enumerates different charges instead of basis.[1] This space is a direct sum of simple spaces $\mathbb{C}^{\vec{D}_\rho}$, dubbed charge sectors. The effective dimension of such space is the sum of dimensions of individual charge sectors

$$D = \sum_\rho \vec{D}_\rho . \tag{8}$$

In the remainder of the text, we will refer to D as the bond dimension when discussing scaling of computational complexity or memory requirements of TN algorithms with symmetric

[1] The conjugation of the leg, i.e., mapping space $\mathcal{H}$ to its dual space $\mathcal{H}^*$, is equivalent to flip of the signature and complex conjugation of elements.

tensors. The Abelian symmetric tensor of rank-N is specified by the product of N such vector spaces

$$T : \otimes_{i=1}^{N} V(s^{[i]}, \vec{t}^{[i]}, \vec{D}^{[i]}) \to \mathbb{C}\,. \tag{9}$$

The following is an example creating a random $U(1)$-symmetric tensor with total charge $n = 0$ with specified legs:[2]

```
import yastn
from yastn.backend import backend_np
from yastn.sym import sym_U1

u1 = yastn.make_config(sym=sym_U1, backend=backend_np)
l = yastn.Leg(u1, s=1, t=(-1,1), D=(1,1))
lc = l.conj()
H = yastn.rand(u1, legs=[l,l,lc,lc], n=0)
```

which is, for example, compatible with a Hamiltonian $H = \vec{S}_1 \cdot \vec{S}_2$ of two spin-$\frac{1}{2}$ degrees of freedom. The configuration created by `yastn.make_config` specifies the symmetry, e.g., `yastn.sym.sym_U1` for the $U(1)$ in the example, and dense linear algebra backend (see below), e.g., `yastn.backend.backend_np` for NumPy.

The covariant transformation property of T is imposed by the charge conservation of non-zero blocks. Any block of tensor T can be identified by selecting a charge sector $\rho_i \in \vec{t}^{[i]}$ on each of the legs, i.e., a N-tuple of charges $(\rho_0, \rho_1, \ldots, \rho_N)$. All non-zero blocks must satisfy

$$\sum_{i=1}^{N} s^{[i]} \rho_i = n\,, \tag{10}$$

which is the block-sparse version of the element-wise charge conservation rule of Eq. (6).

Finally, we remark on the storage of tensor elements. In YASTN, the block data is initialized *lazily*. The storage is allocated only for the blocks that have been assigned a non-zero value, i.e., blocks allowed by the charge conservation but not assigned any value are not stored. All allocated blocks are serialized together in a 1D array of dense linear algebra backend.

2.2 Fusion and contractions

The key operations on symmetric tensors, necessary for manipulating tensor networks, are tensor reshape and permutation, commonly dubbed *fusion* in this context, and tensor contractions. Fusion resolves the tensor product of several spaces as a new space, i.e., the fusion of legs into a new leg. Unlike reshaping the dense tensor, the shape cannot be freely chosen. Instead, it is determined by the structure of the fused spaces. In particular, fusion orders and accumulates tensor products of charge sectors on selected legs into new charge sectors under the joint leg

$$V(s^{[i]}, \vec{t}^{[i]}, \vec{D}^{[i]}) \otimes V(s^{[j]}, \vec{t}^{[j]}, \vec{D}^{[j]}) \to V(s^{[r]}, \vec{t}^{[r]}, \vec{D}^{[r]})\,, \tag{11}$$

with new charge sectors $\vec{t}^{[r]}$ given by the unique combinations of charges $\vec{t}^{[i]} \otimes \vec{t}^{[j]}$

$$\vec{t}^{[r]} := \{\nu = s^{[r]}(s^{[i]}\rho + s^{[j]}\rho') : \rho \in \vec{t}^{[i]}, \rho' \in \vec{t}^{[j]}\}\,. \tag{12}$$

The dimension of new charge sector $\nu \in \vec{t}^{[r]}$ is[3]

$$\vec{D}^{[r]}_{\nu} = \sum_{\substack{\rho,\rho' \\ \nu = s^{[r]}(s^{[i]}\rho + s^{[j]}\rho')}} \vec{D}^{[i]}_{\rho} \vec{D}^{[j]}_{\rho'}\,. \tag{13}$$

[2] One can always define tensors with an extra dummy leg $V(-1,(n,),(1,))$, having a single charge sector of a unit dimension, making it invariant under symmetry transformations.

[3] Following a *lazy* approach adopted in YASTN, a new fused leg contains only charges for which some non-zero tensor block exists. As such, fusion in YASTN is always done in the context of particular tensor.

The fusion and un-fusion calls are demonstrated below on previously constructed rank-4 tensor H, first fusing pairs of legs resulting in a matrix form

```
H_mat = H.fuse_legs(axes=((0,1), (2,3)))
H = H_mat.unfuse_legs(axes=(0,1))
```

where the `axes` argument of `fuse_legs` is a tuple that describes how the legs should be permuted and fused in the resulting tensor. Following `NumPy` convention, each leg of the original tensor is labeled by an integer according to its position starting at 0. The groups of legs to be fused are specified by nested tuples in `axes`. Several alternative ways of fusing H are given below

```
H_long_mat = H.fuse_legs(axes=(0, (1,2,3)))
H_thin_mat = H.fuse_legs(axes=((0,1,2), 3))
H_thin_mat_transposed = H.fuse_legs(axes=(3, (0,1,2)))
H_rank3_permuted = H.fuse_legs(axes=(0, (3,2), 1))
```

When fusing legs, YASTN first calculates the structure of the resulting tensor with fused leg(s), i.e., the tuple $(s^{[r]}, \vec{t}^{[r]}, \vec{D}^{[r]})$ and a set of dense linear algebra jobs (permutes, reshapes, and copies) to be executed by the backend to populate the new 1D storage array with serialized blocks. The resulting tensor records the original structure, and hence, the fusion can be reverted by `unfuse_legs`.

```
H_1fusionlevel = H.fuse_legs(axes=(0,(1,2),3))
H_2fusionlevels = H_1fusionlevel.fuse_legs(axes=(0,(1,2))
H = H_2fusionlevels.unfuse_legs(axes=1).unfuse_legs(axes=1)
```

The tensor contraction of symmetric tensors is realized by a commonly adopted workflow. First, the tensors are fused into block-sparse matrices,[4] then matrix-multiplied along the contracted legs, and finally unfused to obtain the desired form:

$$\begin{aligned}\sum_{x_0x_1\cdots} A_{i_0i_1x_0i_2x_1i_3x_2\cdots}B_{x_1j_0x_0x_2j_1\cdots} &\xrightarrow{\text{permute}} \sum_{\{x\}} A_{\{i\}\cup\{x\}}B_{\{x\}\cup\{j\}} \xrightarrow{\text{reshape}} \sum_X A_{IX}B_{XJ} \\ &\overset{\text{multiply}}{=} C_{IJ} \xrightarrow{\text{unfuse}} C_{\{i\}\{j\}} = C_{i_0i_1i_2\cdots j_0j_1j_2\cdots},\end{aligned} \tag{14}$$

where $\{x\}$ is a set of common legs that become fused into single leg X, and original uncontracted legs $\{i\}$ and $\{j\}$ are restored from the fused I and J to obtain the final tensor. We again adhere to `NumPy` convention, where `axes` argument of `tensordot` contains two tuples. The first tuple specifies legs to contract on the first tensor operand A, and the second tuple specifies legs to contract on the second tensor operand B. The legs in the same position in the two tuples are then contracted, i.e. `axes=((2,4,6,...),(2,0,3,...))` for the above example. The relative order of the remaining uncontracted legs of the first operand $\{i\}$ and the second operand $\{j\}$ is unchanged. The legs of the resulting tensor C are ordered exactly as the uncontracted legs of the two tensor operands. First, we have the uncontracted legs $\{i\}$ of the first operand followed by the uncontracted legs of the second operand $\{j\}$. Alternative contraction workflows, which avoid the fusion to the block-sparse matrix form, will be introduced in the future version.

Here, we first show an example call for pairwise tensor contraction, and second, an equivalent given in terms of explicit operations

```
H2 = yastn.tensordot(H, H, axes=((2,3), (0,1)))
H2 = (H.fuse_legs(axes=((0,1), (2,3))) @
      H.fuse_legs(axes=((0,1), (2,3)))).unfuse_legs(axes=(0,1))
```

[4] For valid contraction, the structures of the contracted legs must be compatible, including the origins of any fused leg. YASTN automatically resolves a situation when some charges in the fusion history of a to-be-contracted fused leg are missing but are present in its contraction partner. This is done by utilizing information on the tensor's fusion history stored in each `yastn.Tensor` object.

For both fusion and multiplication, the YASTN first precomputes what the non-zero blocks are, so the backend performs only the relevant operations. Contractions of more general tensor diagrams are supported through convenience functions, such as `einsum` and `ncon` [42] (in our case differing only by syntax), which are based on the elementary operations discussed above.

2.3 Tensor network algorithms

The symmetric tensor serves as the basis for higher-level tensor network structures and algorithms. Here, YASTN comes with MPS and PEPS modules. The MPS module supports finite-size MPS with the implementation of a range of standard algorithms, including DMRG for ground-state optimization, TDVP [43,44] for time evolution,[5] and overlap maximization [46] against a general target, i.e., MPS, sum of MPSs, or sum of MPO-MPS products. This is complemented by a versatile high-level (Hamiltonian) MPO generator. The MPS module provides subroutines for some PEPS methods; e.g., it was utilized in Ref. [47] for boundary MPS contraction and long-range correlations calculation in a finite PEPS defined on a cylinder. At the same time, it is a versatile computational toolbox on its own. For example, it has been used in simulations of Lindbladian dynamics in the context of fermionic quantum transport [48], where the $U(1)$ symmetry reflects a lack of correlations between different particle-number sectors of a density matrix.

The PEPS module features the implementation of fermionic PEPS, dubbed fPEPS (which also allows simulations of systems without fermionic statistics). It covers both finite PEPS defined on a square lattice and its infinite versions for translationally invariant (over a unit cell) systems in the thermodynamic limit. It supports a range of time evolution algorithms, starting with *neighborhood tensor update* (NTU) scheme [49, 50], its refinement to a family of larger environmental clusters [47], ending on a full-update type of schemes [16, 51]. It is viable for imaginary-time evolution, e.g., in the context of finite-temperature simulation of density matrix purification [52] or minimally-entangled typical thermal states [53], and real-time simulations, e.g., of pure-state quench-dynamics in disordered spin systems [47].

3 Examples

To demonstrate the use and versatility of YASTN, we present three end-to-end numerical examples[6] centered on iPEPS. We provide benchmarks of MPS-type contractions and algorithms with comparisons to ITensor and TeNPy in a dedicated repository [54].

We show computational speed-up and reduced memory footprint obtained with YASTN by utilizing Abelian symmetries for the following examples:

1. Sec. 3.1: variational optimization of iPEPS with $D \leq 8$ for antiferromagnetic spin-$\frac{1}{2}$ model on a square lattice using $U(1)$ symmetry,

2. Sec. 3.2: variational optimization of iPEPS with $D \leq 13$ for SU(3) model on Kagome lattice using $U(1)\times U(1)$ symmetry,

3. Sec. 3.3: observables of thermal iPEPS of Hubbard model at finite temperature using $\mathbb{Z}_2$, $U(1)$, and $U(1)\times U(1)$ symmetry, with D up to 36 for the latter.

[5] In TDVP, we employ adaptive Krylov subspace exponential integrator of Ref. [45].

[6] Examples 1 and 2 were run on a single Intel® Xeon® Silver 4110 processor. Example 3 was run on a single Intel® Xeon® Gold 6416H processor.

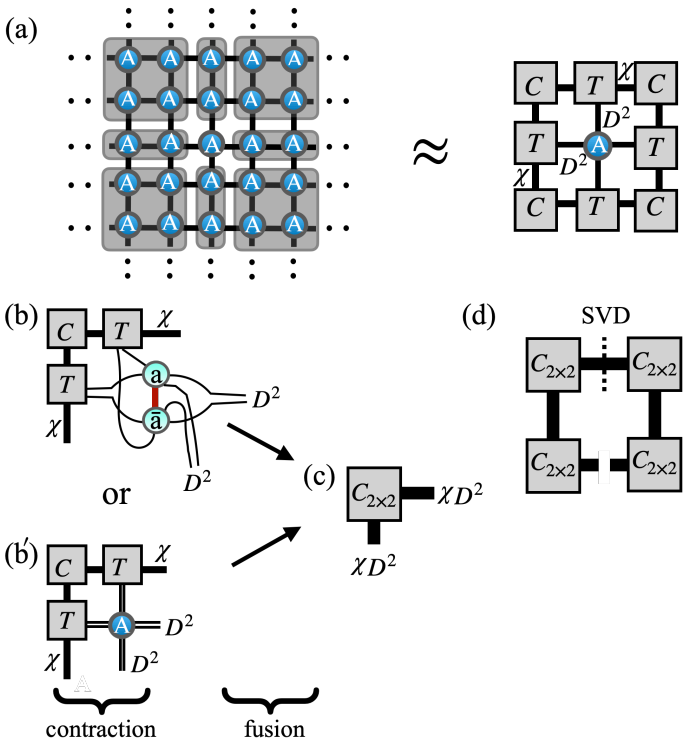

Figure 3: **Corner transfer matrix iteration**. In (a), CTM approximates parts of an infinite tensor network by a set of finite environment tensors characterized by environment bond dimension χ. We depict elements of the CTM algorithm step (iteration) that dominate the computational effort shown in Figs. 4–6. Panels (b), (b'), and (c) depict the construction of an enlarged corner combining CTM environment tensors (rectangular) with PEPS tensors (circle). Panel (d) shows an SVD decomposition of a product of four enlarged corners that is then used to construct the CTM projections from enlarged virtual spaces.

In Sec. 3.1 and Sec. 3.2 we optimize iPEPS for SU(2) model on square and SU(3) model on Kagome lattices, respectively. First, given an iPEPS generated by a set of tensors $\vec{a} = \{a, b, \ldots\}$, we compute an approximate environment tensors $\vec{E}(\vec{a})$ (specified below) with the precision governed by the environment dimension χ. Then, the environment $\vec{E}$ and the tensors $\vec{a}$ are combined to evaluate the energy per site e of the Hamiltonian. Finally, the reverse mode of AD (i.e., backpropagation) is invoked to calculate the gradient $\partial e/\partial \vec{a}$. The most computationally intensive stage is the construction of environments, scaling as the cube of $D^2\chi$, which assuming the necessary $\chi \propto D^2$ gives the overall complexity $\mathcal{O}(D^{12})$, where D is the bond dimension of the iPEPS tensors. We use iPEPS optimization implemented in peps-torch [35], here operating on the YASTN symmetric tensors.

For the examples presented, we employ the *corner transfer matrix* (CTM) algorithm [55–57] to compute the environments. CTM approximates environments $\vec{E} = \{C, T\}$ of iPEPS by a set of $\chi \times \chi$ corner matrices C and $\chi \times D^2 \times \chi$ transfer tensors T, as shown in Fig. 3(a). Alternatively, one can use the boundary MPS methods [16, 46, 58]. The computational complexity of CTM arises from two sources (see Fig. 3), tensor contractions, and *singular value decomposition* (SVD) when computing low-rank approximations, both scaling as $\mathcal{O}(D^{12})$. In practice, for simulations without symmetries, the SVD gives a substantially greater contribution due to a higher scaling prefactor and a poor speed-up offered by multithreading or GPU acceleration compared to tensor contractions. However, for symmetric iPEPS the situation becomes more nuanced, as we demonstrate in the following examples.

In Sec. 3.3, the purification techniques are applied to compute thermal expectation values for the Fermi-Hubbard model on a 2D square lattice. To effectively transform the thermal density matrix into a purified wave function, we use the ancilla trick and perform imaginary time evolution to reach the target temperature. We adopt the NTU algorithm to optimize the time evolution, with computational cost scaling of $\mathcal{O}(D^8)$ dominated by tensor contractions. We focus on the final calculation of the expectation values using CTM with $\chi = 5D$, translating to $\mathcal{O}(D^9)$ scaling. We quantify the sources of advantage offered by incorporating various symmetries.

3.1 Heisenberg antiferromagnet with anisotropy

We revisit the Heisenberg model with anisotropy in the couplings describing a system of coupled spin-$\frac{1}{2}$ ladders

$$\mathcal{H} = J \sum_R \mathbf{S}_R \cdot \mathbf{S}_{R+\hat{x}} + \sum_R J_R \mathbf{S}_R \cdot \mathbf{S}_{R+\hat{y}}\,, \tag{15}$$

where $\mathbf{S}_R = (S^x_R, S^y_R, S^z_R)$ is the $S = \frac{1}{2}$ operator at site $R = (x, y)$ on a square lattice spanned by the $\hat{x}$ and $\hat{y}$ unit vectors. The coupling $J_R = J$ for the odd position and $J_R = \alpha J$ for the even position along the y axis, respectively. For $\alpha = 1$, the Hamiltonian in Eq. (15) realizes the Heisenberg model on the square lattice, and for $\alpha = 0$ it corresponds to a system of decoupled two-legged ladders. The model was previously addressed with iPEPS in Ref. [59]. We adopt the same description that uses 2×2 unit cell with four non-equivalent tensors $\vec{a} = \{a, b, c, d\}$.[7]

The ground states possess $U(1)$ symmetry corresponding to the conservation of the S^z component. The symmetry can be exploited by utilizing $U(1)$-symmetric iPEPS. The results, summarized in Fig. 4, show a rapidly growing computational advantage of symmetric iPEPS for bond dimensions $D > 4$. While at $D = 4$ the overhead due to block-sparse logic is still significant, at the largest bond dimension considered, $D = 8$, a 30-fold speed-up is observed.

In practical terms, the convergence of the CTM towards the desired precision, here measured by the error on the energy per site becoming lower than $\epsilon < 10^{-8}$, typically requires $\mathcal{O}(10)$ iterations; thus without $U(1)$ symmetry a single optimization step would already take hours. Details of the block-sparse structure and its impact on the CTM are visualized in Fig. 4(b,c). At the largest bond dimension, $D = 8$, the fusion of the enlarged corner into a block-diagonal matrix requires processing of approximately $\mathcal{O}(100)$ blocks by performing dense permutes, reshapes, and copies, with the largest block having $\mathcal{O}(10^5)$ elements. The cost of subsequent SVD is dominated by the largest block(s) of fused enlarged corner, which are $L \times L$ matrices with $L = 1,500 \sim 2,000$. As a result, computational time contributions are shared between SVD and tensor contractions with fusions roughly as 3:2, with SVD being the dominant factor.

3.2 SU(3) model on Kagome lattice

We consider an SU(3)-symmetric model on Kagome lattice, analyzed recently in Ref. [60], where each site holds a single degree of freedom from the fundamental representation **3** of the SU(3) group spanned by three states $\{|\alpha\rangle, |\beta\rangle, |\gamma\rangle\}$. The Hamiltonian reads

$$H = J \sum_{\langle i,j \rangle} P_{ij} + \sum_{\triangle ijk} (K P_{ijk} + h.c.)\,, \tag{16}$$

where P_{ij} is a permutation, $P_{ij}|\alpha\rangle_i|\beta\rangle_j = |\beta\rangle_i|\alpha\rangle_j$, of local states on nearest-neighbour bonds. P_{ijk} is a clockwise permutation of local states in nearest-neighbor triangles such that

[7] A more efficient description might generate all tensors in 2×2 unit cell from a single parent tensor a by use of unitary $-i\sigma^y$ acting on physical index and/or permutation of virtual indices generated by the reflection along the x-axis. However, such a parameterization would not change the complexity of the CTM.

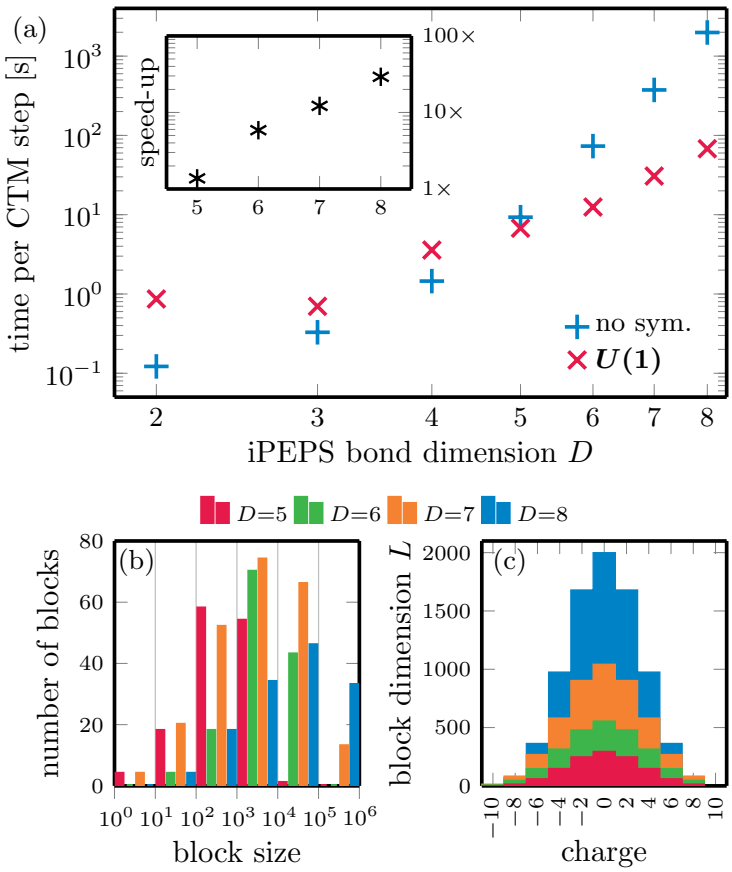

Figure 4: **Use case 1: optimization of $U(1)$-symmetric iPEPS for model of spin-$\frac{1}{2}$ coupled ladders.** In (a), we show the scaling of the wall time per CTM step (in seconds) for the entire gradient optimization step of iPEPS. The bond dimension of the environment is $\chi = 2D^2$. The inset shows the relative speed-up compared to an implementation without symmetry. In (b), we show a distribution of blocks of an enlarged corner by their size (number of elements) before fusion to a block-diagonal matrix as shown in Fig. 3(c). In (c), we show the sizes of $L \times L$ blocks after the fusion.

$P_{ijk}|\alpha\rangle_i|\beta\rangle_j|\gamma\rangle_k = |\gamma\rangle_i|\alpha\rangle_j|\beta\rangle_k$, with fixed choice of orientation of triangles, and J and K are real and complex-valued couplings, respectively.

In this section, we demonstrate an advantage of $U(1)\times U(1)$-symmetric iPEPS, utilizing maximal Abelian subgroup of SU(3), over implementation without symmetries.[8] To compute CTM environments on the Kagome lattice, we coarse-grain three sites on each down-pointing triangle into a single tensor, resulting in an effective square lattice. The local Hilbert space dimension thus grows to $3^3 = 27$, making the optimizations memory intensive. In Fig. 5, we demonstrate the dramatic speed-up achieved by utilizing $U(1)\times U(1)$ symmetry. For $D = 9$ iPEPS, a single gradient step is already accelerated by more than a factor of 100. For larger bond dimensions, the simulations without symmetries become prohibitive, and we estimate the speed-up based on extrapolation of the scaling at smaller bond dimensions.

In contrast to the example in Sec. 3.1 utilizing the $U(1)$-symmetry, the speed-up in this case is not monotonic in D. This happens because of the varying structure of the iPEPS tensors, i.e., the allowed symmetry sectors and their sizes. In Fig. 5(b,c) we illustrate the block structure of

[8] In this example, besides iPEPS, one can also use different ways to construct two-dimensional TN Ansatz on Kagome lattice, i.e., infinite projected simplex states (iPESS), however, the computational complexity $\mathcal{O}(D^{12})$, attributable to CTM, remains unchanged.

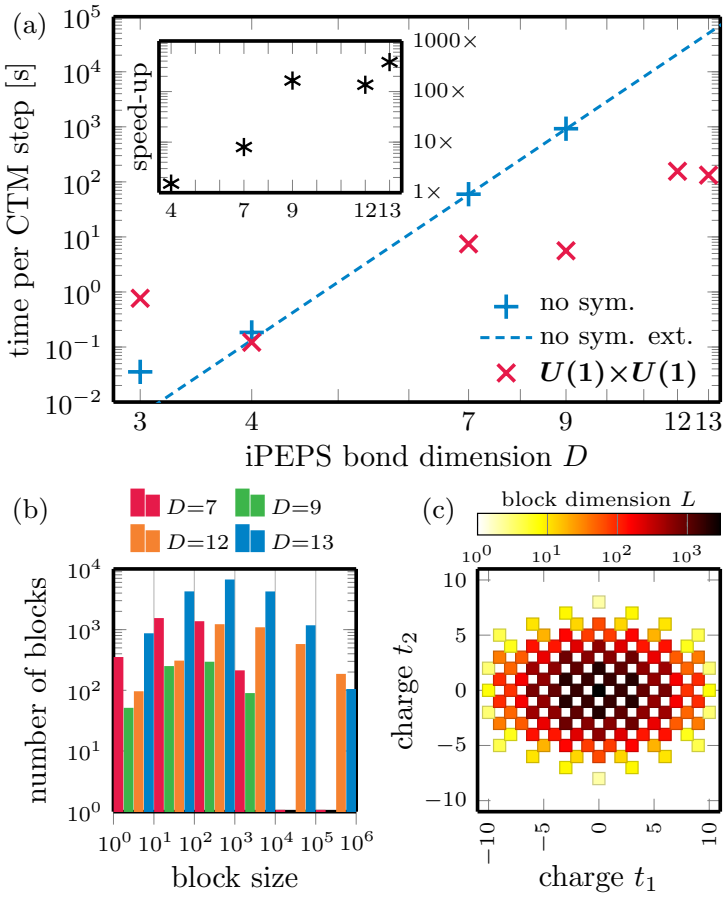

Figure 5: **Use case 2: optimization of $U(1)\times U(1)$-symmetric iPEPS for SU(3) Kagome model.** In (a), scaling of the wall time per CTM step (in seconds) for the entire gradient optimization step of iPEPS. The bond dimension of the environment is $\chi = D^2$. The inset shows the relative speed-up compared to an implementation without symmetry, with $D = 12$ and 13 simulation wall times estimated from the extrapolation (blue dashed line). In (b), a distribution of blocks of an enlarged corner by their size (number of elements) before the fusion to a matrix. In (c), $L \times L$ blocks of the block-diagonal enlarged corner after fusion. We plot them as a heatmap, with different $U(1)$ charges on x- and y-axes.

enlarged corners before and after the fusion to a block-diagonal matrix. Generally, for larger groups, the number of blocks of enlarged corners before fusion is substantially higher. Even at $D = 7$, the total number of blocks is already more than 3,000 whereas for $U(1)$-symmetric enlarged corner in Sec. 3.1 it was below 300. For $D = 13$ Ansatz, the fusion of the enlarged corner into a block-diagonal matrix requires processing of more than 16,000 blocks, with more than half of them being small in size, having roughly $\mathcal{O}(10^3)$ elements or less. This granularity defines the bottleneck of the simulations. For $D = 12$ and $D = 13$ the ratios between the computational time of SVD and contractions including fusion to block-diagonal enlarged corners are 3:4 and 1:10, respectively. Overall, the $U(1)\times U(1)$ simulations become dominated by fusion, with SVD being sub-leading. The precise speed-up depends on the sizes of the blocks, such as here, where $D = 12$ has a slightly higher proportion of largest blocks compared to $D = 13$.

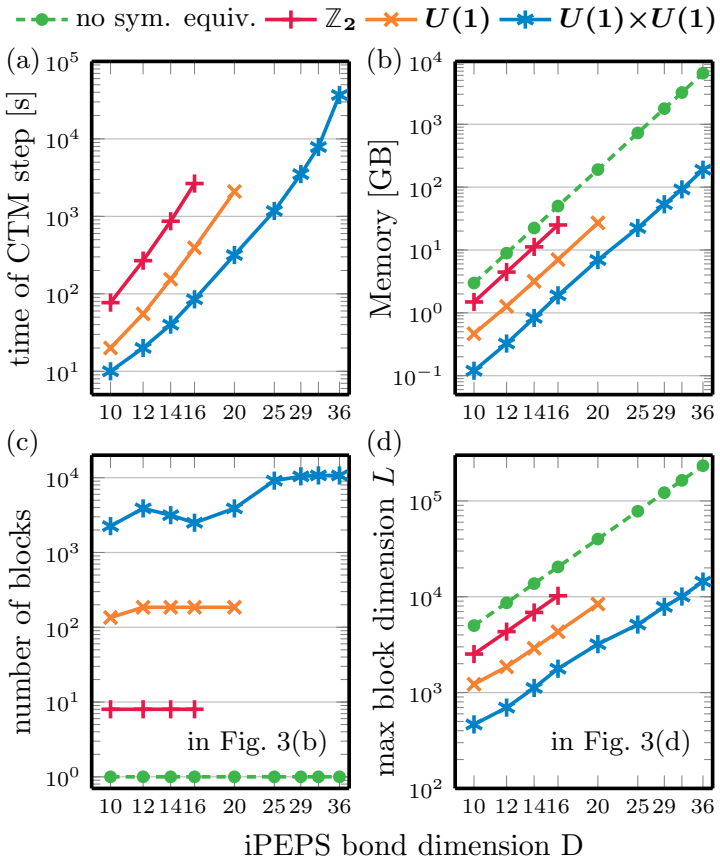

Figure 6: **Use case 3: expectation values from CTM environments in fermionic iPEPS for finite-temperature Hubbard model.** The bond dimension of the environment is $\chi = 5D$, which is sufficient here to converge the expectation values. We show, in (a), the wall time per CTM step and, in (b), the data size (memory requirement) of the biggest intermediate tensor appearing while building enlarged CTM corners in Fig. 3(b). Panel (c) shows the number of blocks in an enlarged corner before the fusion in Fig. 3(c), and panel (d) is the size of the largest $L \times L$ block for SVD in Fig. 3(d).

3.3 2D Fermi-Hubbard model on a square lattice

We consider a two-dimensional Fermi-Hubbard model (FHM) with on-site repulsion as studied in Ref. [52]. The Hamiltonian reads

$$H = -t \sum_{\langle i,j\rangle,\sigma} \left(c^\dagger_{i\sigma} c_{j\sigma} + c^\dagger_{j\sigma} c_{i\sigma}\right) + U \sum_i \left(n_{i\uparrow} - \frac{1}{2}\right)\left(n_{i\downarrow} - \frac{1}{2}\right) - \mu \sum_i n_i \,, \tag{17}$$

where $c^\dagger_{i\sigma}$ and $c_{i\sigma}$ are the creation and annihilation operators for an electron with spin σ at site i, $n_{i\sigma} = c^\dagger_{i\sigma} c_{i\sigma}$ is a corresponding number operator, t is the hopping amplitude, U is the on-site Coulomb repulsion strength, and μ is the chemical potential.

The iPEPS Ansatz employs a checkerboard lattice with a 2-site unit cell. The thermal state for the inverse temperature β is obtained by evolving the infinite-temperature purification $|\psi(0)\rangle$ under the non-unitary propagator $U(\beta) = e^{-\beta H/2}$. The initial purification is a product of maximally entangled pairs between each physical site and its corresponding ancilla, $|\psi(0)\rangle = \prod_j \prod_{m=\uparrow,\downarrow} \frac{1}{\sqrt{2}} \sum_{s_{m_j}=a_{m_j}=0,1} |s_{m_j} a_{m_j}\rangle$, translating to local Hilbert spaces of dimension

$4^2 = 16$. In the following examples, we run the imaginary time evolution employing the NTU scheme targeting $\beta = 2$.

In YASTN, the fermionic exchange order is implemented following the scheme of Refs. [61, 62] by projecting the lattice Ansatz onto a plane, imposing a canonical fermionic order, and applying swap gates on every line crossing; see Fig. 3(b). The swap gate introduces sign changes for blocks with charges of odd parity on both swapped legs. This makes $\mathbb{Z}_2$ the minimal symmetry needed for fermionic system simulations. The Hamiltonian in Eq. (17) preserves the number of particles per spin direction, which allows us to implement the model under $U(1)\times U(1)$ symmetry as the highest Abelian symmetry.

The expectation values of the thermal state at $\beta = 2$ are calculated using the CTM. Fig. 6 shows an advantage of symmetric tensors by comparing $\mathbb{Z}_2$ (parity; minimal requirement for fermionic statistics), $U(1)$ (total charge conservation) and $U(1)\times U(1)$ (total charge conservation for each spin) symmetries. We also show equivalent values for the corresponding tensors with no symmetry. We choose the environmental bond dimension $\chi = 5D$, which is sufficient to converge the expectation values in this example.

Fig. 6(a) presents the computational wall time for a CTM step as a function of the iPEPS bond dimension for all the symmetries tested, with the systematic improvement offered by higher symmetries. Fig. 6(b) highlights the memory usage bottleneck, showcasing the size of the largest object formed during the CTM iteration, i.e., an intermediate step of the contraction in Fig. 3(b); this tensor has to be later fused, and there are other tensors in the memory, so the memory peak is roughly two times higher. This illustrates that those simulations are ultimately memory-limited. Employing $U(1)\times U(1)$ symmetry offers a systematic 30-fold memory gain compared to tensors without symmetries, which ultimately allows for successful simulations up to $D = 36$.

Following the previous examples, in Fig. 6(c), we present the number of blocks processed during the fusion that form enlarged corners in Fig. 3(c). A particular challenge for $U(1)\times U(1)$ case is the number of blocks that can exceed $10,000$. However, SVD is a dominant factor that takes at least half the simulation time for $D \geq 25$ in our numerical experiments. In Fig. 6(d), we show the (sectorial) bond dimension of the largest block decomposed in Fig. 3(d), which is $\mathcal{O}(10^4)$ for each employed symmetry. Among them, the $U(1)\times U(1)$-symmetry offers here a 15-fold improvement for given D as compared to a setup with no symmetries involved.

4 Conclusion and future outlook

Tensor networks are becoming increasingly popular tools for numerical treatment of quantum systems, ranging from ground-state simulations of condensed-matter systems to simulations of quantum circuits. The landscape of associated software is continuously growing. For 1D and quasi-1D geometries, well-established and mature packages offer a rich set of MPS algorithms that cover direct energy minimization, (imaginary) time evolution, and much more. For two-dimensional geometries, predominantly targeted by iPEPS, the field remains nascent.

Here, we have introduced YASTN, a Python-based TN library with a strong emphasis on simulations of two-dimensional systems by iPEPS, which is motivated by the need for both Abelian symmetries and automatic differentiation. By design, the dense linear algebra and the AD engine are provided by different backends, allowing for implementation-specific optimizations. YASTN, with its rich set of examples covering ground-state simulations of various 2D spin lattice models (through peps-torch) and finite-temperature simulations of the 2D Hubbard model, thus joins similar efforts by VariPEPS [37], PEPSkit [38], ad-peps [36], and peps-torch [35] together lowering the barrier for entry.

The wide separation between the high-level description of iPEPS algorithms and their fast execution, optimized down to low-level dense linear algebra, especially for symmetric tensors, remains a challenge. Unlike MPS simulations, iPEPS contraction algorithms for computing environments and evaluation of observables involve a more diverse set of tensor contractions, varying in ranks and block sparsity patterns. Furthermore, flexible deployment and the ability to leverage heterogeneous clusters, which account for the iPEPS-specific block sparsity, is vital to address the sharp $\mathcal{O}(D^8 \sim D^{12})$ (albeit polynomial) scaling with the bond dimension, which is the key resource governing the precision of iPEPS. Thus, these challenges call for further development.

Acknowledgments

We thank Philippe Corboz, Piotr Czarnik, Jacek Dziarmaga, Boris Ponsioen, Yintai Zhang and Yi Xu for inspiring discussions that were invaluable in the development of this package.

Funding information We acknowledge the funding by the National Science Center (NCN), Poland, under projects 2019/35/B/ST3/01028 (A.S.), 2020/38/E/ST3/00150 (G.W.), and project 2021/03/Y/ST2/00184 within the QuantERA II Programme that has received funding from the European Union Horizon 2020 research and innovation programme under Grant Agreement No. 101017733 (M.M.R.). J.H. acknowledges support from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 101001604) and from the Swiss National Science Foundation through a Consolidator Grant (iTQC, TMCG-2_213805).

References

[1] R. Orús, *A practical introduction to tensor networks: Matrix product states and projected entangled pair states*, Ann. Phys. **349**, 117 (2014), doi:10.1016/j.aop.2014.06.013.

[2] R. Orús, *Tensor networks for complex quantum systems*, Nat. Rev. Phys. **1**, 538 (2019), doi:10.1038/s42254-019-0086-7.

[3] J. I. Cirac, D. Pérez-García, N. Schuch and F. Verstraete, *Matrix product states and projected entangled pair states: Concepts, symmetries, theorems*, Rev. Mod. Phys. **93**, 045003 (2021), doi:10.1103/RevModPhys.93.045003.

[4] S. R. White, *Density matrix formulation for quantum renormalization groups*, Phys. Rev. Lett. **69**, 2863 (1992), doi:10.1103/PhysRevLett.69.2863.

[5] S. R. White, *Density-matrix algorithms for quantum renormalization groups*, Phys. Rev. B **48**, 10345 (1993), doi:10.1103/PhysRevB.48.10345.

[6] S. Rommer and S. Östlund, *Class of Ansatz wave functions for one-dimensional spin systems and their relation to the density matrix renormalization group*, Phys. Rev. B **55**, 2164 (1997), doi:10.1103/PhysRevB.55.2164.

[7] J. Dukelsky, M. A. Martín-Delgado, T. Nishino and G. Sierra, *Equivalence of the variational matrix product method and the density matrix renormalization group applied to spin chains*, Europhys. Lett. **43**, 457 (1998), doi:10.1209/epl/i1998-00381-x.

[8] G. Vidal, *Efficient simulation of one-dimensional quantum many-body systems*, Phys. Rev. Lett. **93**, 040502 (2004), doi:10.1103/PhysRevLett.93.040502.

[9] N. Schuch, M. M. Wolf, F. Verstraete and J. I. Cirac, *Entropy scaling and simulability by matrix product states*, Phys. Rev. Lett. **100**, 030504 (2008), doi:10.1103/PhysRevLett.100.030504.

[10] U. Schollwöck, *The density-matrix renormalization group in the age of matrix product states*, Ann. Phys. **326**, 96 (2011), doi:10.1016/j.aop.2010.09.012.

[11] F. Verstraete and J. I. Cirac, *Renormalization algorithms for quantum-many body systems in two and higher dimensions*, (arXiv preprint) doi:10.48550/arXiv.cond-mat/0407066.

[12] F. Verstraete, M. M. Wolf, D. Perez-Garcia and J. I. Cirac, *Criticality, the area law, and the computational power of projected entangled pair states*, Phys. Rev. Lett. **96**, 220601 (2006), doi:10.1103/PhysRevLett.96.220601.

[13] R. Orús, *Exploring corner transfer matrices and corner tensors for the classical simulation of quantum lattice systems*, Phys. Rev. B **85**, 205117 (2012), doi:10.1103/PhysRevB.85.205117.

[14] P. C. G. Vlaar and P. Corboz, *Simulation of three-dimensional quantum systems with projected entangled-pair states*, Phys. Rev. B **103**, 205137 (2021), doi:10.1103/PhysRevB.103.205137.

[15] G. Vidal, *Classical simulation of infinite-size quantum lattice systems in one spatial dimension*, Phys. Rev. Lett. **98**, 070201 (2007), doi:10.1103/PhysRevLett.98.070201.

[16] J. Jordan, R. Orús, G. Vidal, F. Verstraete and J. I. Cirac, *Classical simulation of infinite-size quantum lattice systems in two spatial dimensions*, Phys. Rev. Lett. **101**, 250602 (2008), doi:10.1103/PhysRevLett.101.250602.

[17] I. P. McCulloch, *From density-matrix renormalization group to matrix product states*, J. Stat. Mech.: Theory Exp. P10014 (2007), doi:10.1088/1742-5468/2007/10/P10014.

[18] S. Singh, R. N. C. Pfeifer and G. Vidal, *Tensor network decompositions in the presence of a global symmetry*, Phys. Rev. A **82**, 050301 (2010), doi:10.1103/PhysRevA.82.050301.

[19] S. Singh, R. N. C. Pfeifer and G. Vidal, *Tensor network states and algorithms in the presence of a global U(1) symmetry*, Phys. Rev. B **83**, 115125 (2011), doi:10.1103/PhysRevB.83.115125.

[20] S. Singh and G. Vidal, *Tensor network states and algorithms in the presence of a global SU(2) symmetry*, Phys. Rev. B **86**, 195114 (2012), doi:10.1103/PhysRevB.86.195114.

[21] 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 8 (2019), doi:10.21468/SciPostPhysLectNotes.8.

[22] M. M. Rams, G. Wójtowicz, A. Sinha and J. Hasik, *YASTN: Yet another symmetric tensor network* (2024), https://github.com/yastn/yastn.

[23] P. Corboz, *Variational optimization with infinite projected entangled-pair states*, Phys. Rev. B **94**, 035133 (2016), doi:10.1103/PhysRevB.94.035133.

[24] 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), doi:10.1103/PhysRevB.94.155123.

[25] H.-J. Liao, J.-G. Liu, L. Wang and T. Xiang, *Differentiable programming tensor networks*, Phys. Rev. X **9**, 031041 (2019), doi:10.1103/PhysRevX.9.031041.

[26] M. Fishman, S. White and E. Stoudenmire, *The ITensor software library for tensor network calculations*, SciPost Phys. Codebases 4 (2022), doi:10.21468/SciPostPhysCodeb.4.

M. Fishman, S. White and E. Stoudenmire, *Codebase release 0.3 for ITensor*, SciPost Phys. Codebases 4-r0.3 (2022), doi:10.21468/SciPostPhysCodeb.4-r0.3.

[27] J. Hauschild and F. Pollmann, *Efficient numerical simulations with tensor networks: Tensor network Python (TeNPy)*, SciPost Phys. Lect. Notes 5 (2018), doi:10.21468/SciPostPhysLectNotes.5.

[28] H. Zhai et al., *Block2: A comprehensive open source framework to develop and apply state-of-the-art DMRG algorithms in electronic structure and beyond*, J. Chem. Phys. **159**, 234801 (2023), doi:10.1063/5.0180424.

[29] M. Ballarin et al., *Quantum tea: Qtealeaves*, Zenodo (2024), doi:10.5281/zenodo.10498929.

[30] C. R. Roberts et al., *Tensornetwork: A library for easy and efficient manipulation of tensor networks* (2019), https://github.com/google/TensorNetwork.

[31] K.-H. Wu, C.-T. Lin, K. Hsu, H.-T. Hung, M. Schneider, C.-M. Chung, Y.-J. Kao and P. Chen, *The Cytnx library for tensor networks*, (arXiv preprint) doi:10.48550/arXiv.2401.01921.

[32] Y. Motoyama, T. Okubo, K. Yoshimi, S. Morita, T. Kato and N. Kawashima, *TeNeS: Tensor network solver for quantum lattice systems*, Comput. Phys. Commun. **279**, 108437 (2022), doi:10.1016/j.cpc.2022.108437.

[33] J. Haegeman et al., *TensorKit.jl*, Zenodo (2024), doi:10.5281/zenodo.10959683.

[34] A. Weichselbaum, *QSpace - An open-source tensor library for Abelian and non-Abelian symmetries*, SciPost Phys. Codebases 40 (2024), doi:10.21468/SciPostPhysCodeb.40.

A. Weichselbaum, *Codebase release 4.0 for QSpace*, SciPost Phys. Codebases 40-r4.0 (2024), doi:10.21468/SciPostPhysCodeb.40-r4.0.

[35] J. Hasik et al., *peps-torch: Solving two-dimensional spin models with tensor networks (powered by PyTorch)* (2020), https://github.com/jurajHasik/peps-torch.

[36] B. Ponsioen, *ad-peps: iPEPS ground- and excited-state implementation based on automatic differentiation* (2021)), https://github.com/b1592/ad-peps.

[37] J. Naumann, E. L. Weerda, M. Rizzi, J. Eisert and P. Schmoll, *An introduction to infinite projected entangled-pair state methods for variational ground state simulations using automatic differentiation*, SciPost Phys. Lect. Notes 86 (2024), doi:10.21468/SciPostPhysLectNotes.86.

[38] P. Brehmer, L. Burgelman and L. Devos, *PEPSKit.jl*, Zenodo, (2024), doi:10.5281/zenodo.13938736.

[39] T. Chanda, *Tennetlib.jl* (2023), https://github.com/titaschanda/TenNetLib.jl.

[40] C. R. Harris et al., *Array programming with NumPy*, Nature **585**, 357 (2020), doi:10.1038/s41586-020-2649-2.

[41] A. Paszke et al., *PyTorch: An imperative style, high-performance deep learning library*, in *Proceedings of the 33rd international conference on neural information processing systems*, Curran Associates, Red Hook, USA, ISBN 9781713829546 (2019).

[42] R. N. C. Pfeifer, G. Evenbly, S. Singh and G. Vidal, *NCON: A tensor network contractor for MATLAB*, (arXiv preprint) doi:10.48550/arXiv.1402.0939.

[43] J. Haegeman, J. I. Cirac, T. J. Osborne, I. Pižorn, H. Verschelde and F. Verstraete, *Time-dependent variational principle for quantum lattices*, Phys. Rev. Lett. **107**, 070601 (2011), doi:10.1103/PhysRevLett.107.070601.

[44] J. Haegeman, C. Lubich, I. Oseledets, B. Vandereycken and F. Verstraete, *Unifying time evolution and optimization with matrix product states*, Phys. Rev. B **94**, 165116 (2016), doi:10.1103/PhysRevB.94.165116.

[45] J. Niesen and W. M. Wright, *Algorithm 919: A Krylov subspace algorithm for evaluating the ϕ-functions appearing in exponential integrators*, ACM Trans. Math. Softw. **38**, 1 (2012), doi:10.1145/2168773.2168781.

[46] F. Verstraete, V. Murg and J. I. Cirac, *Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems*, Adv. Phys. **57**, 143 (2008), doi:10.1080/14789940801912366.

[47] A. D. King et al., *Computational supremacy in quantum simulation*, (arXiv preprint) doi:10.48550/arXiv.2403.00910.

[48] G. Wójtowicz, A. Purkayastha, M. Zwolak and M. M. Rams, *Accumulative reservoir construction: Bridging continuously relaxed and periodically refreshed extended reservoirs*, Phys. Rev. B **107**, 035150 (2023), doi:10.1103/PhysRevB.107.035150.

[49] J. Dziarmaga, *Time evolution of an infinite projected entangled pair state: Neighborhood tensor update*, Phys. Rev. B **104**, 094411 (2021), doi:10.1103/PhysRevB.104.094411.

[50] J. Dziarmaga, *Simulation of many-body localization and time crystals in two dimensions with the neighborhood tensor update*, Phys. Rev. B **105**, 054203 (2022), doi:10.1103/PhysRevB.105.054203.

[51] P. Czarnik, J. Dziarmaga and P. Corboz, *Time evolution of an infinite projected entangled pair state: An efficient algorithm*, Phys. Rev. B **99**, 035115 (2019), doi:10.1103/PhysRevB.99.035115.

[52] A. Sinha, M. M. Rams, P. Czarnik and J. Dziarmaga, *Finite-temperature tensor network study of the Hubbard model on an infinite square lattice*, Phys. Rev. B **106**, 195105 (2022), doi:10.1103/PhysRevB.106.195105.

[53] A. Sinha, M. M. Rams and J. Dziarmaga, *Efficient representation of minimally entangled typical thermal states in two dimensions via projected entangled pair states*, Phys. Rev. B **109**, 045136 (2024), doi:10.1103/PhysRevB.109.045136.

[54] M. M. Rams et al., *yastn_benchmarks* (2024), https://github.com/yastn/yastn_benchmarks.

[55] T. Nishino and K. Okunishi, *Corner transfer matrix renormalization group method*, J. Phys. Soc. Jpn. **65**, 891 (1996), doi:10.1143/JPSJ.65.891.

[56] R. Orús and G. Vidal, *Simulation of two-dimensional quantum systems on an infinite lattice revisited: Corner transfer matrix for tensor contraction*, Phys. Rev. B **80**, 094403 (2009), doi:10.1103/PhysRevB.80.094403.

[57] P. Corboz, T. M. Rice and M. Troyer, *Competing states in the $t-J$ model: Uniform d-wave state versus stripe state*, Phys. Rev. Lett. **113**, 046402 (2014), doi:10.1103/PhysRevLett.113.046402.

[58] M. T. Fishman, L. Vanderstraeten, V. Zauner-Stauber, J. Haegeman and F. Verstraete, *Faster methods for contracting infinite two-dimensional tensor networks*, Phys. Rev. B **98**, 235148 (2018), doi:10.1103/PhysRevB.98.235148.

[59] J. Hasik, G. B. Mbeng, S. Capponi, F. Becca and A. M. Läuchli, *Symmetric projected entangled-pair states analysis of a phase transition in coupled spin-1/2 ladders*, Phys. Rev. B **106**, 125154 (2022), doi:10.1103/PhysRevB.106.125154.

[60] Y. Xu, S. Capponi, J.-Y. Chen, L. Vanderstraeten, J. Hasik, A. H. Nevidomskyy, M. Mambrini, K. Penc and D. Poilblanc, *Phase diagram of the chiral SU(3) antiferromagnet on the kagome lattice*, Phys. Rev. B **108**, 195153 (2023), doi:10.1103/PhysRevB.108.195153.

[61] P. Corboz and G. Vidal, *Fermionic multiscale entanglement renormalization Ansatz*, Phys. Rev. B **80**, 165129 (2009), doi:10.1103/PhysRevB.80.165129.

[62] P. Corboz, R. Orús, B. Bauer and G. Vidal, *Simulation of strongly correlated fermions in two spatial dimensions with fermionic projected entangled-pair states*, Phys. Rev. B **81**, 165104 (2010), doi:10.1103/PhysRevB.81.165104.