

Practical algorithm for simulating thermal pure quantum states

Wei-Bo He(何伟博), Yun-Tong Yang(杨赟彤), and Hong-Gang Luo(罗洪刚)
Citation: Chin. Phys. B, 2026, 35 (**1**): 010101. DOI: 10.1088/1674-1056/ae1c21
http://cpb.iphy.ac.cn; http://iopscience.iop.org/cpb

What follows is a list of articles you may be interested in

Duality symmetry, two entropy functions, and an eigenvalue problem in generalized Gibbs' theory

Jeffrey Commons, Ying-Jen Yang(杨颖任), and Hong Qian(钱纮)
Chin. Phys. B, 2025, 34 (**8**): 080201. DOI: 10.1088/1674-1056/adbed7

Trajectory tracking on the optimal path of two-dimensional quadratic barrier escaping

Zengxuan Zhao(赵曾轩), Xiuying Zhang(张秀颖), Pengchen Zhao(赵鹏琛), Chunyang Wang(王春阳), Chunlei Xia(夏春雷), Mushtaq Rana Imran, and Joelous Malamula Nyasulu
Chin. Phys. B, 2025, 34 (**5**): 050205. DOI: 10.1088/1674-1056/adcb99

GPIC: A GPU-based parallel independent cascade algorithm in complex networks

Chang Su(苏畅), Xu Na(那旭), Fang Zhou(周方), and Linyuan Lü(吕琳媛)
Chin. Phys. B, 2025, 34 (**3**): 030204. DOI: 10.1088/1674-1056/adb67c

ScatterX: A software for fast processing of high-throughput small-angle scattering data

Fei Xie(谢飞), Mei Xie(解梅), Baoyu Song(宋宝玉), Qiaoyu Guo(郭桥雨), and Xuechen Jiao(焦学琛)
Chin. Phys. B, 2024, 33 (**12**): 120101. DOI: 10.1088/1674-1056/ad8b36

Two-dimensional finite element mesh generation algorithm for electromagnetic field calculation

Chun-Feng Zhang(章春锋), Wei Wang(汪伟), Si-Guang An(安斯光), and Nan-Ying Shentu(申屠南瑛)
Chin. Phys. B, 2021, 30 (**1**): 010101. DOI: 10.1088/1674-1056/abaedf

---

COMPUTATIONAL PROGRAMS FOR PHYSICS

Practical algorithm for simulating thermal pure quantum states

Wei-Bo He(何伟博)[1,2], Yun-Tong Yang(杨赟彤)[1,2], and Hong-Gang Luo(罗洪刚)[1,2,†]

[1] *School of Physical Science and Technology, Lanzhou University, Lanzhou* 730000, *China*

[2] *Lanzhou Center for Theoretical Physics, Key Laboratory of Theoretical Physics of Gansu Province, Key Laboratory of Quantum Theory and Applications of MoE, Gansu Provincial Research Center for Basic Disciplines of Quantum Physics, Lanzhou University, Lanzhou* 730000, *China*

(Received 3 September 2025; revised manuscript received 21 October 2025; accepted manuscript online 6 November 2025)

The development of novel quantum many-body computational algorithms relies on robust benchmarking. However, generating such benchmarks is often hindered by the massive computational resources required for exact diagonalization or quantum Monte Carlo simulations, particularly at finite temperatures. In this work, we propose a new algorithm for obtaining thermal pure quantum states, which allows efficient computation of both mechanical and thermodynamic properties at finite temperatures. We implement this algorithm in our open-source C++ template library, *Physica*. Combining the improved algorithm with state-of-the-art software engineering, our implementation achieves high performance and numerical stability. As an example, we demonstrate that for the 4×4 Hubbard model, our method runs approximately 10^3 times faster than $\mathcal{H}\Phi$ 3.5.2. Moreover, the accessible temperature range is extended down to $\beta = 32$ across arbitrary doping levels. These advances significantly push forward the frontiers of benchmarking for quantum many-body systems.

Keywords: *Physica*, thermal pure quantum states, Hubbard model, strong correlated electron systems

PACS: 01.50.hv, 02.70.–c, 05.30.–d, 71.10.–w

DOI: 10.1088/1674-1056/ae1c21 **CSTR:** 32038.14.CPB.ae1c21

1. Introduction

Quantum many-body physics stands as one of the most challenging and fascinating areas of modern physics. In particular, the complex and rich emergent phases in strongly correlated systems often elude description by conventional frameworks such as Landau Fermi liquid theory and the Landau–Ginzburg paradigm. Alongside physical experiments and quantum simulations, numerical methods offer a powerful avenue for validating theoretical predictions. Although a variety of computational techniques exist, including exact diagonalization (ED), quantum Monte Carlo (QMC),[1–3] density matrix renormalization group (DMRG),[4,5] tensor networks (TN),[6] and dynamical mean field theory (DMFT),[7–9] only sign-problem-free QMC and ED are generally considered reliable enough for benchmarking. However, these two face significant limitations: for instance, at arbitrary temperature the Hubbard model is sign-problem-free only at half-filling, which excludes much of the experimentally relevant doping regime, including potential superconducting phases. While ED is primarily a ground-state method, extensions such as the finite-temperature Lanczos method (FTLM)[10,11] have been developed to approximate finite-temperature properties in small systems. Although FTLM can produce accurate results with a reduced number of Lanczos steps and randomized state sampling, it requires careful balancing between computational cost and numerical accuracy, including the trade-off between sufficient step count and expensive reorthogonalization procedures.[10]

The open-source community plays a vital role in advancing research by providing fast and reliable code for reproducible benchmarking. In recent years, several open-source software packages for quantum many-body computations have been developed. For example, ALF[15,16] and hubbard-DQMC[17] are QMC packages targeting fermionic systems; however, both are significantly affected by the notorious sign problem, especially at low temperatures. On the other hand, QuSpin[12] offers a user-friendly Python environment for ED and dynamics simulations, but is limited to small and medium-sized quantum systems. XDiag,[13] a recently developed ED library written in C++ with a Julia interface, strikes a balance between computational performance and usability. Meanwhile, $\mathcal{H}\Phi$[14] delivers an ED package optimized for large-scale distributed-memory clusters and supports finite-temperature calculations via thermal pure quantum (TPQ) states.

In the present work, we focus on the TPQ technique. The TPQ states provide a powerful approach for approximating various mechanical and thermodynamic properties of finite-temperature many-body systems. The results obtained from TPQ calculations converge exponentially in probability toward the exact values as the system size increases.[18–20] A key advantage of the TPQ approach over full exact diagonalization (Full-ED) is its reduced memory requirement: instead of storing $O(N)$ eigenvectors for finite-temperature computa-

†Corresponding author. E-mail: luohg@lzu.edu.cn

© 2026 Chinese Physical Society and IOP Publishing Ltd. All rights, including for text and data mining, AI training, and similar technologies, are reserved.

tions, TPQ requires only a single random vector. Since a TPQ state is a pure state in the Hilbert space, nearly all techniques developed for ED, such as symmetry exploitation[21] to reduce memory and enhance performance, can be directly applied. Efficient indexing of basis states, for instance via perfect hashing methods,[22,23] remains essential for high performance in such simulations.

In the original formulation of the TPQ method,[19] the thermal state is constructed by expanding the matrix exponential via a Taylor series and iteratively applying each term to a random vector until convergence. Although this approach is conceptually simple and easy to implement, it is computationally inefficient and prone to significant numerical errors. In this work, we introduce a practical and high-performance algorithm for generating TPQ states that achieves greater accuracy. The improvement is demonstrated through direct comparisons with results from $\mathcal{H}\Phi$ 3.5.2, confirming both the enhanced precision and efficiency of our method.

The article is structured as follows. In Section 2, we provide a brief overview of the Full-ED and TPQ algorithms, followed by the introduction of our improved method for TPQ state simulation. In Section 3, we conduct a comparative analysis of the numerical stability and performance between our proposed algorithm and the original implementation. We conclude with remarks and outlook in Section 4.

2. Formalism

2.1. Full exact diagonalization

For a Hamiltonian $\hat{H}$, the partition function is defined as $Z = \mathrm{tr}[\mathrm{e}^{-\beta\hat{H}}]$, where $\beta = 1/(kT)$ denotes the inverse temperature. The canonical ensemble average of an observable $\hat{O}$ is given by $\langle\hat{O}\rangle_{\mathrm{ens}} = \frac{1}{Z}\mathrm{tr}[\mathrm{e}^{-\beta\hat{H}}\hat{O}]$. To evaluate the trace, one may diagonalize the Hamiltonian to obtain its eigenbasis $|i\rangle$ and eigenvalues E_i, allowing the partition function and observable expectation values to be expressed as sums over eigenstates:

$$Z = \sum_{i=1}^{D} \mathrm{e}^{-\beta E_i}, \quad \langle\hat{O}\rangle = \frac{1}{Z}\sum_{i=1}^{D} \mathrm{e}^{-\beta E_i}\langle i|\hat{O}|i\rangle, \tag{1}$$

where D is the dimension of the many-body Hilbert space. This approach constitutes the Full-ED method. However, since D grows exponentially with system size, the $\mathcal{O}(D^2)$ space complexity of Full-ED restricts its practical use to small or toy models.

A common strategy to mitigate this limitation is to retain only the M lowest-energy eigenstates, which dominate thermodynamic properties at low temperatures. This reduces the space complexity to $\mathcal{O}(MD)$. Nevertheless, the choice of M lacks rigorous convergence guarantees and must be determined empirically. Moreover, even with this truncation, the memory requirement remains substantially higher than that of ground-state-specific methods such as the Lanczos algorithm.[24]

2.2. Thermal quantum pure states

The TPQ states offer an approach to finite-temperature simulations with space complexity comparable to ground-state methods. This is achieved by distributing part of the Hilbert space complexity into the imaginary-time dimension through a stochastic construction. Consider a canonical ensemble at inverse temperature β with fixed particle number N, and let $|i\rangle$ denote the energy eigenstates of the system Hamiltonian $\hat{H}$. Sampling random complex coefficients c_i from the unit hypersphere such that $\sum_i |c_i|^2 = 1$, we define the infinite-temperature canonical TPQ state as $|0\rangle = \sum_i c_i|i\rangle$. The finite-temperature canonical TPQ state is then constructed via imaginary-time evolution:

$$|\beta\rangle = \mathrm{e}^{-\beta\hat{H}/2}|0\rangle. \tag{2}$$

The expectation value of an observable $\hat{O}$ is estimated using the expression

$$Z = \overline{\langle\beta|\beta\rangle}, \quad \langle\hat{O}\rangle^{\mathrm{TPQ}} = \frac{1}{Z}\overline{\langle\beta|\hat{O}|\beta\rangle}, \tag{3}$$

where $\overline{\cdots}$ indicates the average over random realizations.[19] It can be shown that for any $\varepsilon > 0$, the deviation probability $P\left(\left|\langle\hat{O}\rangle^{\mathrm{TPQ}} - \langle\hat{O}\rangle_{\mathrm{ens}}\right| > \varepsilon\right)$ decays exponentially with increasing system size N, where $\langle\hat{O}\rangle_{\mathrm{ens}}$ denotes the canonical ensemble average given above.

To numerically obtain $|\beta\rangle$, the original algorithm expands the matrix exponential as a Taylor series and truncates it at a sufficiently high order n_0:

$$|\beta\rangle = \sum_{n=0}^{\infty}\frac{1}{n!}\left(-\frac{\beta\hat{H}}{2}\right)^n|0\rangle \approx \sum_{n=0}^{n_0}\frac{1}{n!}\left(-\frac{\beta\hat{H}}{2}\right)^n|0\rangle. \tag{4}$$

2.3. Improvements on TPQ

The original algorithm suffers from several limitations. First, the truncation order n_0 is not known a priori and must be determined empirically. Second, even when identified, n_0 is typically large due to the slow convergence of the Taylor series expansion. Moreover, numerical overflow may occur due to the rapidly growing terms in the expansion. To address these issues, we adopt the algorithm proposed in Ref. [25] which is suitable for scalable matrix exponential operations in large Hilbert spaces. The matrix exponential is approximated as

$$\mathrm{e}^{\boldsymbol{A}} \approx \mathrm{e}^{\overline{A}}\left[T_m\left(\frac{1}{n}(\boldsymbol{A} - \overline{A}\boldsymbol{I})\right)\right]^n, \tag{5}$$

where $\boldsymbol{A} = -\frac{\beta}{2}\boldsymbol{H}$, with $\boldsymbol{H}$ being the matrix representation of the Hamiltonian in a chosen basis and $\boldsymbol{I}$ the identity matrix of the same dimension. Here, $\overline{A} = \frac{1}{D}\mathrm{tr}[\boldsymbol{A}]$ is a real scalar

that isolates the diagonal contribution to the partition function, thereby enhancing numerical stability. The function T_m denotes the m-th order Taylor expansion of the exponential function, and n represents the number of imaginary-time segments. The optimal values of m and n that minimize computational cost while achieving machine precision can be expressed using the 1-norm of matrices.

In this work, we employ power-based 1-norm estimator[26] rather than the blocked estimator recommended in the original literature, as the latter requires storing multiple eigenstates and BLAS3 matrix–matrix products — a prohibitive requirement for larger systems, whereas the power-based 1-norm estimator only uses BLAS2 matrix–vector product. The matrix polynomial is carefully normalized to avoid overflow (Algorithm 1). Typically, the matrix–vector product either directly operates on sparse matrix $\boldsymbol{H}$ or is implemented using the on-the-fly trick. However, simply applying these approaches to matrix polynomial $\left(\frac{1}{N}(\boldsymbol{A}-\overline{A}\boldsymbol{I})\right)^p$ would introduce significant inefficiency. Since modifications to either the large-scale sparse matrix or the on-the-fly Hamiltonian matrix elements have an $O(n^2)$ time complexity, which is generally undesired. Here we incorporate the template expression technique, which lowers the custom matrix–vector product with a standard matrix–vector product and highly optimized BLAS1 operations for arbitrary Hamiltonian without coding effort:

$$\frac{1}{N}(\boldsymbol{A}-\overline{A}\boldsymbol{I})\boldsymbol{x}=-\frac{\beta}{2N}(\boldsymbol{H}\boldsymbol{x})-\frac{\overline{A}}{N}\boldsymbol{x}, \tag{6}$$

where $\boldsymbol{x}$ is a given vector. Parameters (m,n) reflect the inherent properties of the system's imaginary time evolution of duration β. As a result, they may be saved and reused for all independent TPQ state simulations, thus avoiding unnecessary computational cost.

Algorithm 1: Determination of (m,n)

Data: Matrix $\boldsymbol{A}$, $m_{\max}=55$, $p_{\max}=8$, $\{\theta_i : 1\le i\le m_{\max}\}$ as provided in TABLE. 3.1 of Ref. [25]

Result: Parameters (m,n)

$N_*\leftarrow 2p_{\max}(p_{\max}+3)\theta_{m_{\max}}/m_{\max}$;

$N\leftarrow ||\boldsymbol{A}-\overline{A}\boldsymbol{I}||_1$;

if $N<N_*$ **then**

  $m\leftarrow \mathrm{argmin}_i\, i\lceil N/\theta_i\rceil$;

  $n\leftarrow \lceil N/\theta_m\rceil$;

else

  Calculate $\{d_p=N\cdot||\left(\frac{1}{N}(\boldsymbol{A}-\overline{A}\boldsymbol{I})\right)^p||^{1/p} : 2\le p\le p_{\max}+1\}$;

  Calculate $\{\alpha_p=\max(d_p,d_{p+1}) : 2\le p\le p_{\max}\}$;

  Calculate $\{C_i=m\lceil\alpha_p/\theta_i\rceil : 2\le p\le p_{\max}, p(p-1)-1\le m\le m_{\max}\}$;

  $m\leftarrow \mathrm{argmin}_i\, C_i$;

  $n\leftarrow \max(\lceil C_m/\theta_m\rceil, 1)$;

end

Leveraging available symmetries and working within the irreducible Hilbert space are essential for enabling large-scale simulations. Taking the Hubbard model and particle number conservation as an example, the Hamiltonian matrix can be diagonalized into a block-diagonal form

$$\boldsymbol{H}=\mathrm{diag}\left(\boldsymbol{H}_0,\boldsymbol{H}_1,\ldots,\boldsymbol{H}_L\right), \tag{7}$$

where $0,1,\ldots,L$ are eigenvalues of $\hat{N}$, $\boldsymbol{H}_l$ is the Hamiltonian matrix in the subspace spanned by states of l particles. One can diagonalize the Hamiltonian in the corresponding subspaces and readily write the imaginary time evolution operator as

$$\mathrm{e}^{\boldsymbol{A}}=\mathrm{diag}\left(\mathrm{e}^{\boldsymbol{A}_0},\mathrm{e}^{\boldsymbol{A}_1},\ldots,\mathrm{e}^{\boldsymbol{A}_L}\right), \tag{8}$$

where $\boldsymbol{A}_l=-\frac{\beta}{2}\boldsymbol{H}_l$. The matrix–vector product may be processed in each sector independently:

$$\begin{aligned}\mathrm{e}^{\boldsymbol{A}}\boldsymbol{x} &= \mathrm{diag}\left(\mathrm{e}^{\boldsymbol{A}_0},\mathrm{e}^{\boldsymbol{A}_1},\ldots,\mathrm{e}^{\boldsymbol{A}_L}\right)\boldsymbol{x}\\ &= \mathrm{e}^{\boldsymbol{A}_0}\boldsymbol{x}_0\oplus\mathrm{e}^{\boldsymbol{A}_1}\boldsymbol{x}_1\oplus\cdots\oplus\mathrm{e}^{\boldsymbol{A}_L}\boldsymbol{x}_L.\end{aligned} \tag{9}$$

The expectation value of observable $\hat{O}$ in the grand canonical ensemble is given by

$$\langle\hat{O}\rangle=\frac{1}{\Xi}\overline{\langle\beta,\mu|\hat{O}|\beta,\mu\rangle}, \tag{10}$$

where Ξ is the grand canonical partition function. The grand canonical TPQ state $|\beta,\mu\rangle$ is related to the canonical TPQ state $|\beta\rangle$ via

$$|\beta,\mu\rangle=\mathrm{e}^{\frac{\beta}{2}\mu\hat{N}}|\beta\rangle, \tag{11}$$

with μ denoting the chemical potential. If the operator $\hat{O}$ commutes with $\hat{N}$, we can further reduce grand canonical expecta-

tion values to a weighted sum over symmetry sectors:

$$\begin{aligned}\langle\hat{O}\rangle &= \frac{1}{\Xi}\overline{\langle\beta|\hat{O}\,\mathrm{e}^{\beta\mu\hat{N}}|\beta\rangle} \\ &= \frac{1}{\Xi}\sum_{l}\mathrm{e}^{\beta\mu l}\sum_{\boldsymbol{x}_l}\boldsymbol{x}_l^{\dagger}\,\mathrm{e}^{\boldsymbol{A}_l}\boldsymbol{O}\,\mathrm{e}^{\boldsymbol{A}_l}\boldsymbol{x}_l,\end{aligned} \tag{12}$$

where $\boldsymbol{x}_l$ is a random initial state in the subspace with fixed particle number l. Similarly, the grand canonical partition function can be expressed as

$$\Xi = \overline{\langle\beta,\mu|\beta,\mu\rangle} = \sum_{l}\mathrm{e}^{\beta\mu l}\overline{\langle\beta|\beta\rangle}_l, \tag{13}$$

which requires computing the canonical partition function $Z_l = \overline{\langle\beta|\beta\rangle}_l = \sum_{\boldsymbol{x}_l}|\,\mathrm{e}^{\boldsymbol{A}_l}\boldsymbol{x}_l|^2$. Note that the partition function scales exponentially with system size and inverse temperature β, but acts as a normalization factor of TPQ states and does not affect the essential physics. Therefore, in practice, we accumulate the logarithmic partition function for each imaginary time step separately, rather than explicitly forming $\mathrm{e}^{\boldsymbol{A}_l}\boldsymbol{x}$. This approach is critical for reaching arbitrary low temperatures while keeping enough effective digits (Algorithm 2). We have implemented the above algorithms in our open-source C++ template library, *Physica* (Appendix A). To keep the article tight, we have provided guidance on reproducing our results in Appendix B.

Algorithm 2: Evaluation of Eq. (5)

Data: Matrix $\boldsymbol{A}$, parameters $(m, n$, arbitrary vector $\boldsymbol{x}$, machine precision ϵ

Result: Vector $\boldsymbol{y} = \mathrm{e}^{\boldsymbol{A}-\eta\boldsymbol{I}}\boldsymbol{x}$, real scalar η

$i \leftarrow 0$;
$\boldsymbol{y} \leftarrow \boldsymbol{x}$;
$\eta \leftarrow 0$;
for $i < n$ **do**
  $N_\infty \leftarrow ||\boldsymbol{y}||_\infty$;
  $\boldsymbol{y} \leftarrow \boldsymbol{y}/N_\infty$;
  $\boldsymbol{z} \leftarrow \boldsymbol{y}$;
  for $j < m$ **do**
    $\boldsymbol{z} \leftarrow \frac{1}{mj}(\boldsymbol{A} - \overline{A}\boldsymbol{I})\boldsymbol{z}$;
    $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{z}$;
    if $N_\infty + ||\boldsymbol{z}||_\infty \le \epsilon||\boldsymbol{y}||_\infty$ **then**
      break;
    end
    $j \leftarrow j + 1$;
  end
  $\eta \leftarrow \eta + \ln N_\infty$;
  $i \leftarrow i + 1$;
end

3. Results and discussion

We use the Hubbard model[27] as our test case, which is widely regarded as the minimal two-dimensional (2D) model for cuprate superconductors. The Hamiltonian, including nearest-neighbor hopping, is given by

$$\begin{aligned}\hat{H} = &-\sum_{ij,\sigma}t_{ij}\left(\hat{c}_{i\sigma}^{\dagger}\hat{c}_{j\sigma} + \mathrm{h.c.}\right) \\ &+U\sum_{i}\hat{n}_{i\uparrow}\hat{n}_{i\downarrow} - \mu\sum_{i}\left(\hat{n}_{i\uparrow}+\hat{n}_{i\downarrow}\right),\end{aligned} \tag{14}$$

where $\hat{c}_{i\sigma}^{\dagger}$, $\hat{c}_{i\sigma}$, and $\hat{n}_{i\sigma}$ denote the creation, annihilation, and particle number operators, respectively, for an electron at site i with spin σ ($\uparrow$ or $\downarrow$). The hopping parameter t_{ij} is equal to t for nearest-neighbor pairs and zero otherwise. We consider a strong interaction strength of $U/t = 8$ and simulate multiple systems under periodic boundary conditions (PBC).

3.1. Numerical stability

To validate the correctness of our implementation, we begin by studying the one-dimensional Hubbard model with 4 sites at $\mu = 0$ — a system size amenable to Full-ED. The electron density ρ as a function of inverse temperature β is computed using TPQ methods implemented independently in our framework and in $\mathcal{H}\Phi$. Using the Full-ED results obtained from $\mathcal{H}\Phi$ as the reference, the relative error R and its corresponding standard deviation σ_R for the electron density is defined as

$$R = \frac{\rho - \rho_{\text{Full-ED}}}{\rho_{\text{Full-ED}}}, \quad \sigma_R = \frac{\sigma_\rho}{\rho_{\text{Full-ED}}}, \tag{15}$$

where σ_ρ is the standard deviation of electron density.

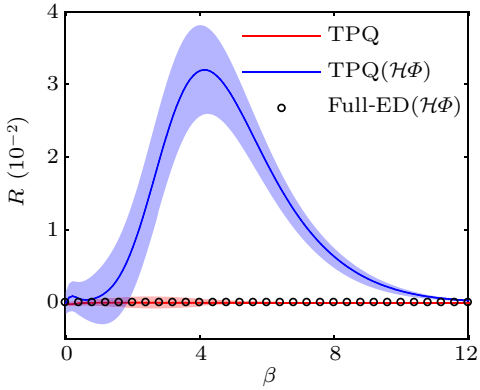


Fig. 1. The relative error R in the electron density for the one-dimensional 4-site Hubbard model at $\mu = 0$, showing its dependence on inverse temperature β for our improved algorithm (red solid line) and the original algorithm implemented in $\mathcal{H}\Phi$ (blue solid line). The statistical uncertainty ($2\sigma_R$) for each is indicated by the lighter shaded regions. Results are benchmarked against the Full-ED results from $\mathcal{H}\Phi$ (black circles). TPQ expectation values are computed using a combination of 8 independent runs and $2^{13} = 8192$ random initial states per run. The imaginary time step size is taken to be $\Delta\tau = 0.1$.

As shown in Fig. 1, our TPQ results demonstrate excellent agreement with Full-ED across the temperature range. We

further note that the initial states are not necessarily generated in complex number space, but satisfying the uncorrelated phase condition $\langle c_i c_j \rangle = \delta_{ij}/D$ is sufficient to allow destructive interference between different eigenstates. Thus, the statistical uncertainty is much smaller due to the selection of real number initial states. In contrast, the original TPQ algorithm implemented in $\mathcal{H}\Phi$ exhibits rapidly growing relative error, reaching approximately 3% as temperature decreases, highlighting its numerical instability at low temperatures. While increasing the expansion order in $\mathcal{H}\Phi$ can improve accuracy, the associated computational cost becomes prohibitively high, especially for larger systems. As a result, we attribute this error accumulation to the original algorithms' use of a non-adaptive truncation of the expansion order. The eventual decrease in relative error at very low temperatures occurs as the system approaches its ground state, where contributions from high energy states are less significant.

3.2. Performance

We simulate the Hubbard model on a 4×4 square lattice, which, to the best of our knowledge, still lacks exact numerical results across arbitrary temperatures and doping levels. The simulations are carried out at multiple temperatures, extending down to $\beta t = 32$. The system is hole-doped, with the chemical potential μ/t varied over the interval $[-8, 0]$, a range sufficiently broad to encompass regimes from Fermi liquids to potential superconducting phases (Fig. 2). Taking the $\beta t = 16$ curve as an example, we compute full profiles of electron density and double occupancy as functions of chemical potential in approximately 5×10^3 core-hours on our test platform (Intel® Xeon® Platinum 8358, 256 GB RAM). In contrast, under equivalent computational conditions, $\mathcal{H}\Phi$ produces only a limited number of data points; we estimate that generating the complete curve would require approximately 10^6 core-hours.

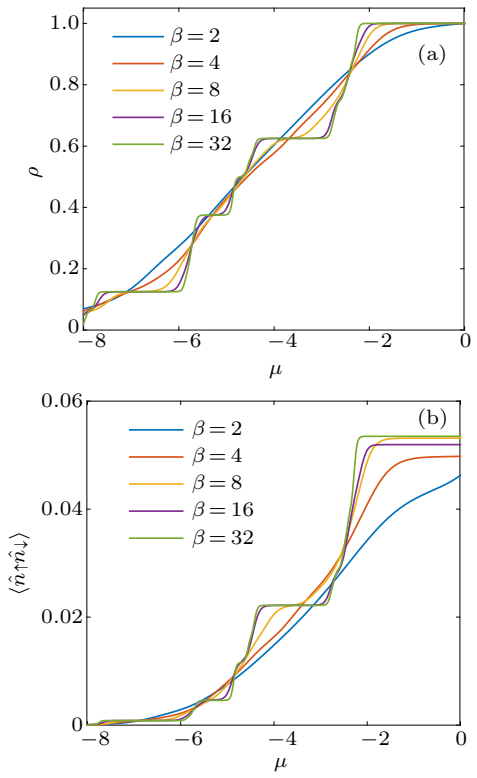


Fig. 2. (a) Electron density (ρ) and (b) double occupation ($\langle \hat{n}_\uparrow \hat{n}_\downarrow \rangle$) as a function of chemical potential (μ) for a 4×4 Hubbard model with PBC. Results are plotted for several inverse temperatures.

4. Conclusion

We present a high-performance and numerically stable algorithm for simulating TPQ states, rendering the TPQ approach highly practical for solving quantum many-body problems. This algorithm is implemented within an open-source C++ template library, combining computational efficiency with robust numerical reliability. Our improved method demonstrates significantly better agreement with exact numerical benchmarks compared to conventional approaches. Moreover, the implementation achieves a speedup of three orders of magnitude over existing methods in $\mathcal{H}\Phi$. By extending the limits of scalable and accurate benchmarking in quantum many-body systems, this work provides a solid and efficient foundation for validating emerging computational algorithms in the field.

Appendix A: High level design of *Physica*

A1. Project structure

The project structure is often the first aspect users encounter when engaging with open-source software. A clean and goal-oriented project organization is as crucial as well-designed and well-documented code. The top-level directory structure of *Physica* follows conventions commonly adopted by open-source C++ projects, consisting of the following directories: `3rdparty`, `benchmark`, `doc`, `examples`, `include`, `src`, and `test`. Within *Physica*, the secondary structure is organized modularly. The modules included in *Physica* are listed in Table A1.

Table A1. Modules and description.

Module	Description
Core	Implementation of *Physica*'s core functionality
Gui	Includes 2D and 3D drawing support, using Qt as the drawing backend
Logger	A high-performance logging library based on NanoLog[28]
Python	Backend of *Physica* python interface (WIP)
phypy	*Physica* python interface (WIP)

The secondary project structure extends to each directory within the top-level layout. Tertiary and finer-grained structures encompass APIs and implementation details. The organization follows scientific — rather than purely engineering — logic, allowing domain experts to more readily adopt *Physica*. Engineering complexities are encapsulated within directories

suffixed with "Impl", with lower-level logic nested deeper in the directory hierarchy.

Overall, *Physica* employs a goal-oriented, layered structure that progressively exposes complexity. Users not concerned with implementation details will generally encounter fewer such details, as most common use cases are addressed at shallower directory levels. Since scientific workflows are diverse and often require flexibility, we ensure users retain the ability to access and modify underlying implementation details. The pervasive use of templates further facilitates non-intrusive customization and extension.

A2. The rational of templates

The development of scalable programs presents significant challenges, as each problem exhibits both universal and particular aspects. A central difficulty lies in balancing these dimensions: generic implementations may require adaptation to improve performance or insight in specific contexts, while optimizations tailored to one system may not transfer effectively to others. In essence, each case demands its own optimally suited implementation. Manually developing specialized solutions for every scenario, however, is often impractical and inefficient.

To address this, we employ template metaprogramming techniques, which allow code generation rules to instruct the compiler to automatically produce optimized implementations for each use case. Conventional scientific computing programs face a fundamental trade-off: they often struggle to incorporate problem-specific optimizations without sacrificing generality. Efforts to introduce specificity frequently lead to uncontrolled growth in input and output configurations, resulting in substantial maintenance and efficiency costs.

Consider, for example, floating-point numeric types: common options include `bfloat16`, `float16`, `float32`, `float64`, and `float128`. When extended to complex numbers, the number of available types doubles. Incorporating automatic differentiation — as often required in deep learning — further doubles this number, leading to at least 20 possible floating-point type combinations, even before considering future extensions. Without templates, each function would require over 20 separate implementations, resulting in repetitive and hard-to-maintain code. By contrast, with C++ template metaprogramming, all valid type combinations are resolved automatically at compile time. Since template parameters are evaluated during compilation, this approach ensures both performance and extensibility, readily accommodating new numerical types without code modification.

However, excessive reliance on templates can lead to long compilation time, large binary sizes, and increased complexity. To mitigate these issues, we adopt the following strategies.

1. Although *Physica* is primarily header-based — a natural consequence of heavy template use that also simplifies usage and modification — we compile sufficiently general and performance-insensitive modules (such as exceptions, I/O, and common utilities) into dynamic libraries. This reduces executable size and compilation time while preserving the benefits of templates where they matter most.

2. To lower the barrier to entry, we encourage users to consult the example cases provided in the `examples` directory. These serve as practical prototypes that can be adapted as needed and will be continuously updated based on community feedback.

3. The expressive power of templates enables the formal composition of existing features and straightforward extension to new functionality, supporting both generality and specificity without invasive code changes.

A3. Input and output

Physica intentionally does not provide traditional input or output files — a deliberate design choice that reflects its modern approach to scientific computing. As software functionalities expand, the rigid input–compute–output model of the past has become increasingly inadequate for contemporary research needs. Many large-scale scientific computing packages rely on cumbersome input files with hundreds of keywords,[29,30] where each new feature introduces additional tags that complicate usage and obscure intent. Like comments and documentation, input files are inherently decoupled from the code, creating a risk of silent discrepancies between user intent and actual computation. Moreover, the limited expressiveness of input files restricts users' ability to finely customize functionality.

Output files in traditional frameworks also present challenges: it is common for large-scale programs to generate numerous lengthy files, forcing users to navigate extensive irrelevant data to locate meaningful results. This not only impedes productivity but also consumes substantial computational resources for superfluous outputs. Simply adding more tags to control output would lead to a "tag explosion," further complicating the input specification without solving the underlying inflexibility.

To address these issues, *Physica* adopts an object-oriented design in which each computational problem is represented as an object. Input parameters are hardcoded into the program to form computational objects, enabling compile-time optimization and ensuring that only relevant objects are constructed, effectively avoiding tag proliferation. This embedded input approach guarantees "what you see is what you compute," eliminating any disconnect between specification and execution. Furthermore, using a general-purpose programming language offers significantly greater expressivity than domain-specific languages (DSLs), allowing more nuanced and precise descriptions of physical processes.

For data I/O, *Physica* uses HDF5 as its standard format. Inspired by Unix philosophy, the API treats every com-

putational object as readable and writable. Through object-oriented composition, simple computational objects can be encapsulated into more complex structures. Users can selectively compute and output only the objects, or sub-objects, they need, avoiding unnecessary data generation and streamlining result analysis.

Appendix B: Usage guide

Physica is an open-source C++ template library distributed under the GNU General Public License version 3. The source code is publicly accessible on Gitee.[31] For a detailed installation guide, users may refer to the official documentation.[32] In the following, we introduce several core concepts of *Physica* before outlining the steps required to reproduce the results presented in Fig. 1. The complete source code, along with additional usage examples, is provided in the accompanying examples folder.

B1. Scalar and linear algebra

Physica provides a comprehensive implementation of scalar algebra, and further extends this foundation with robust support for differentiable linear algebra,[33] making it suitable for general-purpose scientific computing. Both scalar and linear algebra components are systematically unified using C++20 concepts. Drawing inspiration from established C++ linear algebra libraries such as Eigen[34] and Armadillo,[35] the linear algebra module makes extensive use of template expression techniques to eliminate unnecessary temporary objects and enable compile-time expression optimization. SIMD (Single Instruction Multiple Data) intrinsics[36] are also heavily utilized to improve instruction-level parallelism. Additionally, users can interface with high-performance vendor libraries such as OneMKL[37] and CUDA[38] to further accelerate linear algebra operations and enable GPU offloading.

To begin a simulation, the first step is to select an appropriate scalar type. The real number module, accessible through the corresponding header, provides three floating-point types: `float16`, `float32`, and `float64`. For reasons of numerical stability, we use `float64` throughout this work. A `using namespace` declaration can be employed to conveniently expose these types in the current scope.

```
#include "Physica/Core/Scalar/Real.h"

using namespace Physica;

float64 HoppingT = 1;
float64 RepelU = 8;
```

We are interested in studying the electron density ρ as a function of inverse temperature β. From a numerical perspective, we discretize the imaginary-time axis into multiple intervals and store the corresponding values in an N-dimensional vector. The application programming interface (API) is designed to be consistent with conventions in both NumPy[39] and MATLAB.[40] Starting from an infinite-temperature random quantum state, we gradually "cool" the system to a sufficiently low temperature. In this example, the stopping condition is set to $\beta t = 4$, and the imaginary-time domain is discretized into `40` slices.

```
#include "Physica/Core/Math/Algebra/LinearAlgebra/Vector/DenseVector.h"

int NumBeta = 40 + 1; // Both 0 and 4 are included
auto betas = VectorND<float64>::linspace(0, 4, NumBeta);
```

B2. Modeling of many-body problem

In *Physica*, any many-body problem is modeled through three core concepts: `Representation`, `State`, and `Hamiltonian`. The `Representation` class defines a mapping between elements of the Hilbert space and numerical indices. Currently, the library provides two primary representations: `FermiRepr` for fermionic systems and `SpinRepr` for spin systems. Elements of the Hilbert space are represented as `State` objects, with `FermiState` and `SpinState` corresponding to their respective representations. Efficient indexing of basis configurations is essential for performance in ED type algorithms. To this end, we employ a hash table-based indexing mechanism that ensures $O(1)$ time complexity for state lookups. A Hilbert space can be instantiated by constructing a representation object:

```
#include "Physica/Core/Physics/ManyBody/ReprSpace/FermiRepr.h"

using ReprType = FermiRepr<Dim, NumSite, UseInversionSymm>;
ReprType repr(numSpinUp, numSpinDown);
```

The `FermiRepr` representation is implemented as a template class with three template parameters: the system's dimension, the total number of sites, and a Boolean flag indicating whether to employ inversion symmetry. When `UseInversionSymm` is set to `true` and the system satisfies `numSpinUp == numSpinDown`, memory usage can be reduced by approximately half. States belonging to the representation are automatically generated during object construction.

Once the representation object is constructed, the next step is to define the parameters of the Hamiltonian, including the lattice geometry, boundary conditions, interaction strengths, and other relevant terms. The template class `SquareLattice` accepts one template parameter specifying the spatial dimension of the system. In this example, we consider a one-dimensional Hubbard model with `NumSite` sites and one site per unit cell. We can construct the corresponding Hamiltonian in the given representation:

```
#include "Physica/Core/Physics/ManyBody/Hamilton/HubbardMatrix.h"

using Hamiltonian = HubbardMatrix<float64, ReprType>;
SquareLattice<Dim> lattice({NumSite}, 1);
Hamiltonian H(HoppingT, RepelU, lattice, repr);
```

where we define the precision of the Hamiltonian matrix as `float64`. The resulting Hamiltonian matrix may be calculated on-the-fly or stored in a sparse matrix, which is provided by our linear algebra submodule.

B3. Simulation of TPQ state

Since states under a given representation can be naturally modeled as continuous vectors, the `TPQ` class is inherited from the n-dimensional vector and can be accessed like normal vectors. We construct an infinity temperature TPQ state with the size of the Hilbert space and initialize it with gaussian random numbers:

```
#include "Physica/Core/Physics/ManyBody/TPQ.h"

TPQ<float64> psi(H.getNumState());
psi.random_normal<Random<>>();
```

where `Random<>` is the Mersenne Twister pseudo-random generator. The imaginary time evolution can be carried out as easy as one line of code:

```
psi.nvt_step<Hamiltonian>(H, deltaT);
```

where we evolve the TPQ state by an imaginary time of `deltaT`.

B4. Writing results to HDF5

We adopt HDF5 (Hierarchical Data Format version 5)[41] as the standard output format for numerical simulation results. Every C++ class in *Physica* provides two straightforward member functions, `read` and `write`, which facilitate loading data from and saving data to HDF5 files, respectively.

```
T data{};
auto h5f = H5File::open("data.h5");
data.read(h5f, "x");
// Operations on data...
data.write(h5f, "x");
```

For any data of type `T`, an HDF5 file can be created or opened by calling the static member function `open` from the class `H5File`. Here, the HDF5 file is named 'data.h5'. We then read the dataset labeled 'x' into memory, perform the necessary processing, and ultimately write the updated data back to the file.

B5. Data visualization

We provide native data visualization capabilities using Qt[42] as the backend, supporting a wide range of 2D plots as well as basic 3D visualizations.

```
#include "Physica/Gui/Plot/Plot.h"

QApplication app(argc, argv);
Plot* plot = new Plot(-5, 5, -1.1, 1.1, 2, 0.5);
auto x = VectorND<float64>::linspace(-5, 5, 100);
auto y = tanh(x);
plot->spline(x, y);
plot->show();
QApplication::exec();
```

It is essential to initialize Qt by constructing a `QApplication` object before performing any plotting operations. The constructor of `QApplication` accepts the command-line arguments from the `main` function of a standard C++ program. Once Qt is initialized, a `Plot` object can be created to handle 2D plotting tasks. Its constructor accepts six parameters: the minimum and maximum values of the x-axis, the minimum and maximum values of the y-axis, and the step sizes for the x-axis and y-axis. In the example above, we plot the `tanh` function from -5 to 5. The x-axis range is discretized into 100 equal intervals, and the curve between data points is interpolated using a B-spline algorithm. The plotting process is finalized by invoking the `show()` member function of the `Plot` class, and the Qt backend is instructed to render the plot by calling `QApplication::exec()`.

B6. Debugging

We provide a comprehensive assertion mechanism to assist with error checking, comprising both static and dynamic assertions:

- Static assertions are used to detect errors in template parameter usage. Evaluated at compile time, they allow users

to identify issues early and impose no runtime performance overhead. Static assertions are always enabled.

• Dynamic assertions are employed where static checks are insufficient — only during runtime. These assertions incur a runtime performance cost and are enabled only in Debug mode to minimize overhead in production builds. Users can compile their program in debug mode or define the macro `NDEBUG` to control the enabling of dynamic assertions.

This two-tiered approach ensures robust error detection while maintaining high runtime efficiency in release builds.

Program availability

The source code is publicly accessible on Gitee at https://gitee.com/newsigma/Physica.

Acknowledgements

The authors acknowledge Fu-Zhou Chen for helpful discussions. The work is partly supported by the National Key Research and Development Program of China (Grant No. 2022YFA1402704) and the National Natural Science Foundation of China (Grant No. 12247101).

References

[1] Hirsch J E 1985 *Phys. Rev. B* **31** 4403
[2] Sandvik A W and Kurkijärvi J 1991 *Phys. Rev. B* **43** 5950
[3] Gull E, Millis A J, Lichtenstein A I, Rubtsov A N, Troyer M and Werner P 2011 *Rev. Mod. Phys.* **83** 349
[4] Schollwöck U 2005 *Rev. Mod. Phys.* **77** 259
[5] Verstraete F, Nishino T, Schollwöck U, Bañuls M C, Chan G K and Stoudenmire M E 2023 *Nature Reviews Physics* **5** 273
[6] Bañuls M C 2023 *Annual Review of Condensed Matter Physics* **14** 173
[7] Georges A, Kotliar G, Krauth W and Rozenberg M J 1996 *Rev. Mod. Phys.* **68** 13
[8] Kotliar G, Savrasov S Y, Haule K, Oudovenko V S, Parcollet O and Marianetti C A 2006 *Rev. Mod. Phys.* **78** 865
[9] Aoki H, Tsuji N, Eckstein M, Kollar M, Oka T and Werner P 2014 *Rev. Mod. Phys.* **86** 779
[10] Jaklič J and Prelovšek P 1994 *Phys. Rev. B* **49** 5065
[11] Jaklič J and Prelovšek P 2000 *Advances in Physics* **49** 1
[12] Weinberg P and Bukov M 2017 *SciPost Phys.* **2** 003
[13] Wietek A, Staszewski L, Ulaga M, Ebert P L, Karlsson H, Sarkar S, Shackleton H, Sinha A and Soares R D 2025 arxiv 2505.02901
[14] Ido K, Kawamura M, Motoyama Y, Yoshimi K, Yamaji Y, Todo S, Kawashima N and Misawa T 2024 *Computer Physics Communications* **298** 109093
[15] Assaad F F, Bercx M, Goth F, Götz A, Hofmann J S, Huffman E, Liu Z, Toldin F P, Portela J S E and Schwab J 2022 *SciPost Phys. Codebases* 1
[16] Assaad F F, Bercx M, Goth F, G̈tz A, Hofmann J S, Huffman E, Liu Z, Toldin F P, Portela J S E and Schwab J 2022 *SciPost Phys. Codebases* 1-r2.0
[17] Huang E W, Mendl C B, Liu S X, Johnston S, Jiang H C, Moritz B and Devereaux T P 2017 *Science* **358** 1161
[18] Sugiura S and Shimizu A 2012 *Phys. Rev. Lett.* **108** 240401
[19] Sugiura S and Shimizu A 2013 *Phys. Rev. Lett.* **111** 010401
[20] Hyuga M, Sugiura S, Sakai K and Shimizu A 2014 *Phys. Rev. B* **90** 121110
[21] Jung J H and Noh J D 2020 *Journal of the Korean Physical Society* **76** 670
[22] Liang S 1995 *Computer Physics Communications* **92** 11
[23] Jia C J and Wang Y and Mendl C B and Moritz B and Devereaux T P 2018 *Computer Physics Communications* **224** 81
[24] Lanczos C 1950 *J. Res. Natl. Bur. Stand. B* **45** 255
[25] Al-Mohy A H and Higham N J 2011 *SIAM Journal on Scientific Computing* **33** 488
[26] Higham N J 1990 *SIAM Journal on Scientific and Statistical Computing* **11** 804
[27] Hubbard J and Flowers B H 1967 *Proc. Roy. Soc. Lond. Ser. A Math. Phys. Sci.* **296** 82
[28] Yang S, Park S J and Ousterhout J 2018 *2018 USENIX Annual Technical Conference (USENIX ATC 18)* 335
[29] Sandia Corporation 2003-2025 https://docs.lammps.org/
[30] CP2K Developers 2000-2025 https://manual.cp2k.org/
[31] He W B 2025 https://gitee.com/newsigma/Physica
[32] He W B 2025 https://gitee.com/newsigma/Physica/tree/Physica/doc
[33] Golub G H and Van Loan C F 2013 *Johns Hopkins University Press*
[34] Guennebaud G, Jacob B, *et al.* 2010 http://eigen.tuxfamily.org
[35] Sanderson C and Curtin R 2025 *2025 17th International Conference on Computer and Automation Engineering (ICCAE)* 303
[36] Fog A 2023 https://github.com/vectorclass/version2
[37] Intel Corporation 2025 https://www.intel.cn/content/www/cn/zh/developer/tools/oneapi/onemkl.html
[38] NVIDIA Corporation 2025 https://developer.nvidia.com/cuda-toolkit
[39] Harris C R, Millman K J, van der Walt S and *et al.* 2020 *Nature* **585** 357
[40] The MathWorks, Inc. 1994-2025 https://www.mathworks.com/help/index.html
[41] The HDF Group 2006 https://www.hdfgroup.org/HDF5/
[42] The Qt Company 2025 https://doc.qt.io/qt-6/