

torch_vn_algebra: batched random operators, states, channels and constrained optimisation in finite-dimensional Type I von Neumann algebras

Irina Nikolaeva

Andrej Novikov

Abstract

`torch_vn_algebra` is a PyTorch library for numerical work in algebras $\mathcal{M} = \bigoplus_{c=1}^C M_{k_c}(\mathbb{C})$, the finite-dimensional Type I von Neumann algebras that describe quantum systems with superselection rules. Operators, states and channels are stored as batched, block-diagonal tensors, so that thousands of random samples, or thousands of independent starts of an optimisation, are processed at once on a CPU or a GPU. The library provides random operators with prescribed spectra and Haar-distributed eigenvectors, the three trace functionals that are natural on $\mathcal{M}$, states and Lüders conditioning, completely positive maps within and between sectors, plain and charge-conserving (fused) tensor products with partial traces, spin chains with a conserved magnetisation and momentum, sparse Lanczos and Krylov solvers for large sectors, Hamiltonian and Lindblad dynamics (including jumps between sectors), differentiable parametrisations with a prescribed Michelson contrast, and safeguards that estimate run time and memory before expensive steps. It also relates algebras to each other and to their symmetries: inclusions of algebras with conditional expectations, Jones, Watatani and Pimsner–Popa indices and the basic construction; a numerical Wedderburn decomposition of the algebra generated by given matrices or by a finite group; the $SU(2)$ multiplet structure of spin- $\frac{1}{2}$ chains; and Petz and sandwiched Rényi divergences with Petz recovery maps. We describe the design, show how the library is used on short complete examples, and validate it against exact results from random matrix theory, quantum information and integrable spin chains (free-fermion spectra and dynamics, Heisenberg ring energies, Peschel’s entanglement formula, the GOE–Poisson crossover of disordered chains). Applied to the trace inequalities of our earlier work, constrained optimisation replaces Monte Carlo sampling: it finds extremal values that sampling misses in all but the smallest dimensions, and reproduces the bounds that are known in closed form. Without being told where they lie, the same method brackets the data-processing thresholds $\alpha = \frac{1}{2}$ and $\alpha = 2$ of the sandwiched and Petz Rényi divergences, with the violations certified in high-precision arithmetic.

Program summary

Program title: `torch_vn_algebra`

Developer’s repository link: https://github.com/andre-a-no/von_neumann_type_i

Licensing provisions: MIT

Programming language: Python (≥ 3.9), PyTorch (≥ 2.0)

Nature of problem: Numerical experiments in finite-dimensional Type I von Neumann algebras $\bigoplus_c M_{k_c}(\mathbb{C})$, which describe quantum systems with superselection rules or conserved charges: random operators with prescribed spectra, several trace functionals, states, channels within and between sectors, symmetric tensor products, spin chains, Hamiltonian and Lindblad dynamics, extremal problems for trace inequalities under spectral constraints, inclusions of algebras, symmetry reduction, and quantum Rényi divergences.

Solution method: Batched block-diagonal tensors processed by PyTorch linear algebra on CPUs and GPUs; exact Haar sampling; spectral functional calculus; Kraus representations; Lanczos and Krylov methods for sparse sector Hamiltonians; differentiable parametrisations with batched multistart gradient optimisation; run-time and memory estimates before expensive steps; inclusion matrices; randomised Wedderburn decomposition from random elements of the commutant; highest-weight (Yamanouchi) bases and exact Clebsch–Gordan coefficients.

Additional comments: about 800 unit tests, run in continuous integration; every number, table and figure of this paper is produced by the scripts and examples in the repository and regenerated by `scripts/run_all.sh`.

1 Introduction

Where block-diagonal algebras arise. Whenever a conserved quantity, a symmetry or a superselection rule splits a Hilbert space into sectors, the physically relevant operators are block diagonal. Their algebra is a finite-dimensional Type I von Neumann algebra

$$\mathcal{M} = \bigoplus_{c=1}^C M_{k_c}(\mathbb{C}) \quad \text{acting on} \quad \mathcal{H} = \bigoplus_{c=1}^C \mathbb{C}^{k_c}, \quad (1)$$

where C is the number of sectors (direct summands) and $M_k(\mathbb{C})$ is the algebra of complex $k \times k$ matrices. Every finite-dimensional von Neumann algebra is $*$ -isomorphic to one of this form [18], and its centre is spanned by the sector projections. The library also accepts nominal factor sizes $n_c \geq k_c$: this describes computations in the corner $p\mathcal{M}_0p \cong \bigoplus_c M_{k_c}(\mathbb{C})$ of a larger algebra $\mathcal{M}_0 = \bigoplus_c M_{n_c}(\mathbb{C})$, with $p = \bigoplus_c p_c$ and p_c a projection of rank k_c . The corner is again of the form (1); all operators, traces and normalisations refer to it, and in most applications $k_c = n_c$.

From a symmetry to the algebra. The link between physics and (1) is the commutant. If a group G (or a set of conserved quantities) acts unitarily on $\mathcal{H}$ and $\mathcal{H} \cong \bigoplus_{\lambda} V_{\lambda} \otimes \mathbb{C}^{m_{\lambda}}$ is its decomposition into irreducible representations V_{λ} with multiplicities m_{λ} , then the operators commuting with the action form the algebra $\bigoplus_{\lambda} \mathbb{K}_{V_{\lambda}} \otimes M_{m_{\lambda}}(\mathbb{C}) \cong \bigoplus_{\lambda} M_{m_{\lambda}}(\mathbb{C})$. For one conserved self-adjoint charge Q (particle number, magnetisation, Hamming weight) the irreducibles are one-dimensional and m_{λ} is the dimension of the eigenspace of Q with eigenvalue λ . A Hamiltonian that conserves Q , a time evolution generated by it, and every observable compatible with a superselection rule for Q lie in this commutant; the sectors are the eigenvalues of Q , and the centre consists of the functions of Q . Physical modelling with a symmetry is therefore computation in an algebra of the form (1); for non-abelian symmetries the block sizes m_{λ} are smaller than the dimensions of the charge sectors, and Section 8 computes the decomposition numerically from the group or its generators.

Examples are abundant:

- superselection rules for charge, parity or particle number [19, 20] and the classical-quantum structure produced by decoherence and einselection [21];
- many-body Hamiltonians with conserved charges, whose exact diagonalisation and dynamics are carried out sector by sector [22, 23], and symmetry-resolved entanglement [24, 25];
- quantum information: decoherence-free subspaces and noiseless subsystems [26, 27], operator-algebra quantum error correction [28], and quantum algorithms that preserve a conserved quantity, such as the XY mixers of the quantum alternating operator ansatz, which keep the Hamming weight [29];

- random matrix theory, where symmetries decompose ensembles into blocks and spectral statistics are meaningful only inside a block [30, 12];
- machine learning: by Schur’s lemma the linear maps that commute with a (complex, unitary) group representation (equivariant layers) on one space form an algebra of the form (1), whose block sizes are the multiplicities of the irreducible representations, and equivariant layers between two representations are the corresponding off-diagonal blocks [31, 32] (for real representations the blocks can also be real or quaternionic); random-matrix methods are used to analyse trained weights [33];
- operator theory, our original motivation: inequalities between traces of products characterise the trace and the centre of an operator algebra [2, 3, 4, 5], and their failure can be quantified by spectral quantities such as the Michelson contrast [6, 7].

On $\mathcal{M}$ the trace is not unique: every positive weighting of the sectors gives one, and statements such as “trace preserving” or “the dual map” depend on that choice as soon as maps change the sector structure.

What existing software offers. Table 1 compares the features relevant here. Dense and sparse linear algebra (NumPy/SciPy [34, 35], with ARPACK [36] for sparse eigenproblems) handles single matrices; QuTiP [8] and the quantum-information module of Qiskit [9] represent states, operators and channels on one Hilbert space; **dynamiqs** [10] adds batching, GPUs and automatic differentiation for Schrödinger and Lindblad dynamics; QuSpin [23] builds symmetry-reduced bases for exact diagonalisation; tensor-network libraries (ITensor [37], TeNPy [38]) store charge-conserving tensors in block-sparse form; RTNI [39] integrates Haar-random tensor networks symbolically. None of them treats the algebra (1) itself, with its centre, several traces, states and maps between algebras of different sector structure, as the basic object, and none combines it with batched sampling and optimisation over operators with prescribed spectra.

	NumPy/ SciPy	QuTiP	Qiskit q. info	dynamiqs	QuSpin	ITensor/ TeNPy	this work
GPU execution	–	(✓)	–	✓	–	(✓)	✓
batched over many operators	(✓)	–	–	✓	–	–	✓
automatic differentiation	–	(✓)	–	✓	–	(✓)	✓
symmetry sectors of many-body systems	–	–	–	–	✓	✓	✓
algebra $\bigoplus_c M_{k_c}$ with centre and several traces	–	–	–	–	–	–	✓
channels within and between sectors	–	(✓)	(✓)	–	–	–	✓
prescribed spectra, Haar/COE/CSE per sector	(✓)	(✓)	(✓)	–	–	–	✓
optimisation under spectral constraints	–	–	–	–	–	–	✓
large single systems	✓	✓	✓	✓	✓	✓	(✓)

Table 1: Features relevant to computations in the algebra (1), to the best of our knowledge at the time of writing (✓: supported; (✓): partly, through plug-ins, or with user code; –: not supported). The last row is where the specialised packages are the better choice.

Why not one large matrix? Nothing forces one to treat $\mathcal{M}$ as a separate object: it embeds into $M_D(\mathbb{C})$, $D = \sum_c k_c$, as the block-diagonal matrices, and every package above handles a single $D \times D$ matrix (the case $C = 1$ of (1)). The embedding is exact for linear algebra, so the question is only what it costs and what has to be supplied by hand.

- *Cost.* A product, an inverse or a spectral decomposition in M_D costs $O(D^3)$ time and $O(D^2)$ memory, in $\mathcal{M}$ it costs $\sum_c k_c^3$ and $\sum_c k_c^2$. For C equal blocks this is a factor C^2 in time and C

in memory; for the half-filling sector of a chain of $L = 16$ spins ($D = 2^{16}$, largest block 12870) the dense M_D computation is out of reach, the block computation is routine.

- *Measures.* The unitary group of $\mathcal{M}$ is $\prod_c U(k_c)$, for $C \geq 2$ a subgroup of $U(D)$ of measure zero. A Haar-random element of $U(D)$ is not in $\mathcal{M}$, and a random element of $\mathcal{M}$ has to be produced block by block. The same holds for random states, for the circular ensembles and for any optimisation: an unconstrained gradient step in M_D leaves $\mathcal{M}$.
- *Traces.* On M_D the trace is unique up to a factor; on $\mathcal{M}$ the faithful traces form a C -parameter family $\sum_c w_c \text{Tr}_c$. Densities, “trace preserving” and the dual of a map between algebras with different sectors depend on the choice of the weights; on the embedded matrices this information is no longer visible.
- *Centre.* The centre of $\mathcal{M}$ (the sector projections) is what physicists call the classical or superselected part of the system; conditioning on a sector, post-selection and sector-resolved quantities are operations with the centre, and they have to be reconstructed from the block indices when $\mathcal{M}$ is stored as one matrix.

The library keeps these four pieces of structure inside the data type. For $C = 1$ it reduces to batched linear algebra in $M_k(\mathbb{C})$, and all its functions accept that case. The first version of this library [1] shows that the bookkeeping is error-prone: its complex Haar sampler was not Haar distributed, its circular ensembles were not the circular ensembles, and several conclusions drawn from Monte Carlo samples were wrong; all of this is corrected and tested here.

Challenges. Turning the algebra into an efficient and reliable tool raises concrete problems, each addressed by a design decision described in Section 3: (i) heterogeneous block sizes have to be mapped onto batched GPU kernels; (ii) trace conventions and dual maps must stay consistent when maps change the sectors; (iii) random ensembles must be exactly Haar (or COE/CSE) in every block; (iv) gradients propagated through eigen- or singular-vector decompositions (as in $f(A)$ or $|X|$) are numerically singular at degenerate spectra, exactly where extremal operators tend to lie; (v) extremal values sit on sets of tiny measure that random sampling does not reach in higher dimensions (Section 6); (vi) batched workloads easily exceed the time or memory budget of a device; (vii) the number field (real or complex) and the precision change both cost and results (Section 3).

Objective and scope. `torch_vn_algebra` is a library for numerical experiments *in* the algebra (1): sampling and optimising operators, states and channels, evaluating trace and spectral functionals, and formulating physical models (spin chains, open systems, noisy symmetric quantum circuits) whose symmetry is encoded in the sector structure. It is not a circuit simulator and not a tensor-network code, and for single large systems the specialised packages above remain the better choice; Section 9 quantifies where the library gains and where it does not.

Contributions and outline. Section 2 fixes notation and conventions. Section 3 describes the software stack, the workflow, how the challenges (i)–(vii) are met, and the choice of number field and precision. Section 4 shows the library on short complete programs, including a noisy quantum circuit. Section 5 validates it against exact results and checks the trace functionals. Section 6 replaces Monte Carlo sampling by constrained optimisation for the trace inequalities of [1] and, in Section 6.1, locates the data-processing thresholds of the Rényi divergences. Section 7 covers charge-conserving tensor products and spin chains, including momentum sectors, sparse Krylov solvers and particle loss. Section 8 treats inclusions of algebras, conditional expectations and indices, the Wedderburn decomposition of the algebra generated by a symmetry, and the $SU(2)$ symmetry of spin chains.

Section 9 compares with NumPy/SciPy, Section 10 reports performance, and Section 11 discusses limitations.

2 Mathematical setting

Algebra and traces. Let $\mathcal{M} = \bigoplus_{c=1}^C M_{k_c}(\mathbb{C})$ act on $\mathcal{H} = \bigoplus_c \mathbb{C}^{k_c}$; $A \in \mathcal{M}$ is block diagonal, $A = \bigoplus_c A_c$. Every trace on $\mathcal{M}$ is $\text{tr}_w(A) = \sum_c w_c \text{Tr } A_c$ with weights $w_c \geq 0$, and it is faithful iff all $w_c > 0$. The library implements

$$\text{Tr}_{\text{blunt}} : w_c = 1, \quad \text{Tr}_{\text{norm}} : w_c = \frac{1}{k_c}, \quad \tau_{\text{vN}} : w_c = \frac{1}{Ck_c},$$

where τ_{vN} is the faithful normal tracial state that gives every sector the same weight (the library allows empty sectors, $k_c = 0$; they carry no weight, and C in τ_{vN} counts the non-empty ones). For $A \in \mathcal{M}$, $|A| = (A^*A)^{1/2}$; for $A \geq 0$ the Michelson contrast is $\Delta(A) = (\lambda_{\max} - \lambda_{\min})/(\lambda_{\max} + \lambda_{\min}) \in [0, 1]$, with the extreme eigenvalues taken over all sectors, for $A \neq 0$ (the library returns $\Delta(0) = 0$); for $A \geq 0$, $A \neq 0$, $\Delta(A) = 0$ if and only if A is a multiple of the identity.

States. A state ω on $\mathcal{M}$ is represented by its density $\rho \geq 0$ with respect to Tr_{blunt} , $\omega(A) = \text{Tr}(\rho A)$, $\sum_c \text{Tr } \rho_c = 1$. The restriction of ω to the centre is the classical distribution $p_c = \text{Tr } \rho_c$ over sectors. The density of the same state with respect to tr_w is $\rho_c^{(w)} = \rho_c/w_c$; the library converts between the conventions explicitly, because statements such as “trace preserving” or “the dual map” depend on them as soon as the sectors change.

Maps between sectors. Let $\mathcal{M}_{\text{in}} = \bigoplus_c M_{k_c}$ and $\mathcal{M}_{\text{out}} = \bigoplus_d M_{m_d}$. Every normal completely positive map $\Phi : \mathcal{M}_{\text{in}} \rightarrow \mathcal{M}_{\text{out}}$ (in the Schrödinger picture) is a sum of completely positive maps between blocks,

$$\Phi(\rho)_d = \sum_c \sum_i K_i^{dc} \rho_c (K_i^{dc})^*, \quad K_i^{dc} : \mathbb{C}^{k_c} \rightarrow \mathbb{C}^{m_d}.$$

It preserves Tr_{blunt} iff $\sum_{d,i} (K_i^{dc})^* K_i^{dc} = 1_c$ for all c , and $T_{dc} = \text{Tr } \Phi_{dc}(\rho_c) / \text{Tr } \rho_c$ are the transition probabilities between sectors for the input state ρ (defined for $\text{Tr } \rho_c > 0$) (independent of ρ iff every $\sum_i (K_i^{dc})^* K_i^{dc}$ is a multiple of 1_c ; by default the library uses the maximally mixed state of each input sector). The dual with respect to tr_w on both sides is $\Phi^{*,w}(A)_c = \sum_{d,i} (w_d^{\text{out}}/w_c^{\text{in}}) (K_i^{dc})^* A_d K_i^{dc}$; for sector-preserving maps on one algebra the weights cancel and all three traces give the same dual.

Tensor products and fusion of sectors. $(\bigoplus_c M_{k_c}) \otimes (\bigoplus_d M_{m_d}) = \bigoplus_{(c,d)} M_{k_c m_d}$ is again of Type I with sectors labelled by pairs. If the sectors carry conserved charges q_c, q'_d (particle numbers, magnetisation), the operators of the composite system that conserve only the *total* charge form the larger algebra

$$\mathcal{M}_1 \hat{\otimes} \mathcal{M}_2 = \bigoplus_q M_{n_q}, \quad n_q = \sum_{q_c + q'_d = q} k_c m_d,$$

in which all pairs with the same total charge are merged into one sector. It contains $\mathcal{M}_1 \otimes \mathcal{M}_2$ (block-diagonally in every merged sector), but also operators such as $S_i^+ S_j^-$ that move charge between the factors. For additive charges, and distinct charges on distinct sectors of $\mathcal{M}_1$, the partial trace $(\text{Tr}_2 X)_c = \sum_d \text{Tr}_{m_d} X_{(c,d),(c,d)}$ still lands in $\mathcal{M}_1$: a block between pairs $(c,d) \neq (c',d)$ with equal total charge would need $q_c = q_{c'}$, hence $c = c'$. In both cases the partial trace is a map between sectors in the sense above, and its dual with respect to Tr_{blunt} is the embedding $A \mapsto A \otimes 1$.

Spin chains. For L spins $1/2$ with conserved S_{tot}^z , iterating the fused product of single-site algebras $M_1 \oplus M_1$ (charges $0, 1$) gives $\mathcal{M} = \bigoplus_{N=0}^L M_{\binom{L}{N}}(\mathbb{C})$, N the number of up spins, which contains every S_{tot}^z -conserving Hamiltonian, e.g. the XXZ chain $H = \sum_{\langle ij \rangle} [\frac{J}{2}(S_i^+ S_j^- + S_i^- S_j^+) + J\Delta S_i^z S_j^z] + \sum_i h_i S_i^z$ (the anisotropy Δ is unrelated to the Michelson contrast $\Delta(\cdot)$).

Dynamics. For $H = H^* \in \mathcal{M}$ and jump operators $L_j \in \mathcal{M}$ the Gorini–Kossakowski–Sudarshan–Lindblad equation $\dot{\rho} = -i[H, \rho] + \sum_j \gamma_j (L_j \rho L_j^* - \frac{1}{2}\{L_j^* L_j, \rho\})$ preserves every sector, so the sector probabilities p_c are conserved (superselection). Jumps between sectors, such as particle loss, are described by maps between sectors as above (Section 7).

3 Design and implementation

Software stack. The library is pure Python on top of PyTorch [64] (≥ 2.0) and NumPy [34]. All dense linear algebra goes through `torch.linalg`, i.e. LAPACK (MKL or OpenBLAS) on CPUs and cuSOLVER, cuBLAS and MAGMA on NVIDIA GPUs; sparse products use PyTorch’s CSR tensors. Automatic differentiation is PyTorch’s reverse mode. The validation and paper scripts additionally use SciPy, pandas, Matplotlib and mpmath (for the high-precision certification of Section 6.1). Packaging follows `pyproject.toml`; the test suite runs in continuous integration on Python 3.10 and 3.12 (CPU), and the paper scripts state the device and mode in the table captions and record the matmul precision, the PyTorch, CUDA and cuDNN versions and the compute capability in a JSON file next to every table.

Workflow. A computation proceeds in four steps. (1) An algebra is declared by its sector sizes, optionally sector charges, number field, precision and device (`TypeIAlgebra`, or a model such as `SpinChain` that creates it). (2) Objects are created in it: random operators with prescribed spectra, states, channels, Hamiltonians, or differentiable parametrisations. (3) They are combined lazily (products, functional calculus, channel application, tensor products, time evolution). (4) Functionals are evaluated for the whole batch at once (traces, spectra, contrasts, entropies, sector probabilities) and reduced to statistics, or fed into an optimiser that updates the parameters of step (2). Every step keeps the batch dimension, so the same code serves one operator and 10^5 samples.

challenge	solution in the library
(i) blocks of different size on batched kernels	padded $(B, C, k_{\max}, k_{\max})$ layout; products and traces in one batched call, decompositions in one batched call per distinct block size; single-sector algebras and sparse sector Hamiltonians where padding would dominate
(ii) conventions between sectors	states stored as Tr_{blunt} -densities with explicit conversions; duals of maps between sectors computed with the weights of the chosen trace
(iii) exact random ensembles	Mezzadri’s QR algorithm with phase correction, $\det^{-1/n}$ for $SU(n)$, $W^T W$ and $W^R W$ for COE/CSE, validated against Weingarten moments and spacing statistics
(iv) singular gradients at degenerate spectra	functional calculus differentiated by the Daleckiĭ–Krein formula (divided differences, finite at degenerate spectra); parametrisations that build operators from their spectra in the optimisation, exact $ X $ from the parametrisation
(v) extremes missed by sampling	batched multistart optimisation with exact contrast constraints
(vi) time and memory budgets	probing of batched kernels with an ETA warning, memory estimates checked against the free device memory, ETA for time stepping
(vii) field and precision	two switches per algebra, conversions between them, measured trade-offs

Table 2: Implementation challenges (Section 1) and how they are addressed.

Data layout. An operator is a tensor of shape $(B, C, k_{\max}, k_{\max})$: B samples, C sectors, block c in the top-left $k_c \times k_c$ corner and zeros elsewhere. Linear algebra is applied to the batch of active blocks of each sector, so one call to a batched LAPACK/cuSOLVER routine serves all samples. The padding costs memory when the sector sizes differ strongly (for a chain of 14 spins, 77 % of a padded operator is zeros); the library warns in that case, and `SpinChain` lets one work in single sectors $M_{(N)}^{(L)}$ without padding, which is what dense diagonalisation needs anyway.

Objects. `Operator` (with `DensityMatrix` as a subclass for states), `Channel` (sector preserving) and `InterSectorChannel` (between sectors or algebras) are thin wrappers around such tensors; Table 3 lists the modules. Operations on operators are lazy: they record how to compute the result and do so on first use; the result is cached and the recipe dropped, so intermediates of long expressions can be freed. Laziness defers and skips work but does not fuse operations; where this matters the library uses direct formulas, e.g. $\text{Tr}(\rho A) = \sum_{ij} \rho_{ij} A_{ji}$ in $O(k^2)$ instead of forming ρA .

module	content
<code>algebra</code>	<code>TypeIAlgebra</code> , <code>Operator</code> : random operators, unitary ensembles, traces, functional calculus
<code>states</code>	<code>DensityMatrix</code> , random states, Gibbs states, Lüders rule, entropies, fidelity
<code>channels</code>	<code>Channel</code> , <code>InterSectorChannel</code> , Choi matrices, standard and random channels
<code>composite</code>	plain and fused (charge-conserving) tensor products, <code>kron</code> , partial traces
<code>chains</code>	<code>SpinChain</code> : $U(1)$ and momentum sectors, XXZ Hamiltonians with batched disorder, sparse form, reduced states, jump operators between sectors
<code>krylov</code>	sparse sector Hamiltonians, batched Lanczos ground states, Krylov $e^{-itH}\psi$
<code>dynamics</code>	exact Schrödinger/von Neumann evolution, RK4, Lindblad equation, $e^{t\mathcal{L}}$
<code>functional</code>	functional calculus: self-adjoint, normal, singular values, holomorphic (contour); spectral cache
<code>optimize</code>	parametrisations with prescribed contrast, batched multistart <code>extremize</code>
<code>cost</code>	run-time and memory estimates, warnings and errors
<code>hilbert_space</code>	the padded block layout of $\bigoplus_c \mathbb{C}^{k_c}$ and its random subspace bases
<code>inclusions</code>	<code>Inclusion</code> : inclusions $\mathcal{N} \subset \mathcal{M}$ by inclusion matrices, conditional expectations, Markov trace, Jones, Watatani and Pimsner–Popa indices, basic construction, spin-chain tower
<code>decompose</code>	numerical Wedderburn decomposition of the algebra generated by matrices or a finite group; commutants; real, complex and quaternionic types
<code>su2</code>	<code>SU2Chain</code> : $SU(2)$ multiplets of spin- $\frac{1}{2}$ chains, highest-weight bases, Clebsch–Gordan coefficients, Wigner–Eckart theorem
<code>divergences</code>	Petz and sandwiched Rényi divergences and their limits, Petz and universal recovery maps, L^p and Kosaki norms

Table 3: Modules of `torch_vn_algebra`.

Random unitaries. Haar unitaries on $U(n)$ and $O(n)$ are obtained from the QR decomposition of a Ginibre matrix with i.i.d. complex (respectively real) Gaussian entries after fixing the phases (signs) of $\text{diag } R$ [11]; $SU(n)$ and $SO(n)$ by multiplying with $\det^{-1/n}$ or by a sign flip of one column; the circular orthogonal and symplectic ensembles as $W^T W$ and $W^R W$ with W Haar on $U(n)$ (n even for $W^R W$, W^R the quaternion dual) [12]. Section 5 checks all of them.

Functional calculus. The module `functional` provides $f(A)$ at four levels of generality, all built on batched kernels, one call per distinct block size [40]. (1) For self-adjoint A , $f(A) = V f(\Lambda) V^*$

from the Hermitian eigensolver, for any function on the real spectrum. (2) For normal $A = H + iK$ (unitaries), H and K commute, and the eigenvectors of the Hermitian matrix $H + \alpha K$ diagonalise A for all but finitely many α ; the library tries a few irrational values, refines the eigenvectors of nearly tied values with a second coefficient and corrects the remaining residual to first order, so normal operators also need only the Hermitian eigensolver (in single precision this decomposition is computed in double precision). (3) Singular values, $|A|$, the pseudo-inverse and the polar decomposition come from the Hermitian eigensolver of the dilation $\begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix}$, whose eigenvalues are $\pm s_i$: batched SVD on GPUs is limited to blocks of size at most 32, while the batched Hermitian eigensolver is not. (4) For arbitrary A and f holomorphic near the spectrum, $f(A) = (2\pi i)^{-1} \oint f(z)(z - A)^{-1} dz$ is evaluated by the trapezoidal rule on a circle [41], i.e. by batched LU solves, on a circle just around the spectrum, with the number of nodes doubled until two results agree; the error estimate includes the rounding error of cancelling terms, and an unconverged result raises an error instead of returning a wrong matrix. An opt-in cache keeps the eigendecomposition of an operator, so that many functions of it (for example e^{-itH} for many t) cost matrix products only.

Differentiation. All operations are differentiable. Gradients propagated naively through eigenvectors contain factors $1/(\lambda_i - \lambda_j)$ and fail at degenerate spectra, which are typical here (projections, symmetric operators, extremal operators). The functional calculus $f(A) = Vf(\Lambda)V^*$ therefore has its own backward pass: by the Daleckii–Krein formula [42, 43], the derivative of f at a self-adjoint A is $dF = V(L \circ V^*dAV)V^*$ with the Loewner matrix of divided differences $L_{ij} = (f(\lambda_i) - f(\lambda_j))/(\lambda_i - \lambda_j)$, replaced by f' for coinciding eigenvalues, which is finite whenever f is differentiable on the spectrum (see also [44]). Gradients with respect to parameters of f (a time in e^{-itA}) are propagated as usual. The same applies to $\sqrt{A}$, $\log A$, A^p , and to $|A|$ and the inverse of self-adjoint A , which use the Hermitian eigensolver instead of the SVD. For normal A the same formula with complex divided differences is the derivative of the holomorphic calculus, for holomorphic f . $|A|$ of a general A is differentiated as the square root of A^*A , with $L_{ij} = 1/(s_i + s_j)$, and the pseudo-inverse by the constant-rank formula; the contour integral is differentiated through the solves. Only gradients through the eigenvectors returned by `eigh` keep the factors $1/(\lambda_i - \lambda_j)$; the parametrisations of Section 6 avoid them. `Operator.eigenvalues()` computes spectra without forming eigenvectors, which also saves time and memory.

GPUs, tensor cores and TF32. The library computes in FP32 or FP64 (real or complex); it does not use half precision. GPU generations differ strongly in what this costs. Tensor cores, absent on the P100 used for the first version, accelerate matrix products on Volta and later GPUs; from Ampere (A100) on, FP32 products may be executed in the TF32 format with a 10-bit mantissa, which is fast but accurate only to about 10^{-3} relative. PyTorch (since 1.12) keeps true FP32 for matrix products by default, and the library does not change this setting; the paper scripts expose a `-tf32` switch to measure the effect and record the setting in every table. TF32 is acceptable where the statistical error dominates (moments and histograms of many samples) and not where small differences matter (eigenvalues near degeneracies, the validation tables, trace inequalities near equality). Eigen- and singular-value solvers do not use tensor cores. FP64 throughput is half of FP32 on data-centre GPUs (P100, V100, A100, H100), the latter two with FP64 tensor cores, but only 1/32–1/64 on consumer GPUs. We therefore recommend a data-centre GPU for double-precision work. The full-size runs of this paper were made on a consumer GPU (the tables state the hardware); the double-precision timings there are therefore pessimistic for the GPU.

Number field and precision

Every algebra is created with two independent switches, `complex_valued` (real or complex matrices) and `precision` ('single' or 'double'), giving the storage types `float32`, `float64`, `complex64` and `complex128`; everything built on the algebra (random operators, states, channels, chains, parametrisations) inherits them. `alg.like(complex_valued=..., precision=...)` creates the same sector structure with other numerics and `op.cast(alg2)` moves an operator there, e.g. to explore in single precision and re-evaluate the interesting cases in double precision.

The field is part of the model. Choosing real matrices is not only a matter of speed: the symmetry group and the random ensembles change. Haar matrices of a real algebra are orthogonal, so moments, level statistics and extremal values refer to $O(n)$ instead of $U(n)$ (e.g. $\mathbb{E}|U_{11}|^4 = 3/(n(n+2))$ instead of $2/(n(n+1))$, Table 6); random real symmetric and complex Hermitian matrices have different level statistics, with mean spacing ratios 0.5307 and 0.5996 for GOE and GUE [17] (our samples: 0.5313 and 0.5993). Real arithmetic is *exact* (loses nothing) when the problem is real: spectra and eigenvectors of real symmetric matrices, ground states and their entanglement for Hamiltonians without complex hoppings, statistics of time-reversal-invariant systems, and also some extremal problems — for real $X, Y \geq 0$ the supremum of $|\text{Tr}(XUY)|$ over unitaries is attained by a real orthogonal matrix (the polar decomposition of the real matrix YX is real). Complex matrices are *required* for unitary time evolution e^{-iHt} and Lindblad dynamics with a Hamiltonian (the library refuses to run them in a real algebra), for Haar unitaries, CUE/CSE and $SU(n)$, for random pure states and quantities such as Page’s entropy, and for systems without an antiunitary symmetry squaring to +1 (orbital fluxes, complex couplings).

Precision. Single precision resolves about 10^{-7} relative to the largest entry. It is sufficient for Monte Carlo statistics whose statistical error is much larger (moments, histograms, spacing ratios with 10^3 – 10^5 samples) and for the first, exploratory pass of an optimisation. Double precision is needed when the quantity of interest is small compared with the operators: an inequality checked near equality, small spectral gaps, entropies and logarithms of nearly singular states, contrasts close to 1, comparisons with exact results beyond six digits, long RK4 integrations and optimisation to high relative accuracy. Table 5 shows how drastic the difference can be: the von Neumann entropy of an almost pure state with eigenvalues of order 10^{-9} is wrong by more than 100 % in single precision, and the round-off floor of a trace inequality that holds with equality is several orders of magnitude higher in single than in double precision (Table 5).

Cost. An entry takes 4, 8, 8 and 16 bytes for `float32`, `float64`, `complex64` and `complex128`, so memory, and the batch that fits, changes by a factor up to four (`alg.bytes_per_operator(B)` reports it). A complex multiply–add costs about four real ones, which bounds the slowdown of complex matrix products; for eigen- and singular-value solvers, and for double against single precision on CPUs, the measured ratios are smaller (Table 4). On GPUs double precision depends on the hardware: data-centre GPUs execute FP64 at half the FP32 rate, consumer GPUs at 1/32–1/64 of it, so that double precision there can be an order of magnitude slower. Table 4 gives the measured ratios for the current run; the recommendation that follows is to use the smallest type that is exact for the model (real if the problem is real) and the precision that the size of the effect requires.

Safeguards. Batched computations make it easy to request hours of work or more memory than the device has. The `cost` module therefore (a) times the first few items of every large batched eigen-, singular-value or QR decomposition and matrix exponential and issues a warning with the

type	B/entry	MB/batch	generate	eigh f(A)	SVD A	channel	expm	opt. step
float32	4	16	1	1	1	1	1	1
float64	8	32	1.2	7.7	21	3.7	2.1	3.4
complex64	8	32	1.1	1.6	1.8	1.8	1.5	1.8
complex128	16	64	3.2	11	65	13	5.4	11

Table 4: Bytes per entry, memory of one batch [MB], and time relative to `float32` for $C = 16$ sectors of size $k = 32$ and a batch of 256 operators (full run on NVIDIA GeForce RTX 5090; float32 times per sample: generate: $918 \mu\text{s}$, eigh f(A): $7.24 \mu\text{s}$, SVD |A|: $11.4 \mu\text{s}$, channel: $1.83 \mu\text{s}$, expm: $4.96 \mu\text{s}$, opt. step: $70.9 \mu\text{s}$).

type	$\max \ UU^* - 1\ $	$\ \sqrt{A}^2 - A\ /\ A\ $	rel. error $S(\rho)$, $\rho \approx \text{pure}$	$ z $ at equality
float32	6.6×10^{-7}	2.0×10^{-6}	2.0×10^0	5.3×10^{-4}
float64	1.3×10^{-15}	4.0×10^{-15}	6.6×10^{-9}	1.4×10^{-12}
complex64	8.3×10^{-7}	2.3×10^{-6}	2.3×10^0	2.0×10^{-4}
complex128	1.6×10^{-15}	4.5×10^{-15}	8.9×10^{-9}	1.8×10^{-12}

Table 5: Accuracy of the same computations in the four storage types: Haar unitaries ($n = 64$), SVD square root, entropy of $\rho = (1 - \varepsilon)|0\rangle\langle 0| + \varepsilon 1/n$ with $\varepsilon = 10^{-7}$, and the round-off floor of $z = \text{Tr}(Y|X|Y) - \text{Tr}|YXY|$ for commuting X, Y (exactly 0); full run on NVIDIA GeForce RTX 5090.

expected total time when it exceeds a threshold (10s by default), (b) compares the estimated peak memory of large allocations with the free memory of the device, warning above 50 % and raising an `InsufficientMemoryError` above 90 %, and (c) after the second step of a time-stepping or optimisation loop, warns with the expected duration if it exceeds the same threshold; a progress bar is optional. All thresholds can be changed, and the checks can be switched off locally.

4 Usage by example

The listings of this section are complete programs from `man/article/listings`; the test suite runs them, so they stay in sync with the code. Each runs on a CPU in under half a minute and uses a GPU when available.

Random operators with a prescribed contrast. Listing 1 builds 10^4 positive operators in $M_2 \oplus M_3 \oplus M_4$ whose spectra have contrast 0.6 and whose eigenvectors are Haar-random in every sector, and evaluates traces and spectral functions.

Listing 1: Random operators (`ex1_random_operators.py`).

```

"""Random operators with a prescribed spectrum in M = M_2 (+) M_3 (+) M_4."""
import torch
from torch_vn_algebra import TypeIAlgebra

torch.manual_seed(0)                                # reproducible output
dev = 'cuda' if torch.cuda.is_available() else 'cpu'
alg = TypeIAlgebra(n_factors=[2, 3, 4], k_factors=[2, 3, 4], device=dev)
B = 10_000                                           # Monte Carlo samples, processed as one batch

def contrast_sampler(delta):

```

```

    """Spectra in [lo, 1] with lambda_max = 1, lambda_min = lo, i.e. Michelson contrast delta."""
    lo = (1 - delta) / (1 + delta)

    def sampler(dim):
        lam = lo + (1 - lo) * torch.rand(B, dim, device=dev)
        lam[:, 0], lam[:, -1] = 1.0, lo
        return lam
    return sampler

# A_c = U_c diag(lambda) U_c^*, with independent Haar unitaries U_c in every block and sample
X = alg.operator_from_eigenvalues(contrast_sampler(0.6), batch_size=B,
                                force_positive=True, force_self_adjoint=True)
print(X.matrix.shape) # (10000, 3, 4, 4): batch, sector, padded block
print(X.michelson_contrast[:4]) # 0.6 for every sample
print(X.Tr_blunt().real.mean(), X.Tr_norm().real.mean(), X.tau_vN().real.mean())

# functional calculus through the spectral theorem, block by block
R = X.sqrt()
print((R @ R - X).frobenius_norm().max()) # ~1e-6 in single precision
print(X.apply_function(torch.log).tau_vN().real.mean()) # tau_vN(log X)

```

States, measurement and trace conventions. Listing 2 conditions random states on the outcome of a projective measurement (Lüders rule), which moves all probability into one sector, and checks that expectation values do not depend on the trace used to represent the state.

Listing 2: States and conditioning (ex2_states_measurement.py).

```

"""States on M, sector probabilities, Lueders conditioning and trace conventions."""
import torch
from torch_vn_algebra import TypeIAlgebra, DensityMatrix, states

torch.manual_seed(0) # reproducible output
dev = 'cuda' if torch.cuda.is_available() else 'cpu'
alg = TypeIAlgebra([2, 3], [2, 3], device=dev) # two superselection sectors

rho = states.random_density_matrix(alg, batch_size=4) # Hilbert-Schmidt random states
print(rho.sector_probabilities()) # p_c = Tr rho_c, rows sum to 1

# measure the projection onto the first basis vector of sector 0
P = alg.from_blocks([torch.diag(torch.tensor([1.0, 0.0])), torch.zeros(3, 3)])
post, prob = rho.condition_on(P) # rho|P = P rho P / Tr(rho P)
print(prob, post.sector_probabilities()) # all weight now in sector 0

# the same state as a density w.r.t. tau_vN: omega(A) = tau_vN(d A) = Tr(rho A)
A = alg.operator_from_eigenvalues(lambda d: torch.randn(4, d), batch_size=4,
                                force_self_adjoint=True)
d_tau = rho.density('tau_vN')
print(rho.expectation(A).real, (alg.operator(d_tau) @ A).tau_vN().real)
back = DensityMatrix.from_density(alg, d_tau, trace='tau_vN')
print((back.matrix - rho.matrix).abs().max())

```

A map between sectors. Two fermionic modes have sectors $N = 0, 1, 2$ of dimensions 1, 2, 1. Particle loss with Kraus operators $\sqrt{1-g}N$, $\sqrt{g}a_1$, $\sqrt{g}a_2$ ($0 \leq g \leq \frac{1}{2}$) maps sector N to N and $N - 1$. Listing 3 builds it from its blocks; the classical part is the transition matrix of a pure death

process (columns: from $N = 0, 1, 2$; rows: to $N = 0, 1, 2$),

$$T = \begin{pmatrix} 1 & g & 0 \\ 0 & 1 - g & 2g \\ 0 & 0 & 1 - 2g \end{pmatrix},$$

which the program prints. The last lines illustrate a simple fact: a unital positive map cannot increase the Michelson contrast, since $\lambda_{\min} 1 \leq A \leq \lambda_{\max} 1$ implies the same bounds for $\Phi(A)$; the non-unital loss channel increases it for most samples.

Listing 3: Particle loss between sectors (`ex3_channels.py`).

```
"""Particle loss between charge sectors of two fermionic modes, as an InterSectorChannel."""
import torch
from torch_vn_algebra import TypeIAlgebra, InterSectorChannel, channels, states

torch.manual_seed(0) # reproducible output
dev = 'cuda' if torch.cuda.is_available() else 'cpu'
# sectors N = 0, 1, 2 with bases |00>, |10>, |01>, |11>
alg = TypeIAlgebra([1, 2, 1], [1, 2, 1], device=dev)
g = 0.2 # loss probability per particle
s, z = g ** 0.5, torch.zeros
# Kraus operators K_0 = sqrt(1 - g) N, K_1 = sqrt(g) a_1, K_2 = sqrt(g) a_2, split into blocks (d, c)
blocks = {
    (0, 0): [torch.ones(1, 1)],
    (1, 1): [(1 - g) ** 0.5 * torch.eye(2)],
    (2, 2): [(1 - 2 * g) ** 0.5 * torch.ones(1, 1)],
    (0, 1): [z(1, 2), s * torch.tensor([[1.0, 0.0]]), s * torch.tensor([[0.0, 1.0]])],
    (1, 2): [z(2, 1), s * torch.tensor([[0.0], [1.0]]), s * torch.tensor([[1.0], [0.0]])],
}
loss = InterSectorChannel.from_blocks(alg, alg, blocks)
print(loss.is_trace_preserving())
print(loss.transition_matrix()[0]) # classical death process on N

rho = states.random_density_matrix(alg, batch_size=3)
print(loss(rho).sector_probabilities()) # still a DensityMatrix

# unital channels cannot increase the Michelson contrast, non-unital ones can
X = alg.operator_from_eigenvalues(lambda d: 0.5 + torch.rand(1000, d, device=dev),
                                batch_size=1000, force_positive=True)
mix = channels.random_mixed_unitary_channel(alg, 3, batch_size=1000)
print(bool((mix(X).michelson_contrast <= X.michelson_contrast + 1e-6).all()))
print((loss(X).michelson_contrast > X.michelson_contrast).float().mean())
```

Open-system dynamics. Listing 4 integrates spontaneous emission of a two-level atom with the RK4 solver and with the exact exponential of the Lindblad generator, and compares both with $P_e(t) = P_e(0)e^{-\gamma t}$ and $|\rho_{eg}(t)| = |\rho_{eg}(0)|e^{-\gamma t/2}$.

Listing 4: Spontaneous emission (`ex4_dynamics.py`).

```
"""Spontaneous emission of a two-level atom: Lindblad evolution against the exact solution."""
import math
import torch
from torch_vn_algebra import TypeIAlgebra, DensityMatrix, dynamics

torch.manual_seed(0) # reproducible output
dev = 'cuda' if torch.cuda.is_available() else 'cpu'
alg = TypeIAlgebra([2], [2], device=dev, precision='double')
```

```

omega, gamma = 1.0, 0.3
H = alg.from_blocks([torch.diag(torch.tensor([0.5 * omega, -0.5 * omega]))]) # basis |e>, |g>
L = alg.from_blocks([torch.tensor([[0.0, 0.0], [1.0, 0.0]])]) # sigma_- = |g><e|
psi = torch.tensor([0.6, 0.8], dtype=torch.complex128)
rho0 = DensityMatrix(alg, matrix=torch.outer(psi, psi.conj())[None, None].to(dev))

times = [0.0, 1.0, 2.0, 5.0]
rk4 = dynamics.lindblad_evolve(rho0, H, [L], times, rates=[gamma], substeps=100)
for t, rho in zip(times, rk4):
    exact = dynamics.lindblad_channel(alg, H, [L], t=t, rates=[gamma])(rho0) # exp(tL)
    p_e = 0.36 * math.exp(-gamma * t) # analytic excited population
    coh = 0.48 * math.exp(-gamma * t / 2) # analytic |rho_eg|
    m = rho.matrix[0, 0]
    print(f"t={t}: P_e {m[0, 0].real:.6f} (exact {p_e:.6f}), |rho_eg| {m[0, 1].abs():.6f} "
          f"(exact {coh:.6f}), RK4 vs exp(tL): {(rho.matrix - exact.matrix).abs().max():.1e}")

```

Tensor products. Listing 5 estimates the mean entanglement entropy of random pure states on $\mathbb{C}^m \otimes \mathbb{C}^n$ and compares it with Page’s formula [13].

Listing 5: Page entropy (ex5_page_entropy.py).

```

"""Average entanglement entropy of random pure states on C^m (x) C^n (Page, 1993)."""
import torch
from torch_vn_algebra import TypeIAlgebra, tensor_product, partial_trace, states

torch.manual_seed(0) # reproducible output
dev = 'cuda' if torch.cuda.is_available() else 'cpu'
m, n, B = 4, 8, 20_000
A = TypeIAlgebra([m], [m], device=dev, precision='double')
Bsys = TypeIAlgebra([n], [n], device=dev, precision='double')
AB = tensor_product(A, Bsys)

psi = states.random_density_matrix(AB, batch_size=B, rank=1) # Haar-random pure states
S = states.von_neumann_entropy(partial_trace(psi, keep=1)) # entropy of rho_A

page = sum(1.0 / k for k in range(n + 1, m * n + 1)) - (m - 1) / (2 * n)
print(f"<S_A> = {S.mean():.5f} +- {S.std() / B ** 0.5:.5f}, Page: {page:.5f}")

```

Searching for a violation. Listing 6 compares Haar sampling with batched gradient ascent for $\sup_U (|\text{Tr}(XUY)| - \text{Tr}(XY))$, whose exact value $\|YX\|_1 - \text{Tr}(XY)$ is positive for non-commuting $X, Y \geq 0$. Section 6 studies this systematically.

Listing 6: Sampling versus optimisation (ex6_inequality_search.py).

```

"""Searching for violations of |Tr(XUY)| <= Tr(XY): random sampling versus gradient ascent."""
import torch
from torch_vn_algebra import TypeIAlgebra

dev = 'cuda' if torch.cuda.is_available() else 'cpu'
torch.manual_seed(0)
k = 16
alg = TypeIAlgebra([k], [k], device=dev, precision='double')
pos = lambda d: 0.2 + torch.rand(1, d)
X = alg.operator_from_eigenvalues(pos, force_positive=True, force_self_adjoint=True)
Y = alg.operator_from_eigenvalues(pos, force_positive=True, force_self_adjoint=True)
trXY = (X @ Y).trace.real

```

```

def z(U):
    # z(U) = |Tr(XUY)| - Tr(XY), batched over U
    return (X @ U @ Y).trace.abs() - trXY

# 1) Monte Carlo over Haar unitaries
print("sampling, 10^5 Haar U:", z(alg.random_unitary_operator(100_000)).max().item())

# 2) gradient ascent over U = U0 exp(G - G^*), 64 random starts optimised in parallel;
# the exponential chart is re-centred at the current point every 50 steps
U0 = alg.random_unitary_operator(64).matrix
for _ in range(10):
    G = torch.zeros(64, 1, k, k, dtype=alg.hilbert.dtype, device=dev, requires_grad=True)
    opt = torch.optim.Adam([G], lr=0.05)
    for step in range(50):
        U = alg.operator(U0) @ alg.operator(G - G.conj().transpose(-2, -1)).expm()
        loss = -z(U).sum()
        opt.zero_grad()
        loss.backward()
        opt.step()
    U0 = U.matrix.detach()
print("gradient ascent:      ", z(alg.operator(U0)).max().item())

# exact supremum: sup_U |Tr(U YX)| = ||YX||_1
print("exact sup:          ", ((Y @ X).trace_norm() - trXY).item())

```

Spin chains. Listing 7 works with a Heisenberg ring of ten spins. A uniform field moves the ground state through the sectors until the chain is fully polarised above the saturation field $h_s = 2J$ (above it the ground state is the all-down state $N = 0$ for our sign $+\sum_i h_i S_i^z$; the last crossing is with the one-magnon level at $k = \pi$); 100 realisations of random fields are diagonalised in one batch to compute the mean level-spacing ratio; and the entanglement entropy of the ground state is computed from reduced states. Section 7 checks these tools against exact results.

Listing 7: Spin chains (ex7_spin_chain.py).

```

"""Spin chains with conserved S^z: sectors, disorder averages and entanglement."""
import torch
from torch_vn_algebra import SpinChain, DensityMatrix

torch.manual_seed(0) # reproducible output
dev = 'cuda' if torch.cuda.is_available() else 'cpu'
chain = SpinChain(L=10, boundary='periodic', device=dev) # sectors N = 0..10, dims binom(10, N)
print(chain.algebra.k_factors)

# Heisenberg ring in a uniform field: the ground state changes sector as h grows
for h in (0.0, 1.0, 2.5):
    w = [blk[0] for blk in chain.heisenberg(h=h).eigenvalues()]
    print(f"h={h}: ground state in sector N={min(range(11), key=lambda N: w[N].min())}")

# random fields h_i ~ U[-W, W]: 100 disorder realisations in one batch, sector N = 5 only
for W in (0.5, 8.0):
    H = chain.random_field_heisenberg(W, batch_size=100, sector=5)
    E = H.eigenvalues()[0] # (100, 252)
    r = SpinChain.level_spacing_ratio(E[:, 63:189]).mean()
    print(f"W={W}: <r> = {r:.3f} (GOE 0.531, Poisson 0.386)")

# entanglement entropy of the ground state of the clean chain, left block of n sites

```

```
E, V = chain.heisenberg(sector=5).eigh()[0]
psi = V[0, :, 0]
rho = DensityMatrix(chain.sector_algebra(5), matrix=torch.outer(psi, psi.conj())[None, None])
print([round(chain.entanglement_entropy(rho, n, sector=5).item(), 4) for n in range(1, 6)])
```

Large sectors and open chains. Dense matrices of a sector of dimension $\binom{L}{L/2}$ stop fitting at about $L = 16$ – 18 . Listing 8 uses the sparse form of the Hamiltonian (a batch of diagonals and one shared hopping matrix) with Lanczos and Krylov solvers, splits a ring into momentum sectors, and lets particles leak out of the chain through Lindblad jumps between sectors.

Listing 8: Large sectors and open chains (ex8_large_and_open_chains.py).

```
"""Beyond dense matrices: Lanczos, Krylov evolution, momentum sectors and particle loss."""
import math
import torch
from torch_vn_algebra import SpinChain, krylov, dynamics

torch.manual_seed(0) # reproducible output
dev = 'cuda' if torch.cuda.is_available() else 'cpu'

# Lanczos in a sector of dimension binom(16, 8) = 12870, sparse Hamiltonian
ring = SpinChain(16, 'periodic', complex_valued=False, device=dev)
E0, psi, res = krylov.ground_state(ring.xx_sparse(sector=8))
print(f"Heisenberg ring L=16: E0 = {E0.item():.9f} (exact -7.142296361), residual {res.item():.1e}")

# Krylov evolution of a domain wall in the XX chain, L = 14 (sector dimension 3432)
chain = SpinChain(14, device=dev)
psi0 = chain.vector_in_sector('1' * 7 + '0' * 7)
psi_t = krylov.evolve(chain.xx_sparse(J=1.0, Delta=0.0, sector=7), psi0[None], [0.0, 3.0])
mag = (psi_t[-1, 0].abs() ** 2) @ chain.bits(7).to(torch.float64) - 0.5
print("S^z_i(t=3):", [round(x, 3) for x in mag.tolist()])

# translation symmetry: blocks of fixed momentum k = 2 pi m / L inside the sector N = 5 of a ring of
# 10
ring10 = SpinChain(10, 'periodic', device=dev)
Hk = ring10.xx_momentum(J=1.0, Delta=1.0, sector=5)
print("momentum block sizes:", ring10.momentum_dims(5))
e0 = {m: w[0].min().item() for m, w in zip(Hk.algebra.charges, Hk.eigenvalues())}
print("ground state momentum m =", min(e0, key=e0.get), "(pi for L/2 odd)")

# particle loss S_-i on every site with rate g: the sector distribution stays binomial
small = SpinChain(5, 'periodic', device=dev)
rho = dynamics.lindblad_evolve(small.basis_state('11011'), small.xx(1.0, 0.7),
                               [small.lowering(i) for i in range(5)], [0.0, 1.0], rates=[0.4] * 5)
q = math.exp(-0.4)
print("p_N(t=1):", [round(x, 6) for x in rho.sector_probabilities()[0].tolist()])
print("binomial:", [round(math.comb(4, n) * q ** n * (1 - q) ** (4 - n), 6) for n in range(5)] +
      [0.0])
```

A noisy quantum circuit with a conserved quantity. The quantum alternating operator ansatz solves constrained optimisation problems with “XY mixers” (normalised here as $H_M = \frac{1}{4} \sum_{\langle ij \rangle} (X_i X_j + Y_i Y_j)$, the XXZ form with $J = 1$, $\Delta = 0$) that keep the Hamming weight of the computational basis states, so the evolution stays in the feasible subspace [29]. In the language of this paper the qubit register carries the algebra $\bigoplus_{N=0}^n M_{\binom{n}{N}}$ of Hamming-weight sectors, the mixer is an element of it,

amplitude damping (time T_1) is a Lindblad jump between sectors and pure dephasing (time T_φ , which together with T_1 sets the coherence time T_2 , $1/T_2 = 1/(2T_1) + 1/T_\varphi$) a jump inside them, and reading out the Hamming weight is the measurement of a central projection. Listing 9 evolves $|111000\rangle$ (weight $w = 3$) on a ring of six qubits under H_M for a time $T = 2$ with both kinds of noise, jump operators S_i^- at rate γ_1 (damping) and $S_i^z = \frac{1}{2}Z_i$ at rate γ_2 (dephasing), so that $T_1 = 1/\gamma_1$, $1/T_\varphi = \gamma_2/2$ and $1/T_2 = (\gamma_1 + \gamma_2)/2$, and post-selects on the initial weight. With damping only (equal rates on all qubits), the post-selected fidelity is exactly one and the acceptance probability is $e^{-w\gamma_1 T}$: the no-jump evolution is generated by $H - \frac{i}{2}\gamma_1 N$, and N is central, so it only rescales the state. This exact statement is part of the test suite. Dephasing stays inside the sector and is not detected; because N is central, the post-selected state does not depend on γ_1 even with dephasing (fidelity with the noiseless state):

γ_1	γ_2	$P(\text{weight } 3)$	fidelity	after post-selection
0.02	0	0.8869	0.8869	1.0000
0	0.02	1.0000	0.9730	0.9730
0.02	0.02	0.8869	0.8629	0.9730
0.1	0.1	0.5488	0.4805	0.8755

Listing 9: XY mixer with amplitude damping, dephasing and post-selection (ex9_qaoa_xy_mixer.py).

```

"""
Quantum-computing example: an XY mixer (QAOA for constrained problems) on 6 qubits keeps the
Hamming weight, i.e. it acts inside one sector of  $M = (+)_N$   $M_{\text{binom}}(6, N)$ . Amplitude damping ( $T_1$ )
moves weight to lower sectors, pure dephasing does not; measuring the Hamming weight - a central
projection of  $M$  - detects the first kind of error and post-selection restores part of the fidelity.
"""

import torch
from torch_vn_algebra import SpinChain, dynamics, states

torch.manual_seed(0) # reproducible output
dev = 'cuda' if torch.cuda.is_available() else 'cpu'
n, w, T = 6, 3, 2.0 # qubits, Hamming weight, evolution time
qubits = SpinChain(n, 'periodic', device=dev) # ring of qubits, sectors = Hamming
weights
H = qubits.xx(J=1.0, Delta=0.0) # XY mixer sum ( $X_i X_j + Y_i Y_j$ ) / 4
rho0 = qubits.basis_state('111000') # feasible initial state, weight 3
ideal = dynamics.von_neumann(rho0, H, [T])[0] # noiseless reference (pure)
P = qubits.algebra.central([1.0 if N == w else 0.0 for N in range(n + 1)]) # Hamming-weight
projector

T1_jumps = [qubits.lowering(i) for i in range(n)] #  $|1\rangle \rightarrow |0\rangle$ : between sectors
dephasing_jumps = [qubits.sz(i) for i in range(n)] # pure dephasing: inside sectors
print(" gamma1 gamma2 P(weight 3) fidelity fidelity after post-selection")
for g1, g2 in ((0.0, 0.0), (0.02, 0.0), (0.0, 0.02), (0.02, 0.02), (0.1, 0.1)):
    rho = dynamics.lindblad_evolve(rho0, H, T1_jumps + dephasing_jumps, [0.0, T],
                                   rates=[g1] * n + [g2] * n, substeps=80)[-1]
    post, p_ok = rho.condition_on(P) # Lueders rule for the central
    projection
    F = states.trace_of_product(rho, ideal).real.item() #  $\langle \psi_{\text{ideal}} | \rho | \psi_{\text{ideal}} \rangle$ 
    F_post = states.trace_of_product(post, ideal).real.item()
    print(f" {g1:5.2f} {g2:5.2f} {p_ok.item():10.4f} {F:8.4f} {F_post:8.4f}")

```

5 Validation against exact results

Table 6 compares Monte Carlo estimates produced with the library (double precision) with exact values: fourth moments of Haar unitaries and orthogonal matrices; the Weingarten-calculus value $\mathbb{E} \tau((AUBU^*)^2) = -1/(n^2 - 1)$ for traceless involutions $A = B$ and Haar U in $M_n(\mathbb{C})$; Page's formula for $\langle S(\rho_A) \rangle$; and the mean purity $(k + r)/(kr + 1)$ of induced random states of rank r [14]. The last column is the deviation in units of the standard error; for a correct sampler these should look like standard normal deviates. For the circular ensembles the Wigner surmise is only an approximation to the exact spacing distribution, so no z -score is given; Figure 1 shows the distributions, with L^1 distances 0.022, 0.007 and 0.006 for $\beta = 1, 2, 4$ from the surmise; level repulsion $P(s) \sim s^\beta$ is clearly visible. The spontaneous-emission example agrees with the analytic solution to 2.2×10^{-16} (exact exponential) and 2.1×10^{-8} (RK4).

quantity	parameters	exact	Monte Carlo	std. error	z
$\mathbb{E} U_{11} ^4, U(n)$	$n = 2$	0.333333	0.333604	4.7×10^{-4}	+0.6
$\mathbb{E} U_{11} ^2, U(n)$	$n = 2$	0.5	0.500277	4.6×10^{-4}	+0.6
$\mathbb{E} U_{11} ^4, U(n)$	$n = 4$	0.1	0.10028	2.2×10^{-4}	+1.3
$\mathbb{E} U_{11} ^4, U(n)$	$n = 8$	0.0277778	0.0277966	7.5×10^{-5}	+0.3
$\mathbb{E} U_{11} ^4, U(n)$	$n = 16$	0.00735294	0.00735803	2.3×10^{-5}	+0.2
$\mathbb{E} U_{11} ^4, U(n)$	$n = 32$	0.00189394	0.00189625	6.2×10^{-6}	+0.4
$\mathbb{E} U_{11} ^2, U(n)$	$n = 32$	0.03125	0.0312403	4.8×10^{-5}	-0.2
$\mathbb{E} U_{11} ^4, O(n)$	$n = 2$	0.375	0.375106	5.8×10^{-4}	+0.2
$\mathbb{E} U_{11} ^2, O(n)$	$n = 2$	0.5	0.500155	5.6×10^{-4}	+0.3
$\mathbb{E} U_{11} ^4, O(n)$	$n = 4$	0.125	0.125085	3.1×10^{-4}	+0.3
$\mathbb{E} U_{11} ^4, O(n)$	$n = 8$	0.0375	0.0376573	1.3×10^{-4}	+1.2
$\mathbb{E} U_{11} ^4, O(n)$	$n = 16$	0.0104167	0.0104046	4.3×10^{-5}	-0.3
$\mathbb{E} U_{11} ^4, O(n)$	$n = 32$	0.00275735	0.0027558	1.2×10^{-5}	-0.1
$\mathbb{E} U_{11} ^2, O(n)$	$n = 32$	0.03125	0.0312731	6.7×10^{-5}	+0.3
$\mathbb{E} \tau((AUBU^*)^2)$	$n = 4$	-0.0666667	-0.0661548	6.3×10^{-4}	+0.8
$\mathbb{E} \tau((AUBU^*)^2)$	$n = 8$	-0.015873	-0.0159874	2.8×10^{-4}	-0.4
$\mathbb{E} \tau((AUBU^*)^2)$	$n = 16$	-0.00392157	-0.00414221	1.4×10^{-4}	-1.6
$\langle s^2 \rangle$, COE	$n = 64$	1.27324	1.28582	3.7×10^{-4}	—
$\langle s^2 \rangle$, CUE	$n = 64$	1.1781	1.18024	2.4×10^{-4}	—
$\langle s^2 \rangle$, CSE	$n = 64$	1.10447	1.10392	2.0×10^{-4}	—
$\langle S(\rho_A) \rangle$	$m = 2, n = 16$	0.646516	0.646516	1.6×10^{-4}	0.0
$\langle S(\rho_A) \rangle$	$m = 8, n = 16$	1.83367	1.83388	1.7×10^{-4}	+1.2
$\langle S(\rho_A) \rangle$	$m = 16, n = 16$	2.27487	2.27495	1.4×10^{-4}	+0.6
$\mathbb{E} \text{Tr} \rho^2$	$k = 4, r = 2$	0.666667	0.666312	3.2×10^{-4}	-1.1
$\mathbb{E} \text{Tr} \rho^2$	$k = 4, r = 4$	0.470588	0.470523	2.1×10^{-4}	-0.3
$\mathbb{E} \text{Tr} \rho^2$	$k = 16, r = 4$	0.307692	0.307726	6.4×10^{-5}	+0.5
$\mathbb{E} \text{Tr} \rho^2$	$k = 16, r = 16$	0.124514	0.124527	1.7×10^{-5}	+0.8

Table 6: Monte Carlo estimates versus exact values (for $\langle s^2 \rangle$ the Wigner surmise, an approximation, hence no z); samples: 400000 per Haar moment, 400000 per Weingarten row, 20000 matrices per circular ensemble, 50000 states per Page row, 100000 per purity row (full run on NVIDIA GeForce RTX 5090, PyTorch 2.14.1+cu130).

Trace functionals. The three traces are checked on random non-Hermitian operators in $M_2 \oplus M_3 \oplus M_5$ (Table 7): normalisation on the identity ($\sum_c k_c, C$ and 1), the tracial property $\text{tr}(AB) = \text{tr}(BA)$, invariance under unitaries of $\mathcal{M}$, and positivity $\text{tr}(A^*A) > 0$ (faithfulness); the last column is the

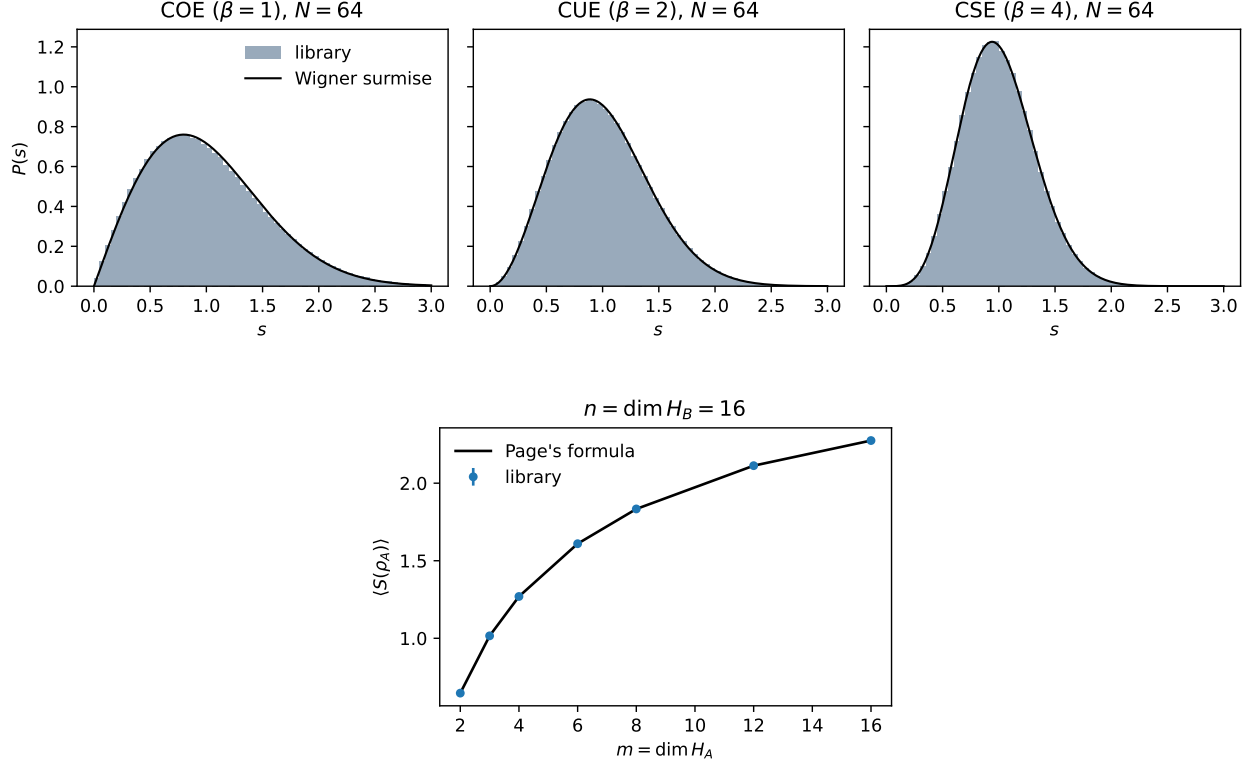


Figure 1: Top: nearest-neighbour spacings of COE, CUE and CSE eigenphases generated by the library ($n = 64$, 20000 matrices per ensemble), against the Wigner surmise. Bottom: mean entanglement entropy of random pure states against Page’s formula (50000 states per point) (error bars: two standard errors).

smallest ratio $\text{tr}(A^*A)/\|A\|_F^2$ over the samples, which is bounded below by $\min_c w_c$. The relation $\text{Tr}_{\text{norm}} = C \tau_{\text{vN}}$ holds to 9.0×10^{-16} . All three return values in the number field of the algebra (complex traces of non-Hermitian operators).

trace	error of $\text{tr}(1)$	$\max \text{tr}(AB) - \text{tr}(BA) $	$\max \text{tr}(UAU^*) - \text{tr}(A) $	$\min \text{tr}(A^*A)/\ A\ _F^2$
Tr_{blunt}	0	4.0×10^{-15}	5.9×10^{-15}	1.000
Tr_{norm}	0	1.4×10^{-15}	1.9×10^{-15}	0.223
τ_{vN}	0	4.4×10^{-16}	6.1×10^{-16}	0.074

Table 7: Identities of the three trace functionals, 2000 random operators in double precision; full run on NVIDIA GeForce RTX 5090.

6 From sampling to optimisation

The first version of this work [1] studied three trace functionals by Monte Carlo:

- Exp. 1: $z = |\text{Tr}(XUY)| - \text{Tr}(XY)$, $X, Y \geq 0$, $U \in \mathcal{M}$ unitary,
- Exp. 2: $z = \text{Tr}|XY| - \text{Tr}(|X||Y|)$, $X = UX_0$, $Y = VY_0$, $X_0, Y_0 \geq 0$, $U, V \in \mathcal{M}$ unitary,
- Exp. 3: $z = \text{Tr}(Y|X|Y) - \text{Tr}|YXY|$, $X = X^*$, $Y \geq 0$,

with $\text{Tr} = \text{Tr}_{\text{blunt}}$, operators normalised to $\lambda_{\max} = 1$, and the contrasts of the operators as parameters (for the self-adjoint X of Exp. 3 and the factors of Exp. 2, the contrasts of $|X|, |Y|$). Two observations motivate a different approach.

Sampling misses the extremes. For fixed X, Y the supremum in Exp. 1 is known exactly, $\sup_U z = \|YX\|_1 - \text{Tr}(XY) > 0$. Table 8 and Figure 2 show that the best of 1 000 000 Haar samples falls short of the supremum for every k , misses the violation for all pairs from $k = 4$ on, and moves further away from it as k grows, while batched gradient ascent (128 starts, 500 steps) finds it to a relative error of at most 3×10^{-5} (Table 8). This is concentration of measure: the unitaries close to the maximiser form a set whose Haar measure decays rapidly with the dimension. This search is done in the complex algebra $M_k(\mathbb{C})$ ($C = 1$, eigenvalues of X, Y uniform in $[0.2, 1.2]$, double precision); the same concentration effect explains the absence of violations in the (real) Monte Carlo samples for $k = 16$ reported in [1]: it is a property of the sampling, not of the inequality.

k	violation found		$\sup z / \text{Tr } XY$	best sample $z / \text{Tr } XY$	ascent rel. error		time [s]	
	sampling	ascent			median	max	sampl.	ascent
2	14/16	16/16	7.1×10^{-4}	5.6×10^{-4}	1.2×10^{-8}	5.4×10^{-6}	64	2.8
4	0/16	16/16	2.3×10^{-3}	-0.12	4.4×10^{-8}	3.1×10^{-6}	66	2.8
8	0/16	16/16	5.6×10^{-3}	-0.48	2.9×10^{-7}	2.0×10^{-6}	70	2.8
16	0/16	16/16	6.7×10^{-3}	-0.72	2.4×10^{-6}	3.9×10^{-6}	88	2.7
32	0/16	16/16	8.9×10^{-3}	-0.86	7.4×10^{-6}	1.6×10^{-5}	1.8e+02	6.6
64	0/16	16/16	7.0×10^{-3}	-0.93	2.0×10^{-5}	2.8×10^{-5}	2.7e+02	11

Table 8: Exp. 1 for fixed random X, Y in $M_k(\mathbb{C})$ (eigenvalues uniform in $[0.2, 1.2]$, double precision): Haar sampling versus gradient ascent (medians over 16 pairs; full run on NVIDIA GeForce RTX 5090).

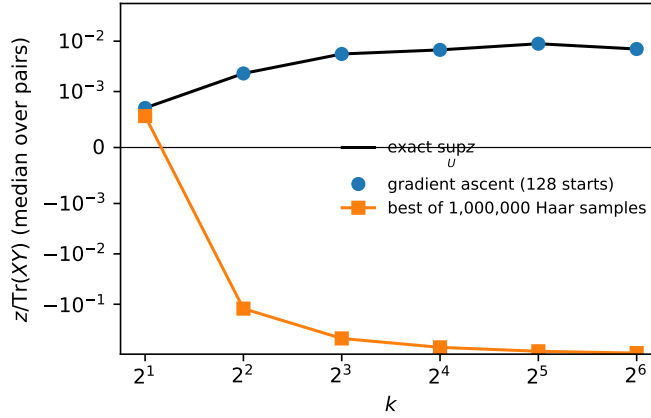


Figure 2: Best z found by sampling and by gradient ascent, relative to $\text{Tr}(XY)$, against the exact supremum (symmetric log scale).

Constrained optimisation. The natural objects are the envelopes

$$\bar{z}(\delta) = \sup\{z : \Delta(X) = \delta\}, \quad \underline{z}(\delta) = \inf\{z : \Delta(X) = \delta\},$$

(and the same with $\Delta(Y)$, or with both contrasts prescribed), the supremum being taken over all remaining variables. The module `optimize` parametrises the constraint set exactly: the spectrum is $\lambda_{\max} = 1$, $\lambda_{\min} = (1 - \delta)/(1 + \delta)$ and the other eigenvalues in between, as $\lambda_{\min} + (1 - \lambda_{\min})(1 + \sin \theta)/2$ (unlike a sigmoid, this reaches both ends at finite θ ; extremal spectra often have all eigenvalues at the ends); the two eigenvalue positions that carry $\lambda_{\max}$ and $\lambda_{\min}$ are drawn at random per start, in the same sector or in different ones, so that the starts together cover a dense subset of the constraint set, the eigenvectors are $U = U_0 e^{G - G^*}$ with a Haar-random U_0 per start and a chart that is re-centred after every round. For Exp. 3 the signs of the eigenvalues of X are discrete variables: the number of negative eigenvalues is drawn per sector (for half of the starts as one fraction common to all sectors), and between the gradient rounds single signs are flipped where this improves z . Every iterate is an admissible point, so the best value seen per start is kept. Every grid point and every start is one element of the batch, so each envelope (supremum or infimum along one contrast) is computed in one batched optimisation. All envelopes are computed in real algebras (real symmetric X, Y and orthogonal U), the setting of the Monte Carlo study of [1].

Figures 3 and 4 and Table 9 show the result. The search is local, so the computed envelopes are checked in three independent ways. (i) Where an envelope is known in closed form (below), it is compared with the formula. (ii) Since z is additive over orthogonal blocks (in Exp. 1 the trace $|\text{Tr}(XUY)|$ is only subadditive, but replacing U by $-U$ on a block aligns the phases of the block traces, so the bound below holds) and the contrast of a direct sum is the largest contrast of its summands, a direct sum of $kC/2$ copies of a solution for $k = 2$, $C = 1$ is an admissible configuration: with $s(\delta)$ the upper envelope for $k = 2$, $C = 1$,

$$\bar{z}_{k,C}(\delta) \geq s(\delta) + \left(\frac{kC}{2} - 1\right) \max_{\delta' \leq \delta} s(\delta') \quad (k \text{ even}),$$

and the same with min for the lower envelopes; the computed envelopes must not lie below this value (above, for the infima). (iii) Each Monte Carlo sample of [1] and each of *structured samples* is checked against both envelopes, along $\Delta(X)$ and along $\Delta(Y)$; samples that fall outside the linear interpolation between grid points are re-optimised at their exact contrast. The structured samples are computed with plain NumPy: all spectra take only the two values 1 and $\lambda_{\min}$, and the eigenbases and U are random signed permutations composed with direct sums of 2×2 rotations. Unlike Haar samples they come close to the extremal configurations in every dimension, so they test the optimiser where Monte Carlo cannot. Of the 468186 Monte Carlo samples with both contrasts in the grid (11814 have a contrast above 0.999), 0 remain outside after re-optimisation, and – of the structured samples; the envelopes fall short of the direct sums (ii) by at most $\sim 1\%$ of the largest value of the curve. For a single sector of size $k = 2$ the Haar samples come close to the envelopes; with two sectors they already miss a substantial part of the range (Table 9), and for larger k or C they stay in a narrow band far from both envelopes (Figure 3, bottom row; Table 8): the extremes are attained at degenerate spectra with aligned eigenbases, a set that Haar sampling essentially never reaches.

Four envelopes can be checked analytically. In Exp. 3, $z \geq 0$ always:

Proposition 1. *For $X = X^*$ and $Y \geq 0$, $\text{Tr}|YXY| \leq \text{Tr}(Y|X|Y)$, with equality if and only if $YX_+Y^2X_-Y = 0$, in particular if Y commutes with X .*

Proof. With $X = X_+ - X_-$, $|X| = X_+ + X_-$, the triangle inequality for the trace norm gives $\|YX_+Y - YX_-Y\|_1 \leq \text{Tr}(YX_+Y) + \text{Tr}(YX_-Y)$; for positive A, B , $\|A - B\|_1 = \text{Tr} A + \text{Tr} B$ if and only if $AB = 0$. If $XY = YX$, then $YX_+Y^2X_-Y = Y^4X_+X_- = 0$. $\square$

The computed lower envelope of Exp. 3 is zero to within 3×10^{-9} . In Exp. 1, the lower envelope for prescribed $\Delta(X) = \delta$ equals $-(kC - 2\delta/(1 + \delta))$ when all sectors have size $k \geq 2$: $\text{Tr}(XY) \leq \text{Tr} X \leq kC - 1 + \lambda_{\min}(X)$ with equality for $Y = 1$, while $|\text{Tr}(UYX)|$ can be made zero by a cyclic permutation

of the eigenbasis of X_c in every sector. The optimiser reproduces this curve to a relative deviation of 1.0e-02 (along either contrast: in a real algebra $\text{Tr}(XUY) = \text{Tr}(YU^T X)$, so X and Y exchange roles). In Exp. 2, $|X| = X_0$ and $|Y| = Y_0$ and the unitary U drops out, so $z = \|X_0 V Y_0\|_1 - \text{Tr}(X_0 Y_0)$. Since V and the eigenbases act sector by sector, von Neumann's trace inequality $\|AB\|_1 \leq \sum_i s_i(A) s_i(B)$, its lower counterpart $\|AB\|_1 \geq \sum_i s_i(A) s_{k+1-i}(B)$ and rearrangement give, for given spectra,

$$\sup z = -\inf z = \sum_c \sum_i x_{c,i}^\downarrow (y_{c,i}^\downarrow - y_{c,i}^\uparrow),$$

with $x_{c,i}^\downarrow$ the eigenvalues of X_c in decreasing order and $y_{c,i}^\downarrow, y_{c,i}^\uparrow$ those of Y_c in decreasing and increasing order. Pairing i with $k+1-i$, $\sum_i x_{c,i}^\downarrow (y_{c,i}^\downarrow - y_{c,i}^\uparrow) = \sum_{i \leq k/2} (x_{c,i}^\downarrow - x_{c,k+1-i}^\downarrow) (y_{c,i}^\downarrow - y_{c,k+1-i}^\downarrow) \leq \lfloor k/2 \rfloor (1 - \lambda_{\min}(X))(1 - \lambda_{\min}(Y))$, with equality for two-level spectra. Hence, for sectors of sizes k_c , $\sup z = -\inf z = \sum_c \lfloor k_c/2 \rfloor (1 - \lambda_{\min}(X))(1 - \lambda_{\min}(Y))$, and for C sectors of size k , $\Delta(|X|) = \delta$ prescribed and $\Delta(|Y|) \leq 0.999$ free,

$$\sup z = -\inf z = C \left\lfloor \frac{k}{2} \right\rfloor \frac{2\delta}{1+\delta} \left(1 - \frac{0.001}{1.999}\right),$$

which tends to $C \lfloor k/2 \rfloor \frac{2\delta}{1+\delta}$ as $\Delta(|Y|) \rightarrow 1$. The optimiser reproduces the upper and lower envelopes of Exp. 2 for all eight configurations, along either contrast, to a relative deviation of 6.9e-04. For $k=2$, $C=1$ the computed upper envelopes of Exp. 1 and Exp. 3 along $\Delta(X) = \delta$, with the free contrast capped at $D = 0.999$, agree with $\delta^2 D^2 / ((1+\delta)(1+D))$ and $(1-\delta)(2D/(1+D))^2$ (for $D \rightarrow 1$: $\delta^2/(2(1+\delta))$ and $1-\delta$) to a relative deviation of 4.8e-03 and 4.0e-03; we have no proof of these closed forms. For larger k and C the upper envelopes of Exp. 1 and Exp. 3 are, to our knowledge, not known in closed form; the computed curves are lower bounds for them (a local search), consistent with the direct sums (ii), and suggest conjectures for sharp contrast-dependent inequalities (for instance, that the direct sums are optimal), which we leave for future work.

6.1 Rediscovering the data-processing thresholds of the Rényi divergences

The envelopes above are largely unknown. A test of the method on an inequality whose range of validity is known exactly is to ask it where that range ends, without telling it. For states ρ, σ on $\mathcal{M}$ (densities with respect to Tr_{blunt}) and $\alpha \in (0, 1) \cup (1, \infty)$ the Petz [45] and the sandwiched [47, 48] Rényi divergences are

$$D_\alpha(\rho\|\sigma) = \frac{1}{\alpha-1} \log \text{Tr}(\rho^\alpha \sigma^{1-\alpha}), \quad \tilde{D}_\alpha(\rho\|\sigma) = \frac{1}{\alpha-1} \log \text{Tr}(\sigma^{\frac{1-\alpha}{2\alpha}} \rho \sigma^{\frac{1-\alpha}{2\alpha}})^\alpha.$$

Both tend to the Umegaki relative entropy $D(\rho\|\sigma) = \text{Tr} \rho(\log \rho - \log \sigma)$ as $\alpha \rightarrow 1$; $\tilde{D}_\alpha \leq D_\alpha$, $\tilde{D}_{1/2} = -\log F(\rho, \sigma)$ with the fidelity $F = \|\sqrt{\rho}\sqrt{\sigma}\|_1^2$, and $\tilde{D}_\alpha \rightarrow D_{\max}$ as $\alpha \rightarrow \infty$. The module **divergences** evaluates them, batched and differentiable (Daleckii–Krein gradients), with exact support conventions (powers are taken on the numerical support, and the divergence is $+\infty$ for $\alpha > 1$ when $\text{supp } \rho \not\subseteq \text{supp } \sigma$), as logarithms of sums over sectors, so that large α neither overflows nor underflows. On $\mathcal{M}$ they split into a classical part and the conditional divergences of the sectors, and they do not depend on the trace used to represent the states (the central weights cancel); the tests compare them with dense references and check these properties.

The data-processing inequality $D(\Phi(\rho)\|\Phi(\sigma)) \leq D(\rho\|\sigma)$ for all channels Φ holds for the Petz divergences for $\alpha \in (0, 1) \cup (1, 2]$ [45] and for the sandwiched ones for $\alpha \geq \frac{1}{2}$ [49, 50, 47, 48]; outside these ranges it fails in general. For the relative entropy, equality characterises the channels that can be reversed on ρ and σ , and the reversal is the Petz recovery map [46] $R_{\sigma, \Phi}(X) =$

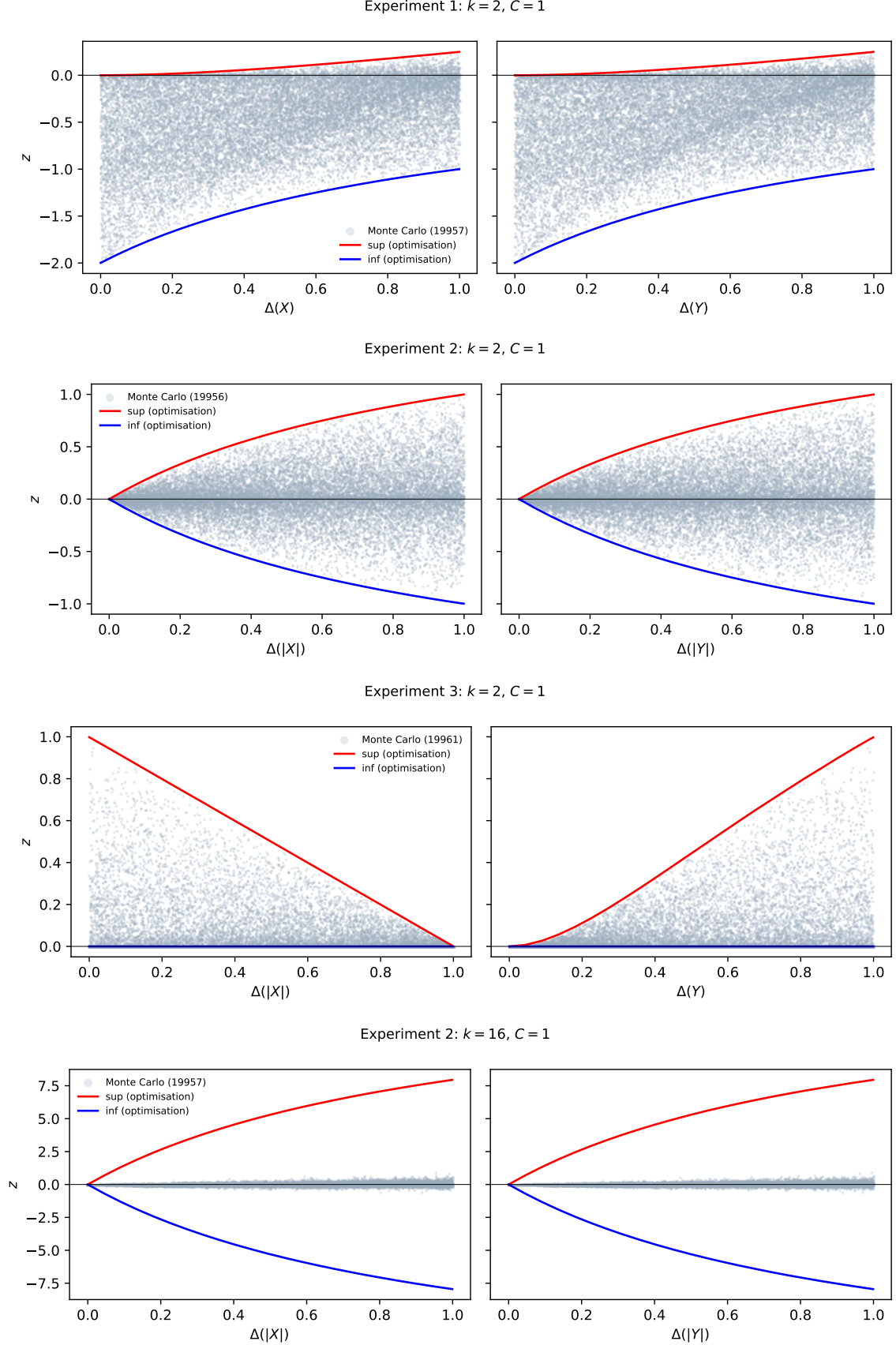


Figure 3: Envelopes from constrained optimisation versus sampling for the three trace functionals (rows 1–3: $k = 2, C = 1$; row 4: Exp. 2 for $k = 16, C = 1$). Horizontal axis: Michelson contrast of the first operator (left) or of the second operator (right); vertical axis: z for operators normalised to $\lambda_{\max} = 1$. Grey: the Monte Carlo samples of [1] (Haar eigenbases, uniform spectra); orange: structured samples (two-level spectra, permutation and rotation bases, computed with NumPy, in

Exp.	k	C	$\inf z$	$\sup z$	Monte Carlo range of z	outside
1	2	1	-1.999	+0.2492	$[-1.98, +0.227]$	0
1	2	2	-3.998	+0.4943	$[-3.39, +0.328]$	0
1	2	16	-32	+3.73	$[-20, -4.37]$	0
1	2	32	-64	+7.28	$[-36.4, -15.9]$	0
1	16	1	-16	+1.913	$[-15.5, -2.18]$	0
1	16	2	-32	+3.706	$[-29, -5.73]$	0
1	16	16	-256	+28.55	$[-166, -89]$	0
1	16	32	-512	+57.05	$[-295, -193]$	0
2	2	1	-0.9985	+0.9988	$[-0.921, +0.975]$	0
2	2	2	-1.989	+1.991	$[-1.43, +1.72]$	0
2	2	16	-15.86	+15.85	$[-3.33, +3.83]$	0
2	2	32	-31.7	+31.68	$[-5.22, +6.1]$	0
2	16	1	-7.948	+7.944	$[-0.526, +0.89]$	0
2	16	2	-15.88	+15.88	$[-0.549, +0.869]$	0
2	16	16	-126.9	+126.9	$[-1.12, +2.29]$	0
2	16	32	-253.7	+253.8	$[-1.07, +3.1]$	0
3	2	1	0	+0.9981	$[0, +0.946]$	0
3	2	2	0	+1.989	$[0, +1.39]$	0
3	2	16	0	+14.86	$[+0.00115, +3.02]$	0
3	2	32	0	+25.63	$[+0.0764, +4.51]$	0
3	16	1	0	+7.958	$[-7.63e - 06, +2.15]$	0
3	16	2	0	+15.89	$[+1.72e - 05, +3.33]$	0
3	16	16	0	+116.8	$[+1.9, +12.7]$	0
3	16	32	0	+223	$[+6.11, +20.1]$	0

Table 9: Global extremes of z over both envelopes and contrasts in $[0, 0.999]$ found by constrained optimisation (real algebras, orthogonal U), the range of the Monte Carlo samples with both contrasts in that interval, and the number of samples outside the envelopes (along either contrast) after re-optimisation (64 starts per grid point, 500 steps; full run on NVIDIA GeForce RTX 5090); a sample counts as outside if it exceeds the envelope by more than $10^{-6} + 10^{-3} \max |z|$; “-”: no Monte Carlo data of [1] for this configuration. Monte Carlo values of z slightly below 0 in Exp. 3 (where $z \geq 0$ is proved) are single-precision round-off of the samples of [1].

$\sigma^{1/2} \Phi^* (\Phi(\sigma)^{-1/2} X \Phi(\sigma)^{-1/2}) \sigma^{1/2}$, which satisfies $R_{\sigma, \Phi}(\Phi(\sigma)) = \sigma$. The module provides it for channels within and between sectors, together with its rotated versions and the universal recovery map R_{univ} of [51], for which $D(\rho \| \sigma) - D(\Phi(\rho) \| \Phi(\sigma)) \geq -\log F(\rho, R_{\text{univ}}(\Phi(\rho)))$; the tests check the inequalities inside their ranges on random states and channels, Petz’s equality condition and the recovery bound.

`scripts/renyi_threshold.py` maximises the violation

$$v(\alpha) = D(\Phi(\rho) \| \Phi(\sigma)) - D(\rho \| \sigma)$$

for both families on a grid of 31 values $\alpha \in [0.2, 3]$, over states ρ, σ on M_d and over two kinds of channels: the complete dephasing (the conditional expectation onto the diagonal subalgebra) and all channels with a d -dimensional environment, given by Stinespring isometries $V = Z(Z^*Z)^{-1/2}$ with a free complex Z . All values of α and all starts form one batch (α is given per sample), optimised with the batched multistart ascent of Section 6. The states are restricted to $\rho, \sigma \geq \varepsilon/d$; this keeps the supremum finite and the sandwiched divergence resolved in double precision for $\alpha \geq 0.2$. The bracket between the last violating and the first non-violating grid point is then refined by searches at interior points, started from random points and from perturbations of the best violating point. Every reported

Experiment 1, $k = 2$, $C = 1$: both contrasts prescribed

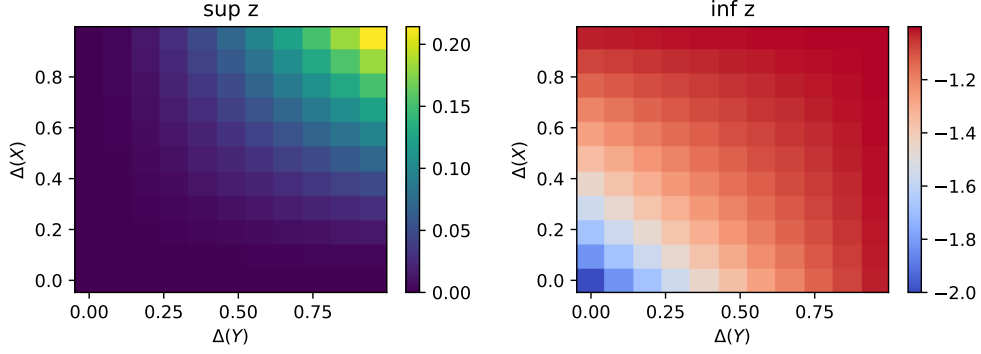


Figure 4: Exp. 1, $k = 2$, $C = 1$, both contrasts prescribed: $\sup z$ (left) and $\inf z$ (right) on a grid of $\Delta(Y)$ (horizontal) and $\Delta(X)$ (vertical). The violation $\sup z > 0$ needs both operators to be far from scalar and grows with both contrasts.

violation is attained at an explicit point (two states and the Kraus operators of a channel), and v is recomputed there independently in mpmath with 50 significant digits, after re-orthonormalising the Kraus operators so that the point is an exactly trace-preserving channel; it counts as a violation only if the certified value exceeds a tolerance far above the error of the certification. A certified violation therefore proves that the inequality fails at that α ; the absence of a violation at the other end of a bracket is evidence only, since a local search can miss small violations. (Results not generated: run `scripts/renyi_threshold.py`.) Table 10 and Figure 5 show $v(\alpha)$ on the grid.

sandwiched $\tilde{D}_\alpha$		Petz D_α	
α	dephasing channels	dephasing channels	channels
<i>run scripts/renyi_threshold.py</i>			

Table 10: Largest certified violation $v(\alpha)$ of the data-processing inequality on M_2 , for the complete dephasing and for all channels with a two-dimensional environment (“–”: no violation above the tolerance found). The theorems guarantee $v \leq 0$ for $\alpha \geq \frac{1}{2}$ (sandwiched) and $\alpha \leq 2$ (Petz).

figure renyi_threshold.pdf not generated

Figure 5: Largest certified violation $v(\alpha)$ (logarithmic axis) for the sandwiched (left) and the Petz (right) divergences, for the complete dephasing and for channels with a d -dimensional environment (solid: $d = 2$, dotted: $d = 3$); open symbols on the bottom line: no violation above the tolerance (dotted horizontal line) found. Dashed: the known thresholds; shaded: the range in which the data-processing inequality is a theorem.

7 Symmetric tensor products and spin chains

The basis of `SpinChain` is, by construction, that of the iterated fused product of single sites: the tests check that S_i^z built by the chain coincides with $S_i^z \otimes 1$ formed with `kron` in the fused product of the algebra of `SpinChain(L-1)` with a single site. Beyond that, integrable chains provide exact results for every part of the module (Table 12, Figure 6; script `scripts/chains.py`):

- *Free fermions.* By the Jordan–Wigner transformation the open XX chain ($\Delta = 0$) is a free-fermion model with single-particle energies $\varepsilon_k = J \cos \frac{\pi k}{L+1}$, so the spectrum of sector N consists of all sums of N distinct ε_k . For $L = 12$ all 2^L levels agree to 7.1×10^{-15} .
- *Heisenberg ring.* Ground-state energies agree with exact values (Bethe ansatz for $L = 4$, independent exact diagonalisation for the others, rounded to nine digits) to the nine digits of the table; E_0/L approaches the Bethe-ansatz value $\frac{1}{4} - \ln 2 \approx -0.4431$. Every level of sector $N < L/2$ reappears in sector $N + 1$ (SU(2) multiplets), up to 2.9×10^{-15} ; Section 8 uses this symmetry to reduce the blocks further.
- *Dynamics.* The melting of a domain wall $|\uparrow \cdots \uparrow \downarrow \cdots \downarrow\rangle$ in the open XX chain, evolved in the sector $N = L/2$ (dimension 12870 for $L = 16$), reproduces the single-particle prediction $\langle n_i(t) \rangle = [e^{iht} C(0) e^{-iht}]_{ii}$, with h the single-particle hopping matrix and $C_{ij} = \langle c_i^\dagger c_j \rangle$ the correlation matrix, to 4.9×10^{-15} .
- *Entanglement.* The entropy of a block of n sites of the XX ground state, obtained with **reduced_state**, agrees with Peschel’s formula $S = -\sum_\nu [\nu \ln \nu + (1 - \nu) \ln(1 - \nu)]$ over the eigenvalues ν of the restricted correlation matrix [15], including the parity oscillations of open chains, to 3.8×10^{-12} .
- *Disorder.* For the Heisenberg chain with random fields $h_i \in [-W, W]$ the mean ratio of consecutive level spacings in the middle of the spectrum crosses over from the GOE value 0.5307 to the Poisson value $2 \ln 2 - 1 \approx 0.3863$ [16, 17] as W grows; at these sizes this is a finite-size crossover, the standard spectral diagnostic used in studies of many-body localisation. Here the batch dimension holds the disorder realisations (400 per point).
- *Momentum.* For periodic chains the basis $|a, k\rangle = N_a^{-1/2} \sum_{j=0}^{L-1} e^{-ikj} T^j |a\rangle$ over orbit representatives a of the translation T (orbit length R_a , $N_a = L^2/R_a$; a contributes to momentum k iff $kR_a \in 2\pi\mathbb{Z}$) splits sector N into blocks of fixed momentum $k = 2\pi m/L$ (for $L = 14$ and $N = L/2$ of sizes 246, 245, 245, 245, 245, 245, 245, 246, 245, 245, 245, 245, 245, 245); the union of their spectra is the spectrum of the sector to 3.6×10^{-14} , and the ground state of the antiferromagnetic Heisenberg ring has $k = 0$ for even and $k = \pi$ for odd $L/2$ (Marshall’s sign rule). Momentum blocks with $k \neq 0, \pi$ are complex in the plane-wave basis even for real Hamiltonians, one of the cases of Section 3 where the field is fixed by the problem, at least in this basis: combined with reflection, time reversal leaves each k -block invariant, so for non-integrable translation-invariant Hamiltonians these blocks show GOE statistics and admit a real basis; the XXZ ring itself is integrable and its blocks are not of GOE type.
- *Beyond dense matrices.* Lanczos with full re-orthogonalisation and restarts, batched over Hamiltonians, reproduces the free-fermion ground-state energies of open XX chains and the Heisenberg ring in sectors up to the sizes of Table 11 (in the full run beyond the reach of dense diagonalisation); Krylov evolution of the domain wall for $L = 22$ (sector dimension 705432) agrees with the single-particle solution to 1.7×10^{-14} .
- *Open chains.* With jump operators S_i^- on every site (Lindblad terms between sectors $N \rightarrow N - 1$) and any N -conserving Hamiltonian the particle number performs a pure death process, so the sector distribution must be binomial, $p_n(t) = \binom{N_0}{n} e^{-n\gamma t} (1 - e^{-\gamma t})^{N_0-n}$; the library reproduces it to 2.4×10^{-10} .

model	sector dim	E_0 (Lanczos)	E_0 (exact)	difference	time [s]
XX open, $L = 16$	12870	-4.9189757237	-4.9189757237	1.8×10^{-15}	0.14
XX open, $L = 20$	184756	-6.1907449998	-6.1907449998	8.9×10^{-16}	0.07
XX open, $L = 24$	2704156	-7.4629855550	-7.4629855550	2.7×10^{-15}	0.56
Heisenberg ring, $L = 16$	12870	-7.1422963606	-7.142296361	$< 5 \times 10^{-10}$	0.059

Table 11: Batched Lanczos on sparse sector Hamiltonians (time includes building the sparse Hamiltonian): ground-state energies against free fermions ($\sum$ of the $L/2$ lowest $\cos \frac{\pi k}{L+1}$) and the tabulated Heisenberg value (nine digits); full run on NVIDIA GeForce RTX 5090.

L	dim ($N = L/2$)	E_0 (library)	E_0 (exact)	difference	E_0/L
4	6	-2.000000000	-2.000000000	$< 5 \times 10^{-10}$	-0.50000
6	20	-2.802775638	-2.802775638	$< 5 \times 10^{-10}$	-0.46713
8	70	-3.651093409	-3.651093409	$< 5 \times 10^{-10}$	-0.45639
10	252	-4.515446354	-4.515446354	$< 5 \times 10^{-10}$	-0.45154
12	924	-5.387390917	-5.387390917	$< 5 \times 10^{-10}$	-0.44895
14	3432	-6.263549534	-6.263549534	$< 5 \times 10^{-10}$	-0.44740
16	12870	-7.142296361	-7.142296361	$< 5 \times 10^{-10}$	-0.44639

Table 12: Ground-state energy of the Heisenberg ring $H = \sum_i \mathbf{S}_i \cdot \mathbf{S}_{i+1}$; exact values to nine digits (full run on NVIDIA GeForce RTX 5090).

8 Subalgebras, inclusions and symmetries

The algebras of the previous sections are declared by their sector sizes. Three modules relate algebras to each other and to the symmetries that produce them: **inclusions** treats a subalgebra $\mathcal{N} \subset \mathcal{M}$ with its conditional expectations and indices, **decompose** computes the algebra generated by given matrices or by a finite group, and **su2** uses the full $SU(2)$ symmetry of spin- $\frac{1}{2}$ chains instead of S_{tot}^z alone.

Inclusions and conditional expectations. Let $\mathcal{N} = \bigoplus_{c=1}^C M_{k_c}$ and $\mathcal{M} = \bigoplus_{d=1}^D M_{m_d}$. Every unital inclusion $\iota : \mathcal{N} \rightarrow \mathcal{M}$ is, up to an inner automorphism $\text{Ad } u$ of $\mathcal{M}$, of the form $\iota(x)_d = \bigoplus_c x_c \otimes 1_{\lambda_{dc}}$, given by the inclusion matrix $\Lambda = (\lambda_{dc}) \in \mathbb{Z}_{\geq 0}^{D \times C}$ with $m_d = \sum_c \lambda_{dc} k_c$ [53]: λ_{dc} is the multiplicity of the summand c of $\mathcal{N}$ in the summand d of $\mathcal{M}$. **Inclusion** stores Λ and an optional (batched) unitary u . With the isometries $V_{dc,i} : \mathbb{C}^{k_c} \rightarrow \mathbb{C}^{m_d}$ onto the i -th copy of $\mathbb{C}^{k_c}$, $\iota(x)_d = \sum_{c,i} u_d V_{dc,i} x_c V_{dc,i}^* u_d^*$, so ι is an **InterSectorChannel**, and inclusion matrices of towers multiply. A faithful trace $\tau = \sum_d w_d \text{Tr}_d$ on $\mathcal{M}$ restricts to $\mathcal{N}$ with the weights $\nu = \Lambda^T w$, and

$$E(y)_c = \nu_c^{-1} \sum_{d,i} w_d V_{dc,i}^* u_d^* y_d u_d V_{dc,i}$$

is the unique τ -preserving conditional expectation of $\mathcal{M}$ onto $\mathcal{N}$: it is unital and completely positive, $E \circ \iota = \text{id}$, $E(\iota(n_1)y \iota(n_2)) = n_1 E(y) n_2$, and it is the adjoint of ι with respect to τ and $\tau|_{\mathcal{N}}$. The tests check these properties against dense matrices for the three traces of Section 2, for the Markov trace and for random weights. For a connected inclusion (the bipartite graph of Λ is connected) the Perron–Frobenius vector of $\Lambda \Lambda^T$ gives the Markov trace, and the index is $[\mathcal{M} : \mathcal{N}] = \|\Lambda\|^2$ [52, 53]: $(m/k)^2$ for factors $M_k \subset M_m$, n^2 for $\mathbb{C} \subset M_n$ and n for the diagonal subalgebra $\mathbb{C}^n \subset M_n$. The Watatani index $\text{Ind } E = \sum_j u_j u_j^*$ over a quasi-basis $\{u_j\}$ [55] is central, $(\text{Ind } E)_d = (\Lambda \Lambda^T w)_d / w_d$, and equals $[\mathcal{M} : \mathcal{N}]$ exactly for the Markov trace. The Pimsner–Popa constant $\lambda(E) = \max\{\lambda : E(y) \geq$

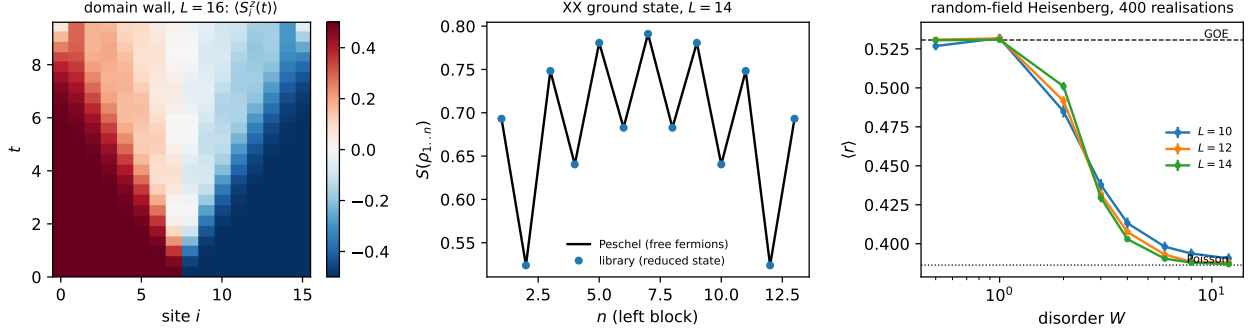


Figure 6: Left: $\langle S_i^z(t) \rangle$ after a domain-wall quench of the XX chain (light cone). Middle: entanglement entropy of the XX ground state, library vs. Peschel’s formula. Right: mean level-spacing ratio of the random-field Heisenberg chain, with the GOE and Poisson values.

λy for all $y \geq 0$] [54] is $(\max_d \sum_c (\nu_c/w_d) \min(k_c, \lambda_{dc}))^{-1}$. For II_1 subfactors $\lambda(E) = [\mathcal{M} : \mathcal{N}]^{-1}$ [54], but for Type I inclusions the two differ: for $\mathbb{C} \subset M_n$, $\lambda(E) = 1/n$, while the constant of the complete order, $\max\{\lambda : E - \lambda \text{id} \text{ completely positive}\} = 1/\|\text{Ind } E\|$ [56], is $1/n^2$. The tests compare both constants with brute-force values (bisection over rank-one y , and Choi matrices). The basic construction $\mathcal{M}_1 = \langle \mathcal{M}, e_{\mathcal{N}} \rangle$ has the sectors of $\mathcal{N}$, of sizes $(\Lambda^T m)_c$, and $\mathcal{M} \subset \mathcal{M}_1$ has the inclusion matrix Λ^T [52, 53]; the library builds the Jones projection $e_{\mathcal{N}}$ (tested: $e_{\mathcal{N}} y e_{\mathcal{N}} = E(y) e_{\mathcal{N}}$, $e_{\mathcal{N}} \in \mathcal{N}'$, the Markov property $\tau_1(y e_{\mathcal{N}}) = [\mathcal{M} : \mathcal{N}]^{-1} \tau(y)$), the Jones tower $\mathcal{N} \subset \mathcal{M} \subset \mathcal{M}_1 \subset \dots$ with constant index, and the relative commutant $\mathcal{N}' \cap \mathcal{M} = \bigoplus_{\lambda_{dc} > 0} M_{\lambda_{dc}}$. All maps act on batches by index gathers, without host synchronisation, and are differentiable.

The spin-chain tower. The $U(1)$ algebras $A_L = \bigoplus_{N=0}^L M_{\binom{L}{N}}$ of Section 7 form the tower $A_L \subset A_{L+1}$, $x \mapsto x \otimes 1_2$, whose Bratteli diagram is Pascal’s triangle: sector N of A_{L+1} receives the sectors $N-1$ (new spin up) and N (new spin down) of A_L . Λ is the incidence matrix of the path with $2L+3$ vertices, so

$$[A_{L+1} : A_L] = \|\Lambda\|^2 = 4 \cos^2 \frac{\pi}{2L+4} \nearrow 4,$$

values $4 \cos^2(\pi/n)$, $n = 4, 6, 8, \dots$ ($2, 3, 2 + \sqrt{2}, \dots$), of the discrete series below 4 in Jones’ index theorem [52]. The tests check the spectrum of the path graph and the index for $L \leq 8$, that the embedding is $x \otimes 1_2$ in the basis of `SpinChain`, and that the conditional expectation for the normalised trace of $(\mathbb{C}^2)^{\otimes(L+1)}$ is the normalised partial trace over the new site. Listing 10 shows the interface.

Listing 10: Inclusions, conditional expectations and indices (abridged from `example_inclusions.py` in `examples`).

```
import math, torch
from torch_vn_algebra import TypeIAlgebra, inclusions as inc

N = TypeIAlgebra([2, 1], [2, 1], precision='double')          # M_2 (+) C
M = TypeIAlgebra([4, 3], [4, 3], precision='double')          # M_4 (+) M_3
I = inc.Inclusion(N, M, [[1, 2], [1, 1]], unitary=M.random_unitary_operator(3))
print(I.index())                                                # ||Lambda||^2 = (7 + sqrt 45)/2
E = I.conditional_expectation('markov')                          # tau-preserving E: M -> N
y = M.operator(torch.randn(3, M.C, M.k_max, M.k_max, dtype=M.dtype))
x = N.operator(torch.randn(3, N.C, N.k_max, N.k_max, dtype=N.dtype))
print((E(I.embed(x) @ y).matrix - (x @ E(y)).matrix).abs().max()) # bimodule property
print(E.index().matrix[0, :, 0, 0].real)                        # Watatani index: [M:N] 1
```

```

print(E.pimsner_popa_constant(), E.pimsner_popa_constant(complete=True))
J, e = I.basic_construction(), I.jones_projection('markov') # M c M_1 = <M, e_N>
eye = e.matrix @ J.embed(y).matrix @ e.matrix # e_N y e_N = E(y) e_N
print((eye - J.embed(I.embed(E(y))).matrix @ e.matrix).abs().max())
for L, T in enumerate(inc.chain_tower(6)): # A_L c A_{L+1}, x -> x (x) 1_2
    print(L, T.index(), 4 * math.cos(math.pi / (2 * L + 4)) ** 2)

```

From a symmetry to the algebra: Wedderburn decomposition. Section 1 obtained the algebra (1) from a symmetry through the commutant. `decompose` carries out this step numerically. Given matrices $S_1, \dots, S_J \in M_n(\mathbb{C})$ (for instance the generators of a Lie algebra representation) or all elements $\pi(g)$ of a unitary representation of a finite group, it returns a unitary V with

$$V^* \mathcal{A} V = \bigoplus_c M_{k_c}(\mathbb{C}) \otimes 1_{m_c}, \quad V^* \mathcal{A}' V = \bigoplus_c 1_{k_c} \otimes M_{m_c}(\mathbb{C}),$$

for the von Neumann algebra $\mathcal{A} = \{S_j, S_j^*\}''$ generated by the matrices, together with maps between $n \times n$ matrices and operators of the algebras $\bigoplus_c M_{k_c}$ and $\bigoplus_c M_{m_c}$ of the library. The algorithm is that of Murota, Kanno, Kojima and Kojima [57] (see also [58]). The eigenspaces of a random Hermitian element of $\mathcal{A}'$ are, generically, irreducible $\mathcal{A}$ -submodules, one per copy of each irreducible representation. For a second random element $Y \in \mathcal{A}'$ and orthonormal bases Q_i, Q_j of two such submodules, $Q_i^* Y Q_j$ intertwines them; by Schur's lemma it vanishes if they are inequivalent and is a non-zero multiple of a unitary (generically) if they are equivalent, so the non-zero blocks give the equivalence classes and their polar factors align the bases of the copies. Random elements of $\mathcal{A}'$ come from the twirl $|G|^{-1} \sum_g \pi(g) X \pi(g)^*$ for a finite group, or from the null space of $X \mapsto ([H_j, X])_j$ with the Hermitian parts H_j of the generators, whose size is reduced by first diagonalising a random Hermitian element of $\mathcal{A}$. Every result is verified before it is returned: the generators lie in $\mathcal{B} = V(\bigoplus_c M_{k_c} \otimes 1_{m_c})V^*$, fresh random elements of $\mathcal{A}'$ lie in $\mathcal{B}'$, and $\sum_c m_c^2 = \dim \mathcal{A}'$, which together give $\mathcal{A} = \mathcal{B}$ by the double commutant theorem; a failed check triggers a new random draw. The decomposition is over $\mathbb{C}$ and uses dense linear algebra. For an algebra closed under complex conjugation (real generators) entrywise conjugation maps every sector onto a sector, $\overline{V_c} = V_d(J_c \otimes M_c)$ by Schur's lemma, and `real_structure` reads off the type of every simple summand of the real algebra: complex for $d \neq c$ (a pair of conjugate sectors), real for $d = c$ and $J_c \overline{J_c} = 1$, quaternionic for $J_c \overline{J_c} = -1$; the sign has a gap of 2, so the decision needs no tolerance, and `real_basis` returns a real orthogonal basis adapted to the types (Takagi factorisation of the symmetric unitary J_c). In a quaternionic sector the multiplicity block of a real symmetric commuting operator has doubly degenerate eigenvalues (Kramers). The tests recover block structures hidden by random unitaries, the Fourier basis of the regular representation of $\mathbb{Z}_n$, and Schur–Weyl duality [59]: on $(\mathbb{C}^2)^{\otimes L}$, $L \leq 6$, the algebra generated by the permutations of the tensor factors and the algebra generated by the total spin have the same central projections, with k_c and m_c exchanged (the dimensions of the irreducible representations of S_L of shape $(L - j, j)$ and $2S + 1$, $S = \frac{L}{2} - j$); Listing 11 shows it for $L = 4$.

SU(2) symmetry of spin chains. The Heisenberg Hamiltonian $H = \sum J_{ij} \mathbf{S}_i \cdot \mathbf{S}_j$ commutes with all of SU(2), not only with S_{tot}^z . On $(\mathbb{C}^2)^{\otimes L} = \bigoplus_S \mathbb{C}^{m_S} \otimes V_S$, with V_S the spin- S representation,

$$\text{SU}(2)' = \bigoplus_S M_{m_S}(\mathbb{C}), \quad m_S = \binom{L}{L/2 - S} - \binom{L}{L/2 - S - 1}, \quad \sum_S (2S + 1) m_S = 2^L,$$

and an invariant Hamiltonian is $\bigoplus_S H_S \otimes 1_{2S+1}$: every multiplet is computed once instead of $2S + 1$ times, and the blocks shrink from $\binom{L}{L/2+S}$ to m_S . `SU2Chain` realises $\mathbb{C}^{m_S}$ as the highest-weight vectors

$\ker S^+ \cap \{S_{\text{tot}}^z = S\}$ in the $U(1)$ sector $N = L/2 + S$ of `SpinChain`. Their orthonormal basis B_S is built site by site with Clebsch–Gordan coefficients (the Yamanouchi basis, labelled by the sequence of intermediate spins $S_1 = \frac{1}{2}, \dots, S_L = S$); it is orthonormal by construction, so no null space or orthogonalisation is needed, and it is expressed directly in the $U(1)$ basis, where the block of an invariant operator is $A_S = B_S^* A_N B_S$. Clebsch–Gordan coefficients are computed exactly in rational arithmetic (Racah’s formula [60], Condon–Shortley phases). By the Wigner–Eckart theorem [61] the reduced matrix elements of the site spins follow from highest-weight vectors alone, and $\mathbf{S}_i \cdot \mathbf{S}_j$ is assembled from them as a sum over intermediate spins, so spin correlations need no sum over magnetic quantum numbers. For $L = 14$ the largest $U(1)$ block, $\binom{14}{7} = 3432$, becomes $m_1 = m_2 = 1001$, and the cubic cost of diagonalising all blocks, $\sum_N \binom{L}{N}^3$ against $\sum_S m_S^3$, drops by a factor of about 48; for $L = 16$ the largest block shrinks from 12870 to 3640. The bases of all spins hold $\sum_S m_S \binom{L}{L/2+S}$ numbers ($1.0 \cdot 10^8$ for $L = 16$), which limits dense $SU(2)$ computations to about $L = 16$. The tests compare the full spectra of open and periodic chains, with a field and next-nearest-neighbour couplings, for $L \leq 12$ with dense diagonalisation (without next-nearest-neighbour couplings also with the $U(1)$ sectors; to 10^{-10}), and reproduce the ring energies of Table 12 for $L \leq 14$ (to 10^{-9} , with total spin 0), the theorem of Lieb and Mattis [62] for open chains and even rings with $L \leq 12$ (the ground state has $S = 0$ for even and $S = \frac{1}{2}$ for odd L , and the lowest energy $E_0(S)$ increases strictly with S), the two exactly degenerate singlets with $E = -3L/8$ at the Majumdar–Ghosh point $J_2 = J/2$ [63], textbook and SymPy values of the Clebsch–Gordan coefficients, and Wigner–Eckart matrix elements and spin correlations against direct evaluation.

Listing 11: Schur–Weyl duality from the symmetries alone, and the $SU(2)$ blocks of a Heisenberg ring (cf. `examples/example_decompose.py` and `example_su2.py`).

```
import torch_vn_algebra.decompose as dc
from torch_vn_algebra import SU2Chain

S4 = dc.from_group(dc.permutation_representation(4, 2))      # all 24 permutations of 4 qubits
SU2 = dc.from_generators(dc.total_spin(4))                  # S_x, S_y, S_z
print(S4.k_factors, S4.multiplicities)                       # (1, 2, 3), (5, 1, 3)
print(SU2.k_factors, SU2.multiplicities)                     # (1, 3, 5), (2, 3, 1): swapped
chain = SU2Chain(14, 'periodic', complex_valued=False)
print(chain.multiplicities)                                   # m_S for S = 0..7, largest 1001
H = chain.heisenberg(J=1.0)                                  # (+)_S H_S in (+)_S M_{m_S}
gs = chain.ground_state(H)
print(gs.energy, gs.spin, chain.lowest_energies(H))          # S = 0; E_0(S) increases with S
```

9 Comparison with NumPy/SciPy

To separate what the library adds from what plain array code already gives, the same computations were written three ways (`scripts/baselines.py`; Table 13). (A) The Monte Carlo of Exp. 1: B pairs of positive operators with prescribed contrast and Haar-orthogonal eigenvectors in every sector, a Haar-orthogonal $U \in \mathcal{M}$, and z . The direct NumPy/SciPy formulation loops over samples and sectors with `scipy.stats.ortho_group`; a hand-vectorised NumPy version uses stacked QR decompositions with the phase correction; the library needs three calls. (B) Ground states of sector Hamiltonians: `scipy.sparse.linalg.eigsh` (ARPACK [36]), one call per Hamiltonian, against the batched Lanczos solver. The three Monte Carlo versions sample the same distribution: their means differ by at most 1.2 standard errors, a two-sample Kolmogorov–Smirnov test of the library against vectorised NumPy gives $p \geq -$, and the ground-state energies agree to round-off. All times are medians of up to five runs after a warm-up.

The comparison is instructive in both directions. On a CPU the gain over the loop depends on the block size: it is large for small blocks, where the Python overhead of the loop dominates, and small for large blocks, where both spend their time in the same LAPACK routines. Against carefully vectorised NumPy the library is not systematically faster on a CPU (Table 13): both call the same kernels. Its value there is that the vectorised, correctly sampled, block-aware version is the default rather than an expert rewrite, and that the same code runs on a GPU and can be differentiated. The NumPy versions work in double precision, so the library’s CPU and GPU columns do too; the last column repeats the GPU run in single precision, since double precision is 32–64 times slower on consumer GPUs. For ground states, ARPACK is the reference for a single Hamiltonian on a CPU; our Lanczos solver re-orthogonalises fully for robustness and is batched over Hamiltonians, which is what lets disorder averages and GPUs pay off (Table 13).

C	k	NumPy loop	NumPy vectorised	library (CPU)	library (GPU)	GPU, FP32
1	2	160	2.49	1.50	124	
16	16	2852	690	884	2000	
32	16	5702	1380	1688	4005	

L	sector dim	Hamiltonians	eigsh [s]	library, CPU [s]	library, GPU [s]	max $ \Delta E_0 $
16	12870	1	0.0288	0.0552	6.2e-15	
16	12870	20	0.621	0.114	4.3e-14	
18	48620	1	0.115	0.0588	3.6e-15	
20	184756	1	0.540	0.0642	1.1e-14	

Table 13: Top: time per Monte Carlo sample [μ s] for (A); bottom: ground states of Heisenberg rings (one Hamiltonian) and random-field chains ($W = 3$, several Hamiltonians) in the sector $N = L/2$, total time for all Hamiltonians (“–”: no GPU in this run) (full run on NVIDIA GeForce RTX 5090).

10 Performance

Table 14 and Figure 7 report the wall-clock time per sample of library operations (not of isolated kernels), in single precision, with batch $B = \min(4096, 2^{22}/(Ck^2))$ (shown in the table) (full run on NVIDIA GeForce RTX 5090; CPU with 16 threads; PyTorch 2.14.1+cu130, CUDA 13.0, cuSOLVER 12004). The versions matter: batched Hermitian eigensolvers on GPUs cover all block sizes only in recent cuSOLVER and PyTorch releases (older releases fall back to a loop over the matrices for blocks larger than 32), and batched SVD on GPUs is limited to blocks of size at most 32, which is why the library uses the Hermitian eigensolver wherever the operator is self-adjoint (and the eigensolver of the Hermitian dilation $\begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix}$ for singular values of larger blocks). Batched QR on GPUs factorises small matrices one at a time; Haar sampling therefore uses a batched Gram–Schmidt orthonormalisation with reorthogonalisation on GPUs, which gives the same matrices as QR with the phase correction. On the CPU, PyTorch’s batched eigen-, singular-value and QR decompositions loop over the matrices on one core; the library splits such batches into one chunk per thread (identical results), so the CPU column uses all cores. The operations are: generating random operators with prescribed spectra, $f(A)$ by eigendecomposition, $|A|$ (singular values), the Michelson contrast, applying a channel, the Lindblad channel $e^{t\mathcal{L}}$, and one gradient step of the optimisation; the table gives the CPU and GPU times and their ratio.

Besides speed, the structure of the computation is checked directly (`scripts/gpu_check.py` on a GPU, `tests/test_sync_free.py` on a CPU): the operations read no values back to the host (each

such read synchronises a GPU), and their number of kernel launches does not grow with the number of sectors C at a fixed total size. Per-sector work of this kind made the random generation of an earlier version of the library slower on a GPU than on a CPU; the check would have found it.

The first version of this work reported GPU speedups for raw kernels on a P100 and found them only for part of the configurations. That observation reflects where GPUs help: a GPU needs enough independent work per kernel launch. For computations in (1) this means many samples, many sectors or large blocks, and matrix products rather than decompositions of small matrices. The design follows from it: everything is batched, so that the batch dimension supplies the parallel work, and the cost module warns before configurations that would be slow. For workloads that are small or dominated by a single eigenproblem, the CPU (or SciPy, Section 9) is the right tool, and the library runs there unchanged.

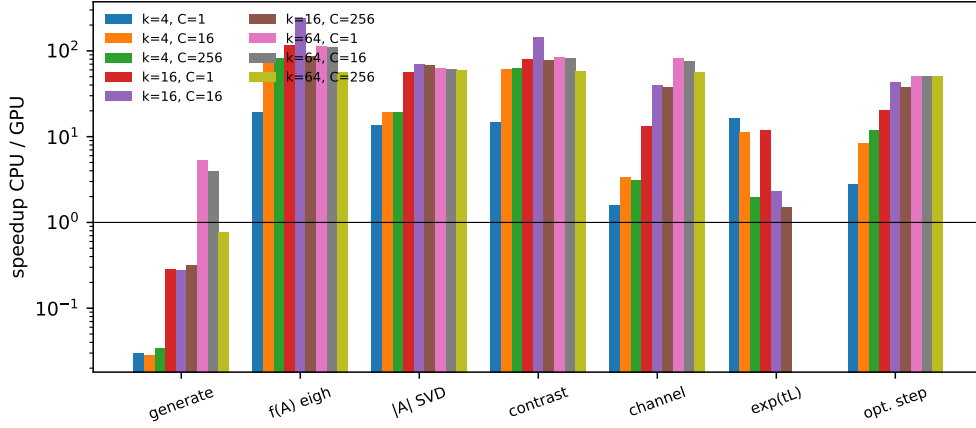


Figure 7: Speedup CPU/GPU (logarithmic axis; above 1 the GPU is faster) of the operations of Table 14, one bar per configuration (k, C) .

11 Limitations and roadmap

- *Layout.* Blocks are padded to $k_{\max}$. For very different sector sizes (binomial sectors of spin chains) one works in single sectors, and **SU2Chain** builds the block of a single total spin without padding; operators acting on all sectors at once would need a bucketed layout, which is not implemented yet.
- *Symmetries.* Abelian charges with an arbitrary fusion rule and lattice translations are supported, and for spin- $\frac{1}{2}$ chains the $SU(2)$ multiplets reduce the blocks further (**su2**; dense up to $L \approx 16$, Section 8). Not yet implemented: $SU(2)$ combined with momentum sectors, chains of spins larger than $\frac{1}{2}$, reflections and spin flip, and sparse (Lanczos) solvers in the $SU(2)$ blocks. The numerical Wedderburn decomposition handles any symmetry given by matrices, but with dense linear algebra; it identifies the real, complex and quaternionic summands of real algebras, but the core algebra class stores only real and complex blocks (quaternionic blocks are kept as complex blocks together with their J_c). Dense methods reach about 16–18 spins in the $U(1)$ sectors, the sparse Lanczos and Krylov solvers handle the half-filled sector of 24 spins (dimension $2.7 \cdot 10^6$; Table 11 in the full run), and about 26 spins on a GPU with tens of gigabytes (an estimate from the memory of the Krylov basis); tensor-network methods are out of scope.

- *Optimisation.* The search is local; envelopes are lower bounds for suprema (upper bounds for infima). Gradients of the functional calculus are finite at degenerate spectra, but gradients through the eigenvectors returned by `eigh` are not; second derivatives of $f(A)$ are correct at non-degenerate spectra only.
- *Divergences.* For $\alpha < 1$ the sandwiched divergence depends on all eigenvalues of $\sigma^g \rho \sigma^g$, $g = (1 - \alpha)/(2\alpha)$, whose condition number grows without bound as $\alpha \rightarrow 0$; for small α and ill-conditioned states the computed value is not accurate (hence the restriction $\alpha \geq 0.2$, $\rho, \sigma \geq \varepsilon/d$ in Section 6.1).
- *Scale.* Choi matrices, superoperators and $e^{t\mathcal{L}}$ work with $k_c^2 \times k_c^2$ matrices and are limited to blocks of a few tens; RK4 integration scales to larger blocks.
- *Scope.* Finite dimensions and Type I only; no tensor-network methods, and sparse methods only for sector Hamiltonians (Lanczos and Krylov).

12 Availability and reproducibility

The code is available under the MIT licence at

https://github.com/andre-a-no/von_neumann_type_i.

`bash scripts/run_all.sh full cuda` regenerates every table and figure of this text on a GPU; `bash scripts/run_all.sh` runs a reduced version on a CPU in a few minutes (written to `.quick_output/`, not into the paper’s tables), and `python scripts/gpu_check.py` checks a GPU in a few minutes (correctness against the CPU, absence of per-sector work, timings). The tables state the device, the mode and the commit of the code they were produced with.

References

- [1] I. Nikolaeva, A. Novikov, “Finite-dimensional Type I von Neumann algebras in PyTorch: a GPU-accelerated framework for random block-diagonal operators”, arXiv:2606.15882v1 (2026).
- [2] L. T. Gardner, “An inequality characterizes the trace”, *Canad. J. Math.* 31 (1979), 1322–1328.
- [3] D. Petz, J. Zemánek, “Characterizations of the trace”, *Linear Algebra Appl.* 111 (1988), 43–52.
- [4] A. Novikov, O. Tikhonov, “Characterization of central elements of operator algebras by inequalities”, arXiv:1502.01267 (2015).
- [5] D. Virostek, “Connections between centrality and local monotonicity of certain functions on C^* -algebras”, *J. Math. Anal. Appl.* 453 (2017), 221–226.
- [6] S. A. Abed, I. A. Nikolaeva, A. A. Novikov, “Generalisation of Michelson contrast for operators and its properties”, *Lobachevskii J. Math.* 45 (2024), 3835–3848.
- [7] A. Novikov, S. A. Abed, I. Nikolaeva, “Generalisation of Michelson contrast for operators”, arXiv:2010.10130 (2020).
- [8] J. R. Johansson, P. D. Nation, F. Nori, “QuTiP 2: A Python framework for the dynamics of open quantum systems”, *Comput. Phys. Commun.* 184 (2013), 1234–1240.

- [9] Qiskit contributors, “Qiskit: An open-source framework for quantum computing”, 2019, <https://qiskit.org>.
- [10] P. Guilmin, R. Gautier, A. Bocquet, E. Genois, D. Weiss, “dynamiqs: an open-source Python library for GPU-accelerated and differentiable simulation of quantum systems”, <https://github.com/dynamiqs/dynamiqs> (2024).
- [11] F. Mezzadri, “How to generate random matrices from the classical compact groups”, *Notices Amer. Math. Soc.* 54 (2007), 592–604.
- [12] M. L. Mehta, *Random Matrices*, 3rd ed., Elsevier, 2004.
- [13] D. N. Page, “Average entropy of a subsystem”, *Phys. Rev. Lett.* 71 (1993), 1291–1294.
- [14] K. Życzkowski, H.-J. Sommers, “Induced measures in the space of mixed quantum states”, *J. Phys. A* 34 (2001), 7111–7125.
- [15] I. Peschel, “Calculation of reduced density matrices from correlation functions”, *J. Phys. A* 36 (2003), L205–L208.
- [16] V. Oganesyan, D. A. Huse, “Localization of interacting fermions at high temperature”, *Phys. Rev. B* 75 (2007), 155111.
- [17] Y. Y. Atas, E. Bogomolny, O. Giraud, G. Roux, “Distribution of the ratio of consecutive level spacings in random matrix ensembles”, *Phys. Rev. Lett.* 110 (2013), 084101.
- [18] M. Takesaki, *Theory of Operator Algebras I*, Springer, 2002.
- [19] G. C. Wick, A. S. Wightman, E. P. Wigner, “The intrinsic parity of elementary particles”, *Phys. Rev.* 88 (1952), 101–105.
- [20] S. D. Bartlett, T. Rudolph, R. W. Spekkens, “Reference frames, superselection rules, and quantum information”, *Rev. Mod. Phys.* 79 (2007), 555–609.
- [21] W. H. Zurek, “Decoherence, einselection, and the quantum origins of the classical”, *Rev. Mod. Phys.* 75 (2003), 715–775.
- [22] A. W. Sandvik, “Computational studies of quantum spin systems”, *AIP Conf. Proc.* 1297 (2010), 135–338.
- [23] P. Weinberg, M. Bukov, “QuSpin: a Python package for dynamics and exact diagonalisation of quantum many body systems part I: spin chains”, *SciPost Phys.* 2 (2017), 003.
- [24] M. Goldstein, E. Sela, “Symmetry-resolved entanglement in many-body systems”, *Phys. Rev. Lett.* 120 (2018), 200602.
- [25] J. C. Xavier, F. C. Alcaraz, G. Sierra, “Equipartition of the entanglement entropy”, *Phys. Rev. B* 98 (2018), 041106.
- [26] P. Zanardi, M. Rasetti, “Noiseless quantum codes”, *Phys. Rev. Lett.* 79 (1997), 3306–3309.
- [27] E. Knill, R. Laflamme, L. Viola, “Theory of quantum error correction for general noise”, *Phys. Rev. Lett.* 84 (2000), 2525–2528.

- [28] C. Bény, A. Kempf, D. W. Kribs, “Generalization of quantum error correction via the Heisenberg picture”, *Phys. Rev. Lett.* 98 (2007), 100502.
- [29] S. Hadfield, Z. Wang, B. O’Gorman, E. G. Rieffel, D. Venturelli, R. Biswas, “From the quantum approximate optimization algorithm to a quantum alternating operator ansatz”, *Algorithms* 12 (2019), 34.
- [30] F. J. Dyson, “The threefold way. Algebraic structure of symmetry groups and ensembles in quantum mechanics”, *J. Math. Phys.* 3 (1962), 1199–1215.
- [31] T. Cohen, M. Welling, “Group equivariant convolutional networks”, *Proc. ICML 2016*, PMLR 48, 2990–2999.
- [32] M. Finzi, M. Welling, A. G. Wilson, “A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups”, *Proc. ICML 2021*, PMLR 139, 3318–3328.
- [33] C. H. Martin, M. W. Mahoney, “Implicit self-regularization in deep neural networks: evidence from random matrix theory and implications for learning”, *J. Mach. Learn. Res.* 22 (2021), 1–73.
- [34] C. R. Harris et al., “Array programming with NumPy”, *Nature* 585 (2020), 357–362.
- [35] P. Virtanen et al., “SciPy 1.0: fundamental algorithms for scientific computing in Python”, *Nat. Methods* 17 (2020), 261–272.
- [36] R. B. Lehoucq, D. C. Sorensen, C. Yang, *ARPACK Users’ Guide*, SIAM, 1998.
- [37] M. Fishman, S. R. White, E. M. Stoudenmire, “The ITensor software library for tensor network calculations”, *SciPost Phys. Codebases* 4 (2022).
- [38] J. Hauschild, F. Pollmann, “Efficient numerical simulations with tensor networks: Tensor Network Python (TeNPy)”, *SciPost Phys. Lect. Notes* 5 (2018).
- [39] M. Fukuda, R. König, I. Nechita, “RTNI – a symbolic integrator for Haar-random tensor networks”, *J. Phys. A* 52 (2019), 425303.
- [40] N. J. Higham, *Functions of Matrices: Theory and Computation*, SIAM, 2008.
- [41] L. N. Trefethen, J. A. C. Weideman, “The exponentially convergent trapezoidal rule”, *SIAM Rev.* 56 (2014) 385–458.
- [42] Ju. L. Daleckii, S. G. Krein, “Integration and differentiation of functions of Hermitian operators and applications to the theory of perturbations”, *Amer. Math. Soc. Transl. (2)* 47 (1965) 1–30.
- [43] R. Bhatia, *Matrix Analysis*, Graduate Texts in Mathematics 169, Springer, 1997.
- [44] M. F. Kasim, “Derivatives of partial eigendecomposition of a real symmetric matrix for degenerate cases”, *arXiv:2011.04366* (2020).
- [45] D. Petz, “Quasi-entropies for finite quantum systems”, *Rep. Math. Phys.* 23 (1986), 57–65.
- [46] D. Petz, “Sufficient subalgebras and the relative entropy of states of a von Neumann algebra”, *Commun. Math. Phys.* 105 (1986), 123–131.
- [47] M. Müller-Lennert, F. Dupuis, O. Szehr, S. Fehr, M. Tomamichel, “On quantum Rényi entropies: a new generalization and some properties”, *J. Math. Phys.* 54 (2013), 122203.

- [48] M. M. Wilde, A. Winter, D. Yang, “Strong converse for the classical capacity of entanglement-breaking and Hadamard channels via a sandwiched Rényi relative entropy”, *Commun. Math. Phys.* 331 (2014), 593–622.
- [49] R. L. Frank, E. H. Lieb, “Monotonicity of a relative Rényi entropy”, *J. Math. Phys.* 54 (2013), 122201.
- [50] S. Beigi, “Sandwiched Rényi divergence satisfies data processing inequality”, *J. Math. Phys.* 54 (2013), 122202.
- [51] M. Junge, R. Renner, D. Sutter, M. M. Wilde, A. Winter, “Universal recovery maps and approximate sufficiency of quantum relative entropy”, *Ann. Henri Poincaré* 19 (2018), 2955–2978.
- [52] V. F. R. Jones, “Index for subfactors”, *Invent. Math.* 72 (1983), 1–25.
- [53] F. M. Goodman, P. de la Harpe, V. F. R. Jones, *Coxeter Graphs and Towers of Algebras*, MSRI Publications 14, Springer, 1989.
- [54] M. Pimsner, S. Popa, “Entropy and index for subfactors”, *Ann. Sci. École Norm. Sup. (4)* 19 (1986), 57–106.
- [55] Y. Watatani, “Index for C^* -subalgebras”, *Mem. Amer. Math. Soc.* 83 (1990), no. 424.
- [56] M. Frank, E. Kirchberg, “On conditional expectations of finite index”, *J. Operator Theory* 40 (1998), 87–111.
- [57] K. Murota, Y. Kanno, M. Kojima, S. Kojima, “A numerical algorithm for block-diagonal decomposition of matrix $*$ -algebras with application to semidefinite programming”, *Japan J. Indust. Appl. Math.* 27 (2010), 125–160.
- [58] T. Maehara, K. Murota, “A numerical algorithm for block-diagonal decomposition of matrix $*$ -algebras with general irreducible components”, *Japan J. Indust. Appl. Math.* 27 (2010), 263–293.
- [59] W. Fulton, J. Harris, *Representation Theory: A First Course*, Graduate Texts in Mathematics 129, Springer, 1991.
- [60] G. Racah, “Theory of complex spectra. II”, *Phys. Rev.* 62 (1942), 438–462.
- [61] A. R. Edmonds, *Angular Momentum in Quantum Mechanics*, Princeton University Press, 1957.
- [62] E. Lieb, D. Mattis, “Ordering energy levels of interacting spin systems”, *J. Math. Phys.* 3 (1962), 749–751.
- [63] C. K. Majumdar, D. K. Ghosh, “On next-nearest-neighbor interaction in linear chain. I”, *J. Math. Phys.* 10 (1969), 1388–1398.
- [64] A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library”, *NeurIPS* 2019.

operation	k	C	batch	GPU [μ s]	CPU [μ s]	speedup
generate	4	1	4096	63.5	1.90	0.0299
f(A) eigh	4	1	4096	0.160	3.06	19.2
A SVD	4	1	4096	0.388	5.27	13.6
contrast	4	1	4096	0.0979	1.44	14.7
channel	4	1	4096	0.116	0.186	1.60
exp(tL)	4	1	4096	2.05	34.1	16.6
opt. step	4	1	4096	2.76	7.78	2.82
generate	4	16	4096	995	28.1	0.0283
f(A) eigh	4	16	4096	0.514	44.7	86.9
A SVD	4	16	4096	4.17	80.7	19.4
contrast	4	16	4096	0.358	22.0	61.5
channel	4	16	4096	1.34	4.53	3.38
exp(tL)	4	16	1024	49.2	556	11.3
opt. step	4	16	4096	9.54	80.2	8.41
generate	4	256	1024	15710	534	0.0340
f(A) eigh	4	256	1024	9.01	742	82.3
A SVD	4	256	1024	66.3	1283	19.4
contrast	4	256	1024	5.63	359	63.7
channel	4	256	1024	20.9	65.0	3.12
exp(tL)	4	256	64	7743	15365	1.98
opt. step	4	256	1024	133	1604	12.0
generate	16	1	4096	60.9	17.3	0.283
f(A) eigh	16	1	4096	0.206	23.8	116
A SVD	16	1	4096	0.780	44.5	57.0
contrast	16	1	4096	0.137	10.9	79.5
channel	16	1	4096	0.129	1.72	13.4
exp(tL)	16	1	64	1149	13608	11.8
opt. step	16	1	4096	3.61	74.3	20.6
generate	16	16	1024	1032	290	0.281
f(A) eigh	16	16	1024	1.76	419	238
A SVD	16	16	1024	10.7	746	69.6
contrast	16	16	1024	1.26	182	145
channel	16	16	1024	1.63	65.0	39.8
exp(tL)	16	16	4	79221	181417	2.29
opt. step	16	16	1024	34.9	1505	43.1
generate	16	256	64	18838	5937	0.315
f(A) eigh	16	256	64	78.0	6736	86.3
A SVD	16	256	64	172	11860	68.9
contrast	16	256	64	36.9	2871	77.7
channel	16	256	64	26.3	989	37.6
exp(tL)	16	256	1	2349043	3558629	1.51
opt. step	16	256	64	567	21657	38.2
generate	64	1	1024	62.0	329	5.30
f(A) eigh	64	1	1024	3.47	393	113
A SVD	64	1	1024	13.1	820	62.4
contrast	64	1	1024	2.48	211	85.0
channel	64	1	1024	0.859	71.2	82.9
opt. step	64	1	1024	34.2	1737	50.8
generate	64	16	64	1262	4995	3.96
f(A) eigh	64	16	64	59.2	6495	110
A SVD	64	16	64	211	12966	61.4
contrast	64	16	64	41.2	3424	83.0
channel	64	16	64	13.7	1054	76.8
opt. step	64	16	64	550	28154	51.2
generate	64	256	4	132631	103301	0.779
f(A) eigh	64	256	4	1753	99206	56.6
A SVD	64	256	4	3398	204740	60.2
contrast	64	256	4	933	54526	58.4
channel	64	256	4	217	12346	57.0