DMRGPY user guide

Contents

1 Introduction **1**

2 Models and Hilbert spaces **3**

3 Hamiltonians and operators **5**

4 Ground state **6**

5 Controlling the solver **10**

6 Excited states and gaps **12**

7 Entanglement **17**

8 Correlators and distributions **21**

9 Mean-field decoupling **22**

10 Topological invariants **24**

11 Fidelity susceptibility **25**

12 Dynamical correlators **26**

13 Choosing a correlator method **30**

14 Real-time evolution **34**

15 STM and Kondo spectra **36**

16 Finite temperature **59**

17 Open systems **62**

18 Infinite chains **63**

19 Cookbook **67**

20 Post-processing tools **71**

21 Performance **71**

1 Introduction

DMRGPY is a Python library for quasi-one-dimensional quantum many-body systems: spin chains, spinless and spinful fermions, bosons and parafermions. Ground states, excitations, dynamical correlators, real-time evolution

and finite temperature are computed with matrix product states. Small systems can also be solved with exact diagonalization (ED), so every DMRG result can be checked on a short chain.

This guide explains how to compute physical quantities. For each capability it gives the quantity, its formula, the code and the keywords that matter. The code itself is on GitHub, and the whole guide is also available as one PDF.

1.1 Installation

```
pip install dmrgpy
```

This installs ED and a DMRG code written in pure Python. It needs no compiler and works on Linux, Mac and Windows. The faster C++ backend (ITensor) is compiled from a clone of the repository with `python install.py`; the README explains how.

1.2 A first calculation

The ground-state energy of a spin-1/2 Heisenberg chain of 12 sites, with DMRG and with ED:

```
from dmrgpy import spinchain
n = 12
sc = spinchain.Spin_Chain(["S=1/2" for i in range(n)])
h = sum(sc.Sx[i]*sc.Sx[i+1] + sc.Sy[i]*sc.Sy[i+1] + sc.Sz[i]*sc.Sz[i+1]
        for i in range(n-1))
sc.set_hamiltonian(h)
print(sc.gs_energy())              # DMRG
print(sc.gs_energy(mode="ED"))     # exact diagonalization
```

Both print $E_0 \approx -5.1420906$. Every chain class has the same methods, and most of them accept `mode="ED"` in the same way.

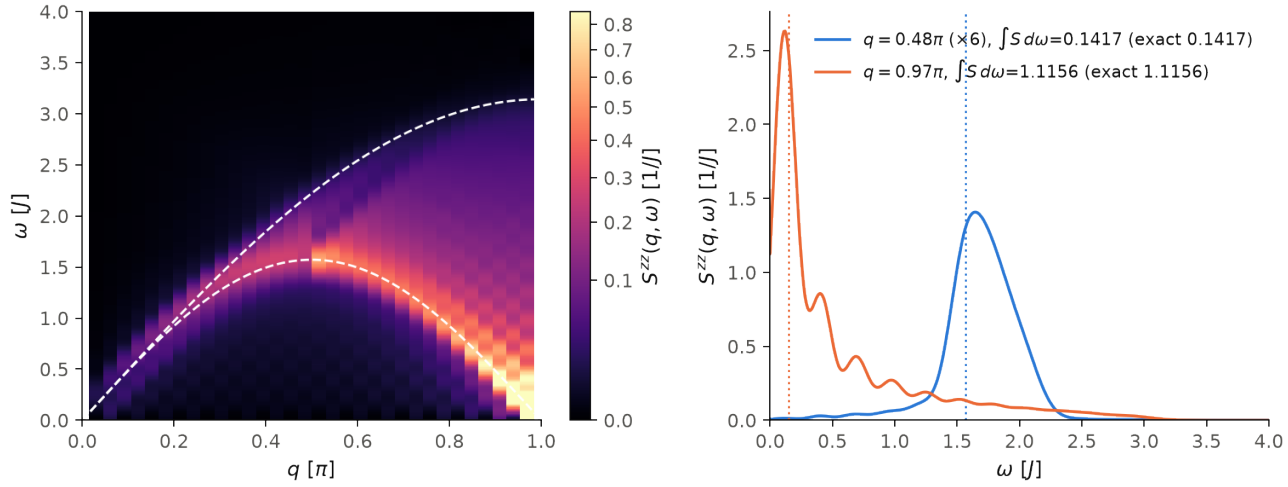


Figure 1: Dynamical spin structure factor of the Heisenberg chain

A Heisenberg chain with $n = 32$ gives the dynamical spin structure factor $S(q,\omega)$, the quantity measured by inelastic neutron scattering. The weight fills the two-spinon continuum between its exact edges (dashed). See Dynamical correlators.

The code of this figure is in `docs/figures/dynamical-correlators_sqw/`.

1.3 Backends

The keyword `itensor_version=` of a chain chooses the DMRG code. The results are the same; only the speed and a few features differ.

- `3`: ITensor v3 in C++, the default when it is compiled. The fastest.
- `"python"`: the pure-Python DMRG, the default after `pip install`. It can also run on a GPU (Performance).
- `2`: ITensor v2 in C++. Some features are missing.
- `"julia_live"`: ITensors.jl in Julia, after `pip install "dmrgpy[julia]"`.

`mode="ED"` uses exact diagonalization instead of DMRG, for small chains.

1.4 Contents of the guide

- Getting started: the chain classes, their local operators, and how to write a Hamiltonian.
- Ground state: energies, expectation values, correlators and conserved sectors; the solver settings and warm starts; excited states and degeneracies; entanglement; four-point correlators and distributions; mean-field decoupling.
- Phases: Berry phases, fermion parity and the fidelity susceptibility.
- Dynamics: dynamical correlators, spectral functions and the local density of states, real-time evolution and STM spectra.
- Finite temperature: purification, METTS and ED.
- Infinite chains: iDMRG and VUMPS directly in the thermodynamic limit.
- More: the common models written out, post-processing tools and performance.

2 Models and Hilbert spaces

Every model is a chain of n local Hilbert spaces, $\mathcal{H} = \bigotimes_{i=1}^{n} \mathcal{H}_i$. The chain class fixes the local space and gives lists of local operators, indexed by site. The Hamiltonian and the observables are then built from these operators (Hamiltonians and operators).

Chain class	Local Hilbert space	Operators
`spinchain.Spin_Chain(sites)`	spin $S = \frac{1}{2}$ to 3, one label per site (`"S=1/2"`, `"S=1"`, …)	`Sx`, `Sy`, `Sz`
`fermionchain.Fermionic_Chain(n)`	spinless fermion, empty or occupied	`C`, `Cdag`, `N`, Jordan-Wigner string `F`
`fermionchain.Majorana_Chain(n)`	n Majorana modes, built from $n/2$ fermions	`G`, with $\gamma_{2j} = (c_j + c_j^\dagger)/\sqrt{2}$, $\gamma_{2j+1} = i(c_j - c_j^\dagger)/\sqrt{2}$
`fermionchain.Spinful_Fermionic_Chain(n)`	spin-1/2 fermion: $0, \uparrow, \downarrow, \uparrow\downarrow$	`Cup`, `Cdagup`, `Cdn`, `Cdagdn`, `Nup`, `Ndn`, `Ntot`, `Sx`, `Sy`, `Sz`, `Delta`
`fermionchain.Spinful_Fermionic_Chain_Native(n)`	the same, one 4-state site per orbital	the same as above
`bosonchain.Bosonic_Chain(n, maxnb=...)`	boson with 0 to $\text{maxnb}_i - 1$ particles; `maxnb` is a list with one entry per site, 4 everywhere by default	`A`, `Adag`, `N`, projectors `D[i][k]` $= \lvert k\rangle\langle k\rvert$

Chain class	Local Hilbert space	Operators
`bosonchain.SpinBoson_Chain(sites)`	boson and spin sites mixed, labels `"B"`, `"B6"`, `"S=1/2"`, …	`A`, `Adag`, `N`, `D` on boson sites; `Sx`, `Sy`, `Sz` on spin sites
`parafermionchain.Parafermionic_Chain(n, Z=3)`	$\mathbb{Z}_N$ clock model, $N = 2, 3, 4$	clock and shift `Sig`, `Tau`, parafermions `Chi`, `Psi`
`mixedchain.Mixed_Spin_Fermion_Chain(sites)`	spin sites and spinful-fermion sites mixed, labels `"1/2"`, `"1"`, …, `"F"`	`Sx`, `Sy`, `Sz` everywhere; `Cup`, `Cdagup`, `Cdn`, `Cdagdn`, `Nup`, `Ndn`, `Ntot`, `Delta` on fermion sites

The spin operators of a spinful fermion are $S^z_i = \frac{1}{2}(n_{i\uparrow} - n_{i\downarrow})$, $S^+_i = c^\dagger_{i\uparrow} c_{i\downarrow}$, and the onsite pairing is $\Delta_i = \frac{1}{2} c_{i\uparrow} c_{i\downarrow}$. In the mixed chains, an operator that does not exist on a site (for example `Cup` on a spin site) is zero.

`Spinful_Fermionic_Chain` stores each orbital as two spinless sites, and `Spinful_Fermionic_Chain_Native` as one 4-state site. The physics and the sign conventions are the same. In practice the native class is not faster, so `Spinful_Fermionic_Chain` is the default choice. `Mixed_Spin_Fermion_Chain` is for models with a true local moment next to conduction electrons, such as the Kondo lattice (`examples/fermion_models/mixed_spin_fermion_chain`).

`Mixed_Spin_Fermion_Chain` also runs with `mode="ED"`. The Hilbert space is the product of the sites: a spin S has $2S+1$ states, and a fermion site has the four states $0, \uparrow, \downarrow, \uparrow\downarrow$. The fermion operators carry the Jordan-Wigner string over the fermion sites only,

$$c_{j\sigma} = \Big(\prod_{\text{fermion modes } k<j\sigma} (1 - 2n_k) \Big)\, a_{j\sigma},$$

meaning that a spin site between two fermion sites gives no sign, and the spins commute with the electrons. DMRG uses the same convention, so the two solvers can be compared directly:

```
from dmrgpy import mixedchain
mc = mixedchain.Mixed_Spin_Fermion_Chain(["F", "1/2", "F"])
h = 0.5*mc.SS(0, 1) + 0.35*mc.SS(1, 2) + 0.15*mc.Ntot[0] - 0.2*mc.Ntot[2]
hop = 0.4*(mc.Cdagup[0]*mc.Cup[2] + mc.Cdagdn[0]*mc.Cdn[2])
h = h + hop + hop.get_dagger() + 0.13*sum(mc.Sz)
mc.set_hamiltonian(h)
print(mc.gs_energy(mode="ED"), mc.gs_energy(mode="DMRG"))
```

Both numbers are -0.992144. The conserved sectors are the total $2S^z$, of spins and electrons together, and the electron number. On a chain with both spin and fermion sites they work only with `mode="ED"`: set `mc.mode = "ED"` before `set_conserved_sector`. For the same reason `submode="SECTOR"` is not available there; use `"KPM"`, `"TD"` or `"CVM"`. The example `examples/fermion_models/mixed_spin_fermion_chain_VS_ED` compares the magnetization profile of DMRG and ED on a Kondo chain.

Backends.

Some classes do not run on every backend:

- `Mixed_Spin_Fermion_Chain`: `itensor_version=3`, `"python"` and `"julia_live"`, not 2.
- `SpinBoson_Chain`: no `mode="ED"`. Use `Bosonic_Chain` for a purely bosonic model.
- `Mixed_Spin_Fermion_Chain` with both spin and fermion sites: a conserved sector only with `mode="ED"`, and no `submode="SECTOR"`.
- `Bosonic_Chain` and `SpinBoson_Chain` with a boson site that does not have 4 levels: not on `itensor_version=2`, it raises an error.

Every chain has the methods of this guide. A few methods exist only on the class whose physics they need:

- `Spin_Chain`: `SS(i, j)`, `get_magnetization()`, `get_kondo_spectrum()` (STM and Kondo spectra), `get_full_SS_correlator()`.

- The fermionic chains: `get_density()`, `get_density_fluctuation()`, `get_charge_gap()`, the correlation matrix and its entropies (Entanglement), and `get_fermionic_parity()` on their states (Topological invariants).
- The spinful fermionic chains: `get_magnetization()`, `get_onsite_pairing()`, `get_spectral_function()` with `spin=`, `get_kondo_spectrum()` on the electron spin of a site (STM and Kondo spectra).
- `Bosonic_Chain`: `get_density()`, `get_density_fluctuation()`.

Every constructor takes `itensor_version=` and the chain settings as keywords. A keyword has the same effect as an assignment after construction, so

```
from dmrgpy import spinchain
sc = spinchain.Spin_Chain(["S=1/2"]*20, maxm=60, nsweeps=20)
```

is the same as setting `sc.maxm = 60` and `sc.nsweeps = 20` afterwards. A keyword that is not a setting (for example `hubbard=` or `hamiltonian=`) raises an error.

3 Hamiltonians and operators

A Hamiltonian or an observable is written as its operator expression, with the operator lists of Models. For the spin-1/2 Heisenberg chain,

$$H = J\sum_{i=1}^{n-1} \mathbf{S}_i \cdot \mathbf{S}_{i+1} = J\sum_{i=1}^{n-1} \left(S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + S_i^z S_{i+1}^z\right),$$

```
from dmrgpy import spinchain
n = 30
sc = spinchain.Spin_Chain(["S=1/2" for i in range(n)])
h = 0
for i in range(n-1):
    h = h + sc.Sx[i]*sc.Sx[i+1] + sc.Sy[i]*sc.Sy[i+1] + sc.Sz[i]*sc.Sz[i+1]
sc.set_hamiltonian(h)
```

Any sum of products of local operators is allowed, with any range and complex coefficients. This covers single-ion anisotropy $D(S_i^z)^2$, Dzyaloshinskii-Moriya terms $\mathbf{D}\cdot(\mathbf{S}_i \times \mathbf{S}_{i+1})$, biquadratic exchange $(\mathbf{S}_i \cdot \mathbf{S}_{i+1})^2$, hopping $t\sum_i (c_i^\dagger c_{i+1} + \text{h.c.})$, the Hubbard term $U\sum_i n_{i\uparrow} n_{i\downarrow}$, and so on. The Cookbook gives the complete Hamiltonian of the common models. `h.get_dagger()` returns $h^\dagger$, so `h + h.get_dagger()` adds the Hermitian conjugate. It is best to write H in units where its largest coefficient is of order 1: a few routes, such as real-time evolution, use absolute thresholds.

3.1 Shortcuts

On a spin chain, `sc.SS(i, j)` is the Heisenberg bond $\mathbf{S}_i \cdot \mathbf{S}_j$, so the chain above is `h = sum(sc.SS(i, i+1) for i in range(n-1))`. On every chain, `sc.generate_bilinear(f, A, B)` returns the Hermitian part of $\sum_{ij} f(i,j)\, A_i B_j$ for two operator lists, for example `sc.generate_bilinear(f, sc.Sz, sc.Sz)` for an Ising coupling with any range. A fermionic Hamiltonian is written the same way, as a sum of operators. The Hubbard-like chain with hopping $t = 1$ and nearest-neighbor repulsion $V = 2$ is:

```
from dmrgpy import fermionchain
fc = fermionchain.Fermionic_Chain(4)
h = 0
for i in range(3):
    h = h + fc.Cdag[i]*fc.C[i+1] + fc.Cdag[i+1]*fc.C[i]  # hopping
    h = h + 2.0*fc.N[i]*fc.N[i+1]                        # repulsion
fc.set_hamiltonian(h)
```

3.2 Operators

On an operator `A`, `A.simplify()` collects equal terms, and `A.is_hermitian()`, `A.is_antihermitian()` and `A.is_zero()` answer by reading the terms. A `True` answer is exact. A `False` answer can be wrong when the property depends on an identity of the local space, such as $S^x S^x = 1/4$ for spin 1/2. `sc.is_hermitian(A)` does not have this limitation, since it also checks numerically on the chain.

`sc.toMPO(A)` builds the matrix product operator of `A` once. Two such operators can be added, subtracted, multiplied by numbers and multiplied with each other (`A + 2*B - A`, `A*B`), and `A.trace()` is their trace, so `(Sz[0]*Hmpo*Hmpo).trace()` is a moment of the spectrum. To combine Hamiltonians, it is simpler to sum the operator expressions before calling `set_hamiltonian`. On a small chain, `sc.get_full_matrix(A)` returns the dense matrix of any operator, to check a model against another code.

3.3 States and operators

The chain also gives the basic operations on states. Here `wf` is a state, for example `wf = sc.get_gs()`, and `A` an operator:

Call	Returns
`sc.overlap(a, b)`	$\langle a\|b\rangle$
`sc.aMb(a, A, b)`	$\langle a\|A\|b\rangle$
`sc.applyoperator(A, wf)`, or `A*wf`	$A\|\psi\rangle$
`sc.applyinverse(A, wf)`	$A^{-1}\|\psi\rangle$, solved iteratively
`sc.exponential(A, wf)`	$e^A\|\psi\rangle$
`sc.summps(a, b)`	$\|a\rangle + \|b\rangle$
`sc.scale_mps(x, wf)`, or `wf*x`	$x\|\psi\rangle$
`sc.trace(A)`, `sc.inverse_trace(A)`	$\mathrm{Tr}\, A$, $\mathrm{Tr}\, A^{-1}$
`sc.operator_norm(A)`	an estimate of $\|A\|$
`sc.is_zero_operator(A)`	whether $A = 0$
`wf.dot(wf2)`, `wf.norm()`	$\langle\psi\|\psi_2\rangle$, $\|\psi\|$
`wf.normalize()`	$\|\psi\rangle/\|\psi\|$
`wf.get_conjugate()`	the complex conjugate state
`sc.hadamard_product(a, b, conj=(True, False))`	the state with amplitudes $a(s)^* b(s)$
`sc.elementwise_map(wf, f)`	the state with amplitudes $f(\psi(s))$

A state remembers the backend that made it (an MPS for DMRG, a vector for `mode="ED"`), and these operations run on that backend. Give `mode="ED"` to `overlap` and `aMb` when the states come from ED. On DMRG, `exponential` needs a Hermitian or an anti-Hermitian `A`, so `sc.exponential(-1j*t*h, wf)` is a real-time step. For a general `A` use `mode="ED"` on a small chain. On DMRG, `applyinverse` solves $Ax = \psi$ by two-site sweeps on `itensor_version="python"` when A is not Hermitian, such as the resolvent $z - H$ off the real axis, as DMRG solves the ground state. A Hermitian A, the other backends and a conserved sector take BiCGSTAB. `sc.inverse_solver="bicgstab"` or `"sweep"` forces one (`"sweep"` on `"python"` and `"julia_live"`). The sweeps warn if the residual stays above the tolerance, as it can for an A with eigenvalues of both signs. `sc.inverse_nsweeps` is the largest number of sweeps (default 10).

The elementwise products are taken in the computational basis s of the sites. On `itensor_version=3`, `"python"` and `"julia_live"` they use recursive sketched interpolation (arXiv:2602.17974), with `maxm=` the bond dimension of the result; `method="direct"` is the exact product compressed by SVD, more accurate and more expensive. `elementwise_map` is the extension of the same method to a nonlinear `f`, exact once the bond dimension holds the result. For fermions a product of two amplitudes depends on the sign convention of the basis states, while $|\psi(s)|^2$ does not. The Participation entropy is built on them.

4 Ground state

On this page we compute the ground state of a chain and measure it. All the examples use a spin-1/2 Heisenberg chain:

```
from dmrgpy import spinchain
n = 20
sc = spinchain.Spin_Chain(["S=1/2" for i in range(n)])
h = sum(sc.Sx[i]*sc.Sx[i+1] + sc.Sy[i]*sc.Sy[i+1] + sc.Sz[i]*sc.Sz[i+1]
        for i in range(n-1))
sc.set_hamiltonian(h)
```

4.1 Ground-state energy

The ground state is the state with the lowest energy,

$$E_0 = \min_{|\psi\rangle} \frac{\langle\psi|H|\psi\rangle}{\langle\psi|\psi\rangle}.$$

```
e0 = sc.gs_energy()    # E0
wf = sc.get_gs()       # the ground state |GS> as an MPS
```

DMRG finds this minimum within matrix product states of bond dimension `sc.maxm`. The result is exact when `maxm` is large enough, and an upper bound on E_0 otherwise. The keywords that control accuracy are:

- `sc.maxm`: the bond dimension. Increase it until E_0 stops changing. Gapped chains need a small `maxm`; critical chains and wide ladders need a large one.
- `sc.nsweeps`: the number of DMRG sweeps.
- `sc.dmrg_method = "CBE"`: one-site updates with bond expansion, cheaper for a large local dimension (One-site updates with bond expansion).
- `gs_energy(maxde=1e-4)`: doubles `maxm` until the energy fluctuation per site (Energy fluctuation) is smaller than `maxde`.
- `mode="ED"`: exact diagonalization, for small chains and for checking DMRG.

The chain stores the ground state. A second call to `gs_energy()` returns it without solving again. The sweep schedule and warm starts are on Controlling the solver.

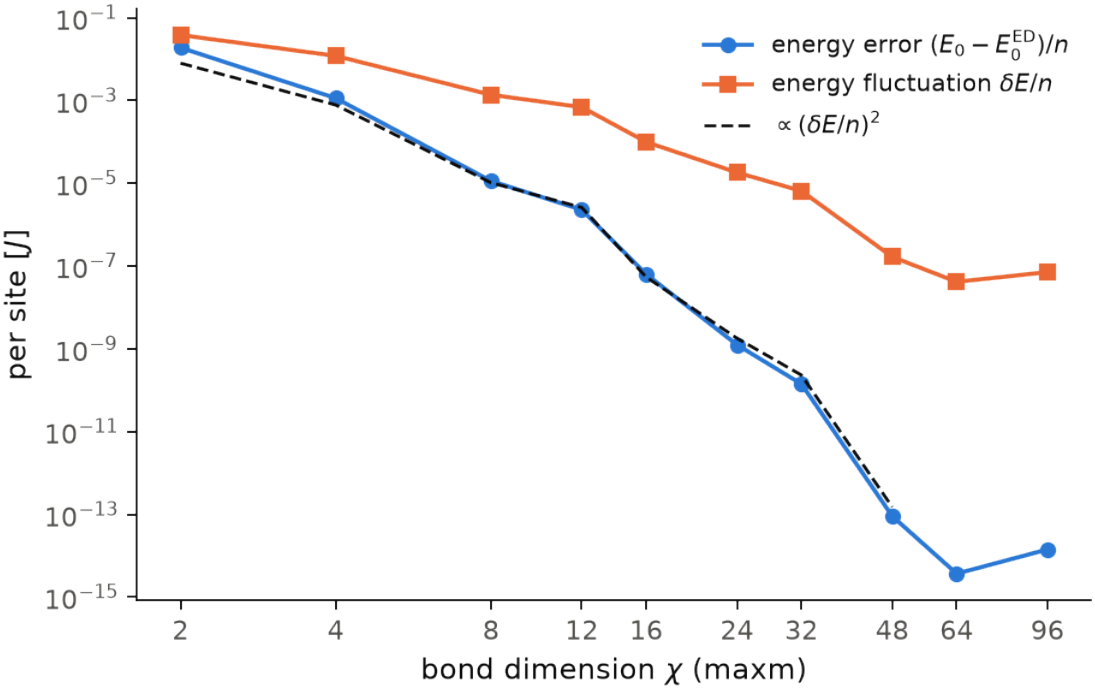


Figure 2: Energy convergence with the bond dimension

The figure shows the energy error and the energy fluctuation per site of a Heisenberg chain with $n = 16$, as a function of `maxm`. The reference is ED. Both fall fast with `maxm`. The energy error follows δE^2 (dashed line). From `maxm=64` on, both stay at the numerical precision.

The code of this figure is in `docs/figures/ground-state_convergence/`.

4.2 Expectation values

For any operator O built from the chain operators, as H is,

$$\langle O \rangle = \langle \mathrm{GS}|O|\mathrm{GS}\rangle, \qquad \langle O^n \rangle = \langle \mathrm{GS}|O^n|\mathrm{GS}\rangle.$$

```
mz = [sc.vev(sc.Sz[i]).real for i in range(n)]    # magnetization profile
e2 = sc.vev(h, npow=2)                            # <H^2>
```

Common profiles have their own methods, each returning one value per site:

- `get_density()`: the occupation $\langle n_i \rangle$, summed over spin. Fermionic chains.
- `get_density_fluctuation()`: $\langle n_i^2 \rangle - \langle n_i \rangle^2$. Fermionic chains.
- `get_onsite_pairing()`: $\langle c_{i\uparrow} c_{i\downarrow} \rangle$. Spinful fermionic chains.
- `get_magnetization()`: $\langle S_i^x \rangle$, $\langle S_i^y \rangle$, $\langle S_i^z \rangle$. A $3 \times n$ array on spin chains, and an $n \times 3$ array on spinful fermionic chains.

4.3 Static correlators

A two-point correlator is the expectation value of a product of two operators:

```
czz = [sc.vev(sc.Sz[0]*sc.Sz[i]).real for i in range(n)]        # <Sz_0 Sz_i>
cdc = fc.vev(fc.Cdag[0]*fc.C[5])    # <c^dag_0 c_5> on a fermionic chain fc
```

The equal-time structure factor is its Fourier transform, $S(q) = \sum_{i,j} e^{iq(i-j)} \langle O_i O_j \rangle$. This is the quantity measured by neutron or X-ray diffraction, and it is the frequency integral of the dynamical correlators.

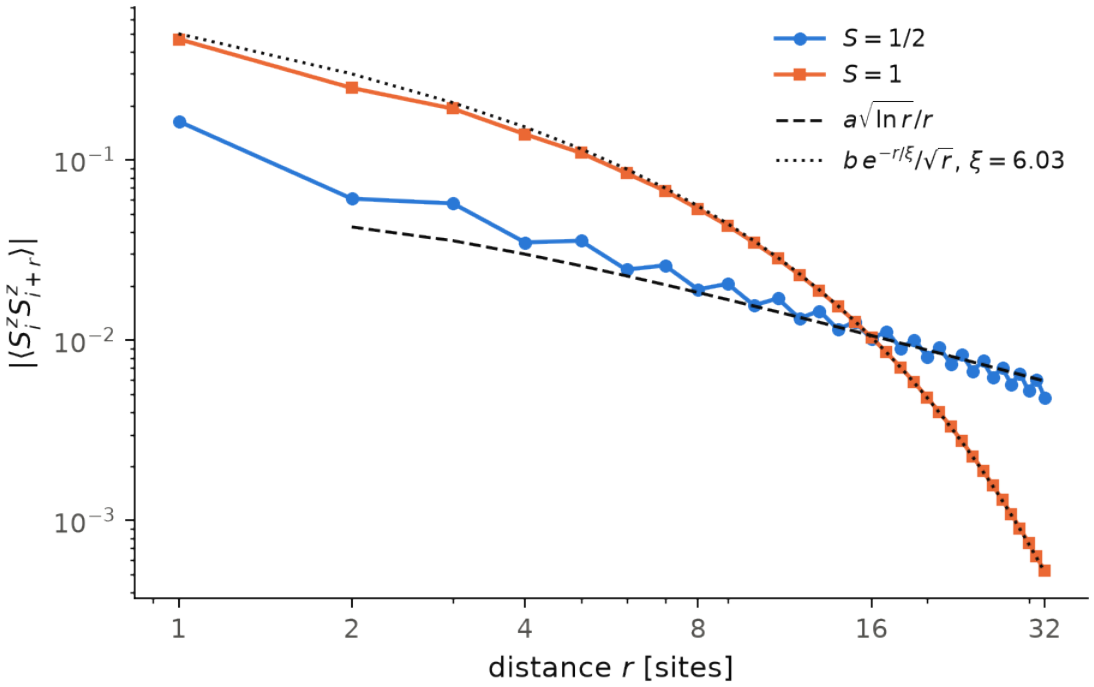


Figure 3: Spin correlations in spin-1/2 and spin-1 chains

The figure shows $|\langle S^z_i S^z_{i+r}\rangle|$ in two open Heisenberg chains with $n = 64$, measured from site $i = 16$. The spin-1/2 chain is gapless, and its correlations decay as a power law, $\sqrt{\ln r}/r$ (dashed line). The small zigzag between even and odd r is an effect of the open ends. The spin-1 chain has the Haldane gap, and its correlations decay exponentially, as $e^{-r/\xi}/\sqrt{r}$ with $\xi = 6.03$ (dotted line). A free fit of the DMRG data gives $\xi = 6.09$. In the spin-1 chain, the two end sites are spin 1/2. They bind the edge states of the open chain, so that the ground state is unique.

The code of this figure is in `docs/figures/ground-state_correlators/`.

4.4 Energy fluctuation

The energy variance tells us how close the state is to an eigenstate of H:

$$\delta E = \sqrt{\langle H^2\rangle - \langle H\rangle^2}.$$

```
de = sc.gs_energy_fluctuation()
```

For an exact eigenstate $\delta E = 0$. In DMRG, δE comes from the truncation of the bond dimension, so it is a direct check of `maxm`. It is a stricter check than the energy itself: roughly, the energy error is of order δE^2, while the error of other observables is of order δE. The figure in Ground-state energy shows this on a Heisenberg chain.

4.5 Fixed particle number or fixed magnetization

Without any constraint, DMRG returns the global ground state, at whatever particle number or total S^z it has. When H conserves a charge Q, we can instead ask for the ground state of one sector,

$$E_0(q) = \min_{\hat{Q}|\psi\rangle = q|\psi\rangle} \frac{\langle\psi|H|\psi\rangle}{\langle\psi|\psi\rangle}.$$

Here `fc` is a `Fermionic_Chain` and `hc` a `Spinful_Fermionic_Chain_Native`.

```
fc.set_conserved_sector(Nf=6)          # exactly 6 fermions
sc.set_conserved_sector(Sz=0)          # total Sz = 0
hc.set_conserved_sector(Nf=8, Sz=0)    # Hubbard chain: both
fc.set_conserved_sector()              # back to the full Hilbert space
```

`Sz` is given in units of 1/2, so `Sz=2` means $S^z_{\rm tot} = 1$. The available charges are `Nf` for fermions, `Sz` for spins, both for the native Hubbard chain, and `Nb` for bosons. Parafermion chains have none.

With sectors we compute the addition spectrum $E_0(N)$, the charge gap $E_0(N+1)+E_0(N-1)-2E_0(N)$, and the magnetization curve, each point being a true ground state of its own sector. The same calls work with `mode="ED"`. On large chains a sector is also faster, by about 4 times at $n = 60$.

The figure shows the magnetization curve of two open chains with $n = 32$. In a field h, the ground state is in the sector $M = S^z_{\rm tot}$ that minimizes $E_0(M) - hM$. So the magnetization jumps from M to $M+1$ at $h_M = E_0(M+1) - E_0(M)$, and one solve per sector gives the whole curve. For the XX chain ($S^xS^x + S^yS^y$) the steps follow the free-fermion curve $\arcsin(h/J)/\pi$ (dashed line), and every $E_0(M)$ agrees with free fermions on the same chain within 3×10^{-10}. The Heisenberg chain saturates close to $h_s = 2J$; on this open chain the last step is at $J(1 + \cos(\pi/n)) = 1.995\,J$.

The code of this figure is in `docs/figures/ground-state_sectors/`.

Inside a sector, every operator must conserve the charge. An operator that changes it, for example c_i at fixed N_f or S^x_i at fixed S^z, raises an error. To apply such an operator to the sector ground state, first convert the state to an ordinary MPS:

```
fc.set_conserved_sector(Nf=6)
fc.gs_energy()
fc.promote_to_dense()                  # same state, no sector
wf = fc.applyoperator(fc.C[3], fc.wf0)   # c_3 |GS_N>, now with 5 fermions
```

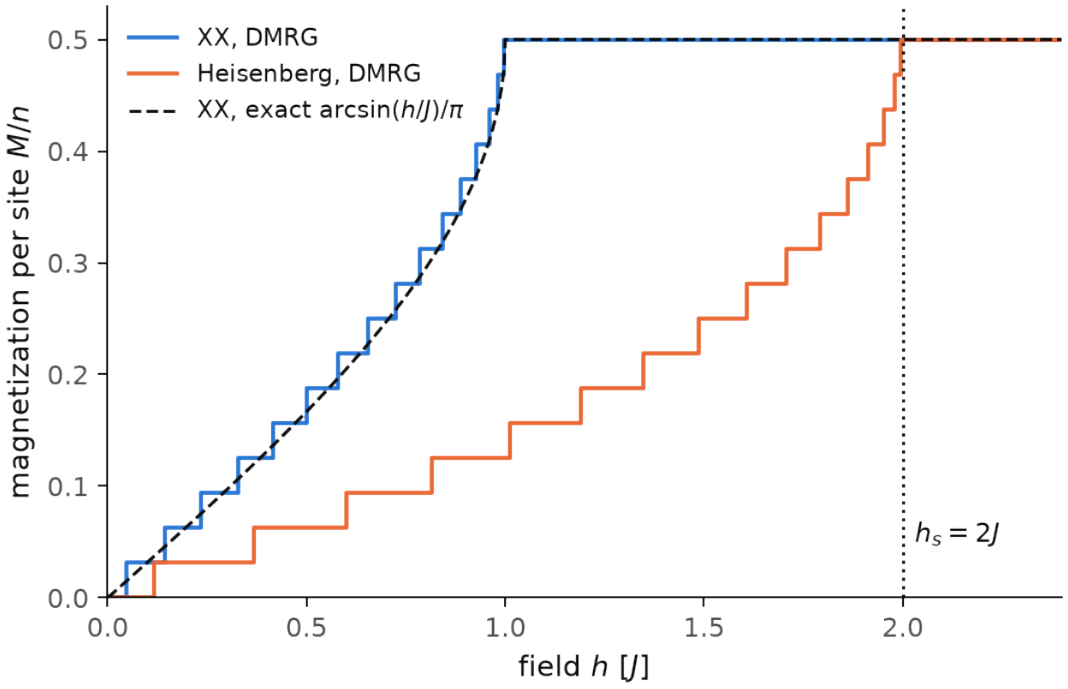


Figure 4: Magnetization curves from sector energies

Two ground states with N and $N-1$ fermions, both converted this way, can be compared directly. This gives for example the photoemission weight $Z_i = |\langle \mathrm{GS}_{N-1}|c_i|\mathrm{GS}_N\rangle|^2$ (see `examples/groundstate/sector_promotion_to_dense`).

Backends.

Sectors work on `itensor_version=3`, `"python"`, `"julia_live"` and `mode="ED"`. They are not available on `itensor_version=2`, with METTS and for infinite chains; these raise an error.

5 Controlling the solver

On this page we control the DMRG solve itself: the sweep schedule, the state it starts from, and the state the chain keeps. The physics of the ground state is on the Ground state page; here `sc` is its Heisenberg chain.

5.1 Sweep schedule

Each sweep moves through the chain and updates two sites at a time, keeping at most `sc.maxm` Schmidt states on every bond. The settings are:

- `sc.maxm` (default 30): the bond dimension. The one that sets the accuracy; increase it until the result stops changing.
- `sc.nsweeps` (default 15): the number of sweeps. A gapped chain converges in a few sweeps, a critical or frustrated one needs more.
- `sc.cutoff` (default 10^{-12}): Schmidt weights below this are discarded, so a weakly entangled bond uses fewer than `maxm` states.
- `sc.noise` (default 10^{-7}): a small density-matrix perturbation during the first half of the sweeps, which helps the state leave a local minimum, for example a product state that is an exact eigenstate. It is zero in the final sweeps, so it does not bias the result. Raise it to 10^{-5} when a solve is stuck, set it to 0 for a model that converges without it.
- `sc.bond_ramp` (default `True`): the first half of the sweeps grow the bond dimension geometrically from `sc.bond_ramp_start` (default 10) to `maxm`, and the second half run at `maxm`. The early sweeps mostly locate the right state and gain little from a large bond dimension, so this makes the solve cheaper without

changing the result. Set it to `False` to run every sweep at `maxm`; `sc.bond_ramp_fraction` (default 0.5) is the fraction of the sweeps spent growing.

- `sc.verbose = True`: prints the progress of each sweep, to see whether `nsweeps` is enough.

These settings are also constructor keywords, `Spin_Chain(sites, maxm=60, nsweeps=20)`. The chain remembers the solved state: `gs_energy()` and `get_gs()` return it without a new solve as long as the Hamiltonian and the settings above do not change, and `sc.restart()` forgets it.

5.2 One-site updates with bond expansion

The default sweep updates two sites at a time. This lets a bond grow, but the local problem has dimension d^2D^2, so it becomes expensive for a large local dimension d, for example $d = 4$ in a Hubbard chain. A one-site update costs only dD^2, but alone it cannot grow a bond: from a product state it stays a product state. Controlled bond expansion (Gleis, Li and von Delft, PRL 130, 246402 (2023)) solves this. Before each one-site update, the bond gets the few new states on which $H|\psi\rangle$ has weight outside the current two-site space, the part that a two-site update would add. After the update the bond is cut back to `sc.maxm`. So it converges like two-site DMRG with one-site local problems.

```
sc.dmrg_method = "CBE"
e0 = sc.gs_energy()
```

- `sc.dmrg_method` (default `"TWO_SITE"`): `"CBE"` for one-site updates with controlled bond expansion. Use it for chains with a large local dimension; for spin-1/2 chains the two-site update is usually faster.
- `sc.cbe_delta` (default 0.1): each bond is expanded to $(1 + \delta)$`maxm` states before the update, as in the paper.

On `itensor_version="python"` it also works with a conserved sector (Fixed particle number or fixed magnetization).

Backends.

`"CBE"` exists on `itensor_version="python"`, on `itensor_version=3` and on `"julia_live"`; on the last two it does not work in a conserved sector. The other backends raise an error, and so do the excited-state search and the non-Hermitian ground state.

See `examples/groundstate/cbe_ground_state`.

5.3 Starting from a given state

By default a solve starts from a state the backend chooses, a random one on `itensor_version=3`, `"python"` and `"julia_live"`. `wf0=` starts the sweeps from a state we give, for example the ground state of the previous point of a parameter scan:

```
import numpy as np
bs = np.linspace(0.0, 0.5, 11)
hst = sum((-1)**i*sc.Sz[i] for i in range(n))      # staggered field
wf = None
es = []
for b in bs:
    sc.set_hamiltonian(h + b*hst)
    es.append(sc.gs_energy(wf0=wf, reconverge=True))
    wf = sc.get_gs()                                # start of the next point
```

`sc.set_initial_wf_guess(wf)` does the same for the next solve. A warm start follows one branch of states along the scan, which is what we want when a symmetry-broken state has to be tracked, or when the solve from a random start lands in a wrong state. It does not make the solve faster in the cases we tried: with the bond ramp, a solve from a random start converges in as few sweeps as one from the previous point. And it can be wrong across a level crossing of a conserved quantity: in a scan over a uniform field $B\sum_i S^z_i$ the ground state jumps from one S^z_{tot} sector to the next, and a state in the old sector leaves it only through the noise term, slowly.

With a few sweeps the warm start stays in the old sector with an energy that is too high (on 16 sites, by 0.1 to 0.2 with 2 or 3 sweeps, and still by 0.03 after 15 sweeps on `"python"`). Start from a random state there.

A state can also be taken as it is, without sweeping:

```
sc.set_hamiltonian(sum((-1)**i*sc.Sz[i] for i in range(n)))
neel = sc.get_gs()                  # Neel product state, as the ground state of a staggered field
sc.set_hamiltonian(h)
sc.set_initial_wf(neel)             # or sc.set_gs(neel)
e_neel = sc.gs_energy()             # <Neel|H|Neel> = -(n-1)/4, no sweep
mz = [sc.vev(sc.Sz[i]).real for i in range(n)]    # measured in the Neel state
```

After `set_initial_wf` or `set_gs`, every method of the chain measures the state we set, with `gs_energy()` returning its energy $\langle\psi|H|\psi\rangle$. This gives the variational energy of a trial state, and the correlators or the time evolution of a prepared state. The state is normalized when it is set. `sc.set_initial_wf(None)`, or a new `set_hamiltonian`, returns to the solved ground state.

Two more states are useful here. `sc.random_mps()` is a random normalized MPS, a start that tests whether a solve depends on where it begins. `sc.get_gs(best=True, n=3)` solves `n` times from independent random starts and keeps the lowest energy, for a frustrated or disordered chain where a single solve can end in an excited state.

5.4 Several chains

`sc.copy()` returns an independent chain with the same Hamiltonian and settings, to compare two values of `maxm` on the same model, or to keep a converged chain while another one is modified. Two chains never share a state.

Two methods solve an auxiliary problem on a copy of the chain and leave `sc` untouched:

```
e_min = sc.lowest_eigenvalue(sum(sc.Sz))    # smallest eigenvalue of any Hermitian operator, here -n/2
W = sc.bandwidth(h)                         # E_max - E_min of H
```

The bandwidth is the energy scale the kernel polynomial method rescales by (Choosing a correlator method).

> **Backends.**
>
> `cutoff`, `noise`, the warm starts and the bond ramp work on every backend. `verbose` prints on `itensor_version=2`, `3` and `"python"`.

6 Excited states and gaps

On this page we compute the low-energy spectrum: excited states, gaps, the degeneracy of the ground state, non-Hermitian and generalized eigenproblems, and effective Hamiltonians of a low-energy manifold. Here `sc` is the Heisenberg chain of Ground state.

6.1 Low-lying spectrum

The lowest eigenvalues $E_0 \le E_1 \le \ldots$ of H and the gap $\Delta = E_1 - E_0$ are

```
es = sc.get_excited(n=4)                # [E0, E1, E2, E3]
gap = sc.get_gap()                      # E1 - E0
es, ws = sc.get_excited_states(n=4)     # energies and states
```

The list has one entry per eigenstate, so a degenerate level appears once for each state of the multiplet. If the ground state is degenerate, `get_gap()` is zero, and we must ask for enough states to reach the first excited level. When DMRG splits a multiplet slightly, identify the states by a quantum number (for example $\langle \mathbf{S}^2 \rangle$) and not by their position in the list.

DMRG finds each excited state as the ground state of H plus a penalty on the states already found. This search can converge to a wrong state. When the energy variance of a state is much larger than that of the ground state, a warning is printed, and the state should be checked with more sweeps, a larger `maxm` or `mode="ED"` on a smaller chain. The keywords are:

- `n`: the number of states.
- `scale`: the weight of the penalty, in units of the bandwidth of H (default 10).
- `purify=True` (default, `get_excited_states`): diagonalizes H again in the space of the states found, which separates nearly degenerate states.

For example, an open spin-1 Heisenberg chain has four low states from the two spin-1/2 edge modes (a singlet and a triplet), and the Haldane gap is the gap to the next state:

```
es = s1.get_excited(n=6)     # s1: an open spin-1 Heisenberg chain
haldane_gap = es[4] - es[0]
```

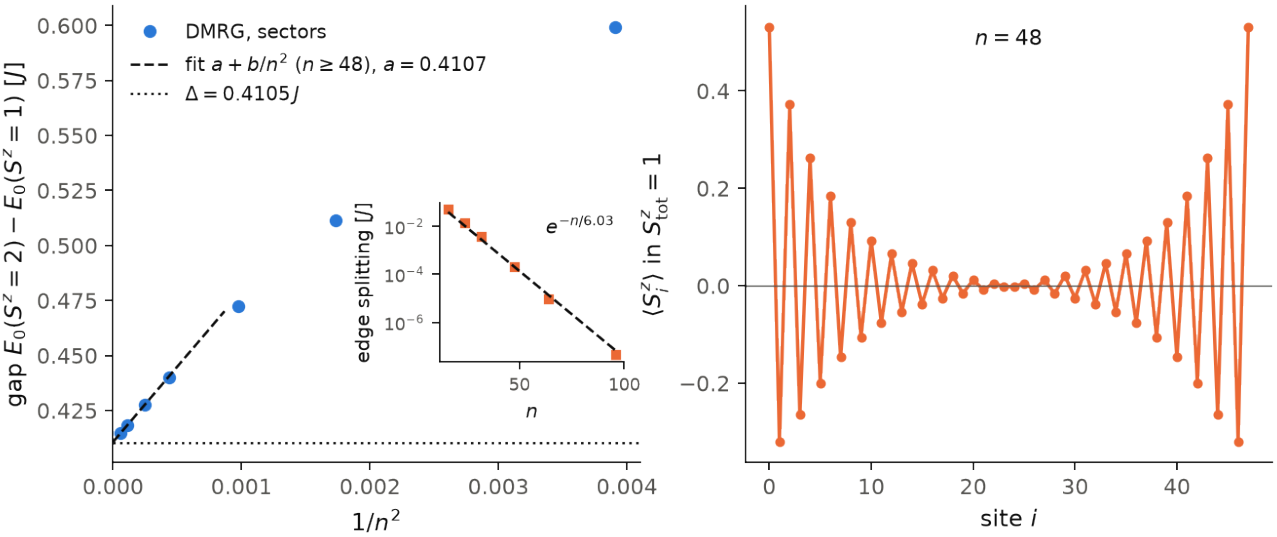


Figure 5: Haldane gap and edge states of the spin-1 chain

The figure shows the same physics on long open spin-1 chains. Here the energies come from the ground states of three sectors, $S^z_{\rm tot} = 0$, 1 and 2, which is faster and safer than `get_excited` at these lengths. The edge splitting is $E_0(1) - E_0(0)$ and the bulk gap is $E_0(2) - E_0(1)$, one magnon on top of the edge triplet. On the left, the gap goes as $1/n^2$, and the fit $a + b/n^2$ on $n = 48$ to 128 gives $0.4107\,J$, against the known $\Delta = 0.4105\,J$. The inset shows that the edge splitting falls as $e^{-n/\xi}$ with $\xi \approx 6$, to below 10^{-7} at $n = 96$, so the four edge states become degenerate. On the right is $\langle S_i^z \rangle$ in the $S^z_{\rm tot} = 1$ ground state at $n = 48$. Each end carries a spin 1/2 (each half sums to exactly 1/2), spread over about 6 sites, and the bulk is not magnetized.

The code of this figure is in `docs/figures/excited-states_haldane/`.

6.2 Ground-state degeneracy

A degenerate ground state signals a broken symmetry or a topological phase. Examples are the two ordered states of an Ising chain and the four edge states of an open Haldane chain, whose splitting is shown in Low-lying spectrum. We count the levels close to E_0,

$$g \approx \sum_i \exp\left[-\left(\frac{(E_i - E_0)^2}{\delta}\right)^2\right],$$

adding excited states E_i until the count stops growing:

```
g = sc.get_gs_degeneracy()              # close to an integer
g = sc.get_gs_degeneracy(delta=1e-4)    # a stricter window
```

A level counts when $|E_i - E_0|$ is smaller than about $\sqrt{\delta}$, which is 0.1 at the default `delta=1e-2`. A result that is not close to an integer means that a level sits at the edge of the window. On a finite chain a degeneracy is often

split by an amount that decays with the length, so choose `delta` relative to that splitting. `mode="ED"` uses exact diagonalization.

The states of the manifold are `sc.get_gs_manifold(n=2, tol=1e-3)`, the eigenstates within `tol` of E_0 (more states are searched when all `n` turn out degenerate), and `sc.vev(A, wf=w)` measures an operator in any of them. Here `sc` is an Ising chain $H = -\sum_i S^z_i S^z_{i+1} - 0.2\sum_i S^x_i$ in its ordered phase:

```
wfs = sc.get_gs_manifold(n=2, tol=1e-3)          # two states
M = sum(sc.Sz)
print([sc.vev(M, wf=w).real for w in wfs])       # 0 and 0
print(sc.aMb(wfs[0], M, wfs[1]).real)            # modulus 4.82 on 10 sites
up = sc.summps(wfs[0], wfs[1]).normalize()       # the ordered states
dn = sc.summps(wfs[0], -1*wfs[1]).normalize()
print(sc.vev(M, wf=up).real, sc.vev(M, wf=dn).real)   # 4.82 and -4.82, in some order
```

The eigenstates are the symmetric and antisymmetric combinations of the two ordered states, so the magnetization vanishes in each of them and appears in the off-diagonal element. The ordered states are their sum and difference. The dynamical correlator with `mode="ED"` averages over such a manifold by itself.

6.3 Gap to another particle number

For a conserved charge A, for example the particle number $N = \sum_i n_i$, `get_charge_gap(d=...)` gives the energy to change $\langle A\rangle$ by d from its ground-state value A_0. It solves

$$H_\lambda = H + \lambda(A - A_0 - d)^2,$$

increasing λ until the constraint holds, and returns $\Delta(d) = E_0(A_0 + d) - E_0(A_0)$.

```
gap2 = fc.get_charge_gap(d=2)     # fc: a fermionic chain; add two particles
```

`d=2` is the cost of adding a pair, and `d=-2` of removing one. `Fermionic_Chain`, `Spinful_Fermionic_Chain` and the native spinful chain have this method. The conserved sectors give the same energies directly, $E_0(N)$ for each N.

6.4 Non-Hermitian Hamiltonians

When $H \neq H^\dagger$ (gain and loss, complex or non-reciprocal hopping), the ground state is the eigenvalue with the smallest real part. Eigenstates come in left and right pairs,

$$H|\psi_R\rangle = E|\psi_R\rangle, \qquad H^\dagger|\psi_L\rangle = E^*|\psi_L\rangle, \qquad \langle\psi_L|\psi_R\rangle = 1.$$

`gs_energy()` detects a non-Hermitian H. With `mode="DMRG"` it evolves a random state in imaginary time with H itself, $|\psi(\tau)\rangle \propto e^{-\tau H}|\psi(0)\rangle$, normalized at every step. The levels of smallest real part grow fastest, so the state ends in their span. Several levels often share that real part, for example a complex-conjugate pair in a PT-symmetric chain. Then the state keeps rotating between them, so the energies are read from a small diagonalization of H in the span of the last states of the evolution. The left state comes from the same evolution with $H^\dagger$. Here `fc` is a fermionic chain with a non-Hermitian Hamiltonian, for example a staggered imaginary potential $i\gamma(-1)^i n_i$:

```
e = fc.gs_energy()                # E; stores the right state as the ground state
es, res = fc.get_nh_levels()      # the levels of smallest real part, with residuals
e, psil, psir = fc.nhdmrg()       # non-Hermitian DMRG: E, left and right states
```

`get_nh_levels()` returns every level it finds, sorted by real part, with its relative residual. When the spectrum is degenerate in the real part, the first entries are all the levels that share it, which a single eigenpair cannot show. `vectors=True` also returns their right states. A level in a symmetry sector the random start has no weight in does not appear. `nhdmrg()` is non-Hermitian DMRG (NH-DMRG), which returns one left and right pair.

The energy is not a variational bound, so the result is checked with the residuals $\|H|\psi_R\rangle - E|\psi_R\rangle\|$ and $\|H^\dagger|\psi_L\rangle - E^*|\psi_L\rangle\|$. The keywords are:

- `tol`: the largest relative residual accepted (default `1e-4`).
- `ntries`: how many times NH-DMRG restarts from a random state when the residual is too large (default 5).
- `fc.nh_gs_method` (default `"imaginary_time"`): `"nhdmrg"` makes `gs_energy()` use NH-DMRG. NH-DMRG is faster where it converges, and the imaginary-time route also runs it when its own result fails the residual check, keeping the better of the two.
- `fc.maxm`: the state has to hold all the levels that share the real part at once, so a multiplet of many tied levels needs a larger bond dimension than one level does. Raise it when the check fails.

Backends.

The imaginary-time route and `get_nh_levels()` run on `itensor_version=3` and `"python"`, without a conserved sector. On the other backends `gs_energy()` uses NH-DMRG, which works on `itensor_version=2`, 3, `"python"` and `"julia_live"`.

See `examples/non_hermitian/nh_levels_imaginary_time`, where the four levels of smallest real part of a PT-symmetric XX chain come out together.

Excited states of a non-Hermitian H come from `get_excited_states(n)` and `get_excited(n)`, which use an Arnoldi method on matrix product states. It is less accurate than NH-DMRG, so check it against `mode="ED"` on a small chain (`examples/non_hermitian/nhdmrg_VS_ED_VS_arnoldi`).

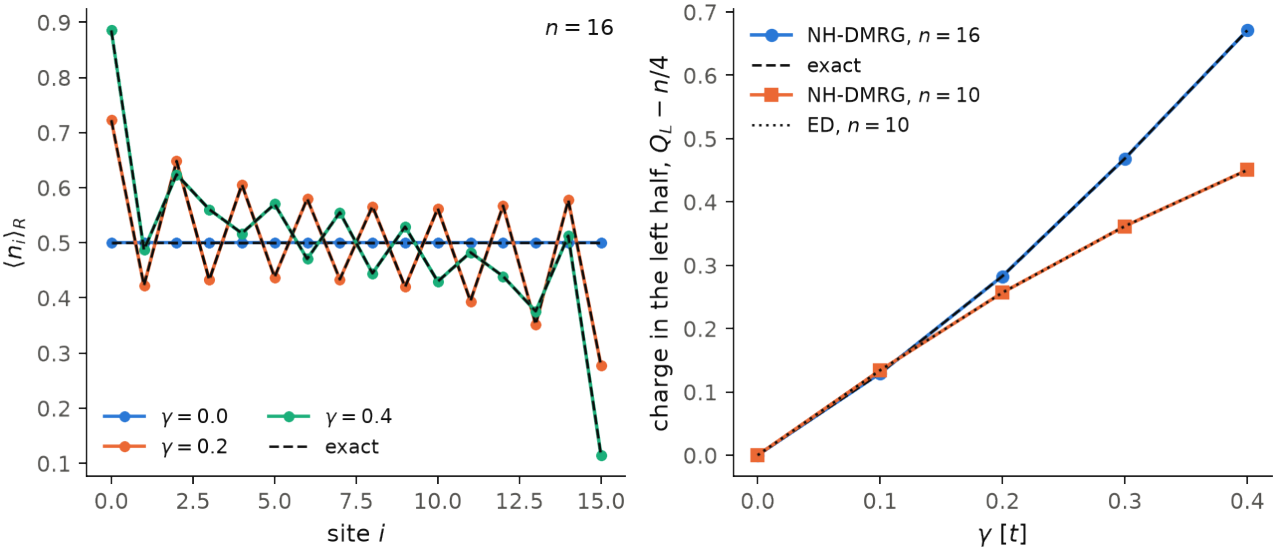


Figure 6: Skin effect of the interacting Hatano-Nelson chain

The figure shows the many-body skin effect of an interacting Hatano-Nelson chain at half filling, with hopping $t \pm \gamma$ and a nearest-neighbor repulsion $V = t$. On the left is the density of the right ground state, $\langle\psi_R|n_i|\psi_R\rangle / \langle\psi_R|\psi_R\rangle$, from `nhdmrg()` on 16 sites. It is flat at $\gamma = 0$, and charge moves to the left end as γ grows. On the right is the charge moved into the left half. The dashed curves are exact, since with open ends H is a similarity transform of a Hermitian chain, and ED on 10 sites agrees with NH-DMRG to 10^{-14}. The problem becomes ill conditioned fast with n and γ; `nhdmrg` warns when it does not converge, and again each time the unconverged state is used.

The code of this figure is in `docs/figures/excited-states_nonhermitian/`.

6.5 Spectrum from time evolution

We will now see a second way to get several low eigenvalues at once, for a Hermitian or a non-Hermitian H. We evolve two random states in imaginary time, $|\rho_m\rangle = e^{Gzm}|\rho_0\rangle$ with $G = -H$, and keep every snapshot. One state

evolves along $z = \delta t$, the other along the complex contour $z = \delta t\, e^{-i\alpha}$, which slows down the decay of modes with $\mathrm{Im}\,\lambda > 0$. Then we diagonalize G in the space spanned by the snapshots. The eigenvalues λ with the largest real part come first, so for $G = -H$ the energies are $E = -\lambda$, lowest first. This is the method of arXiv:2509.07709, where G is a Lindbladian.

```
sc = spinchain.Spin_Chain(["S=1/2" for i in range(n)], itensor_version="python")
sc.set_hamiltonian(h)
ev, res, obs = sc.get_krylov_spectrum()  # eigenvalues of G = -H
es = -ev                                 # energies, lowest first
zz = obs(sc.Sz[0]*sc.Sz[1], 0)           # <Sz0 Sz1> in the lowest state
```

`res[k]` says how much λ_k moves when the last snapshot of each state is left out. A converged eigenvalue does not move, so a small `res[k]` marks the converged part of the list. It is not an error bar: on small chains the true error was 3 to 40 times larger. On the 20-site chain the lowest energy agrees with `gs_energy()` to 10^{-5}. For a Hermitian H each energy is an upper bound on the exact one at the same position. Two random starts give at most two copies of a degenerate level, so a triplet appears twice. The keywords are:

- `G`: the generator, a `MultiOperator` (default $-H$).
- `nk`: the number of snapshots per state (default 20).
- `dt`: the time step (default 0.5). A small `dt` gives snapshots that are almost parallel, and the spectrum becomes worse.
- `alpha`: the contour angle (default: estimated from a short run).
- `gram_tol`: relative threshold below which directions of the snapshot space are dropped (default `1e-12`).
- `exact_residuals=True`: `res[k]` becomes the residual $\|G|r_k\rangle - \lambda_k|r_k\rangle\|$ of the normalized eigenvector. It is a real error measure, but it costs much more on a large chain.
- `mode="ED"`: every eigenvalue of the dense G, in the same order, as a reference.

Backends.

`get_krylov_spectrum` runs on `itensor_version="python"`, `3` and `"julia_live"`, and not yet in a conserved sector. On `"julia_live"` it takes random starts only. The random starts differ between the backends, so the unconverged levels differ too.

6.6 Generalized eigenvalue problem

For a Hermitian positive definite operator A,

$$H|\psi\rangle = \lambda A|\psi\rangle.$$

```
A = fc.Id + 0.5*sum(fc.N)            # for example
lam = fc.gs_energy_generalized(A)    # smallest lambda
```

DMRG solves this directly, with no inverse of A. A non-Hermitian H is also accepted, and then λ is the complex eigenvalue with the smallest real part. Not available on `itensor_version=2`, it raises an error. It also raises if A is not Hermitian, or if $\langle\psi|A|\psi\rangle$ is not positive during the solve.

After this call the chain stores the generalized eigenstate as its ground state, and `fc.e0` is its energy $\langle\psi|H|\psi\rangle/\langle\psi|\psi\rangle$ (with left and right states for a non-Hermitian H), while λ is in `fc.lam_generalized`. Every later calculation (excited states, correlators) uses this state. So call it last, or call `fc.restart()` and `fc.gs_energy()` to get back the true ground state.

6.7 Effective low-energy Hamiltonians

An effective Hamiltonian acts in a small model space and has the same low-energy spectrum as H. For a Hubbard chain at strong coupling the model space is the states with one electron per site, and the effective model is a spin model. We take the n lowest eigenstates $|\psi_k\rangle$, project them on the model space with its projector Q, and orthonormalize them (Löwdin), which gives $|\tilde{k}\rangle$. The effective Hamiltonian is

$$H_{\text{eff}} = \sum_{k<n} E_k |\tilde{k}\rangle\langle\tilde{k}| \simeq \sum_a J_a O_a,$$

and the couplings J_a of the operators O_a are fitted by least squares, with the O_a represented on the $|\tilde{k}\rangle$. This is the des Cloizeaux effective Hamiltonian. It has the eigenvalues E_k, so on a Hubbard dimer the fitted exchange is the singlet-triplet splitting $E_T - E_S$, which tends to $4t^2/U$ at strong coupling:

```
# fc: a Spinful_Fermionic_Chain(2) Hubbard dimer, with 20*(N-2)^2 added
# to H to keep the four lowest states at half filling
from dmrgpy import effectivehamiltonian
Q = effectivehamiltonian.single_occupancy_projector(fc)
J = fc.get_heff(operators=[fc.Sx[0]*fc.Sx[1], fc.Sy[0]*fc.Sy[1],
                           fc.Sz[0]*fc.Sz[1]], n=4, projector=Q)

coef = effectivehamiltonian.get_effective_hamiltonian_couplings(fc, n=4,
                                                                projector=Q)
latex = effectivehamiltonian.get_effective_hamiltonian(fc, n=4, projector=Q)
```

`get_heff` fits the operators given and returns one J per operator. `get_effective_hamiltonian_couplings` fits all products $S_i^\alpha S_j^\beta$ of the chain spin operators and returns a dictionary, and `get_effective_hamiltonian` returns the same fit as a LaTeX string. The keywords are:

- `n`: the number of states. It must not cut a degenerate multiplet, and it must not be larger than the dimension of the model space. Otherwise it raises an error.
- `projector`: the projector on the model space, as an operator or a list of commuting operators. `single_occupancy_projector(fc)` gives the states with one electron per site. Without it, the model space is the span of the n eigenstates themselves. This is right for a spin chain, but for a Hubbard chain the spin operators then miss the doubly occupied weight, and J is too large at small U.
- `operators`: the operators of the fit. They must be linearly independent in the model space, otherwise the couplings are not unique and it raises an error.
- `tol`: couplings smaller than this are set to zero.
- `method`: `"single"` (default) fits $QO_aQ\,QO_bQ$, and `"full"` fits $Q\,O_aO_b\,Q$. They differ when the model space is not closed under O_a.
- `mode="ED"`: exact eigenstates, for small systems.

See `examples/groundstate/effective_hamiltonian`, which fits the twisted exchange of a Hubbard dimer with a Peierls phase.

The figure shows the exchange J that `get_heff` fits on a Hubbard dimer (`n=4`, `mode="ED"`, with $20(N-2)^2$ added to H to keep the four states at half filling). The three couplings come out equal, as for an isotropic exchange. With the projector, J is the splitting $E_T - E_S$ (dashed) at every U, and it joins $4t^2/U$ at large U. Without the projector, J is too large at small U: the singlet has doubly occupied weight, where $\mathbf{S}_0 \cdot \mathbf{S}_1 = 0$, and the fit gives $(E_T - E_S)/(1/4 + 3w/4)$, with w the singly occupied weight. This is 40% above the splitting at $U = t$.

The code of this figure is in `docs/figures/excited-states_heff/`.

7 Entanglement

On this page we measure the entanglement of a state. Here `wf` is a ground state, for example `wf = sc.get_gs()` for the Heisenberg chain of Ground state.

7.1 Entanglement entropy of a cut

We cut the chain between sites i and $i+1$. The Schmidt decomposition of the state is $|\psi\rangle = \sum_\alpha \lambda_\alpha |\alpha\rangle_L |\alpha\rangle_R$, and the von Neumann entropy of the left part is

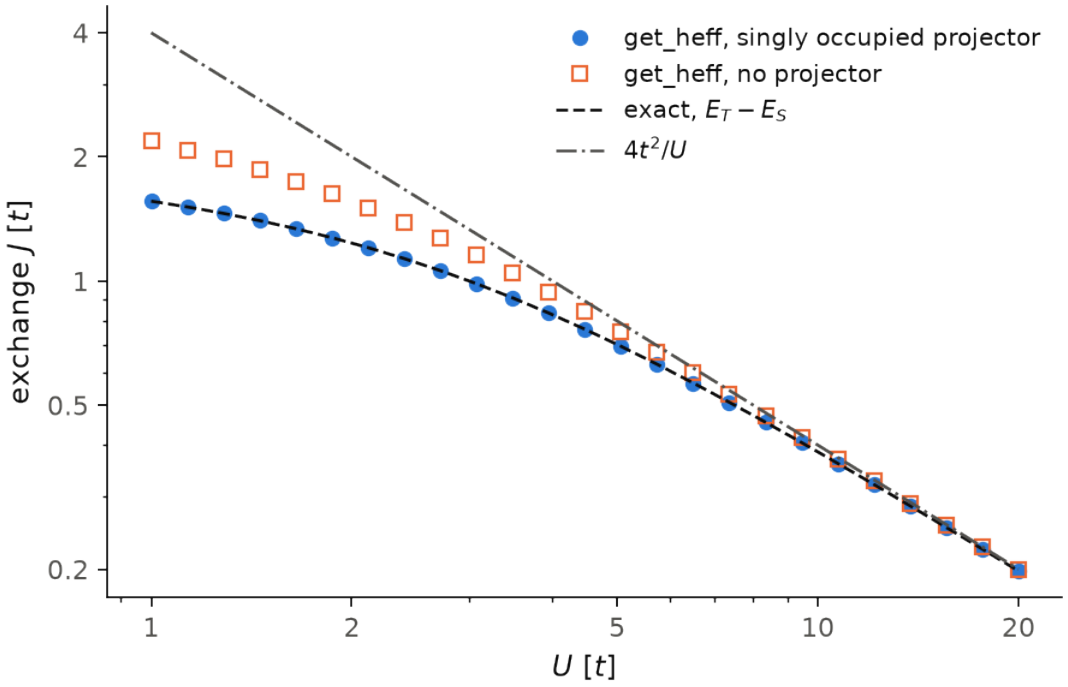


Figure 7: Exchange of the Hubbard dimer

$$S_i = -\sum_\alpha \lambda_\alpha^2 \log \lambda_\alpha^2 = -\mathrm{Tr}\, \rho_L \log \rho_L, \qquad \rho_L = \mathrm{Tr}_R |\psi\rangle\langle\psi|.$$

```
s = wf.get_bond_entropy(i, i+1)      # cut between sites i and i+1
s = wf.get_entropy(b)                # the same as get_bond_entropy(b, b+1)
```

In a gapped chain S_i saturates with the size of the block, and in a critical chain it grows as $\log \ell$ (see the figure in Central charge). It also tells us the bond dimension we need, roughly $\chi \gtrsim e^S$. The bond entropy needs an MPS, so it is not available with `mode="ED"`.

7.2 Reduced density matrix

The reduced density matrix of one site,

$$\rho_i = \mathrm{Tr}_{j \neq i} |\mathrm{GS}\rangle\langle\mathrm{GS}|,$$

contains every local observable of that site, and its eigenvalues give the site entropy of the next section.

```
rho = sc.get_rdm(i=0)      # d x d matrix, d the local dimension
```

It is computed from the MPS, so it is not available with `mode="ED"`, where it raises an error.

7.3 Entropy of one site or two sites

`wf.get_site_entropy(i)` and `wf.get_pair_entropy(i, j)` give the entropy of the reduced density matrix of site i, or of the two sites i and j (not necessarily neighbors), with the rest of the chain. They are available for spin-1/2 chains and spinless fermionic chains. With `mode="ED"`, call them on the chain: `sc.get_site_entropy(wf, i)`, `sc.get_pair_entropy(wf, i, j)`.

7.4 Mutual information

For two sites i and j,

$$I(i,j) = S_i + S_j - S_{ij},$$

```
mi = wf.get_mutual_information(i, j)
```

$I(i,j)$ measures all the correlations, classical and quantum, between the two sites, so it shows which sites are coupled in the state even when no term of H couples them. Same availability as the site entropy.

7.5 Central charge

For a critical open chain of length L, the entropy of a cut at position ℓ follows the Calabrese-Cardy formula,

$$S(\ell) = \frac{c}{6} \log\left[\frac{2L}{\pi} \sin\frac{\pi\ell}{L}\right] + \text{const.}$$

The central charge c identifies the conformal field theory, for example $c = 1/2$ for the critical Ising chain and $c = 1$ for the XX and Heisenberg chains. `get_CFT_central_charge()` fits this formula to the entropy of every bond:

```
c = wf.get_CFT_central_charge()
```

On an open chain the entropy also has a term that alternates between even and odd bonds, and the fit does not remove it. So the result converges slowly with L: for the spin-1/2 XX chain it gives about 0.54 at $L = 20$ and 0.74 at $L = 40$, and on a short chain it can even be negative (-0.63 for the 8-site Heisenberg chain). Use long chains, or fit the bond entropies yourself with only even (or only odd) ℓ.

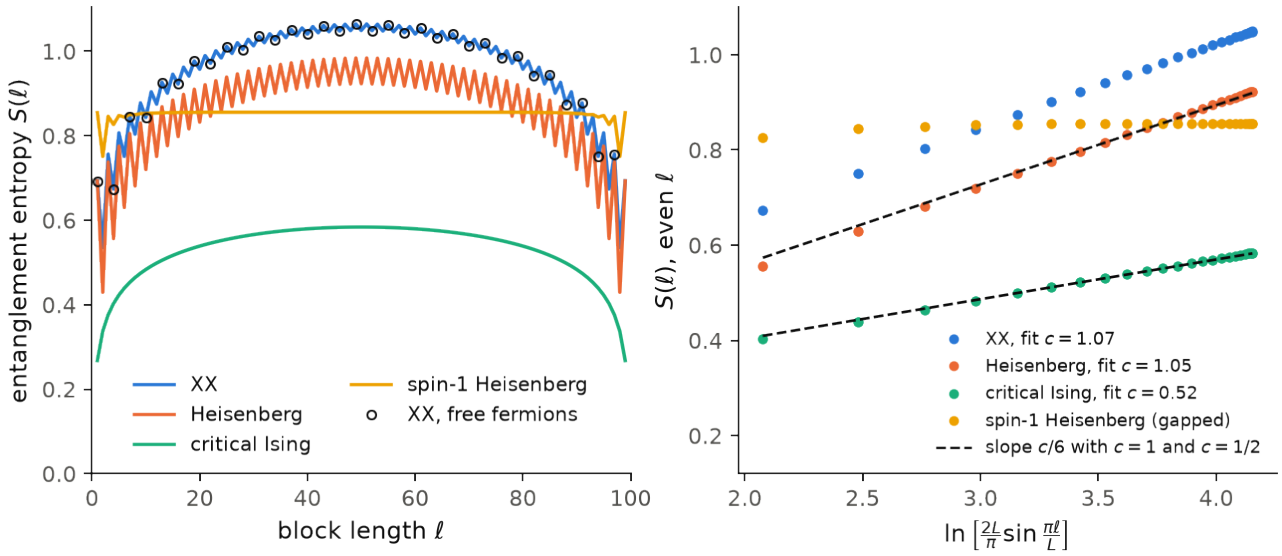


Figure 8: Entanglement entropy and central charge

The figure shows open chains with $L = 100$ sites. On the left is the entropy of every cut. For the XX chain the DMRG entropies lie on the exact free-fermion values (circles). The XX, Heisenberg and critical Ising chains follow an arch. For the XX and Heisenberg chains the even-odd oscillation of the open chain sits on top of it. The gapped spin-1 chain is flat in the bulk: its entropy saturates. On the right, the even bonds lie on straight lines against the Calabrese-Cardy variable. The XX and Heisenberg lines are parallel, with $c \approx 1$, and the Ising line has half their slope, $c \approx 1/2$. At $L = 100$ the fits give 1.07, 1.05 and 0.52; the exact free-fermion entropies of the XX chain give the same 1.07, so the remaining difference is a finite-size effect.

The code of this figure is in `docs/figures/entanglement_central_charge/`.

7.6 Participation entropy

A measurement of every site in the computational basis (S^z for spins, the occupation for fermions) gives a snapshot s with probability $p(s) = |\langle s|\psi\rangle|^2$. Snapshots are what a quantum gas microscope or a Rydberg array records. The Rényi participation entropies

$$S_q = \frac{1}{1-q} \log \sum_s p(s)^q$$

measure how much the state spreads over the basis. They grow linearly with the length of the chain. We build $p(s)$ as an MPS, the elementwise product of ψ^* and ψ, by recursive sketched interpolation (Meng, Khoo, Li and Stoudenmire, arXiv:2602.17974), whose cost grows as χ^3, against χ^4 for the exact product followed by a compression. The sum $\sum_s p(s)^q$ is then an overlap.

```
s2 = sc.get_participation_entropy(q=2)              # of the ground state
s3 = sc.get_participation_entropy(wf, q=3, maxm=80)
```

- `q`: an integer, at least 2.
- `maxm`: the bond dimension of p (default the chain's `maxm`). p needs more than ψ itself, so check that the result is converged in it.
- `mode="ED"`: the exact value, for a small chain.

In `examples/entanglement/participation_entropy` the ground state of the Heisenberg chain from 4 to 12 sites agrees with ED to 10^{-7}, the accuracy of its own DMRG state.

7.7 Single-particle correlation matrix

For fermions, the correlation matrix is

$$C_{ij} = \langle c_i^\dagger c_j \rangle.$$

Its eigenvalues $n_\alpha \in [0, 1]$ are the occupations of the natural orbitals. For a single Slater determinant every n_α is 0 or 1. Interactions move them away from 0 and 1, and the correlation entropy measures how far the state is from a single Slater determinant:

$$S = -\sum_\alpha \Big[n_\alpha \log n_\alpha + (1 - n_\alpha) \log(1 - n_\alpha) \Big].$$

Here `fc` is a fermionic chain with its Hamiltonian set.

```
cm = fc.get_correlation_matrix()          # C_ij
na = fc.get_correlation_eigenvalues()     # n_alpha
s = fc.get_correlation_entropy()          # S
orb = fc.get_correlated_orbitals()        # |orbital|^2, most correlated first
rho = fc.get_correlated_density(n=2)      # weight of the 2 most correlated orbitals
sd = fc.get_gs().get_correlation_entropy_density()   # S resolved by site
```

The most correlated orbitals are those with n_α closest to 1/2. The four-point correlators and the two-particle density matrix are on the Correlators and distributions page. The keywords are:

- `T`: temperature of `get_correlation_matrix` (default 0). Finite T is computed with exact eigenstates, for small chains.
- `dmmode`: how C_{ij} is computed (`"fast"`, `"full"`, `"explicit"`, `"simple"`). The default chooses the fastest one for the state. `"explicit"` works on every backend.

The figure shows the half-filled Hubbard chain with $n = 12$ sites and hopping $t = 1$. On the left are the occupations n_α for several values of U. At $U = 0$ the ground state is one Slater determinant, so half of the occupations are 1 and the other half are 0. As U grows, the step melts and all occupations move towards 1/2.

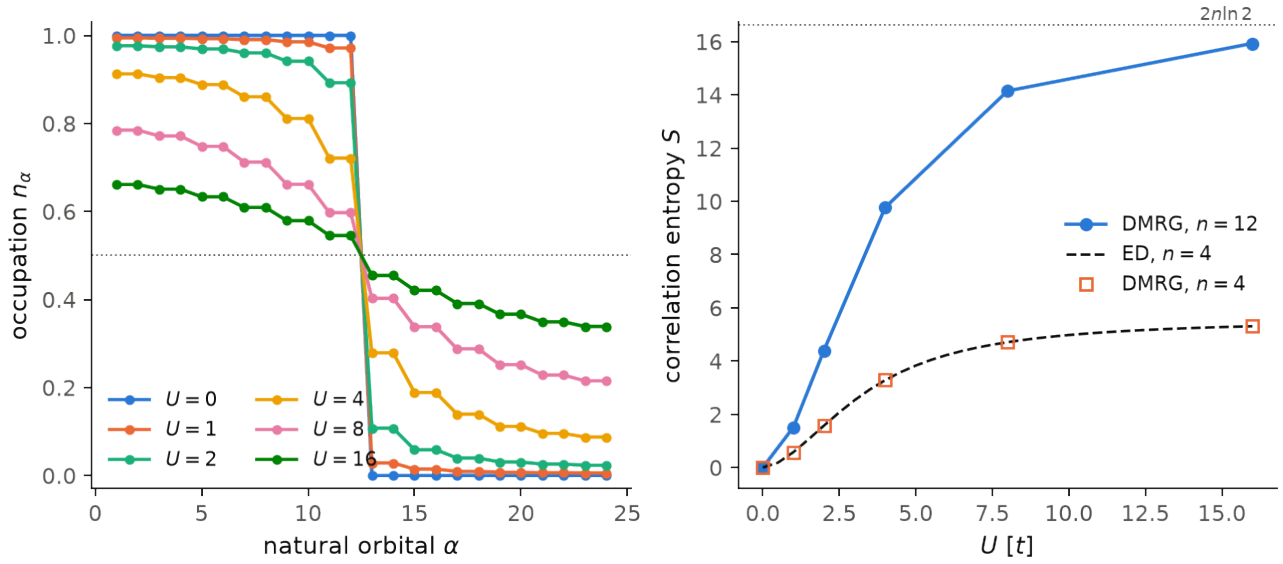


Figure 9: Natural orbitals of the Hubbard chain

On the right is the correlation entropy. It starts at zero and grows towards its maximum $2n \log 2$. For a chain with $n = 4$ the DMRG values (squares) lie on the ED curve (dashed line).

The code of this figure is in `docs/figures/entanglement_natural_orbitals/`.

8 Correlators and distributions

On this page we measure the ground state beyond a two-point correlator: four-point correlators, the two-particle density matrix, and the full distribution of an operator. Two-point correlators and the equal-time structure factor are on the Ground state page, and the single-particle correlation matrix, whose eigenvalues are the natural orbitals, is under Entanglement.

8.1 Four-point correlators

For a fermionic chain `fc`, the four-point tensor $\langle c_i^\dagger c_j c_k^\dagger c_l \rangle$ for all i, j, k, l is

```
t = fc.get_gs().get_four_correlation_tensor()    # shape (n, n, n, n)
```

It holds every density-density correlator $\langle n_i n_k \rangle$ (at $j = i, l = k$) and every pair correlator, so it is the input of a two-particle reduced density matrix or of a cumulant. The keyword `ctmode` chooses the method (`"batched"`, `"sweep"`, `"fold"`, `"full"`, `"explicit"`). The default chooses the fastest one for the backend, and `"explicit"` works on every backend. A single four-point correlator is also `fc.vev(fc.Cdag[i]*fc.C[j]*fc.Cdag[k]*fc.C[l])`, which is cheaper when only a few are needed.

The two-particle density matrix $\langle c_i^\dagger c_j^\dagger c_l c_k \rangle$, with the pairs $i < j$ and $k < l$ as row and column, is

```
from dmrgpy.entropytk.correlationentropy import get_highorder_correlation_matrix
d2 = get_highorder_correlation_matrix(fc)
```

8.2 Operator distributions

For a Hermitian operator X, the expectation value $\langle X \rangle$ is only the mean. The full distribution of the measured values of X in the ground state is

$$P(x) = \langle \mathrm{GS} | \, \delta(X - x) \, | \mathrm{GS} \rangle, \qquad \int P(x)\, dx = 1.$$

For a conserved charge this is the full counting statistics. A sharp $P(x)$ means that the ground state has a well-defined value of X, and a broad one means large fluctuations. It is computed with the kernel polynomial method, as the dynamical correlators:

```
X = sum(sc.Sz[i] for i in range(n))       # total magnetization
x, p = sc.get_distribution(X=X, delta=0.1)
mus, shift, scale = sc.get_distribution_moments(X=X, delta=0.1)
```

`delta` sets the resolution through the number of Chebyshev moments. Unlike in Dynamical correlators, it is not calibrated as a broadening, so compare distributions computed with the same `delta`. `get_distribution` also works with `mode="ED"`. `get_distribution_moments` returns the raw Chebyshev moments of the rescaled operator and has no ED version. A maximum-entropy reconstruction from the moments $\langle X^k \rangle$, `distribution.get_distribution_maxent(sc, X=..., n=6)`, gives the smoothest distribution with the first `n` moments. With few moments it shows the envelope of the distribution, not its separate peaks.

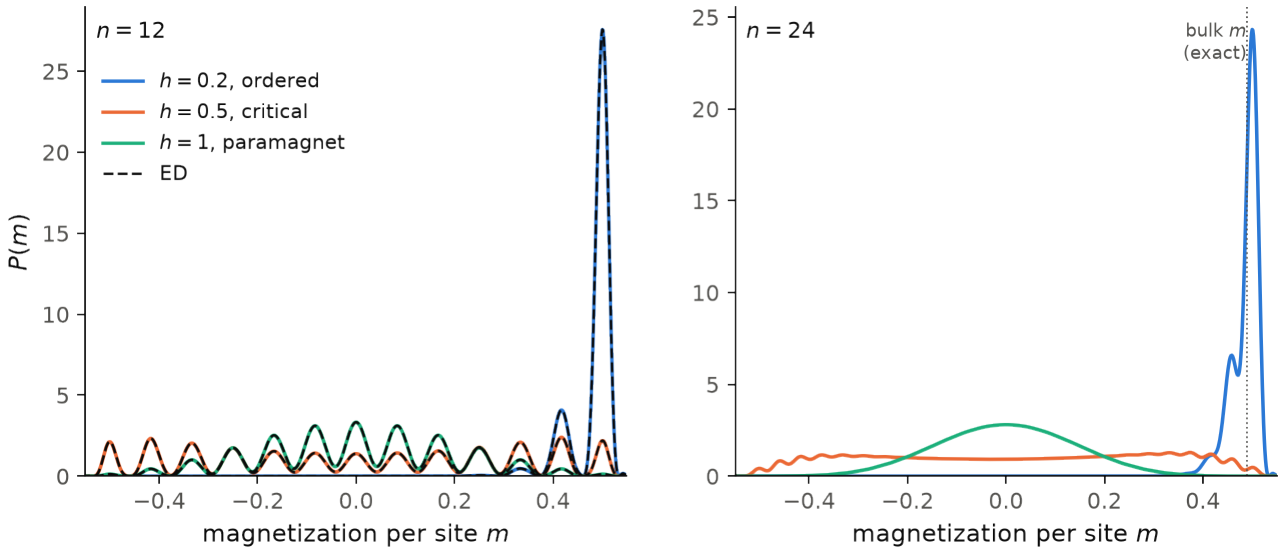


Figure 10: Magnetization distribution of the Ising chain

The figure shows $P(m)$ for the magnetization per site $m = \frac{1}{n}\sum_i S_i^z$ in the transverse-field Ising chain $H = -\sum_i S_i^z S_{i+1}^z - h\sum_i S_i^x$, which is critical at $h = 1/2$. A small field $10^{-4}\sum_i S_i^z$ selects one of the two ordered states. In the ordered phase ($h = 0.2$), $P(m)$ is one sharp peak close to the exact bulk value $m = 0.489$. At the critical field it is broad and flat, and in the paramagnet ($h = 1$) it is one peak at $m = 0$. On the left, with $n = 12$, the DMRG curves lie on the ED curves (dashed lines). There the peaks are separated, because m only takes the values k/n. On the right, with $n = 24$, these values are closer than the resolution set by `delta`, so the distribution looks smooth.

The code of this figure is in `docs/figures/correlators_distribution/`.

The many-body density of states, $\rho(E) = \sum_n \delta(E - E_n)$, is not a distribution of this kind: `get_distribution` computes a ground-state expectation value, not a trace over all the states, and there is no method for $\rho(E)$. For a small chain the full ED spectrum from `get_excited(mode="ED", n=...)` (Excited states) gives it directly.

9 Mean-field decoupling

For a spin Hamiltonian with exchange couplings J_{ij}, the self-consistent mean-field (Weiss-field) decoupling replaces the exchange by a field on each site,

$$H_{\mathrm{MF}} = p\, H_{\mathrm{exch}} + H_{\mathrm{onsite}} + \sum_i \mathbf{h}_i \cdot \mathbf{S}_i, \qquad \mathbf{h}_i = (1-p)\sum_j J_{ij}\langle \mathbf{S}_j \rangle.$$

We solve H_{MF}, compute the new $\langle \mathbf{S}_i \rangle$, rebuild $\mathbf{h}_i$, mix the old and new values, and repeat until the magnetization stops changing. With $p = 0$ this is pure mean-field theory. With $0 < p < 1$ a fraction p of the exchange is kept and treated exactly, together with the field. With $p = 1$ it is the original model. The couplings are read from the Hamiltonian of the chain, so any one- and two-site spin Hamiltonian works, including anisotropic and long-range exchange. Here `sc` is the Heisenberg chain of Ground state:

```
from dmrgpy import meanfield
n = sc.ns
m0 = [[0, 0, 0.5*(-1)**i] for i in range(n)]       # staggered start
mf = meanfield.spinchain_meanfield(sc, p=0.0, m0=m0, mix=0.5)
m = mf.get_magnetization()       # self-consistent <S_i>, a 3 x n array
e = mf.gs_energy()               # mean-field energy
```

The function returns a new chain whose Hamiltonian is the converged H_{MF}. This Hamiltonian includes the constant $-(1-p)\sum_{ij} J_{ij} \langle \mathbf{S}_i \rangle \cdot \langle \mathbf{S}_j \rangle$, so the exchange is not counted twice. At $p = 0$ its `gs_energy()` is the expectation value of the original Hamiltonian in the mean-field state. The keywords are:

- `p`: the fraction of the exchange kept exactly.
- `m0`: the starting magnetization, one 3-vector per site. Random by default. For an antiferromagnet start from a staggered pattern, since simple mixing can oscillate otherwise.
- `mix`: the weight of the new magnetization in each step (default 0.9). Lower it if the loop oscillates.
- `maxerror`, `maxite`: the convergence tolerance and the maximal number of iterations. After `maxite` the loop stops with a warning.
- `mixmode`: `"default"` (linear mixing) or `"broyden"`.
- `mode="ED"`: solve each step with exact diagonalization.

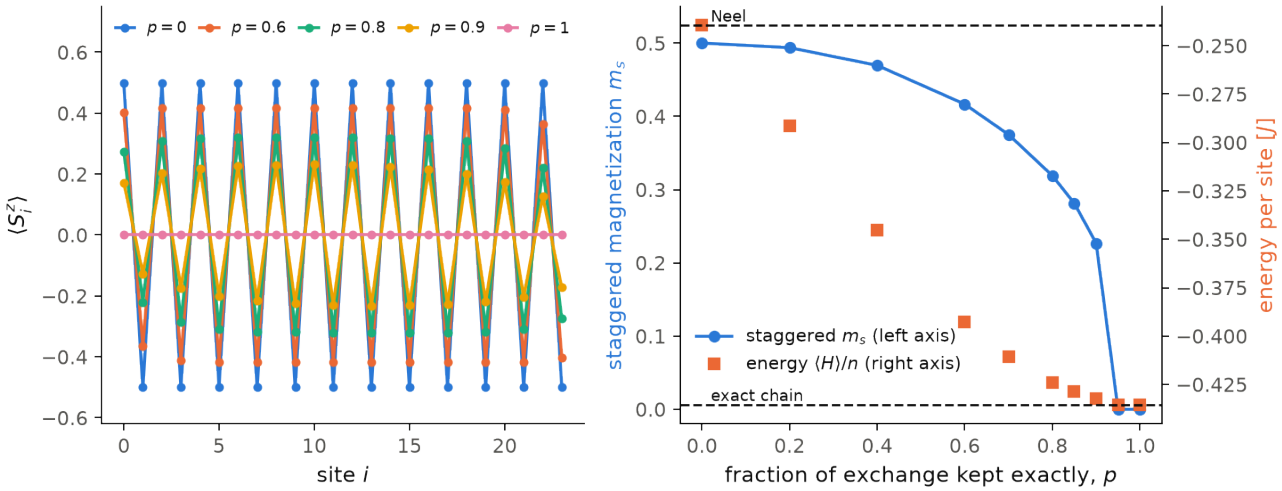


Figure 11: Mean-field Neel order of the Heisenberg chain

The figure shows the Heisenberg chain with $n = 24$ sites, started from the staggered pattern above. On the left is $\langle S_i^z \rangle$. At $p = 0$ it is the classical Neel state with $|\langle S_i^z \rangle| = 1/2$. As p grows, the order gets weaker. On the right, the staggered magnetization of the 12 central sites falls with p and drops to zero between $p = 0.9$ and $p = 0.95$, before $p = 1$: in a finite chain the order disappears once the Weiss field is small compared with the finite-size gap. The squares are $\langle H \rangle$ of the original Heisenberg chain in the self-consistent state, not `gs_energy()` of the returned chain, which differs from it for $0 < p < 1$. The energy goes from the Neel value $-(n-1)/(4n)$ at $p = 0$ to the exact energy of the chain at $p = 1$. In the bulk this exact energy is $1/4 - \log 2 = -0.443$ per site; the open chain with $n = 24$ gives -0.4356, because of its edges.

The code of this figure is in `docs/figures/mean-field_neel/`.

`meanfield.decompose_spin_hamiltonian(sc)` returns the onsite fields `b[i,a]`, the exchange `J[i,j,a,b]` and the constant read from the Hamiltonian. A Hamiltonian that is not of this form, for example with a three-site

term, raises an error.

10 Topological invariants

10.1 Many-body Berry phase

Let the Hamiltonian depend on a parameter ϕ that goes around a closed loop, for example a flux ϕ through a ring, entering as a twist $e^{i\phi}$ on one bond. The discrete Berry phase of the ground state is

$$\gamma = \arg \prod_k \langle \psi(\phi_k) | \psi(\phi_{k+1}) \rangle,$$

with the last state connected to the first one. With inversion symmetry γ is quantized to 0 or π, and the two values are two different phases, for example the two dimerizations of a spin chain. We compute it by solving the ground state at each ϕ_k and multiplying the overlaps. Here `h_of(phi)` returns the twisted Hamiltonian of `sc`:

```
import numpy as np
phis = np.linspace(0, 2*np.pi, 20, endpoint=False)
wfs = []
for phi in phis:
    sc.set_hamiltonian(h_of(phi))
    wfs.append(sc.get_gs())
prod = np.prod([wfs[i].dot(wfs[(i+1) % len(wfs)]) for i in range(len(wfs))])
gamma = np.angle(prod)        # 0 or pi
```

See `examples/topological/zak_phase` for the dimerized Heisenberg chain.

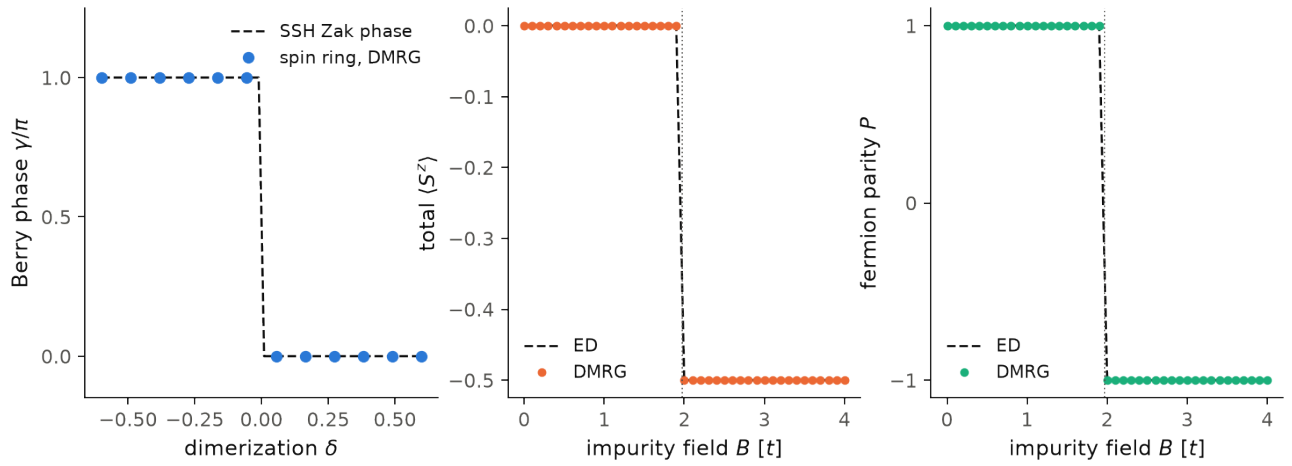


Figure 12: Berry phase and fermion parity

On the left, the figure shows the Berry phase of a ring of 12 spins 1/2 with exchange $J_i = 1 + \delta(-1)^i$ and the twist on the bond with $J = 1 - \delta$, from 16 values of ϕ. It is π when the twisted bond is a strong bond ($\delta < 0$) and 0 when it is a weak bond ($\delta > 0$), and it jumps at $\delta = 0$. The dashed line is the Zak phase of the SSH chain with the same dimerization (see Single-particle Berry phase), and it jumps at the same point. The middle and right panels belong to Fermion parity.

The code of this figure is in `docs/figures/topology_berry_phase/`.

10.2 Single-particle Berry phase

For a Bloch Hamiltonian $H(k)$, the Zak phase of the occupied bands is the phase of the Wilson loop,

$$\gamma = \arg\det \prod_k U(k, k+\delta k), \qquad U_{mn}(k, k+\delta k) = \langle u_m(k) | u_n(k+\delta k)\rangle,$$

with m, n running over the occupied bands. It gives the polarization of the band. Here `hk(k)` returns the Bloch matrix, with k from 0 to 1:

```
from dmrgpy import topology
gamma = topology.berry_phase_matrix(hk, nk=40)    # the lower half of the bands is occupied
```

10.3 Fermion parity

For a fermionic chain the total parity is

$$P = \left\langle \prod_i (1 - 2n_i) \right\rangle.$$

It is $+1$ or -1 for a state with an even or odd number of fermions. In a chain with superconducting pairing the particle number is not conserved, but the parity is. A change of the ground-state parity as a function of a parameter is a level crossing between the two sectors, as in a Kitaev chain or a magnetic impurity in a superconductor. Here `fc` is a `Fermionic_Chain`:

```
wf = fc.get_gs()
p = wf.get_fermionic_parity()                       # +1 or -1
p = wf.get_fermionic_parity(fpmode="iterative")     # same number, other route
```

See `examples/topological/parity` for P as a function of the chemical potential.

The middle and right panels of the figure in Many-body Berry phase show a chain of 6 sites with hopping $t = 1$, chemical potential $\mu = -0.5$, onsite pairing $0.5\,(c_{i\uparrow}c_{i\downarrow} + \text{h.c.})$, a weak Hubbard interaction $U = 0.5$, and a Zeeman field $B\,S^z_0$ on the first site, a magnetic impurity. At $B \approx 1.97$ the impurity binds one quasiparticle. The parity jumps from $+1$ to -1, and the total $\langle S^z \rangle$ jumps from 0 to $-1/2$ at the same field. DMRG (points) agrees with ED (dashed lines).

11 Fidelity susceptibility

Let the Hamiltonian depend on a parameter, $H(\lambda) = H_0 + \lambda H_1$. The fidelity susceptibility measures how fast the ground state changes with λ. In perturbation theory it is

$$\chi(\lambda) = \sum_{n \neq 0} \frac{|\langle 0|H_1|n\rangle|^2}{(E_n - E_0)^2 + \delta^2}.$$

At a quantum phase transition the gap closes and χ has a peak that grows with system size. So a scan of $\chi(\lambda)$ locates a critical point without knowing the order parameter. Here `sc` is a spin chain:

```
from dmrgpy import fidelity
h0 = 4*sum(sc.Sz[i]*sc.Sz[i+1] for i in range(n-1))   # Ising coupling
h1 = 2*sum(sc.Sx[i] for i in range(n))                # transverse field
chi = fidelity.get_fidelity(sc, h0, h1, 1.0)              # default
chi_pt = fidelity.get_fidelity(sc, h0, h1, 1.0, fmode="PT")  # formula above
```

The default, `fmode="derivative"`, does not use the formula above. It computes the second derivative of the ground-state overlap by finite differences, so it needs only ground states. If the ground state is degenerate within `delta`, it uses the overlap matrix of the degenerate states and returns the sum of χ over these states. `fmode="PT"` sums over the `n` lowest states. Increase `n` until χ stops changing. The keywords are:

- `dl`: the finite-difference step in λ (default 0.1). Use a smaller value when χ changes fast.

- `n`, `delta`: for the default estimator, the number of low states searched for a degenerate ground state, and the energy tolerance for calling two states degenerate. For `fmode="PT"`, `n` is the number of states in the sum (default 2) and `delta` is the δ of the formula above.
- `mode="ED"`: exact diagonalization, for small chains.

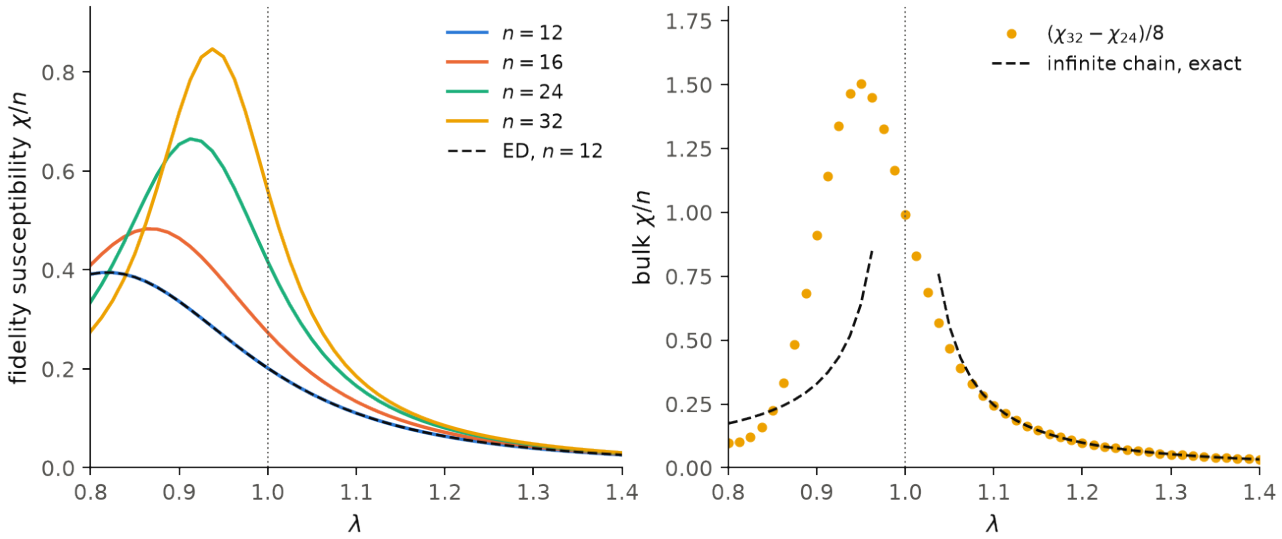


Figure 13: Fidelity susceptibility of the Ising chain

The figure shows open Ising chains, $H = \sum_i \sigma^z_i \sigma^z_{i+1} + \lambda \sum_i \sigma^x_i$, computed with `dl=0.02`. On the left is χ/n for four lengths. The peak grows with n and moves towards the critical point $\lambda = 1$: it is at $\lambda = 0.83$, 0.86, 0.91 and 0.94 for $n = 12$, 16, 24 and 32. For $n = 12$ the ED curve (dashed line) covers the DMRG one. On the right, the difference of the two longest chains removes the contribution of the edges and gives χ/n of the bulk. From about $\lambda = 1.1$ it lies on the exact result of the infinite chain, $1/[16\lambda^2(\lambda^2 - 1)]$. On the ordered side the exact result is $1/[16(1-\lambda^2)]$, and the two chains are still too short to reach it: the difference crosses it at $\lambda = 0.85$ and lies below it at $\lambda = 0.8$. The exact free-fermion solution of the open chain shows that at $\lambda = 0.8$ the difference reaches the bulk value only for n of about 50. Close to $\lambda = 1$ the correlation length is longer than the chains, so they do not reach the infinite-chain result there.

We start the scan at $\lambda = 0.8$. In the ordered phase the two lowest states of an open chain have an energy splitting that decays with n. For $n = 32$ it is 6×10^{-4} at $\lambda = 0.8$ and drops below the default `delta` near $\lambda = 0.75$. Below that point the default estimator returns χ summed over the two states. A smaller `delta` does not help: DMRG does not separate two states this close in energy, and the overlaps are then not meaningful.

The code of this figure is in `docs/figures/fidelity_ising/`.

12 Dynamical correlators

On this page we compute zero-temperature dynamical correlators: the dynamical spin structure factor, the single-particle spectral function and the local density of states, and the correlator of any pair of operators (A, B).

12.1 The quantity

For a pair (A, B), `get_dynamical_correlator` returns the Lehmann spectral density

$$S_{AB}(\omega) = \sum_n M_n \frac{\delta/\pi}{(\omega - \Delta_n)^2 + \delta^2} = \frac{i}{2\pi}\Big[G^R_{AB}(\omega) - G^A_{AB}(\omega)\Big],$$

with $M_n = \langle \mathrm{GS}|A|n\rangle\langle n|B|\mathrm{GS}\rangle$, $\Delta_n = E_n - E_0$, and the resolvents $G^{R/A}_{AB}(\omega) = \langle \mathrm{GS}|A\,(\omega + E_0 \pm i\delta - H)^{-1} B|\mathrm{GS}\rangle$.

The frequency ω is measured from the ground-state energy E_0, and δ is the broadening. When every M_n is real, S_{AB} is real and equal to $-\frac{1}{\pi}\mathrm{Im}\, G^R_{AB}(\omega)$. This is always the case for a Hermitian pair $A = B^\dagger$, where $S_{AB} \geq 0$, and also for real operators on a real Hamiltonian. In general M_n is complex, so the returned array is complex. A converged calculation satisfies the sum rule

$$\int d\omega\, S_{AB}(\omega) = \langle \mathrm{GS}|AB|\mathrm{GS}\rangle,$$

which is a simple check of a calculation. For EX and SECTOR (Choosing a correlator method) the missing weight is the part of the spectrum outside the states kept. Here $|\mathrm{GS}\rangle$ is the ground state of the chain, or the state set with `set_gs()`.

All the examples use the Heisenberg chain `sc` of Ground state.

```
import numpy as np
es = np.linspace(-1, 4, 300)                        # frequencies
(x, y) = sc.get_dynamical_correlator(name=(sc.Sz[0], sc.Sz[0]),
                                     es=es, delta=0.1)
print(np.sum(y.real)*(x[1]-x[0]))                   # sum rule: <Sz_0 Sz_0> = 0.25
```

The keywords that every method shares are:

- `name=(A, B)`: the operator pair, built from the chain operators. A string with two sites also works, `name="ZZ", i=0, j=3`, as do `"XX"`, `"YY"`, `"+-"`, `"cdc"` and `"densitydensity"`.
- `es`: the frequency grid.
- `delta`: the broadening. A pole becomes a line of full width 2δ.
- `submode`: the method, see Choosing a correlator method. The default is `"KPM"`.
- `mode="ED"`: exact diagonalization, for small chains and for checking. It accepts the submodes `"KPM"`, `"ED"`, `"INV"`, `"EX"`, `"CVM"`, `"ROOTN"` and `"TD"`.

The default `es` and `delta` change from one method to another, so it is best to always give them. A KPM line has the width 2δ at the center of the many-body spectrum, and down to about 0.7 of it for low-energy excitations of a long chain. It is not a Lorentzian: its peak is about 1.5 times higher than a Lorentzian of the same width and weight. So compare peak heights only between calculations with the same method.

12.2 Dynamical spin structure factor

The dynamical spin structure factor is

$$S^{zz}_{ij}(\omega) = \sum_n \langle \mathrm{GS}|S^z_i|n\rangle\langle n|S^z_j|\mathrm{GS}\rangle\, \delta(\omega - \Delta_n).$$

The local one, $i = j$, is what inelastic spin-flip spectroscopy with an STM measures on a single magnetic atom. Its Fourier transform

$$S(q,\omega) = \frac{1}{n}\sum_{i,j} e^{iq(i-j)} S^{zz}_{ij}(\omega)$$

is what inelastic neutron scattering measures. We compute it from all the site pairs:

```
es = np.linspace(0, 4, 100)
qs = np.linspace(0, 2*np.pi, 41)
sqw = np.zeros((len(qs), len(es)))
for i in range(n):
    for j in range(n):
        (x, y) = sc.get_dynamical_correlator(name=(sc.Sz[i], sc.Sz[j]),
                                             es=es, delta=0.1)
        sqw += np.real(np.exp(1j*qs*(i-j))[:, None]*y[None, :])/n
```

For a Heisenberg chain the weight is largest at $q = \pi$, the antiferromagnetic wave vector, and its lower edge follows the two-spinon dispersion $\frac{\pi}{2}|\sin q|$ for long chains.

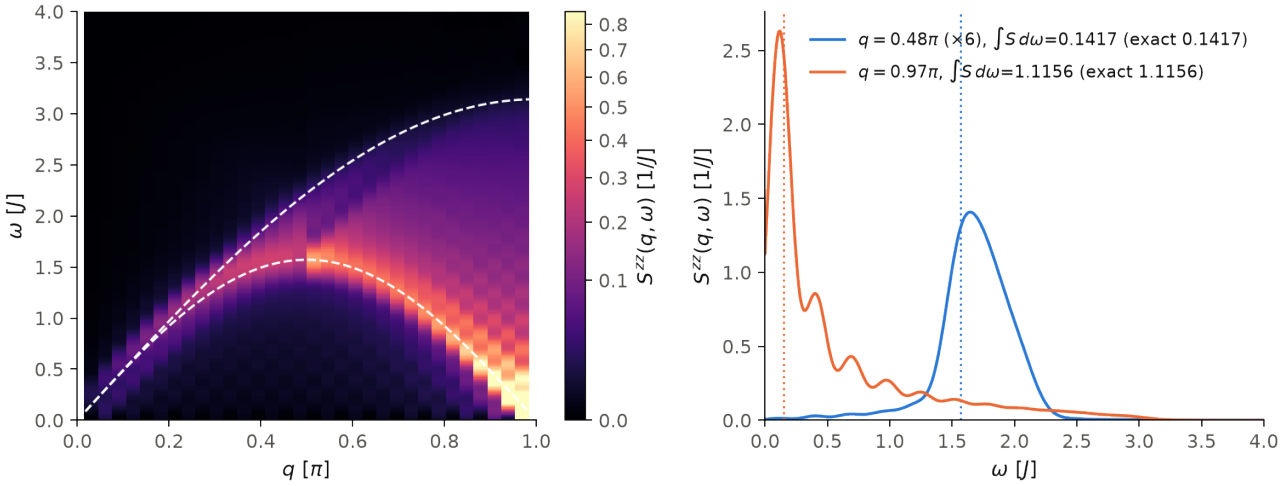


Figure 14: Two-spinon continuum of the Heisenberg chain

The figure shows $S^{zz}(q,\omega)$ of an open Heisenberg chain with $n = 32$, with KPM and $\delta = 0.15$. On an open chain the momentum states are standing waves, so we use one operator per momentum, $S^z_q = \sqrt{2/(n+1)}\sum_i \sin(q(i+1))S^z_i$ with $q = \pi k/(n+1)$, and one KPM run per q. The weight fills the two-spinon continuum between the exact edges $\frac{\pi}{2}J|\sin q|$ and $\pi J|\sin(q/2)|$ (dashed). It is largest just above the lower edge and near $q = \pi$. The right panel shows two cuts, with the lower edge as a dotted line. Their integrals match $\langle S^z_q S^z_q\rangle$ from `vev`. At every q this sum rule holds to a few parts in 10^5.

The code of this figure is in `docs/figures/dynamical-correlators_sqw/`.

Two methods give the spin structure factor directly:

- `sc.get_spin_spectral_function(i, j=None, es=..., delta=..., nex=20)`: returns $S^{zz}_{ij} + (S^{+-}_{ij} + S^{-+}_{ij})/2$, which equals $S^{xx} + S^{yy} + S^{zz}$. With `return_poles=True` it also returns the three channels separately, so we can see which spin channel each excitation belongs to. `connected=True` removes the elastic peak $\langle S^z_i\rangle\langle S^z_j\rangle$ at $\omega = 0$. It uses the SECTOR method, so it needs `itensor_version=3`, `"python"` or `"julia_live"`. The sector states are shared by all the pairs (i, j), so a full $S(q,\omega)$ map costs about as much as one site. This is the better choice on a long chain. See `examples/dynamical_correlator/sector_spin_structure_fa`
- `sc.get_full_SS_correlator(mode="ED", i=0, j=None, es=..., delta=...)`: returns $\frac{1}{3}\sum_\alpha S^{\alpha\alpha}_{ij}(\omega)$, averaged over a degenerate ground-state manifold. This is what an unpolarized measurement on an isotropic magnet sees. ED only.

12.3 Single-particle spectral function

For a fermionic chain the single-particle spectral function is

$$A_{ij}(\omega) = \sum_n \langle \mathrm{GS}|c_i|n^{N+1}\rangle\langle n^{N+1}|c_j^\dagger|\mathrm{GS}\rangle\,\delta(\omega-\omega_n^+) + \sum_m \langle \mathrm{GS}|c_j^\dagger|m^{N-1}\rangle\langle m^{N-1}|c_i|\mathrm{GS}\rangle\,\delta(\omega-\omega_m^-),$$

with $\omega_n^+ = E_n^{N+1} - E_0^N - \mu$ and $\omega_m^- = E_0^N - E_m^{N-1} - \mu$. The first term is electron addition, at $\omega > 0$, and the second is electron removal, at $\omega < 0$. The local one, $A_{ii}(\omega)$, is the local density of states measured by STM (dI/dV); its Fourier transform $A(k,\omega)$ is measured by ARPES (removal part) and inverse photoemission (addition part).

Here `fc` is a `Fermionic_Chain` and `hc` a `Spinful_Fermionic_Chain_Native`:

```
fc.set_conserved_sector(Nf=n//2)                    # the N-particle ground state
(x, A, info) = fc.get_spectral_function(i=0, es=np.linspace(-4, 4, 400),
```

```
                                        delta=0.1, nex=20, return_poles=True)
print(info["mu"], info["gap"])                  # mu and the charge gap

hc.set_conserved_sector(Nf=n, Sz=0)             # half-filled Hubbard chain
(x, A) = hc.get_spectral_function(i=0, spin="up", es=np.linspace(-6, 6, 400),
                                  delta=0.2, nex=20)
```

The chemical potential $\mu = (E_0^{N+1} - E_0^{N-1})/2$ is subtracted by default, so the gap is centered at $\omega = 0$. Pass `shift=None` to keep the raw frequencies. `info` also gives the charge gap $E_0^{N+1} + E_0^{N-1} - 2E_0^N$ and the two halves separately. This method needs `itensor_version=3`, `"python"` or `"julia_live"`. See `examples/dynamical_correlator/sector_spectral_function_hubbard`, where the Mott gap opens with U.

On other backends, or for a broad continuum, we build the two halves with `get_dynamical_correlator`. The second operator acts first on the ground state, so $(c_i, c_i^\dagger)$ is the addition part and $(c_i^\dagger, c_i)$ the removal part:

```
es = np.linspace(0, 4, 200)
(x, yp) = fc.get_dynamical_correlator(name=(fc.C[0], fc.Cdag[0]), es=es, delta=0.1)
(x, yh) = fc.get_dynamical_correlator(name=(fc.Cdag[0], fc.C[0]), es=es, delta=0.1)
w = np.concatenate([-es[::-1], es])             # removal at w<0, addition at w>0
A00 = np.concatenate([yh.real[::-1], yp.real])
```

Here the frequency is measured from E_0^N in both halves, without the μ shift. If the chain is not in a sector, $|\mathrm{GS}\rangle$ is the global ground state, at whatever particle number it has.

12.3.1 Local density of states

The local spectral function $A_{ii}(\omega)$ is the density of states measured by STM (dI/dV) above site i, with electron addition at $\omega > 0$ and electron removal at $\omega < 0$. On a spinful chain, pass `spin="up"` or `spin="dn"` to `get_spectral_function` and add the two.

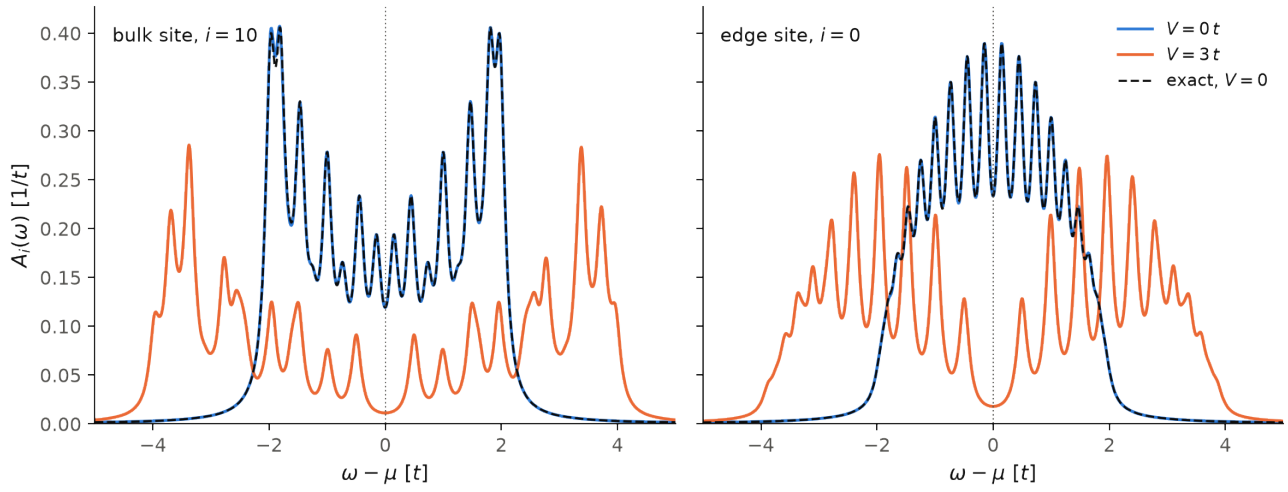


Figure 15: Local density of states of a fermion chain

The figure shows $A_i(\omega)$ of an open chain of 20 spinless fermions at half filling, with $H = -t\sum_i (c_i^\dagger c_{i+1} + \text{h.c.}) + V\sum_i (n_i - \frac{1}{2})(n_{i+1} - \frac{1}{2})$, from `get_spectral_function` with `nex=50` and $\delta = 0.1$. The left panel is a site in the middle of the chain and the right panel the edge site. At $V = 0$ the result lies on the exact free-fermion spectrum (dashed): 20 levels inside the band $|\omega| < 2t$, with a weight that depends on the site. In the middle of the chain the weight piles up at the band edges, as in $1/\sqrt{4t^2 - \omega^2}$. At the edge it is largest at the band center. At $V = 3t$ the chain is a charge density wave, and a gap of about t opens at $\omega = \mu$. The weight moves to higher energies, around $\pm V$. Here the 50 states hold 79 per cent of the weight at the middle site, so part of the spectrum at high energy is missing; `info["particle"]["captured"]` and `info["hole"]["captured"]` tell this.

The code of this figure is in `docs/figures/dynamical-correlators_ldos/`.

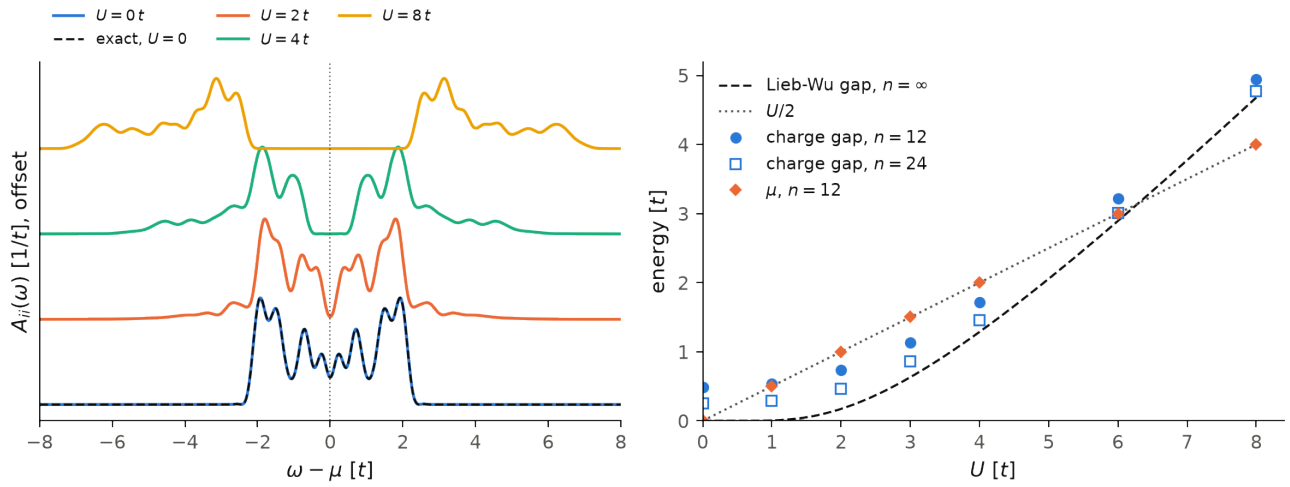


Figure 16: Mott gap of the Hubbard chain

The left panel shows $A_{ii}(\omega)$ at a bulk site of a half-filled Hubbard chain with $n = 12$, from the two KPM correlators above with $\delta = 0.25$. We write the interaction as $U \sum_i (n_{i\uparrow} - \frac{1}{2})(n_{i\downarrow} - \frac{1}{2})$, so the global ground state is at half filling and $\mu = 0$. At $U = 0$ the result lies on the exact free-fermion spectrum (dashed), drawn with the same KPM kernel. As U grows, the single band splits into a lower and an upper Hubbard band. The right panel shows the charge gap from the ground states of the sectors $N-1$, N and $N+1$. It approaches the Lieb-Wu gap of the infinite chain (dashed) as n grows, and the finite chains stay above it because of the level spacing. For these sectors we use $U \sum_i n_{i\uparrow} n_{i\downarrow}$, and the chemical potential comes out exactly at $U/2$, as particle-hole symmetry requires.

The code of this figure is in `docs/figures/dynamical-correlators_spectral_function/`.

12.4 Any pair of operators

`name=(A, B)` accepts any operators built from the chain operators, for example a bond operator:

```
A = hc.Cdagup[0]*hc.Cup[1]
(x, y) = hc.get_dynamical_correlator(name=(A, A.get_dagger()), es=es, delta=0.1)
```

The pair may also be operators already converted with `sc.toMPO(A)` (Hamiltonians and operators). This avoids building them again in a scan over parameters. It works with the DMRG submodes `"EX"`, `"CVM"` and `"ROOTN"`, and with `mode="ED"` when the operators are built with `sc.toMPO(A, mode="ED")`. The other submodes raise an error and need the operator itself.

13 Choosing a correlator method

The `submode` keyword of `get_dynamical_correlator` selects how $S_{AB}(\omega)$ is computed. All the methods return the same quantity, defined in Dynamical correlators.

13.1 The methods

- `"KPM"` (default): the Kernel Polynomial Method. It expands the spectrum in Chebyshev polynomials of H and gives the whole frequency range at once. The cost grows as $1/\delta$. Use it for a first look and for broad spectra.
- `"CVM"`: the correction-vector method. It solves one linear system per frequency, so it is accurate at each ω but the cost grows with the number of points in `es`. Use it to resolve a narrow window or a sharp peak. `"CVM_explicit"` computes the same resolvent in a different way, and is useful as a cross-check.

- `"ROOTN"`: the root-N Krylov correction vector of Nocera and Alvarez (arXiv:2204.03165). It applies the resolvent in `N` smaller steps, each in a Krylov space of size `nkry`. Use it for high frequencies, near the top of the spectrum, where a single correction vector needs a large bond dimension. It is the most expensive method.
- `"TD"`: real-time evolution of $C(t) = \langle \mathrm{GS}|A(t)B(0)|\mathrm{GS}\rangle$ followed by a Fourier transform with the damping $e^{-\delta t}$. Use it for fine resolution and when time evolution is cheap (short-range Hamiltonians, small entanglement growth). The time propagator is the one of Real-time evolution.
- `"TDZ"`: the same, but the evolution follows a complex-time contour (Cao, Lu, Stoudenmire and Parcollet, arXiv:2311.10909). The bond dimension grows much more slowly than in `"TD"`, so it reaches longer times.
- `"EX"`: an explicit sum over the lowest `nex` excited states found by DMRG. Use it when a few low-energy states carry the physics, for example in a small gapped system.
- `"SECTOR"`: an explicit sum over the lowest `nex` eigenstates of the quantum-number sector that $B|\mathrm{GS}\rangle$ belongs to ($N+1$ for $c^\dagger$, S^z+1 for S^+). Use it for sharp low-energy excitations: the poles come out at converged energies with exact weights. It needs `itensor_version=3`, `"python"` or `"julia_live"`. `get_spectral_function` and `get_spin_spectral_function` are built on it.
- `"TASK"`: the tangent-space Krylov method of Kovalska, von Delft and Gleis (arXiv:2510.07279). A Lanczos recursion runs in the space of one-site changes of the ground-state MPS, where the Krylov vectors stay exactly orthogonal. It gives discrete poles and weights for all frequencies at once, and an error bar for each pole. The one approximation is that the excited states are one-site changes of the ground state, which is exact when `maxm` reaches the full Hilbert space and weakest for broad multi-particle continua. It needs `itensor_version=3`, `"python"` or `"julia_live"`.
- `"ED"` and `"INV"` (`mode="ED"` only): the exact Lehmann sum and the exact resolvent. These are the references to check the other methods.
- `"maxent"`: the smoothest spectrum with the first six moments of $H - E_0$ in the state $B|\mathrm{GS}\rangle$. It gives the envelope of the spectrum, not its separate peaks.
- `"CVMimag"`: the correction vector at 20 frequencies above the real axis, continued to it with a Pade approximant. The total weight and the peak positions are right, the peaks are smoothed: on a 6-site Heisenberg chain the main peak is 8% lower than the exact one.

As a rule: KPM for the full spectrum, CVM or TD for a narrow window at high resolution, TDZ when TD needs too large a bond dimension, EX or SECTOR when a few states are enough, TASK for sharp poles with an error bar on each.

The figure shows the local correlator $S^{zz}_{00}(\omega)$ at the edge of a Heisenberg chain with $n = 10$, computed with six methods and the same $\delta = 0.1$. The dashed lines are ED. The number next to each curve is the largest deviation from ED, relative to the peak height. CVM, TD, TDZ and SECTOR fall on the exact Lehmann sum. The KPM lines are narrower and taller than the Lorentzian lines of ED (dotted). Compared with ED and the same KPM kernel, KPM agrees to 0.1 per cent. EX with `nex=20` misses all the weight above $\omega \approx 1.2J$, because the states that carry it are not among its 20 states.

The code of this figure is in `docs/figures/correlator-methods_comparison/`.

```
pair = (sc.Sz[0], sc.Sz[0])
(x, y) = sc.get_dynamical_correlator(submode="CVM", name=pair, es=es, delta=0.1)
(x, y) = sc.get_dynamical_correlator(submode="TD", name=pair, es=es, delta=0.1, dt=0.05)
(x, y) = sc.get_dynamical_correlator(submode="EX", name=pair, es=es, delta=0.1, nex=20)
(x, y) = sc.get_dynamical_correlator(mode="ED", submode="ED", name=pair, es=es, delta=0.1)
```

13.2 Keywords of each method

The accuracy of every method also depends on the ground state itself, so `maxm` and `nsweeps` (Ground state) come first.

- `sc.kpmmaxm` (KPM): the bond dimension of the Chebyshev vectors. Increase it until the spectrum stops changing.
- `sc.kpm_scale` (KPM, default 0.7): how much of $[-1, 1]$ the spectrum of H fills after rescaling. Do not go below 0.5; the calculation raises an error when the expansion becomes unstable.

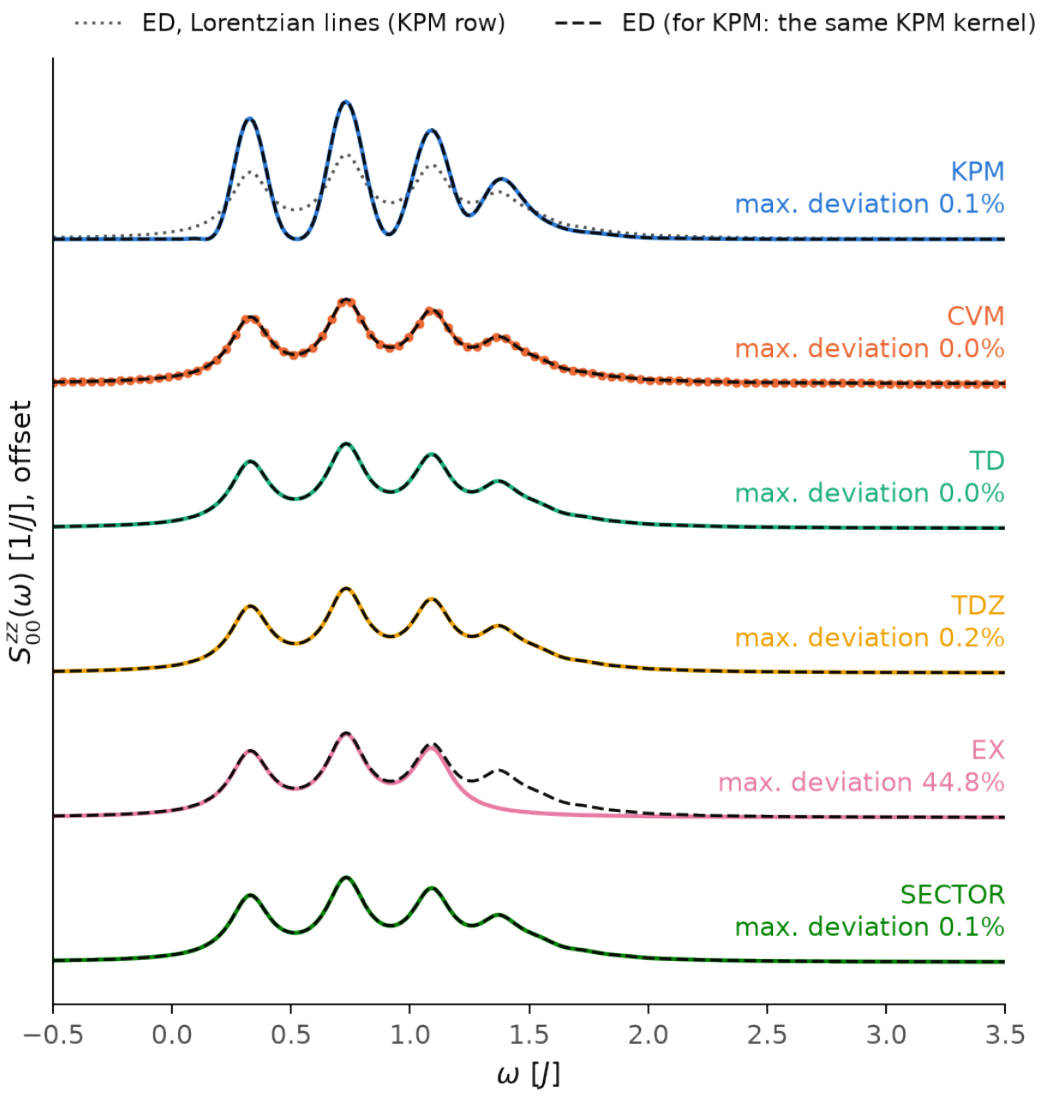


Figure 17: Six methods against ED

- `sc.kpm_n_scale` (KPM, default 1): a positive integer that multiplies the number of Chebyshev moments, for a line sharper than 2δ.
- `sc.kpm_extrapolate = True` (KPM): computes the moments for the broadening `kpm_extrapolate_factor` times `delta` (default 2, so half as many moments) and extrapolates the sequence to the full count. On a Heisenberg chain of 10 sites the line agrees with the full calculation within 2 per cent of its peak, at half the cost. Check it against the full calculation on a small chain before using it in a scan.
- `kernel=` (KPM): how the moments become a spectrum, `"jackson"` (default), `"lorentz"`, `"plain"` or `"hodc"`. The last is the high-order kernel of arXiv:2512.03149, with `hodc_order` (default 6) and `hodc_eta` (default `delta`). It is more accurate where the spectrum is a smooth continuum, but it can be slightly negative and is more sensitive to a small `kpmmaxm`. See `examples/dynamical_correlator/hodc_VS_jackson_kernel`.
- `sc.kpm_energy_truncate=True` (KPM, `itensor_version=3` and `"python"`): energy truncation of Holzner et al., PRB 83, 195115 (2011). It allows `kpm_scale` well below 0.5, so that more moments are spent on the low-energy window. It is slower per moment. The settings are `kpm_truncate_dK` (increase until converged, 30 is a good value), `kpm_truncate_nsweeps` and `kpm_truncate_threshold`. See `examples/dynamical_correlator/dynamical_correlator_kpm_energy_truncation`.
- `sc.cvm_maxm` and `sc.cvm_tol` (CVM): the bond dimension of the correction vector and the relative tolerance of its solver (default 10^{-5}). If a frequency point warns that it did not converge, increase `cvm_maxm`.
- `sc.cvm_solver="variational"` (CVM, `itensor_version=3` and `"python"`, only for `A` equal to `B^dagger`): the dynamical DMRG of Jeckelmann, PRB 66, 045114 (2002). The correction vector is found by minimizing a functional with two-site sweeps, so the truncation is part of the ansatz. Set `cvm_nsweeps` and `cvm_maxm`. The default `"cg"` works on every backend.
- `sc.inverse_solver` (CVM_explicit): the solver of each resolvent, for any pair `A`, `B`. By default two-site sweeps on `"python"`, since the resolvent is not Hermitian, and BiCGSTAB on the other backends (see Operators).
- `N` and `nkry` (ROOTN, defaults 8 and 20): the number of steps and the Krylov dimension. Larger `N` helps at high frequency when `nkry` is small.
- `dt` and `damping_periods` (TD and TDZ): the time step, and the total time T in units of $1/\delta$ (default 6). The resolution is set by T.
- `damping` and `predict` (TD): the time window, `"exp"` (default, a Lorentzian line), `"gaussian"` (faster decaying tails, line 1.2 times wider) or `"parzen"`, and linear prediction of $C(t)$ to longer times (default `True`), which makes the line sharper and more accurate when the simulated time is short.
- `alpha0` and `n_max` (TDZ, defaults 0.1 and 4): the angle of the time contour and the order of the expansion back to real time. A larger `alpha0` needs a smaller bond dimension but a larger `n_max`, at most 4.
- `nex` (EX and SECTOR): the number of states in the sum. With `return_poles=True`, SECTOR also returns the poles and their weights, and `info["captured"]` is the fraction of the weight $\langle \mathrm{GS}|B^\dagger B|\mathrm{GS}\rangle$ they account for. A warning appears below 0.9; increase `nex` then.
- `nkry` (TASK, default 100): the number of Lanczos steps, about the number of poles. With `return_poles=True`, TASK returns `(es, y, info)`: `info["omega"]` and `info["weight"]` are the poles and their weights, `info["elastic"]` the weight $\langle A\rangle\langle B\rangle$ at $\omega = 0$, and `info["variance"]` the two-site variance of each pole, $\|P^{2\perp}H|\alpha\rangle\|^2$. Its square root is an error bar on the energy of the pole; increase `maxm` until it is small for the poles you need. `info["missing_weight"]` is the sum rule $\langle AB\rangle$ minus all the weights. It is not zero when B changes more than one site of the ground state: a bond operator, or c_i on a state of mixed fermion parity, such as a degenerate pair of Majorana ground states. The variance does not see this loss. A warning appears when it is above 10^{-6} of the weight; increase `maxm` then.

13.3 Reusing the KPM moments

The Chebyshev moments are the expensive part of KPM. We can compute them once and build the spectrum with several kernels:

```
from dmrgpy import kpmdmrg
mus, emin, emax, scale, npol, delta = sc.get_dynamical_correlator_moments(
        name=(sc.Sz[0], sc.Sz[0]), delta=0.1)
(x, y) = kpmdmrg.dynamical_correlator_from_moments(mus, emin, emax, scale,
        npol, es, kernel="hodc", delta=delta)
```

13.4 Degenerate ground states

With `mode="ED", submode="ED"`, a degenerate ground state is handled by averaging the correlator over all the states with excitation energy below `dex` (default 10^{-5}), with equal weights. This is a hard cutoff, so in a scan where a level splitting crosses `dex` (a Zeeman field, for example) the result jumps. A warning appears when a level is close to `dex`. For a scan, use a small temperature instead, `T=0.01`, which weights the states with Boltzmann factors and changes smoothly (Finite temperature). See `examples/kondo/atom_iets_orbital_resolved`.

13.5 Non-Hermitian Hamiltonians

For a non-Hermitian H (Excited states) the ground state is a pair of left and right eigenstates, and the correlator is $\langle\psi_L|A(z)B|\psi_R\rangle$. `submode="KPM"` then uses the non-Hermitian KPM of PRL 130, 100401, and needs `E_max`, an upper bound on the spectral radius of H. It recomputes the moments at every frequency, so it is much slower. `"CVM"`, `"CVM_explicit"` and `"EX"` also work; the other submodes raise an error. Non-Hermitian KPM works with `mode="ED"`, `itensor_version=3`, `"python"` and `"julia_live"`. See `examples/non_hermitian/nhkpm_v3_VS_ED`.

14 Real-time evolution

On this page we evolve a state in real time, $|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$, and measure an observable along the way. The typical case is a quantum quench: we prepare the ground state of a Hamiltonian H_0, change the Hamiltonian to H_1 at $t = 0$, and follow how observables oscillate, relax or thermalize. Here `h0` and `h1` are two Hamiltonians of the chain `sc`, for example a staggered field and the Heisenberg chain of Ground state.

```
from dmrgpy import timedependent
sc.set_hamiltonian(h0)
wf0 = sc.get_gs()                              # ground state of H0
sc.set_hamiltonian(h1)                         # quench to H1
(ts, sz) = timedependent.evolve_and_measure(sc, operator=sc.Sz[0],
                                             nt=200, dt=0.02, wf=wf0)
```

This gives $\langle\psi(t)|S^z_0|\psi(t)\rangle$ at the times $t = 0$, dt, $2dt$, ... The keywords are:

- `nt`, `dt`: the number of time steps (default 1000) and the time step (default 0.01). A smaller `dt` is more accurate.
- `wf`: the initial state. Without it, the ground state of the chain is used.
- `h`: the Hamiltonian of the evolution, if not the one of the chain.
- `mode="ED"`: exact evolution, the reference for small chains.
- `return_wf=True` (DMRG): also return the final state as a third output.

The accuracy of the DMRG evolution is set by `sc.maxm` and `dt`. The entanglement grows with time after a quench, so the bond dimension needed grows too. When the result stops agreeing with a larger `maxm`, the simulation has reached its maximum time.

The figure shows a quench in the XX chain, $H = \sum_i (S^x_i S^x_{i+1} + S^y_i S^y_{i+1})$, with $n = 40$. The initial state is a domain wall, all spins up on the left half and all down on the right half. Left: $\langle S^z_x(t)\rangle$ from TDVP. The magnetization moves only inside a cone of slope one (dashed lines), since the fastest excitations have velocity $v = J$. Right: the profile at three times. The model maps to free fermions, so the exact profile is known (dashed lines). TDVP agrees with it to 10^{-4}. At long times the profile approaches $-\frac{1}{\pi}\arcsin(x/t)$ (dotted line).

The code of this figure is in `docs/figures/real-time_light_cone/`.

14.1 Local quench

`evolution_ABA` first applies an operator A to the state, for example a spin flip or an added electron, and then measures B:

$$\langle\psi|A^\dagger e^{iHt} B e^{-iHt} A|\psi\rangle.$$

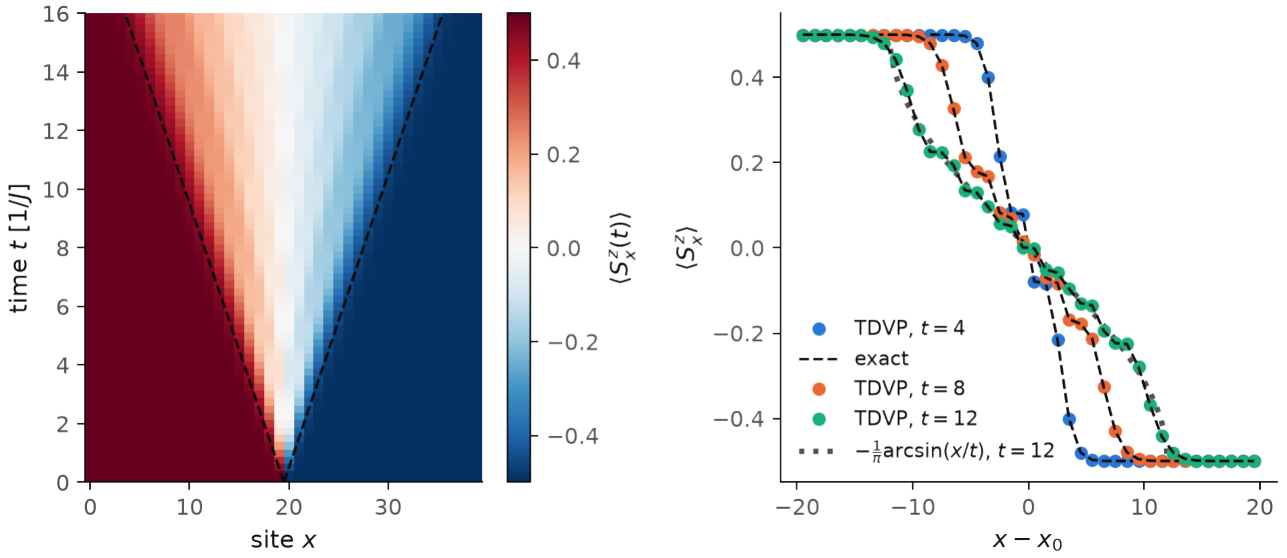


Figure 18: Light cone of a domain wall in the XX chain

Here we flip up the spin at the edge of the Heisenberg chain and follow how the magnetization arrives at site 3:

```
Sp0 = sc.Sx[0] + 1j*sc.Sy[0]                    # S^+ on site 0
(ts, sz3) = timedependent.evolution_ABA(sc, A=Sp0, B=sc.Sz[3], nt=200, dt=0.02)
```

It takes the same keywords as `evolve_and_measure`, and starts from the ground state of the chain unless `wf` is given. Note that A is applied as it is, so the state $A|\psi\rangle$ is not normalized.

14.2 Evolved states

To get the states themselves, `sc.exponential(-1j*t*h, wf)` returns $e^{-iHt}|\psi\rangle$ for one time (Hamiltonians and operators), and `timeevolution.evolve_WF(h, wf, ts=ts)` returns a list of states, one for each time in `ts`. Note that `evolve_WF` applies e^{+iht}, so pass `-h` for the physical evolution. With the states, we compute any overlap, for example the return amplitude $\langle\psi(0)|\psi(t)\rangle$.

14.3 Choosing the propagator

`sc.tevol_method` selects the algorithm:

- `"TDVP"` (default): two-site TDVP. It works for any Hamiltonian and grows the bond dimension as needed. On `itensor_version=2` it is replaced by `"MPO"`.
- `"TDVP_GSE"`: one-site TDVP with global subspace expansion (Yang and White, PRB 102, 094315 (2020)). Use it when the initial state has a small bond dimension, for example a product state. The bond dimension grows only during the first `sc.tdvp_gse_sweeps` steps (default 3); the other settings are `sc.tdvp_gse_krylov_order` (default 3) and `sc.tdvp_gse_cutoff` (default 10^{-8}).
- `"TDVP_CBE"`: one-site TDVP with controlled bond expansion (Li, Gleis and von Delft, arXiv:2208.10972). Before each local step, the bond gets the few new states on which $H|\psi\rangle$ has weight outside the current bond space, and after it the bond is cut back to `sc.maxm`. Like `"TDVP_GSE"` it starts well from a product state, but the bond keeps growing at every step, so it follows a quench whose entanglement grows for a long time. Its accuracy is the one of `"TDVP"`. `sc.cbe_delta` (default 0.1) sets how far a bond is expanded, to $(1+\delta)$`maxm` states. Only on `itensor_version="python"`, 3 and `"julia_live"`; the other backends raise an error. See `examples/time_evolution/tdvp_cbe_VS_ED_time_evolution`.
- `"TEBD"`: second-order Trotter evolution with two-site gates. Faster than TDVP, but only for nearest-neighbor Hamiltonians; otherwise it raises an error. A `Spinful_Fermionic_Chain_Native` Hubbard chain is nearest-neighbor, the interleaved `Spinful_Fermionic_Chain` is not.

- `"AUTO"`: `"TEBD"` when the Hamiltonian is nearest-neighbor, `"TDVP"` otherwise.
- `"MPO"`: a second-order Taylor expansion of $e^{-iH\,dt}$ applied as an MPO. The only option on `itensor_version=2`, and less accurate than TDVP at the same bond dimension.

```
sc.tevol_method = "TEBD"
```

The same propagator is used by the `"TD"` and `"TDZ"` dynamical correlators (Choosing a correlator method). See `examples/time_evolution/time_evolution_quench`.

15 STM and Kondo spectra

On this page we compute the differential conductance dI/dV measured by an STM tip above a magnetic atom or a spin chain on a surface. The electrons tunnel from the tip to the sample, and on the way they exchange spin and energy with the spins. We follow the perturbation theory of Ternes, New J. Phys. 17, 063016 (2015), arXiv:1505.04430, to third order in the coupling. The tip couples to one site of the chain. Here the chain is a single spin-1/2 in a magnetic field, with H in eV:

```
import numpy as np
from dmrgpy import spinchain
muB, g, B = 5.7883818066e-5, 2.0, 10.0    # eV/T, g-factor, field in T
sc = spinchain.Spin_Chain(["S=1/2"])
sc.set_hamiltonian(g*muB*B*sc.Sz[0])
eV = np.linspace(-4e-3, 4e-3, 161)        # bias in eV
eV, dIdV = sc.get_kondo_spectrum(eV, site=0, Jrho_s=-0.05, U=0.25, T=1.0)
```

15.1 Second order

Second order (`order=2`) is the Fermi golden rule for spin-flip and elastic tunneling,

$$\frac{dI}{dV}(eV) \propto \sum_{i,f} p_i \left[\tfrac{1}{2}|\langle f|S_-|i\rangle|^2 + \tfrac{1}{2}|\langle f|S_+|i\rangle|^2 + |\langle f|S_z|i\rangle|^2 \right] \Theta(eV - \epsilon_{if}) + 4U^2,$$

summed over both tunneling directions. Here p_i is the Boltzmann weight of eigenstate i, $\epsilon_{if} = E_f - E_i$, and Θ a step broadened by temperature. Each excitation of the spins gives a step in dI/dV at $eV = \pm\epsilon_{if}$. This is inelastic tunneling spectroscopy (IETS), and the step positions measure the spin excitation energies, for example the Zeeman energy.

15.2 Third order

Third order (`order=3`, the default) adds two terms. Both sum over every eigenstate m as a virtual intermediate state. The Kondo term involves three spin matrix elements and a logarithmic function $F(\epsilon, T)$. It gives a peak at zero bias, which splits into two peaks at the Zeeman steps in a field. The potential-interference term, present when `U` is not zero, makes the spectrum asymmetric in bias. The second-order and Kondo terms are even in eV, and the potential term is odd in eV.

At the parameters of the paper ($S = 1/2$, $B = 0$, $T = 1$ K, $J\rho_s = -0.05$, $\omega_0 = 20$ meV), the zero-bias peak divided by 2π is 1.13 for $U = 0$ and 1.39 for $U = 0.25$, as in the paper's figures. In a field the code departs from the paper in one place: the exchange diagram of the Kondo term has its logarithm at $F(eV - (\epsilon_f - \epsilon_m))$, which is what an exact second-order T-matrix gives, and not at the argument of the paper's eq. 25. The two agree at zero field for $S = 1/2$ and at zero bias. In the cases tested with a field or anisotropy, they differ by 5 to 20 per cent of the third-order term. The keywords are:

- `site`: the site under the tip.
- `Jrho_s`: the dimensionless Kondo coupling $J\rho_s$, in the sign convention of the paper (which uses -0.05).
- `U`: the ratio of potential scattering to spin scattering.
- `T`: the temperature in kelvin. `kB` (default in eV/K) converts it to the energy unit of H.
- `omega0`, `Gamma0`: the band cutoff and the lifetime broadening of the Kondo function F.

- `T0`: the tunneling strength. It only sets the overall scale.
- `order`: 2, 3, or `"LL"` (see Leading logs).

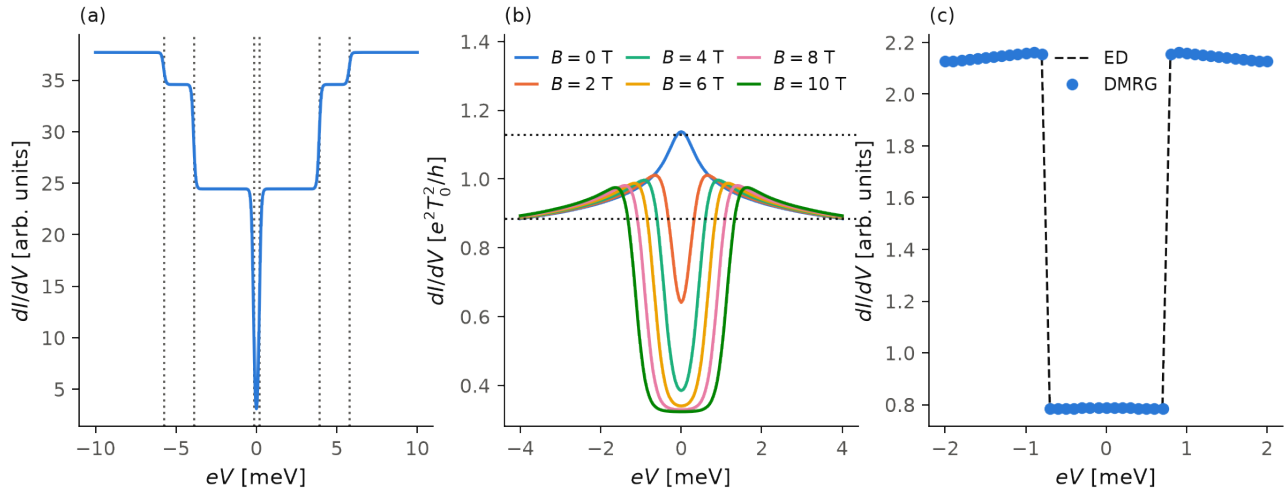


Figure 19: STM spectra of magnetic atoms and chains

Left: the second-order spectrum of an $S = 2$ atom with $H = D(S^z)^2 + E[(S^x)^2 - (S^y)^2]$, $D = -1.55$ meV and $E = 0.31$ meV (Fe on Cu_2N), at $T = 0.5$ K. The steps sit at the excitation energies of the spin multiplet, 0.18, 3.90 and 5.76 meV (dotted lines), which come from the exact 5×5 Hamiltonian. Middle: the third-order spectrum of an $S = 1/2$ with $J\rho_s = -0.05$, $U = 0$ and $T = 1$ K, divided by 2π to give units of $e^2T_0^2/h$. At $B = 0$ there is a Kondo peak at zero bias. Its height, 1.137, and its value at ± 4 mV, 0.885, match the paper's 1.13 and 0.886 (dotted lines). In a field the peak splits. The spectrum drops in the gap $|eV| < g\mu_B B$, and the Kondo term gives two peaks just outside the Zeeman steps. Right: the $T = 0$ spectrum of a 3-site chain from `mode="DMRG"` agrees with `mode="ED"` to 0.5 per cent (see ED and DMRG).

The code of this figure is in `docs/figures/kondo_dIdV/`.

15.3 Leading logs

We will now see how to go beyond third order. Each further order adds one more power of the same logarithm, $J\rho_s \ln(\omega_0/|eV - \epsilon|)$. Near zero bias and near the steps this number is not small, and the third order fails there. `order="LL"` sums these leading logarithms to all orders with the perturbative renormalization group. The couplings run as the band cutoff is lowered, and each logarithm stops at the energy of its own virtual process, so in a field or with anisotropy the couplings depend on the energy of the electron (Rosch, Paaske, Kroha, Wölfle, cond-mat/0408506; Rosch, Costi, Paaske, Wölfle, cond-mat/0301106). The renormalized tip vertex then goes into the second-order formula. To first order in $J\rho_s$ this is exactly `order=3`. At $T > 0$ the cutoff of each logarithm is averaged over the thermal window of the electrodes, with the same weight as the third order, so the first order is `order=3` at the same temperature, and the excited states of the sample are occupied as in `mode="ED"`. At $T > 0$ the memory grows with the number of occupied states, and `mem_gb=4` cuts the biases in groups so that each flow stays below about 4 GB. When H conserves S^z (no field, or a field along z) the flow at $T = 0$ uses this symmetry on its own from 32 states on, with the same result; `sz_blocks=False` turns this off.

```
sc = spinchain.Spin_Chain(["S=1/2"])
sc.set_hamiltonian(0*sc.Sz[0])
eV = np.linspace(-4e-3, 4e-3, 161) + 1e-5
eV, dIdV = sc.get_kondo_spectrum(eV, Jrho_s=-0.05, U=0.0, T=0.0, order="LL")
```

At the coupling of the paper the zero-bias peak is higher than at third order: $dI/dV/2\pi$ is 1.95 at $eV = 10$ μeV against 1.31, and 1.06 at 1 meV against 0.98. Far from zero bias the two agree, 0.92 against 0.88 at 4 meV. The method is still perturbative. It fails at the Kondo temperature, where the running coupling reaches order one, and there it returns NaN with a warning (for example at $J\rho_s = -0.2$ near zero bias). It also warns when

the running coupling passes 0.5, which can happen near a step while the zero-bias coupling is still small. It renormalizes the couplings and not the energies, so the steps stay at the bare excitation energies. The derivation and its checks are in `docs/new_methods/kondo_leading_log/` in the repository.

Backends.

`order="LL"` works with `mode="ED"` only, at any temperature. It runs on a GPU, with the same numbers, when the array backend is JAX: `from dmrgpy.pyitensor import backend` and `backend.set_backend("jax")` before the call. The memory grows as the square of the number of states, so a large impurity needs a card with much memory and full double precision: 13 spins 1/2 take about 110 GB. JAX gives a process only 75% of the card by default, so for such a run set `XLA_PYTHON_CLIENT_MEM_FRACTION=0.95` in the environment before Python starts. Above 2000 states the full diagonalization is refused unless `dmrgpy.algebra.algebra.maxsize` is raised first.

15.4 ED and DMRG

`mode="ED"` (the default) diagonalizes the full Hamiltonian, since every eigenstate enters as a virtual state. It works at any temperature, on small chains. `mode="DMRG"` works at $T = 0$ and never needs excited states (`submode="TASK"` below also takes the temperature of the electrodes). The second-order and potential terms come from the dynamical correlator, and the Kondo term from a real-time evolution of a three-point function. It runs on `itensor_version=3`, `"python"` and `"julia_live"`, on chains of at least 3 sites. Here the chain has 3 sites, with the field on the first one and a weak Heisenberg coupling. Without a field the ground state is degenerate, see `n_gs` below:

```
sc = spinchain.Spin_Chain(["S=1/2"]*3)
h = g*muB*B*sc.Sz[0]
for i in range(2):
    h = h + 0.01*(sc.Sx[i]*sc.Sx[i+1] + sc.Sy[i]*sc.Sy[i+1] + sc.Sz[i]*sc.Sz[i+1])
sc.set_hamiltonian(h)
eV = np.linspace(-2e-3, 2e-3, 41)
es = np.linspace(-3e-3, 3e-3, 800)
eV, dIdV = sc.get_kondo_spectrum(eV, Jrho_s=-0.05, U=0.0, T=0.0, order=3,
                omega0=2e-3, Gamma0=2e-3, mode="DMRG", submode="KPM",
                delta=2e-5, es=es,
                dt2=150.0, n_t2_half=20, dtau=150.0, n_tau_half=20)
```

These keywords have no default and must be chosen for the problem:

- `es`: the frequency grid of the dynamical correlator. For `order=2` it must cover the transitions below $\max|eV|$. With `U` not zero it must cover the whole spin excitation spectrum of the site, plus a few `delta`. The grid may be non-uniform. The grid must start several `delta` below 0 (the elastic line at zero energy is cut otherwise) and pass $\max|eV|$, and a warning says when it does not.
- `delta`, `submode`: the broadening and the method of the dynamical correlator.
- `dt2`, `n_t2_half`, `dtau`, `n_tau_half`: the two time grids of the Kondo term. They must resolve the time scales $1/\omega_0$ and $1/\Gamma_0$, so the default ω_0 and Γ_0 need very long grids. Too coarse a grid gives a wrong result without any error, so compare with `mode="ED"` on a small chain first.
- `n_gs`: the number of degenerate ground states. With a degenerate ground state, `mode="ED"` averages over the multiplet, while DMRG returns one member. `n_gs=g` makes DMRG average over `g` states too.

With `submode="TASK"`, the default on `itensor_version=3`, `"python"` and `"julia_live"`, there is no frequency or time grid at all. On `itensor_version=2` the default stays `"KPM"` with the two-time route above. The tangent-space Krylov method of Choosing a correlator method gives the ground state and the excited states that the spin of the site reaches, as discrete energies, and the exact sums of `mode="ED"` run over them:

```
eV, dIdV = sc.get_kondo_spectrum(eV, Jrho_s=-0.05, U=0.0, T=0.0, order=3,
                omega0=2e-3, Gamma0=2e-3, mode="DMRG", submode="TASK",
                nkry=100)
```

The result is exact when `maxm` holds the whole Hilbert space (with `sc.cutoff = 0`) and `nkry` is large enough; with a smaller `maxm` the excited states are one-site changes of the ground state, which moves the steps slightly.

It takes only `nkry` (default 100, increase until the spectrum stops changing) and `n_gs`.

At $T = 0$ every excited state is a sharp step of dI/dV. On a long chain the Krylov space resolves only the lowest excited states one by one, and the others are steps at energies that still move with `nkry`. A warning says how many, and how large the largest of these steps is. A larger `nkry` makes them fewer. On a long chain some always remain, so compare such curves only at a bias resolution above the spacing of the steps.

With `T>0` the steps and the Kondo peak are broadened by the temperature of the tip and of the substrate, as in an experiment, and the sample stays in its ground state:

```
eV, dIdV = sc.get_kondo_spectrum(eV, Jrho_s=-0.05, U=0.0, T=0.5, order=3,
                omega0=2e-3, Gamma0=2e-3, mode="DMRG", submode="TASK",
                nkry=100)
```

This is the thermal spectrum as long as k_BT is well below the lowest excitation of the sample. The call raises an error when the Boltzmann factor of the lowest excitation that the spin of the site reaches is above 10^{-3}; use `mode="ED"` then. An excitation that this spin does not reach is not checked. A degenerate ground state needs `n_gs`, as at $T = 0$. Once the thermal width $5.4\,k_BT$ is larger than the spacing of the steps, the curve no longer depends on the steps that are not converged, and a smaller `nkry` is enough, as long as the lowest excitation itself is converged.

See `examples/kondo/kondo_third_order_dmrg_VS_ED`, which is the right panel of the figure in Third order, and `examples/kondo/kondo_third_order_task_VS_ED`. The tip couples to one site only; several sites with independent couplings are not implemented.

15.5 Larger clusters at finite temperature

`mode="ED"` diagonalizes the whole Hamiltonian, so it stops at about 10 spins 1/2. At a temperature T only the states within a few k_BT of the ground state are occupied. An occupied state i reaches, through the spin of the tip site, only the states in the span of $S_a|i\rangle$ and of H applied to them. `submode="KRYLOV"` uses both facts. It finds the occupied states with a sparse eigensolver. For each one it builds a single Krylov space started from $S_x|i\rangle$, $S_y|i\rangle$ and $S_z|i\rangle$ together, and it runs the sums of `mode="ED"` over the states of that space. It works with any Hamiltonian, also when S_z is not conserved, for example in a field that turns from site to site:

```
n = 8
sc = spinchain.Spin_Chain(["S=1/2"]*n)
h = 0
for i in range(n-1):
    h = h + 1e-3*(sc.Sx[i]*sc.Sx[i+1] + sc.Sy[i]*sc.Sy[i+1] + sc.Sz[i]*sc.Sz[i+1])
for i in range(n):
    h = h + 2e-4*(np.cos(2*np.pi*i/3)*sc.Sx[i] + np.sin(2*np.pi*i/3)*sc.Sz[i])
sc.set_hamiltonian(h)
eV = np.linspace(-4e-3, 4e-3, 101) + 1e-5
eV, dIdV = sc.get_kondo_spectrum(eV, site=1, Jrho_s=-0.05, U=0.2, T=1.0,
                submode="KRYLOV", nkry=100)
```

The result is the one of `mode="ED"` once every Krylov space holds all the states that the spin reaches, apart from the occupied states that are dropped. On this chain the two differ by about 10^{-8} of the peak at 1 K.

- `nkry`: each Krylov space holds at most `3*nkry` states (default 100). Increase it until the spectrum stops changing.
- `p_cut`: occupied states whose Boltzmann weight is below `p_cut` times that of the ground state are dropped (default 10^{-8}), and dI/dV changes by about that much.
- `su2`: when H conserves the total spin, the default (`None`) takes one member of each spin multiplet with the weight $2S + 1$, which needs fewer occupied states, and otherwise every state. `True` raises when H does not conserve the total spin, and `False` always takes every state.
- `tol`: with a value such as 10^{-8}, a Krylov space stops before `3*nkry` states once every state that can be a final state (up to $\max|eV|$ above the initial state, plus $40k_BT$) is converged to `tol` times the scale of H. Useful with a large `nkry` and a narrow bias window. The default (`None`) never stops early.

The route is third order. `order="LL"` needs every eigenstate and raises with `submode="KRYLOV"`. See `examples/kondo/kondo_krylov_thermal_VS_ED`.

15.6 Hubbard chains

The same method works on the spinful fermion chains, `Spinful_Fermionic_Chain` and `Spinful_Fermionic_Chain_Native`. The tip then couples to the electron spin of the site, $\mathbf{S}_i = \frac{1}{2} c_i^\dagger \sigma c_i$. The spectrum includes the charge fluctuations that a spin model leaves out. At half filling and large U the Hubbard chain maps onto a Heisenberg chain with $J = 4t^2/U$, so the two spectra can be compared with the bias in units of J:

```
import numpy as np
from dmrgpy import fermionchain
n, t, U = 6, 1.0, 16.0
J = 4*t**2/U
fc = fermionchain.Spinful_Fermionic_Chain_Native(n)
h = 0
for i in range(n-1):
    h = h - t*(fc.Cdagup[i]*fc.Cup[i+1] + fc.Cdagdn[i]*fc.Cdn[i+1])
h = h + h.get_dagger()
for i in range(n):
    h = h + U*(fc.Nup[i]-0.5)*(fc.Ndn[i]-0.5)
fc.set_hamiltonian(h)
fc.set_conserved_sector(Nf=n)       # half filling, for mode="ED"
eV, dIdV = fc.get_kondo_spectrum(np.linspace(-3*J, 3*J, 240), Jrho_s=-0.05,
               T=0.0, kB=1.0, omega0=10*J, Gamma0=1e-2*J)
```

With `mode="ED"` a sector of fixed electron number `Nf` is exact at $T = 0$, since $\mathbf{S}_i$ does not change the number of electrons. At $T > 0$ it gives the canonical ensemble at that `Nf`. A sector of fixed `Sz` raises, since $S_\pm$ leaves it. `mode="DMRG"` with `submode="TASK"` works on both classes. Use `Spinful_Fermionic_Chain_Native`: there the site is one tensor, while on `Spinful_Fermionic_Chain` the spin of the site acts on two tensors, so a truncated `maxm` was less accurate in our tests. As U/t grows the Hubbard spectrum approaches the Heisenberg one. The difference falls as $(t/U)^2$ once U/t is a few tens; below that, higher orders in t/U still matter. The weight of the steps drops with the double occupancy, and the steps move with the corrections to J. See `examples/kondo/kondo_third_order_hubbard_VS_heisenberg`.

15.7 Orbital-resolved IETS of a magnetic atom

`dmrgpy.atom` describes a single atom with five d orbitals instead of a chain. `generate_atom` builds a `Spinful_Fermionic_Chain` whose sites are the orbitals, with a crystal field `tij`, spin-orbit coupling `soc`, the interactions `U` and `J`, a magnetic field `B` acting as $2\mathbf{S} + \mathbf{L}$, and `Ne` electrons. Two IETS signals are available:

```
from dmrgpy import atom
orbs = ["dxy", "dyz", "dz2", "dxz", "dx2y2"]
tij = np.diag([0.0, 0.05, 0.6, 0.05, 0.6])          # crystal field, eV
fc = atom.generate_atom(orbs=orbs, tij=tij, Ne=2, soc=0.03, U=4., J=0.5)
es = np.linspace(0., 0.1, 301)
x, y = atom.get_spinflip(fc, es=es, iorb=2, delta=1e-3)
x, y = atom.get_orbital_cotunneling(fc, es=es, iorb=2, delta=1e-3)
```

- `get_spinflip`: the spin-flip signal, $\sum_a S_{aa}(\omega)$ built from $S^a - \langle S^a \rangle$ on orbital `iorb`.
- `get_orbital_cotunneling`: the orbital cotunneling signal, from $c^\dagger_{\mathtt{iorb}\,\sigma} c_{j\sigma'}$ summed over the other orbitals j and both spins.

`iorb` is the orbital the tip couples to, for example d_{z^2} for a tip on top of the atom. With a crystal field, different orbitals give different weights and different peak positions. Both functions use ED. The low multiplet of an atom is often nearly degenerate, so the remarks of Degenerate ground states on `dex` and `T` apply here. See `examples/kondo/atom_iets_orbital_resolved`.

15.8 Kondo screening on a Wilson chain

We now ask a different question: is the spin of an impurity screened by the conduction electrons, and by which electrons? We follow the NRG-MPS method of Stocker and Zilberberg, arXiv:2501.10746. The Anderson impurity

$$H = \epsilon_d \sum_\sigma n_\sigma + U n_\uparrow n_\downarrow + t_D \sum_\sigma \left(d_\sigma^\dagger f_{0\sigma} + \text{h.c.} \right) + \sum_{n\sigma} t_n \left(f_{n\sigma}^\dagger f_{n+1\sigma} + \text{h.c.} \right)$$

is coupled to a Wilson chain: the band on $[-D, D]$ is cut at the energies $\pm D\Lambda^{-m}$, and the Lanczos tridiagonalization of the levels gives the hoppings $t_n \sim \Lambda^{-n/2}$. Site n resolves the energy $D\Lambda^{-n/2}$, so the chain must be long enough to reach below the Kondo temperature T_K. DMRG gives the ground state, and from it we compute

$$M_{\text{imp}} = 4\langle S^z_{\text{imp}} S^z_{\text{imp}} \rangle, \qquad M_n = 4\langle S^z_{\text{imp}} S^z_n \rangle, \qquad M_{\text{env}} = \sum_n M_n .$$

M_{imp} is the probability that the impurity is singly occupied, and M_n says where the screening cloud sits along the chain. A screened moment has M_{imp} close to 1 and $M_{\text{env}} = -M_{\text{imp}}$. We also project the ground state on a singly occupied impurity and take its Schmidt coefficients between impurity and bath: two equal values $1/\sqrt{2}$ for the Kondo singlet, one value 1 for a free, polarized moment.

```
from dmrgpy import wilsonchain
fc = wilsonchain.anderson_chain(13, ed=-0.5, U=1.0, tD=0.3, Lambda=3.0, B=0.0)
fc.maxm = 100
fc.nsweeps = 12
r = fc.get_kondo_screening()
print(r["M_imp"], r["M_env"], r["M_n"], r["schmidt"])
```

The bandwidth is $D = U$ and the density of states is flat and normalized, so $\Gamma = \pi t_D^2/(2D)$, and here $T_K \approx 0.017\,U$. The result is $M_{\text{imp}} = 0.83$, $M_{\text{env}} = -0.83$ and Schmidt coefficients $(0.707, 0.707)$: a Kondo singlet. In zero field $M_{\text{env}} = -M_{\text{imp}}$ holds exactly for any singlet, so the information is in M_{imp}, in the profile M_n, and in the response to a field. With `B` the larger Schmidt coefficient goes from $1/\sqrt{2}$ to 1 when B passes T_K. The field also polarizes the bath, and M_n is not a connected correlator, so in a strong field M_{env} turns positive.

With two levels, `ed` and `tD` are lists of two. Each level couples to its own Wilson chain, and U acts between all four spin-orbitals, so at $\epsilon_d = -U/2$ the impurity holds one electron. Then `M_imp` and `M_n` are given per level, `M_env[l][k]` is the screening of level `l` by chain `k`, and there are four Schmidt coefficients. Equal couplings give an SU(4) Kondo singlet, with four values 1/2. Unequal couplings split them in two pairs, and the splitting grows with the chain length, toward the SU(2) singlet of the level with the larger coupling.

15.8.1 The Kondo cloud along a real-space lead

How far from the impurity does the screening happen? A Wilson chain does not say it, since its sites are energy shells and not positions. We follow Holzner, McCulloch, Schollwöck, von Delft and Heidrich-Meisner, PRB 80, 205114 (2009), and put the dot at the end of a plain tight-binding lead of $L-1$ sites, L sites in total:

$$H = \epsilon_d \sum_\sigma n_{d\sigma} + B S^z_d + U n_{d\uparrow} n_{d\downarrow} - t \sum_{i\sigma} \left(c^\dagger_{i\sigma} c_{i+1\sigma} + \text{h.c.} \right) - \sqrt{2}\, t' \sum_\sigma \left(c^\dagger_{1\sigma} d_\sigma + \text{h.c.} \right), \qquad \Gamma = 2t'^2/t.$$

The cloud is read from the integrated spin correlation

$$\Sigma(x) = 1 + \sum_{i=1}^{x} \frac{\langle \vec{S}_d \cdot \vec{S}_i \rangle}{\langle \vec{S}_d \cdot \vec{S}_d \rangle},$$

meaning that $1 - \Sigma(x)$ is the part of the impurity spin screened by the first x lead sites. In a total-spin singlet $\langle \vec{S}^2_{\text{tot}} \rangle = 0$ gives $\sum_i \langle \vec{S}_d \cdot \vec{S}_i \rangle = -\langle \vec{S}^2_d \rangle$, so $\Sigma(L-1) = 0$ exactly. This needs L even at half filling; with L odd the ground state is a doublet. The screening length is the distance where a fraction a of the spin is screened,

$\xi_a(L) = \min\{x : \Sigma(x) \le 1 - a\}$, with $a = 0.9$ as the reference. It grows with L and saturates once the lead is longer than the cloud. The scale to compare with is $\xi_K^0 = \hbar v_F / T_K$, with $v_F = 2t$ and

$$T_K = \sqrt{U\Gamma}\, \exp\left(-\frac{\pi|\epsilon_d||\epsilon_d + U|}{2U\Gamma}\right).$$

The paper finds $\xi_{0.9} \approx 6.7\, t/T_K$ for $U/\Gamma \gtrsim 2$.

```
from dmrgpy import kondocloud
fc = kondocloud.anderson_lead(40, ed=-0.5, U=1.0, Gamma=0.5)
fc.maxm = 80
fc.nsweeps = 12
fc.cutoff = 1e-10
r = fc.get_kondo_cloud(a=[0.75, 0.9])
print(r["Sigma"], r["xi"], r["sum_rule"])
print(6.7/kondocloud.kondo_temperature(1.0, 0.5))
```

Here $U/\Gamma = 2$ and $T_K = 0.32$. The result is $\xi_{0.75} = 5$ and $\xi_{0.9} = 15$, the same at $L = 60$, and $\Sigma(L-1) = 4 \times 10^{-6}$, the truncation error. $\Sigma(x)$ drops in steps of two sites: the odd sites screen and the even sites add almost nothing, the even-odd effect of a half-filled lead, so ξ_a comes out odd. At $U/\Gamma = 3$ we get $\xi_{0.75} = 9$, again the same at $L = 40$ and 60, so $\xi_{0.75} T_K$ is 1.61 and 1.60 for the two couplings, meaning that the cloud scales as $1/T_K$. The paper's own numbers give 1.8. For $\xi_{0.9}$ at $U/\Gamma = 2$ we get $4.8/T_K$; $U/\Gamma = 2$ is at the edge of the Kondo regime, where the paper also sees deviations from $6.7/T_K$. At $U/\Gamma = 3$ and 4, $\xi_{0.9}$ still grows from $L = 40$ to $L = 60$: the cloud does not fit in the lead yet.

- `L`: the total number of sites, dot included. Take it even.
- `Gamma`, `U`, `ed`: the hybridization, the repulsion and the level, in units of `t`. The cloud grows as $1/T_K$, so it fits only for small U/Γ: on leads of 40 to 60 sites $\xi_{0.75}$ saturates up to $U/\Gamma = 3$.
- `half_filling=True`: fixes $N = L$ electrons, as the paper does. It is needed away from $\epsilon_d = -U/2$, where particle-hole symmetry no longer fixes the filling. On `itensor_version="python"` this sector converged here for leads up to 30 sites, not at 40.
- `B` and `connected=True`: a field on the dot. Then $S^z_{\rm tot}$ is the only conserved spin, and we use the connected form $(\langle S^z_d S^z_i\rangle - \langle S^z_d\rangle\langle S^z_i\rangle)/(\langle S^z_d S^z_d\rangle - \langle S^z_d\rangle^2)$, which has no sum rule. `r["mu"]` is the moment $\langle (S^z_d)^2\rangle - \langle S^z_d\rangle^2$.
- `maxm`: a metallic lead is entangled at every bond, so it needs a large bond dimension. At $L = 40$, `maxm=64` gave $\xi_{0.9} = 14$ and `maxm=80` and 100 gave 15. A too small bond dimension also breaks the singlet, and then $\Sigma(L-1)$ is not 0; a warning says so above `sum_rule_tol`.

The scaling collapses of the paper, Σ against x/ξ_K and $\xi_a(L)/\xi_a$ against L/ξ_a, are fits on these numbers. See `examples/kondo/kondo_cloud_real_space`.

15.8.2 The Wilson chain keywords

- `n`: the number of Wilson chain sites per level. It sets the lowest energy, $D\Lambda^{-n/2}$, which must be below T_K. Take n odd: with n even the ground state has an odd number of electrons, so it is degenerate and not a singlet, and one member of it looks unscreened.
- `Lambda`: the discretization. A larger value reaches low energies with fewer sites, at the price of a coarser band.
- `B`: a field BS^z on the impurity, and a rigid shift $\pm B$ of the bath bands.
- `dos`: a function giving the shape of the density of states, flat if not given.
- `maxm`: the Wilson chain ground state needs a bond dimension of 100 or more. Too small a `maxm` gives a state that breaks spin symmetry, which looks like an unscreened moment: $M_{\rm env} \neq -M_{\rm imp}$ and unequal Schmidt coefficients in zero field. Check against a larger `maxm`.

The diagnostics are expectation values, so `mode="ED"` gives the exact values on a short chain, and `get_kondo_screening(imp=..., bath=...)` works on any spinful chain. See `examples/kondo/kondo_screening_field` and `examples/kondo/kondo_screening_su4`.

15.9 Open Wilson chains

We now see how to keep the bath modes that a Wilson chain drops. We follow Bruognolo et al., arXiv:1611.05291. A chain of N sites stands for the bath at the temperature $T \sim \Lambda^{-N}$, and the modes below that scale are missing. They do not only broaden the levels, they also shift them, because the bath correlator at zero frequency,

$$\operatorname{Re} G_{\text{bath}}(0) = -\int d\omega \, \frac{A(\omega)}{\omega},$$

is not zero when the spectrum $A(\omega)$ is not symmetric. For a bosonic bath it is never symmetric. A chain that misses part of this shift gives a renormalized impurity energy that depends on N, so on T. This is what is called the mass-flow problem.

The open Wilson chain is a continued fraction of the bath correlator. At site k the self-energy is split by energy: the part inside the window $\Lambda^{-(k+1)}$ times the band becomes site $k+1$, the rest stays as the own bath F_k of site k, with the coupling t_k^F. The chain with all these baths is exact. The renormalized chain keeps only the static part of the baths it drops, as onsite shifts,

$$\tilde{\epsilon}_k = \epsilon_k + \delta\epsilon_k^F + \delta_{k,N-1}\,\delta\epsilon_{N-1}^S, \qquad \delta\epsilon_k^X = \operatorname{Re}\Sigma_k^X(0).$$

Chain C0 has no shifts, C1 the fast shifts only, C2 also the slow shift of the last site. C2 gives $\operatorname{Re} G_{\text{bath}}(0)$ exactly at any length, meaning that a static quantity does not drift with N.

```
from dmrgpy import wilsonchain
owc = wilsonchain.open_wilson_chain(20, Lambda=2.0, band=(0.0, 1.0),
                                    dos=lambda e: e**0.4)
for kind in ["C0", "C1", "C2"]:
    eps, t = owc.renormalized(10, kind=kind)
    print(kind, wilsonchain.chain_green(eps, t).real)

bc = wilsonchain.spin_boson_chain(6, alpha=0.1, s=0.4, Delta=0.1, eps=0.01,
                                  kind="C2", maxnb=8)
bc.maxm = 40
bc.nsweeps = 20
print(bc.vev(bc.x_imp).real)
```

For the sub-Ohmic bath $A \propto \omega^{0.4}$ the exact value is $-(s+1)/s = -3.5$. On 10 sites C0 gives -2.44, C1 gives -3.42, and C2 gives -3.5 to 1e-15. The arrays `owc.eps`, `owc.tS`, `owc.tF`, `owc.dF` and `owc.dS` are the chain, and `owc.bath_spectrum(k, omega, eta)` is the spectrum $\Gamma_k^F(\omega)$ of the own bath of site k.

The second part is the spin-boson model, $H = -\frac{\Delta}{2}\sigma_x + \frac{1}{2}\sigma_z\left(\epsilon + t_{\text{imp}}(b+b^\dagger)\right) + H_{\text{bath}}$, with $\Gamma(\omega) = 2\alpha\,\omega_c^{1-s}\omega^s$, on a C2 chain of 6 bosonic sites. The spin is site 0, a two-level site, and `bc.x_imp` is σ_z. Here $\langle\sigma_z\rangle = -0.90$ already at $\epsilon = 0.01$, a strong response of the spin to the bias. Its response to the bias, $\chi = d\langle\sigma_z\rangle/d\epsilon$ on a chain of length N, is the susceptibility at $T \sim \Lambda^{-N}$, as in the paper. `oscillator_chain` builds the dissipative harmonic oscillator in the same way. That model is quadratic, so its susceptibility $1/(\Omega - 2\alpha/s)$ is known exactly. The standard Wilson chain at $\Lambda = 2$, $\alpha = 0.1$, $s = 0.4$, $\Omega = 1$ gives 1.67 on 4 sites and 1.93 on 40 sites instead of 2, while C2 gives 2 at every length, and DMRG on C2 agrees with it to 1e-5.

- `n`: the number of chain sites. Site k has the energy Λ^{-k} times the band, on a bosonic and on a fermionic bath.
- `Lambda`: the factor by which the slow window shrinks per site.
- `band`, `dos`: the support and the shape of the bath spectrum. A bosonic bath has `band=(0, wc)`.
- `kind`: `"C0"`, `"C1"` or `"C2"`. Use C2 to remove the mass flow. `anderson_chain(..., kind="C2")` and `boson_bath_chain` take it too, and `"standard"` is the Wilson chain of the section above. In `anderson_chain` the factor per site is $\Lambda^{1/2}$, so `n` and `Lambda` mean the same there for every kind.
- `nsub`: levels per factor Λ in the fine discretization of the bath. The chain coefficients converge as $1/$`nsub`, about 1% at the default 8. The static identities hold at any value.

- `maxnb`: bosons per chain site. The C2 shift lowers the energy of the last site, so that site needs more bosons near a critical point.

What we neglect is the dynamics of the dropped baths: only their static shift is kept, so the chain has no dissipation. For a fermionic band with a finite density of states at the Fermi level the self-energies have an imaginary part at $\omega = 0$, and then C2 is an approximation, not an identity. For a symmetric band all the shifts vanish, and C0, C1 and C2 are the same chain. The finite-temperature NRG averages of the paper are not used: as in its variational MPS results, the temperature is the length of the chain. See `examples/kondo/open_wilson_chain_mass_flow`.

15.10 Spin excitations on a superconductor

When the spin is weakly coupled to the superconductor, for instance a molecule whose ligand separates the magnetic ion from the surface, we do not need the Wilson chain. The second-order spectrum of Second order is a golden rule between the eigenstates of the spin alone, and the gap only changes the density of states the electron tunnels into. The current is

$$I(V) = 2\pi T_0^2 \sum_{i,f} p_i |M_{fi}|^2 \left[C_{ts}(eV - \epsilon_{fi}) - C_{st}(eV + \epsilon_{fi}) \right], \qquad C_{ts}(x) = \int d\omega\, \rho_t(\omega) f(\omega)\, \rho_s(\omega{+}x) \left[1 - f(\omega + x)\right],$$

with C_{st} the same with f and $1-f$ exchanged, $\epsilon_{fi} = E_f - E_i$, and ρ_t, ρ_s the Dynes densities of states of tip and sample, $\rho(\omega) = \mathrm{Re}\left[(\omega + i\gamma)/\sqrt{(\omega + i\gamma)^2 - \Delta^2}\right]$. For $\rho = 1$ this is the step $\Theta(eV - \epsilon_{fi})$ of the normal metal. With a normal tip at $T = 0$, dI/dV of each transition is $\rho_s(eV - \epsilon_{fi})$, meaning that every step becomes a coherence peak at $eV = \Delta_s + \epsilon_{fi}$. With a superconducting tip the peaks move to $\pm(\Delta_s + \Delta_t + \epsilon_{fi})$, and at $T > 0$ thermal quasiparticles add peaks at $\pm|\Delta_s - \Delta_t| \pm \epsilon_{fi}$. This is how a superconducting tip gives inelastic spectroscopy a resolution better than the thermal width.

```
import numpy as np
from dmrgpy import spinchain
from dmrgpy.ysrtk import bcsperturbative
D, Delta = 0.7e-3, 1.35e-3
sc = spinchain.Spin_Chain(["S=5/2"])
sc.set_hamiltonian(D*sc.Sz[0]*sc.Sz[0])
eV = np.linspace(-6e-3, 6e-3, 2401)
eV, dIdV = bcsperturbative.get_spin_excitation_spectrum(sc, eV, T=1.2,
        U=1.15, Delta_s=Delta, Delta_t=Delta, gamma=1e-2*Delta)
```

This is the spin 5/2 of Fe-OEP-Cl on Pb measured with a Pb tip by Heinrich, Braun, Pascual and Franke (arXiv:1405.6984). The ground doublet is $S_z = \pm 1/2$ and the first excitation is $2D = 1.4$ meV. We find the elastic BCS peak at $2\Delta = 2.70$ meV and the spin excitation peak at $2\Delta + 2D = 4.10$ meV. Their height ratio, 0.44, is set by `U`, which we chose to match the ratio of about 0.4 in the experiment. At 4.8 K the state $\pm 3/2$ is thermally occupied, so the transition $\pm 3/2 \to \pm 5/2$ adds a peak at $2\Delta + 4D = 5.5$ meV, and the thermal quasiparticles of the two equal gaps give a small peak at zero bias. With two superconductors each peak is a Lorentzian of half-width $\gamma_s + \gamma_t$ and weight $\pi\sqrt{\Delta_s \Delta_t}/2$ per unit of the normal-state step, so γ sets its height but not its position.

The gap also protects an excited spin. The substrate relaxes it by scattering one electron, at the rate

$$R_{fi} = \pi (J\rho)^2 |M_{fi}|^2 \int d\omega\, f(\omega) \left[1 - f(\omega - \epsilon_{fi})\right] \left[\rho_s(\omega)\rho_s(\omega - \epsilon_{fi}) + F_s(\omega) F_s(\omega - \epsilon_{fi})\right],$$

with $F_s = \mathrm{Re}\left[\Delta/\sqrt{(\omega + i\gamma)^2 - \Delta^2}\right]$ the anomalous density of states. The $F_s F_s$ term is the coherence factor of the spin vertex. At $T = 0$ the rate is zero for $|\epsilon_{fi}| < 2\Delta_s$, meaning that an excitation below the pair breaking energy cannot decay into the substrate, and it starts continuously above $2\Delta_s$. `bcsperturbative.substrate_relaxation_rates(ks, Jrho_s, Delta_s=..., gamma_s=...)` returns this matrix of rates.

This is second order in the exchange with the substrate. It has no YSR state and no Kondo effect. The spin occupations are thermal, so the spin pumping by the current is not there, and the excited states have no lifetime width. It is valid when ρJ is small, which is the case of a spin separated from the surface by a ligand. The many-body route of A magnetic atom on a superconductor contains the YSR state and the Kondo effect.

- `Delta_s`, `Delta_t`: the gaps of sample and tip; 0 gives back the normal-metal second order.
- `gamma` (or `gamma_s`, `gamma_t`): the Dynes broadening. A gap needs $\gamma > 0$; it sets the height of the peaks.
- `T`: the temperature of the electrons and of the spin occupations.
- `U`: the direct tunneling ratio, as in Second order. It adds to the elastic BCS peak, which the spin also gives through its diagonal matrix elements.
- `de`: the step of the energy integral, by default a twentieth of the smallest of γ, $k_B T$ and $\Delta/20$.

15.11 A magnetic atom on a superconductor

We now put the spin on a superconductor with gap Δ. The substrate is a BCS band,

$$H_{\text{band}} = \sum_{k\sigma} \epsilon_k c^\dagger_{k\sigma} c_{k\sigma} + \Delta \sum_k \left(c^\dagger_{k\uparrow} c^\dagger_{-k\downarrow} + \text{h.c.} \right),$$

and the spin couples to it by $J\,\mathbf{S}\cdot\mathbf{s}_0$. The pairing is diagonal in energy and the same for both spins, so the Wilson map keeps it local: the BCS band is the Wilson chain of the section above with the pairing $\Delta(f^\dagger_{n\uparrow} f^\dagger_{n\downarrow} + \text{h.c.})$ on every site. Two scales compete. The Kondo effect screens the spin below T_K, while the gap removes the electrons that screen it. As a result the spin binds a Yu-Shiba-Rusinov (YSR) state inside the gap, and the ground state changes at $\Delta_c \approx 3.7\,T_K$ from a screened singlet to a free doublet. The two states have opposite fermion parity, so the energy of the YSR state is the difference of the lowest energies of the two parity sectors, $E_{\text{odd}} - E_{\text{even}}$. It is the position of the subgap peaks in dI/dV, which STM resolves on single adatoms and molecules.

```
import numpy as np
from dmrgpy import ysr
J, Lambda = 0.4, 4.0
fc = ysr.kondo_chain(14, J=J*ysr.lambda_factor(Lambda), Delta=0.003,
                     Lambda=Lambda)
fc.maxm = 64
fc.nsweeps = 20
r = ysr.subgap_energies(fc)
print(r["E_ysr"]/0.003, r["ground_state"])
```

This gives $E_{\text{odd}} - E_{\text{even}} = 0.56\,\Delta$ and a singlet. At $\Delta = 0.03$ it is $-0.23\,\Delta$, a doublet, so the transition is in between; `examples/kondo/ysr_state_on_superconductor` scans it. The ratio Δ_c/T_K depends on how T_K is read, and "Two Kondo temperatures" below compares the definitions. With both scales averaged over the twist z of the mesh, Wilson's T_K gives $\Delta_c/T_K = 3.94$ for every ρJ from 0.09 to 0.2. The numerical renormalization group gives 3.7, with T_K read from a crossing at $\Lambda = 2$, and read in the same way, with Δ_c from $\Lambda = 4$, we find 3.83. T_K from the local susceptibility of the impurity gives 4.4 at $\rho J = 0.1$ and 5.0 at $\rho J = 0.2$. At $\Lambda = 4$ the absolute Δ_c depends on where the mesh puts its levels: it is 29% above the value of the numerical renormalization group on the default mesh, and 0.7% below it on the mesh of "The mesh of the band" below. The ratio is the same on both meshes, so we compare ratios.

The STM spectrum comes from the composite tunneling operator of the Kondo section, now exact in J. The tip electron enters the substrate under the spin directly or with a spin flip, $X^\dagger_\sigma = \sum_{\sigma'} f^\dagger_{0\sigma'} \left[U\delta_{\sigma'\sigma} + (\sigma\cdot\mathbf{S})_{\sigma'\sigma} \right]/2$, and with a normal tip at $T = 0$

$$\frac{dI}{dV}(V) \propto \sum_\sigma \sum_n \left| \langle n | X^\dagger_\sigma | GS \rangle \right|^2 \delta(eV - E_n + E_0) + (\text{hole part at } -eV),$$

meaning that dI/dV is the spectral function of X. It shows the YSR peaks at $\pm|E_{\text{odd}} - E_{\text{even}}|$ with different heights at the two signs of the bias, and the coherence peaks above the gap.

```
es = np.linspace(-0.6, 0.6, 241)
fc = ysr.kondo_chain(4, J=0.6, Delta=0.1)
fc.maxm = 60
fc.nsweeps = 15
es, dIdV = ysr.get_stm_spectrum(fc, es, delta=0.01, U=0.3)
Vs, dIdV_sc = ysr.tip_convolution(es, dIdV, 0.1)
```

The chain of 4 sites is a small model that ED can check, not a converged substrate. Its YSR peak is at ± 0.095, and its strongest one is at negative bias. `tip_convolution` gives the spectrum with a superconducting tip of gap Δ_{tip}, where the YSR peaks move to $\pm(\Delta_{\mathrm{tip}} + |E_{\mathrm{odd}} - E_{\mathrm{even}}|)$, here ± 0.195.

- `J`: the exchange on the chain. On the default mesh the chain acts on the spin as a continuum with ρJ smaller by A_Λ, so pass `J*ysr.lambda_factor(Lambda)` to stand for J in the continuum. With `mesh="zp"` pass J itself; `fc.dos_factor` is the factor that applies. Here $\rho = 1/(2D)$, so $\rho J = J/2$ for $D = 1$.
- `mesh`: where each interval of the band keeps its level, `"mean"` (default) or `"zp"` (the representative energy of Zitko and Pruschke, flat band only). See "The mesh of the band" below.
- `Delta`: the gap. The chain must reach below it, the last hopping below $\Delta/10$ (`fc.reaches_gap`), or the continuum starts above Δ.
- `Lambda`, `n`: as for the Wilson chain. $\Lambda = 4$ resolves small energy differences better than $\Lambda = 2$ at the same bond dimension, because the chain is shorter.
- `U`: the ratio of direct to spin-flip tunneling, which sets the asymmetry of the YSR peaks.
- `delta`: the broadening of each peak. Above the gap the continuum is a set of discrete levels of the Wilson chain, so `delta` must be larger than their spacing near Δ; the section "The continuum above the gap" below gives a smooth continuum instead.
- `nrestart`: in `subgap_energies`, the number of DMRG solves per sector, of which the lowest is kept. A difference larger than the gap means one solve stopped in a metastable state, and a warning says so.

The energies are differences of two sector ground states of order Δ, so they need a converged `maxm`: on 20 sites at $\Lambda = 4$, `maxm=64` and `maxm=128` differ by 5e-7 and `maxm=128` and `maxm=200` by 2e-10. The peak widths are not computed: at $T = 0$ a YSR state is a discrete level, and the width measured in experiment comes from the relaxation of quasiparticles, which this model does not contain. See `examples/kondo/ysr_state_on_superconductor` and `examples/kondo/ysr_stm_spectrum`.

15.11.1 The mesh of the band

The Wilson chain keeps one level for each interval of the band. Where in the interval should the level sit? The usual choice is the mean energy. Then the impurity sees a density of states ρ/A_Λ at low energy, with

$$A_\Lambda = \frac{1 + \Lambda^{-1}}{2(1 - \Lambda^{-1})} \ln \Lambda,$$

which is 1.155 at $\Lambda = 4$. The levels of the top interval also pile up below the band edge. Zitko and Pruschke choose the energy $E(z)$ of each level so that the average over the twist z gives back ρ exactly, $\int_I \rho \, d\epsilon / |dE/dz| = \rho(E)$. For a flat band every interval below the top then sits at its logarithmic mean. The top interval $[D\Lambda^{-z}, D]$ sits at $E_1 = D[(1 - \Lambda^{-z})/\ln\Lambda + 1 - z]$, with $z = 1$ in this formula for the untwisted mesh. No A_Λ is needed, and the band edge is right.

```
import numpy as np
from dmrgpy import wilsonchain, ysr
from dmrgpy.ysrtk import zaverage
xi = np.array([0.01, 0.5, 0.7, 0.9])
w = np.sqrt(xi**2 + 1e-3**2)
for mesh in ["mean", "zp"]:
    A = 0*xi
    for z in zaverage.twists(16):
        eps, t = wilsonchain.wilson_chain(20, Lambda=4.0, z=z, mesh=mesh)
        e, wt = zaverage.bdg_poles(eps, t, 1e-3)
```

```
    A = A + zaverage.broaden_poles(w, e, wt, b=0.15, omega0=abs(t[-1]), gap=1e-3)[0]/16
print(mesh, A*xi/w*(ysr.lambda_factor(4.0) if mesh == "mean" else 1.0))
```

This prints the density of states per spin on the first chain site of a superconductor with $\Delta = 10^{-3}$, as a function of the normal-state energy $\xi = \sqrt{\omega^2 - \Delta^2}$. Both meshes give 0.5 in the bulk. At $0.7D$ the mean mesh gives 1.08 and the zp mesh 0.4998. At $0.9D$ the zp curve is 0.41, because the width b smears the step at D. The weight per side is 0.578 on the mean mesh after the factor, and 0.500 on the zp mesh.

On the zp mesh the continuum J goes on the chain as it is:

```
fc = ysr.kondo_chain(14, J=0.4, Delta=0.003, Lambda=4.0, mesh="zp")
fc.maxm = 64
fc.nsweeps = 20
r = ysr.subgap_energies(fc)
print(r["E_ysr"]/0.003, r["ground_state"])
```

This gives $0.51\,\Delta$ and a singlet. At $\Delta = 0.03$ it gives $-0.27\,\Delta$ and a doublet. At $\Lambda = 4$ the absolute scales depend on the mesh: on the zp mesh $\Delta_c = 4.26 \times 10^{-5}$ and Wilson's $T_K = 1.08 \times 10^{-5}$ at $\rho J = 0.1$, averaged over four twists, both about 23% below the mean mesh. The ratio Δ_c/T_K does not depend on the mesh: with Wilson's T_K it is 3.94 on the zp mesh and 3.95 on the mean mesh, and with the local one 4.40 and 4.41. So at $\Lambda = 4$ we compare ratios on either mesh. See `examples/kondo/wilson_mesh_band_edge`.

- `mesh`: `"mean"` (default) or `"zp"`, in `kondo_chain` as in `wilsonchain.wilson_chain`. With `"zp"` the continuum J goes on the chain bare, and the density of states is right up to the band edge.
- `z`: the twist of the mesh. On the zp mesh one twist moves Δ_c and T_K by up to 13%, so average over several.

15.11.2 Two Kondo temperatures

Let us now see how T_K is read, since at finite ρJ two susceptibilities give two different numbers. The local susceptibility is the response of the impurity spin to a field on the impurity alone, $\chi_{\rm loc} = -d\langle S^z_{\rm imp}\rangle/dB$. Wilson's impurity susceptibility is the response of the total moment to a uniform field, with the impurity minus without it,

$$T\chi_{\rm imp}(T) = \left[\langle (S^z_{\rm tot})^2\rangle - \langle S^z_{\rm tot}\rangle^2\right]_T - (\text{the same for the chain without the impurity}).$$

It goes from 1/4 for a free moment to $wT/(4T_K)$ for a screened one, with $w = 0.4128$, and both definitions use $T_K = w/(4\chi(0))$. Note that $\chi_{\rm imp}$ cannot be read from a ground state: $S^z_{\rm tot}$ commutes with H, so a uniform field moves every level linearly. It lives in the spectrum at temperatures of order the level spacing. So we compute it with a numerical renormalization group on the same Wilson chain. The chain is diagonalized site by site in blocks of charge and S^z, keeping the lowest states, and site N gives the temperature $T_N = \omega_N/\bar\beta$, with ω_N the last hopping.

```
from dmrgpy import nrgthermo, ysr
A = ysr.lambda_factor(4.0)
r = nrgthermo.kondo_nrg(40, J=0.2*A, Lambda=4.0, keep=400)
print(nrgthermo.kondo_temperature(r)["T_K"])
print(nrgthermo.kondo_temperature(r, definition="crossing")["T_K"])
rb = nrgthermo.kondo_nrg(40, J=0.2*A, Lambda=4.0, keep=400, B=1.2e-6)
print(nrgthermo.kondo_temperature(rb, definition="local")["T_K"])
```

At $\rho J = 0.1$ this gives $T_K = 1.339\times10^{-5}$ from the slope of $T\chi_{\rm imp}$, 1.494×10^{-5} from the crossing $T\chi_{\rm imp} = 0.0701$, and 1.199×10^{-5} from the local susceptibility, which is the value the DMRG energy in a field gives. The ratio of Wilson's T_K to the local one is 1.12 at $\rho J = 0.1$ and 1.27 at $\rho J = 0.2$. It goes to 1 linearly as $\rho J \to 0$, and it is the same for every twist and on both meshes. The slope does not depend on Λ, while the crossing does: it sits at $1.03\,T_K$ at $\Lambda = 2$ and at $1.11\,T_K$ at $\Lambda = 4$. The numerical renormalization group values compared with on this page read T_K from the crossing at $\Lambda = 2$, so compare with them at that Λ. `examples/kondo/kondo_temperature_wilson` prints the three values for two couplings.

- `n`, `Lambda`, `z`, `mesh`, `D`: the Wilson chain, as in `kondo_chain`. The chain must reach about $10^{-3}T_K$, so that several sites of each parity sit on the linear part. A longer chain does not change T_K.
- `J`: the coupling on the chain, as in `kondo_chain`. Use `J*ysr.lambda_factor(Lambda)` for a continuum J on the default mesh, and J bare with `mesh="zp"`.
- `B`: a field on the impurity only, needed for `definition="local"`. Take it well below T_K: $B = 0.1\,T_K$ moves $\chi_{\rm loc}$ by 4×10^{-4}. Use `B=0` for the other two definitions.
- `keep`: the number of states kept per site. 400 is enough at $\Lambda = 4$, and 800 at $\Lambda = 2$ with `beta_bar=1.5`. `kondo_temperature` warns when the run is truncated.
- `beta_bar`: sets $T_N = \omega_N/\bar{\beta}$. At $\Lambda = 4$, values from 0.5 to 1 give the same T_K to 1.5×10^{-3}. At $\Lambda = 2$, take 1.5.
- `definition`: in `kondo_temperature`. `"wilson"` (default) is the slope of $T\chi_{\rm imp}$ at low T, the same at every Λ. `"crossing"` is where $T\chi_{\rm imp} = 0.0701$, which depends on Λ. `"local"` is $w/(4\chi_{\rm loc})$.
- `window`: in `kondo_temperature`, the range of T_N that is fitted. By default it is everything below 0.03 times the crossing.

15.11.3 Higher spin and magnetic anisotropy

Many magnetic atoms carry a spin $S > 1/2$ and a uniaxial anisotropy,

$$H_{\rm imp} = J\,\mathbf{S}\cdot\mathbf{s}_0 + D_z(S^z)^2 + BS^z.$$

One channel of electrons screens half a unit of spin. So the screened state has spin $S-1/2$, and the free state over the BCS vacuum has spin S. The two states have opposite parity of the bath, so the energy of the YSR state is again a difference of the lowest energies of two parity sectors. The anisotropy splits each multiplet according to $|S^z|$. For $D_z < 0$ the free spin prefers $S^z = \pm S$, and for $D_z > 0$ it prefers the smallest $|S^z|$. As a result the lowest state of a parity can sit at any $|S^z|$ up to $S + 1/2$, and one YSR peak splits into several. Only states that differ from the ground state by $S^z = \pm 1/2$ give a peak in dI/dV.

```
from dmrgpy import ysr
J, Lambda, Delta = 0.4, 4.0, 0.025
fc = ysr.kondo_chain(10, J=J*ysr.lambda_factor(Lambda), Delta=Delta,
                     Lambda=Lambda, S="1", Dz=0.5*Delta)
fc.maxm = 32
fc.nsweeps = 10
r = ysr.subgap_energies(fc)
print(r["ground_state"], r["Sz_ground"])
print({s: (e - r["sectors"][0])/Delta for s, e in r["sectors"].items()})
```

This gives a free ground state with $S^z = 0$. The screened doublet ($2S^z = 1$) is $0.17\,\Delta$ above it, and the free $S^z = \pm 1$ states are $0.33\,\Delta$ above it. So dI/dV has one pair of YSR peaks, at $\pm 0.17\,\Delta$. `examples/kondo/ysr_high_spin_anisotropy` scans D_z for $S = 1$. The screened doublet is the ground state only near $D_z = 0$. For $D_z < 0$ the ground state is the free $S^z = \pm 1$ pair, and for $D_z > 0$ it is the free $S^z = 0$ state.

At the same J, and so the same T_K, the transition moves to a larger gap as S grows. We take 26 sites at $\Lambda = 4$ and $\rho J = 0.1$, and average Δ_c over two meshes, $z = 0$ and $z = 1/2$. This gives $\Delta_c(S)/\Delta_c(1/2) = 1.57$, 2.72 and 5.25 for $S = 1$, 3/2 and 2. The numerical renormalization group gives 1.59, 2.78 and 5.35, so the two agree within about 2%. On a single mesh the ratio moves by up to 5% with z, so average over z before you compare. The absolute Δ_c at $\Lambda = 4$ is about 25% too large for every S on the default mesh. The absolute scales at this Λ depend on the mesh, so compare ratios, not absolute values.

The bath also renormalizes the anisotropy. Near the transition, on the free side, the $S^z = \pm 1$ and $S^z = 0$ levels of $S = 1$ move with D_z at slopes $\langle (S^z)^2 \rangle = 0.86$ and 0.28. So they split by $0.58\,D_z$ and not by D_z. This means that a splitting read off STM gives a renormalized anisotropy, not the bare one.

- `S`: the impurity spin, `"1/2"` (default), `"1"`, `"3/2"`, `"2"`, or a number.
- `Dz`: the uniaxial anisotropy $D_z(S^z)^2$, in energy units.
- `B`: the field on the impurity, BS^z. With `B` set, the sectors of both signs of S^z are solved.

- `z`: the shift of the logarithmic mesh, in $[0, 1)$. Averaging over z removes most of the oscillation of Δ_c with the mesh.
- `sectors`: in `subgap_energies`, the values of $2S^z$ to solve. By default these are every value up to $2S+1$ when `Dz` or `B` is nonzero, and one per parity otherwise. `r["sectors"]` holds the energy of each one, and `r["Sz_ground"]` the $2S^z$ of the lowest. `r["E_free"]` and `r["E_screened"]` are the lowest energies of each parity, and `r["E_ysr"]` is `E_free - E_screened`, positive for a screened ground state.
- `observables`: in `subgap_energies`, operators measured in the lowest state of each sector, for instance $(S^z)^2$ built from `ysr.impurity_spin(fc)`.

For $S > 1/2$ the sector energies need a converged `maxm`. At $D_z = 0$ the members of one multiplet have equal energies, and their difference measures the error. On 26 sites at `maxm=128` and $z = 0$ it is about $10^{-4}\,\Delta$ for $S = 1$, and 2 to $4 \times 10^{-4}\,\Delta$ for $S = 3/2$ and 2. At $z = 1/2$ the spin 3/2 needs `maxm=192` to reach $2 \times 10^{-4}\,\Delta$; at `maxm=128` the difference is $10^{-2}\,\Delta$.

15.11.4 Scans over the gap and the anisotropy

A scan over Δ, or a bisection of Δ_c, solves the same chain many times. The states at two neighbouring gaps are close, so each point can start from the states of the point before. The Hamiltonian is linear in both parameters,

$$H(\Delta, D_z) = H_0 + \Delta \sum_n \left(f^\dagger_{n\uparrow} f^\dagger_{n\downarrow} + \text{h.c.} \right) + D_z \left(S^z_{\text{imp}} \right)^2 ,$$

so a scan changes one coefficient and keeps everything else. In a scan over D_z the charge penalty and the Hund coupling stay fixed. They shift the absolute energies by a constant and do not change any energy difference.

```
import numpy as np
from dmrgpy import ysr
from dmrgpy.ysrtk import subgapscan
J, Lambda, n = 0.4, 4.0, 14
Jchain = J*ysr.lambda_factor(Lambda)
Deltas = np.logspace(-3, -1, 9)
r = subgapscan.subgap_scan(n, Deltas=Deltas, J=Jchain, Lambda=Lambda,
                           maxm=64, nsweeps=20, nsweeps_warm=6)
t = subgapscan.find_transition(n, Jchain, Deltas[4], Deltas[5],
                               Lambda=Lambda, maxm=64, nsweeps=20,
                               nsweeps_warm=6)
print(r["E_ysr"]/Deltas, t["Delta_c"])
```

This gives the same $E_{\text{odd}} - E_{\text{even}}$ as `ysr.subgap_energies` at every point, for instance $0.54\,\Delta$ at $\Delta = 0.0032$ (singlet) and $-0.085\,\Delta$ at $\Delta = 0.018$ (doublet). The bisection puts the transition at $\Delta_c = 0.0137$ on this 14-site chain. The snippet uses 6 warm sweeps, fewer than the default, and on this chain they give the energies of the cold solves. `r` holds the keys of `subgap_energies` as arrays, one entry per gap. `t` holds every point visited, in `t["Delta"]` and in `t["x"]` $= E_{\text{ysr}}/\Delta$. `examples/kondo/ysr_transition_scan` plots both.

- `Deltas`, `Dzs`: the scanned parameter, exactly one of the two. A scan over `Dzs` needs `Delta`, and a scan over `Deltas` takes `Dz`.
- `nsweeps_warm`: the sweeps of every point after the first. Each point starts from the one before, in the order given, so the points should be monotonic and closely spaced. The default is the larger of 8 and a third of `nsweeps`, at most `nsweeps`. Only `cold_every` checks a smaller value. With `"auto"` each sector is swept in blocks of two sweeps until its energy moves by less than `warm_tol` between two blocks, and `r["nsweeps_warm"]` lists the sweeps used at every point.
- `warm_tol`: the stop of `nsweeps_warm="auto"`, relative to $|\Delta|$, 10^{-6} by default. The stop is a change between two blocks, not an error bound, so `cold_every` is still the check.
- `cold_every`: solve every k-th point once more from a fresh start with `subgap_energies`. `r["warm_cold"]` is the largest difference in the sector energies, and a warning names a point where the warm start stayed in a metastable or unconverged state.
- `mesh`: the Wilson mesh, `"mean"` (default) or `"zp"`, as in `kondo_chain`.

- `ratio`: `find_transition` halves the bracket in $\ln\Delta$ until $\Delta_{\rm hi}/\Delta_{\rm lo}$ is below `ratio` (default 1.04). It then interpolates $E_{\rm ysr}/\Delta$ linearly in $\ln\Delta$. `check_cold=True` also solves the two final ends cold.

15.11.5 The continuum above the gap

One Wilson chain has a few discrete levels for each factor Λ in energy. So above the gap the coherence peaks are a comb of poles, not a curve. We fix this as the numerical renormalization group does. We take several meshes, twisted by $z = k/n_z$, and solve one chain for each. Each pole is broadened with a log-Gaussian of width b, and the n_z spectra are averaged.

A pole at $|\omega| > |\Delta|$ is broadened in the normal-state energy $\xi = \sqrt{\omega^2 - \Delta^2}$ and mapped back with the factor $|\omega|/\xi$. This keeps all the weight outside the gap. It also gives the BCS shape $|\omega|/\sqrt{\omega^2 - \Delta^2}$ exactly from a smooth density in ξ. The poles inside the gap, the YSR states, are kept apart and get a small Lorentzian width.

```
import numpy as np
from dmrgpy import ysr
from dmrgpy.ysrtk import zaverage
J, Lambda, Delta = 0.4, 4.0, 0.01
es = Delta*np.linspace(-4, 4, 801)
es, dIdV, info = zaverage.zaverage_stm_spectrum(10, es,
        J=J*ysr.lambda_factor(Lambda), Delta=Delta, Lambda=Lambda,
        nz=4, b=0.4, U=0.3, delta_sub=0.03*Delta, maxm=48, nsweeps=30)
print([np.unique(np.round(s["omega"]/Delta, 4)) for s in info["subgap"]])
```

A chain of 10 sites is enough for the YSR energy. For the continuum near the gap edge the chain must reach below Δ, and `info["reaches_gap"]` says whether it does; here 12 sites are needed. On a shorter chain the lowest levels are strongly coupled to the spin, and the YSR state can sit above Δ.

Without the spin ($J = 0$) the average of 4 meshes at $b = 0.4$ gives the BCS density of states to 10^{-5}. With the spin, the YSR state takes weight from the gap edge. There the density goes to zero as $\sqrt{\omega - \Delta}$, and the coherence peak becomes a broad maximum at about 2Δ. The energy of the YSR state changes by about 10% from one mesh to the other on a chain of 20 sites, so average it over z as well.

- `nz`: number of twisted meshes. Each one is a full DMRG solve.
- `b`: width of the log-Gaussian. Use $b = 0.4$ with `nz=4` for the continuum. Within a few percent of Δ above a bound state, use $b = 0.15$ with `nz=16`, because there the broadened curve is high by $e^{3b^2/2}$.
- `nkry`: size of the Krylov space of each spectrum. It sets how close to the gap edge the levels are resolved. With `nkry=300` the continuum is converged from about $1.3\,\Delta$ up.
- `delta_sub`: width given to the subgap peaks. They are discrete levels at $T = 0$.
- `info["subgap"]`: the subgap poles of each mesh, with their electron and hole weights.

The spectrum is multiplied by `fc.dos_factor`, A_Λ on the default mesh and 1 on `mesh="zp"`, so it stands for the continuum density of states ρ, the same convention as for `J`. On the default mesh the spectrum within a factor Λ of the band edge D is not the continuum one.

15.11.6 The width and height of a YSR peak

The many-body spectrum gives a YSR state as a sharp level. In experiment the peak has a width, and its height does not grow linearly with the tunneling current. The reason is that the tip changes the occupation of the YSR state. An electron from the tip fills the state, and the state must relax to the quasiparticle continuum before the next electron can enter. We now take this into account with the rate model of Ruby, Pientka, Peng, von Oppen, Heinrich and Franke (arXiv:1502.05048).

The YSR state is one level at energy ϵ, with electron weight u^2 and hole weight v^2 at the tip. The tip is a superconductor with gap Δ_t and density of states $\rho(\omega)$, in units of its normal value. The tip fills the level with electrons at the rate $\Gamma_e(\omega) = g\,u^2\rho(\omega - eV)$ and with holes at the rate $\Gamma_h(\omega) = g\,v^2\rho(\omega + eV)$, where $g = 2\pi t^2\nu_t$ is the tip coupling. The level is emptied by relaxation at the rate Γ_1 and filled by a thermal quasiparticle at the rate Γ_2. With $\Gamma = \Gamma_e + \Gamma_h + \Gamma_1 + \Gamma_2$ and the tip occupation n,

$$I_s = \frac{e}{h}\int d\omega\, \frac{\Gamma_1\left[\Gamma_e n(\omega_-) - \Gamma_h n(\omega_+)\right] - \Gamma_2\left[\Gamma_e(1-n(\omega_-)) - \Gamma_h(1-n(\omega_+))\right]}{(\omega-\epsilon)^2 + \Gamma^2/4}, \qquad I_a = \frac{2e}{h}\int d\omega\, \frac{\Gamma_e\Gamma_h\left[n(\omega_-) - n(\omega_+)\right]}{(\omega-\epsilon)^2 + \Gamma^2/4},$$

with $\omega_\pm = \omega \pm eV$. I_s is single-electron tunneling, and it needs relaxation. I_a is resonant Andreev reflection: an electron enters and a hole leaves, so a Cooper pair goes into the condensate. The main peaks are at $eV = \pm(\Delta_t + \epsilon)$, and the thermal peaks at $eV = \pm(\Delta_t - \epsilon)$.

At weak tunneling the peak heights are linear in the normal-state conductance G_N, and their ratio is u^2/v^2. In this limit the peak is the BCS density of the tip convolved with a Lorentzian of full width Γ_1. At strong tunneling the tip empties the level faster than relaxation does, so the single-electron peak saturates and Andreev reflection takes over. Andreev sets in first at the peak with the smaller weight, so the asymmetry inverts. The ratio approaches $(v/u)^{10/3}$ only when Γ_1 is small. At stronger tunneling the hole rate also broadens the level, and the ratio decreases again. The thermal peaks are smaller than the main ones by $\Gamma_2/\Gamma_1 = e^{-\epsilon/T}$, and they go through a maximum as G_N grows.

```
import numpy as np
from dmrgpy import ysr
from dmrgpy.ysrtk import zaverage, ysrcurrent
Lambda, Delta = 4.0, 0.1
fc = ysr.kondo_chain(8, J=0.8*ysr.lambda_factor(Lambda), Delta=Delta,
                     Lambda=Lambda)
fc.maxm = 60
fc.nsweeps = 15
poles = zaverage.stm_poles(fc, U=0.3, nkry=100)
p = ysrcurrent.rate_parameters(poles, Delta, Lambda=Lambda)
g = ysrcurrent.gamma_tip_from_GN(1e-6)
Vs = np.linspace(-1.6, 1.6, 641)*Delta
dIs, dIa = ysrcurrent.didv(Vs, p["eps"], p["u2"], p["v2"], Delta, g,
                           1e-4, 2.5e-5)
h = ysrcurrent.peak_heights(p["eps"], p["u2"], p["v2"], Delta, g, 1e-4, 2.5e-5)
```

Eight sites is the shortest chain at $\Lambda = 4$ that reaches below the gap. It gives $\epsilon = 0.46\,\Delta$ and $u^2/v^2 = 1.55$. The main peaks are at $\pm 1.46\,\Delta$, with heights 0.034 and 0.023 in units of $2e^2/h$. The thermal peaks are at $\pm 0.54\,\Delta$. `examples/kondo/ysr_peak_width` scans G_N. The ratio of the main peaks is 1.55 at $G_N = 10^{-8}$ and 0.78 at $G_N = 0.1$.

The weights u^2 and v^2 are those of the operator the tip couples to. For the bare electron they are the residues of the YSR pole in the local density of states. For the composite vertex X they are divided by its total weight per tip spin, $\sum_\sigma \langle\{X_\sigma, X_\sigma^\dagger\}\rangle/2$, so that they are per unit of G_N. The ratio u^2/v^2 needs no normalization. The rate model treats the YSR state as one level. For the doublet of a spin 1/2 the weak-tunneling heights are right, while the crossover conductances are right only up to a factor of order 2.

Every tunneling vertex adds or removes one electron, so it changes the fermion parity. On a ground state of definite parity there is no elastic weight, $\langle X\rangle = 0$. Near the singlet-doublet transition the two parities are almost degenerate, and a DMRG solve from a random start can stop at a mixture $a|\text{even}\rangle + b|\text{odd}\rangle$. Then dI/dV has a false peak at zero bias, and the YSR pole splits and moves. With `parity=True` the ground state is solved inside its parity sector instead, the lowest of the $2S_z$ sectors of `ysr.subgap_energies`, and the correlator starts from it.

```
Lambda, Delta = 4.0, 0.1
fc = ysr.kondo_chain(8, J=0.6*ysr.lambda_factor(Lambda), Delta=Delta,
                     Lambda=Lambda)
fc.maxm = 60
fc.nsweeps = 15
poles = zaverage.stm_poles(fc, U=0.3, nkry=100, parity=True)
p = ysrcurrent.rate_parameters(poles, Delta, Lambda=Lambda)
```

This chain is close to the transition, with $\epsilon = 0.0190\,\Delta$. In the parity sector the elastic weight is zero, there is one YSR pole on each bias side, and $u^2 = 0.334$, $v^2 = 0.196$. By default, `parity="auto"`, the plain state is solved first and probed with two expectation values per vertex, the elastic weight $w_{\mathrm{el}} = \sum_X 2|\langle X\rangle|^2$ and the sum rule $w_{\mathrm{tot}} = \sum_X \langle XX^\dagger + X^\dagger X\rangle$. The sector solve is done only when $w_{\mathrm{el}} > 10^{-6}\, w_{\mathrm{tot}}$, so a chain far from the transition keeps its plain result. Without a gap there is no parity sector, and the plain result is kept. For `ysr.get_stm_spectrum`, take the state from `paritystate.parity_ground_state(fc)`, which returns it on a copy of the chain:

```
from dmrgpy.ysrtk import paritystate
es = np.linspace(-0.004, 0.004, 401)
gs, info = paritystate.parity_ground_state(fc)
es, dIdV = ysr.get_stm_spectrum(gs, es, delta=1e-4, U=0.3)
```

TASK works in the tangent space of the state, and a state solved inside one sector can have smaller bonds than the mixture a plain solve gives. On a chain of a few sites this space is then too small, and part of the spectrum is missed. The same happens for the product state at $J = 0$. With `pad_bonds="full"` every bond is padded with empty directions to its largest size before the recursion. The state does not change, the tangent space becomes the whole Hilbert space, and the spectrum is the exact one. For a spin 1 with $D_z < 0$, whose free state is a pair $\pm S_z$:

```
fs = ysr.kondo_chain(4, J=0.6, Delta=0.1, S="1", Dz=-0.05)
fs.maxm, fs.nsweeps = 64, 15
gs, info = paritystate.parity_ground_state(fs)
es = np.linspace(-1.2, 1.2, 241)
es, A = ysr.get_stm_spectrum(gs, es, delta=0.08, U=0.3, pad_bonds="full")
```

Against exact diagonalization this spectrum is off by 1.1×10^{-3} of the peak without the padding, and by 10^{-8} with it. `examples/kondo/ysr_task_tangent_space` plots both.

The padding adds directions that are not chosen from the state, and the whole Hilbert space is reachable only on a short chain. `expand_bonds=k` enlarges the space only where it is needed. On every bond it takes the two-site part of $X|\mathrm{GS}\rangle$ and of $HX|\mathrm{GS}\rangle$ that the tangent space drops, and adds its leading singular vectors, at most k of them, as new kept directions. The state does not change.

```
es, A = ysr.get_stm_spectrum(gs, es, delta=0.08, U=0.3, expand_bonds=4)
```

On the same pair this spectrum is the exact one to 1.5×10^{-11} of the peak, in a tangent space much smaller than the padded one. At $J = 0$ add `expand_rounds=2`: the second round starts from the expanded state and reaches the directions where the flipped impurity spin has moved into the bath. The expansion does not make the Krylov recursion converge faster. On a long chain a pole that is not converged needs a larger `nkry`, not `expand_bonds`. `examples/kondo/ysr_task_expansion` plots the spectrum with and without it.

`rate_parameters` warns when the poles show a mixture, and `E_ysr=` compares ϵ with `ysr.subgap_energies`.

- `gamma_tip`: the tip coupling $g = 2\pi t^2 \nu_t$. `gamma_tip_from_GN(GN)` gives it from the normal-state conductance in units of $2e^2/h$. `exact=False` uses the weak-coupling form $G_N = 2\pi\nu_s g$, the default uses the paper's form, and `exact="landauer"` uses the one-channel form. They agree for $G_N \ll 1$, where the model holds.
- `Gamma1`, `Gamma2`: the rates that empty and fill the YSR state. They set the width of the peak at weak tunneling. With `Gamma2=None`, $\Gamma_2 = \Gamma_1 e^{-\epsilon/T}$. For a degenerate excitation use `detailed_balance(Gamma1, eps, T, g_ratio)`.
- `T`: the temperature of the tip and of the relaxation. It gives the thermal peaks.
- `dynes`: the Dynes broadening of the tip. Zero is a sharp BCS tip.
- `Delta_t`: the gap of the tip. Zero is a normal tip, where the Andreev peak is symmetric in bias.
- `Lambda`: in `rate_parameters`, multiplies the weights by A_Λ, so that they stand for the continuum.
- `E_ysr`: in `rate_parameters`, the YSR energy from `ysr.subgap_energies`. It raises when the subgap pole is not at that energy.
- `peak_heights`: the four peaks, each with its single-particle part `"s"` and its Andreev part `"a"`.
- `parity`: in `zaverage.stm_poles` and `zaverage_stm_spectrum`. `"auto"` (default) starts from the ground state solved inside its fermion-parity sector only when the plain state is a parity mixture, `True` always does,

and `False` always takes the plain state. `paritystate.parity_ground_state(fc)` gives the same state for `ysr.get_stm_spectrum`.

- `pad_bonds`: in `zaverage.stm_poles`, `zaverage_stm_spectrum` and `ysr.get_stm_spectrum`, pads every bond of the state before the TASK recursion. `"full"` gives the whole Hilbert space, for a short chain with `parity=True` or at $J = 0$. An integer below the largest bond adds arbitrary directions: on a long chain it does not make the spectrum better, and the pole variances stop being an error bar, so leave it at None (default) there. A padded space above 4×10^6 raises an error.
- `expand_bonds`: in the same three functions, None (default) or an integer k, the largest number of new directions per bond, chosen from the part of $X|\mathrm{GS}\rangle$ and $HX|\mathrm{GS}\rangle$ that the tangent space drops. Use it for a sector state of a degenerate pair, and with `expand_rounds=2` at $J = 0$. It cannot be combined with `pad_bonds`.
- `expand_rounds`: repeats the expansion from the expanded state, 1 by default. A round that adds nothing ends it.

15.12 Several magnetic atoms: the surrogate bath

We now put two or more spins on the superconductor. A Wilson chain per spin is too long for a chain of atoms, so each spin gets a small bath instead, the surrogate model of Baran, Frost and Paaske (arXiv:2307.11646). Seen from the atom, a flat BCS band of half-width D enters only through the function

$$g(\omega) = \frac{1}{\pi} \int_{-D}^{D} \frac{d\xi}{\xi^2 + \Delta^2 + \omega^2},$$

on the Matsubara axis. A few BCS levels with the same gap reproduce it,

$$\tilde{g}(\omega) = \sum_{l=1}^{L} \frac{\gamma_l}{\xi_l^2 + \Delta^2 + \omega^2},$$

with levels in pairs $\pm\xi_l$ and one level at $\xi = 0$ when L is odd. γ_l and ξ_l are fitted on ω from $10^{-3}\Delta$ to D. The levels share one Δ, so they form a short chain of L sites with the pairing Δ on every site, and the spin couples to its first site. The spin sees the same low-energy Green function as in the continuum, so the classical YSR energy is right to a few percent of Δ for any D/Δ.

The atoms talk to each other in two ways: a hopping t_b between the first sites of neighbouring baths, which stands for the substrate between them, and a direct exchange $J_{12}\,\mathbf{S}_i \cdot \mathbf{S}_{i+1}$. In a real metal the first decays as $\sin(k_F r)/(k_F r)\, e^{-r/\xi_0}$; here it is a parameter.

```
import numpy as np
from dmrgpy.ysrtk import surrogate
Delta = 0.1
fc = surrogate.surrogate_kondo_chain(2, J=0.9, Delta=Delta, L=3, J12=0.2)
r = surrogate.sector_energies(fc, mode="ED", sectors=(0, 1, 2), nstates=4)
print({s: (e - r["E0"])/Delta for s, e in r["sectors"].items()})
```

Each spin carries one electron, so the sectors of total S_z are sectors of the bath parity: with N spins, $2S_z = N + N_{\mathrm{bath}} \bmod 2$. For the dimer, $2S_z = 0$ holds the singlets (and one member of each triplet), $2S_z = 1$ the odd states, the YSR excitations, and $2S_z = 2$ the triplets. Here both spins are screened, the two odd states sit at $0.37\,\Delta$, and a second singlet, the singlet of the two atomic spins, is at $0.11\,\Delta$. As J_{12} grows this singlet becomes the ground state, between $J_{12} = 2\,\Delta$ and $3\,\Delta$, and the odd states move to the gap edge: two spins bound in a singlet leave no state inside the gap, which is what STM sees on close antiferromagnetic dimers.

The hopping t_b splits the two odd states into a bonding and an antibonding YSR state, and the subgap peak of dI/dV splits in two. The splitting is linear in t_b and vanishes at particle-hole symmetry, $V = 0$ (in the zero-bandwidth model it is $t_b\,|u^2 - v^2|$, with u and v the electron and hole amplitudes of the YSR state on the coupled site). So it needs a potential scattering `V` at the atom.

```
fc = surrogate.surrogate_kondo_chain(2, J=2.0, V=1.0, Delta=Delta, L=2,
                                     t_b=0.1)
fc.maxm = 64
fc.nsweeps = 20
es = np.linspace(-1.2*Delta, 1.2*Delta, 241)
es, dIdV = surrogate.get_stm_spectrum(fc, es, spin=0, delta=0.01*Delta, U=0.3)
```

This gives the YSR peaks at $\pm 0.57\,\Delta$ and $\pm 0.75\,\Delta$, against $\pm 0.67\,\Delta$ at $t_b = 0$. On a chain of N atoms the N YSR states form a band, like the modes of an open chain of N sites. Every state shows at every atom, with the weight of that mode on the atom: on four atoms the lowest state is about 2.6 times weaker over an end atom than over a middle one, and the second state is 2.6 times stronger.

Near a crossing of the two parities a plain solve can mix them, as for one atom. Then pass `parity=True`:

```
fc = surrogate.surrogate_kondo_chain(2, J=0.671, Delta=Delta, L=3)
fc.maxm = 64
fc.nsweeps = 20
es = np.linspace(-0.2, 0.2, 41)
es, dIdV = surrogate.get_stm_spectrum(fc, es, spin=0, delta=0.005, U=0.3,
                                      parity=True)
```

Here the lowest odd state is $0.017\,\Delta$ above the ground state, and the spectrum is the exact one.

- `N`: the number of spins, each 1/2.
- `J`, `V`: the exchange and the potential scattering in the continuum, with $\rho = 1/(2D)$. The bath gets JW and VW, where W is the weight of the fitted levels.
- `L`: levels per bath (2 by default). Odd L has a level at zero energy and the gap edge exactly at Δ; even L puts the gap edge above Δ.
- `D`: the half-width of the band.
- `t_b`, `J12`: the coupling of neighbouring atoms through the substrate and directly.
- `B`: a field on the spins. With a field, `sector_energies` solves every $2S_z$ from $-(N+1)$ to $N+1$.
- `spin`: in `get_stm_spectrum`, the atom under the tip.
- `parity`: in `get_stm_spectrum`, start from the ground state solved inside its parity sector (the sectors of `sector_energies`). Use it when a state of the other parity is close to the ground state.
- `nkry`: in `get_stm_spectrum`, the size of the Krylov space. More atoms need more: 100 is enough for four atoms, eight atoms need 300. A peak at a different $|eV|$ on the two bias sides means it is too small.
- `nstates`: in `sector_energies`, the number of states per sector.

A few levels have no room for the Kondo logarithm between Δ and D. With $D \gg \Delta$ the surrogate misses the screening: at $\rho J = 0.15$ it gives a free spin at every Δ, where the Wilson chain of the section above has the transition at $\Delta_c = 2 \times 10^{-3} D$. With $\Delta \geq 0.1\,D$ the YSR energy at $L = 4$ agrees with the Wilson chain to $0.06\,\Delta$, and to $0.01\,\Delta$ when Δ is above $D/2$. So use the surrogate for Δ not much smaller than D, or when the exchange is a parameter to be fitted to the measured YSR energy rather than derived from the band. Excited states inside a sector need `itensor_version=3` or `mode="ED"`.

The example is `examples/kondo/ysr_dimer_surrogate`.

15.13 A spin chain on a metal

We now take a chain of spins 1/2 on a normal metal, and we put the tip on one of its spins. The spin under the tip is Kondo coupled to the metal, a Wilson chain. The other spins see the metal only through the chain. The Hamiltonian is

$$H = J_s \sum_i \mathbf{S}_i \cdot \mathbf{S}_{i+1} + J_K\, \mathbf{S}_j \cdot \mathbf{s}_0 + B \sum_i S_i^z + \sum_m \epsilon_m f_m^\dagger f_m + \sum_m t_m \left(f_m^\dagger f_{m+1} + \text{h.c.} \right),$$

with j the spin under the tip, $\mathbf{s}_0$ the spin of the electrons on the first Wilson site, and f_m the Wilson chain of a flat band of half-width D. The spin j can be any spin of the chain, at the end or in the middle. The STM

spectrum uses the composite tunneling vertex of A magnetic atom on a superconductor, with the tip on the spin j and the electron entering at the first Wilson site.

```
import numpy as np
from dmrgpy import ysr
fc = ysr.spin_chain_on_metal(3, 5, site=1, Js=0.3, JK=0.4) # 3 spins, metal under the middle spin
fc.maxm, fc.nsweeps = 40, 12
es = np.linspace(-1.0, 1.0, 201)
(e, didv) = ysr.get_stm_spectrum(fc, es, delta=0.03) # tip on the middle spin
print(didv[100]) # 0.080 at zero bias, and 0.510 with site=2
```

The spectrum depends on the spin under the tip. In the ground state of three spins the two end spins are parallel to the total spin, and the middle spin is antiparallel to it. So the exchange with the metal is antiferromagnetic for an end spin and ferromagnetic for the middle spin. With the metal on an end spin the ground state is a singlet, meaning that the metal screens the chain. With the metal on the middle spin it is a triplet. The end spin has peaks close to zero bias, and the middle spin does not. On a finite Wilson chain the spectrum is a set of discrete lines, and the last hopping of the chain, `fc.t_last`, is its energy resolution.

- `ns`, `n`: the number of spins and of Wilson sites. With `ns+n` odd the ground state is a doublet.
- `site`: the spin coupled to the metal, where the tip is. The default is the last spin.
- `Js`, `JK`: the Heisenberg and the Kondo exchange, antiferromagnetic when positive. On the default mesh, `JK*ysr.lambda_factor(Lambda)` stands for the continuum J_K.
- `B`: a field on the spins only.
- `Lambda`, `D`, `mesh`, `z`: the Wilson chain, as in `kondo_chain`.
- `nkry` (in `get_stm_spectrum`): raise it for sharp lines on a long chain.

The example is `examples/kondo/spin_chain_on_metal_site_resolved`.

15.13.1 Several spins on the metal

When several spins of the chain touch the metal, each of them is Kondo coupled to the first site of its own Wilson chain,

$$H = J_s \sum_i \mathbf{S}_i \cdot \mathbf{S}_{i+1} + \sum_j J_K \, \mathbf{S}_j \cdot \mathbf{s}_0^{(j)} + \sum_j H_{\text{metal}}^{(j)},$$

with j running over the spins on the metal. The metals of different spins are independent, which is the case of spins far apart on the scale of the Fermi wavelength. The tip on spin j puts its electron into the metal of that spin. For a dimer with J_s above the Kondo scale the ground state is a singlet, and the tip excites the triplet: dI/dV has two steps at $eV = \pm J_s$, which is how the exchange of a spin dimer is measured with STM.

```
fc = ysr.spin_chain_on_metal(2, 5, site=[0, 1], Js=0.4, JK=0.4) # each spin on its own metal
fc.maxm, fc.nsweeps = 64, 16
es = np.linspace(-1.0, 1.0, 201)
(e, didv) = ysr.get_stm_spectrum(fc, es, delta=0.03, imp=fc.impurities[0],
                                 expand_bonds=8, expand_rounds=3) # tip on spin 0
print(didv[100]) # 0.135 at zero bias, and 0.579 with Js=0
```

- `site`: the list of the spins on the metal.
- `JK`: a number, or one value per spin of `site`.
- `order`: where the Wilson chains go in the MPS. `"sides"` (default) puts them on the two ends of the chain and gives the best energies, `"comb"` each one after its spin, `"linear"` all after the spins, in the order of `site`. The order changes the bond dimension a result needs, not the result.
- `imp` (in `get_stm_spectrum`): the site of the spin under the tip, one of `fc.impurities`; `fc.spin_sites[i]` is the site of spin `i`.
- The STM spectrum by TASK needs the tip spin first in `site` and `order="linear"`: each metal keeps its own charge, and in any other layout the tangent space of TASK misses most of the spectrum, so `get_stm_spectrum` raises there. For another tip, build the chain again with that spin first.

- `expand_bonds`, `expand_rounds` (in `get_stm_spectrum`): the other route, in any order: `expand_bonds=8, expand_rounds=3` adds most of the missing directions, and the call raises when more than a tenth of the weight is still missing. Check the result against a larger value or against `submode="CVM"` with `fc.cvm_solver="variational"`.

The `submode="NRG"` route takes one metal only. The example is `examples/kondo/spin_dimer_two_metals`.

15.13.2 A spin chain between two leads

With the metal under the two end spins, `site=[0, ns-1]`, the chain sits between two semi-infinite leads, the geometry of a junction. Each lead can be a different metal, and each one can be a superconductor with the onsite pairing of `kondo_chain`,

$$H^{(j)}_{\text{lead}} = \sum_m \epsilon_m N_m + \sum_m t_m \left(f^\dagger_m f_{m+1} + \text{h.c.} \right) + \Delta_j \sum_m \left(f^\dagger_{m\uparrow} f^\dagger_{m\downarrow} + \text{h.c.} \right).$$

The leads couple to the spins by exchange only, which moves no charge in or out of a lead. As a result the phase of one pairing can be gauged away: the energy does not depend on the phase difference of the two leads, and there is no Josephson coupling through the spins. What the model gives is how each end spin is screened by its own lead, how a gap suppresses that screening, and the spin correlations across the junction.

```
fc = ysr.spin_chain_on_metal(4, 8, site=[0, 3], Js=0.3, JK=0.5,
                             Delta=[0.0, 0.5], Lambda=4.0) # normal | chain | superconductor
fc.maxm, fc.nsweeps = 64, 16
p, w = fc.impurities[1], fc.bath_sites[1][0] # the right spin and its lead
print(fc.vev(fc.Sx[p]*fc.Sx[w] + fc.Sy[p]*fc.Sy[w] + fc.Sz[p]*fc.Sz[w]).real)
```

The binding $\langle \mathbf{S}_{\text{end}} \cdot \mathbf{s}_0 \rangle$ of the right spin to its lead is -0.123 at $\Delta = 0$ and -0.034 at $\Delta = 0.5$, while that of the left spin stays at -0.122: the gap removes the states that screen the spin on one end only.

- `Delta`, `D`, `Lambda`: a number, or one value per spin of `site`, so each lead has its own gap, bandwidth and Wilson mesh. They need `site` as a list.
- `JK`: as above. On the mean mesh pass `JK*ysr.lambda_factor(Lambda)` of each lead to stand for the continuum coupling.
- The Wilson chain resolves energies down to about $D\Lambda^{-n/2}$, so it must reach well below the gap and below the Kondo scale of each end.
- `subgap_energies` takes one impurity and raises on this chain.
- With a superconducting lead, the STM spectrum runs with `mode="ED", submode="CVM"` only, on short chains. The DMRG routes raise there, since they do not agree with ED yet. One spin on one lead, `spin_chain_on_metal(1, n, site=[0])`, is not concerned: it is the chain of `kondo_chain`.
- `ground_level` (in `get_stm_spectrum`, ED only): by default a degenerate ground level is averaged with equal weights, the limit $T \to 0$. On a symmetric junction the two members of a level can give different spectra under one tip.

The example is `examples/kondo/spin_chain_between_metal_and_superconductor`.

15.13.3 An Anderson level in the chain

The spin under the tip can also be an orbital with a charge, an Anderson level: an energy ϵ_d, a repulsion U and a hopping t_D onto the metal,

$$H_d = \epsilon_d N_d + U n_{d\uparrow} n_{d\downarrow} + t_D \sum_s \left(d^\dagger_s f_{0s} + \text{h.c.} \right),$$

with the hybridization $\Gamma = \pi \rho t_D^2$. The exchange J_s with the neighbouring spins acts on the spin of the level. The tip puts its electron on the level, so dI/dV is the spectral function of the level: the two Hubbard peaks near ϵ_d and $\epsilon_d + U$, the Kondo resonance, and the spin excitations of the chain. Away from $\epsilon_d = -U/2$, adding and removing an electron cost different energies, and the spectrum is not symmetric in the bias. At $U \gg \Gamma$ the level is a Kondo spin with $J_K = 8t_D^2/U$ at $\epsilon_d = -U/2$, and the chain becomes `spin_chain_on_metal`.

```
fc = ysr.anderson_spin_chain(3, 3, Js=0.3, ed=-0.4, U=1.6, tD=0.3) # the last spin is a level
fc.maxm, fc.nsweeps = 64, 16
es = np.linspace(-1.5, 1.5, 151)
(e, didv) = ysr.get_stm_spectrum(fc, es, delta=0.04)
print(es[:75][np.argmax(didv[:75])], es[75:][np.argmax(didv[75:])]) # -0.56 and 0.10
```

The NRG route resolves the Kondo resonance of the level, at any temperature. For one level at $\epsilon_d = -U/2$ the Friedel sum rule fixes its height, $\pi\Gamma\rho_d(0) = 1$ per spin:

```
Gamma = 0.05 # the hybridization, in units of the half-bandwidth
tD = np.sqrt(2*Gamma/np.pi)*np.sqrt(ysr.lambda_factor(4.0)) # tD on the mesh
fc = ysr.anderson_spin_chain(1, 12, U=0.5, tD=tD, Lambda=4.0, itensor_version=3)
fc.maxm, fc.nsweeps, fc.cutoff = 160, 30, 1e-12
(e, didv) = ysr.get_stm_spectrum(fc, np.array([0.0]), submode="NRG", nz=2)
print(np.pi*Gamma*didv/2) # 1.000
```

- `ed`, `U`, `tD`: the level; `ed` defaults to `-U/2`. On the default mesh, `tD*np.sqrt(ysr.lambda_factor(Lambda))` stands for the continuum t_D.
- `site`: the spin that is a level; a list makes several levels, each on its own metal.
- `q` (in `get_stm_spectrum`): the direct tunneling into the metal relative to the level, `d^dag + q f^dag_0`, which gives Fano line shapes. 0 by default.
- With `submode="NRG"`, `head="dmrg"` only.

The example is `examples/kondo/anderson_level_on_spin_chain`.

15.14 The Kondo resonance by NRG

TASK resolves the spectrum of a Wilson chain only down to about a third of the Kondo temperature, because one Krylov space has to converge at every energy at once. The numerical renormalization group (NRG) resolves one energy scale per Wilson site instead: it adds one site, diagonalizes, keeps the lowest states and repeats. With the discarded states of every site and the full density matrix, the spectrum keeps its sum rule and is valid at any temperature. The quantity is the same $A(\omega)$ as in A magnetic atom on a superconductor, for a spin or a chain of spins on a normal metal (`ysr.kondo_chain` or `ysr.spin_chain_on_metal`). For one spin, the T-matrix $t(\omega) = \pi^2\rho J^2 A(\omega)/2$ reaches the unitary value 1 at $\omega = 0$ and $T = 0$, which is the height of the Kondo resonance in STM.

```
import numpy as np
from dmrgpy import ysr
J = 0.4*ysr.lambda_factor(4.0) # rho J = 0.2 in the continuum
fc = ysr.kondo_chain(40, J=J, Lambda=4.0, itensor_version="python")
es = np.array([0.0, 1e-4, 1e-2])
(e, didv) = ysr.get_stm_spectrum(fc, es, submode="NRG", head="exact", nz=4)
t = np.pi**2*0.5*0.4**2*didv*fc.dos_factor/2 # the T-matrix
print(t) # 1.001, 0.999 and 0.596; Wilson's T_K is 3.5e-3
(e, didv) = ysr.get_stm_spectrum(fc, es, submode="NRG", head="exact", nz=4,
                                 T=3.5e-3) # at T = T_K, t(0) = 0.719
```

There are two ways to start the iteration. With `head="exact"` the spins and the first Wilson site are diagonalized exactly, which works up to seven spins: above the bandwidth of the spins every spin state is a low-energy state, and the number of states to keep grows with the number of spins. With `head="dmrg"` DMRG solves the spins and the Wilson sites of `fc` (the head, a short chain of about 12 Wilson sites), and the NRG goes on from the bond basis of that state a few sites before its end (the tail). The tail gives the spectrum below the energy of that site, and TASK on the head gives it above. On three spins the two ways agree, and on seven spins the tail agrees with the exact route to about 2% of the peak. Beyond that there is no exact reference, so raise `m` and `fc.maxm` until the spectrum stops changing: on seven to ten spins the bond basis at the hand-over needed 300 to 400 states (`m=5` or 6, `fc.maxm=512`).

```
fc = ysr.spin_chain_on_metal(3, 12, Js=0.01, JK=0.3*ysr.lambda_factor(4.0),
                             Lambda=4.0, itensor_version=3) # metal under the end spin
```

```
fc.maxm, fc.nsweeps = 160, 30
es = np.logspace(-8, 0, 50)
(e, a1) = ysr.get_stm_spectrum(fc, es, submode="NRG", head="exact", keep=1600)
(e, a2) = ysr.get_stm_spectrum(fc, es, submode="NRG", head="dmrg", keep=1600,
                               m=4, nkry=30)
print(np.max(np.abs(a2/a1 - 1)[es < 1e-3])) # 4e-4 below the hand-over
```

Keeping only the lowest states of the bare spin chain as the impurity is not a controlled shortcut. The states that are cut take part in the Kondo flow between the band edge and the spin exchange: for three spins at $J_s = 10^{-3}D$ the lowest doublet alone gives a resonance 42 times narrower than all eight states.

- `head`: `"exact"` (the first shell diagonalized, up to seven spins) or `"dmrg"` (the bond basis of a DMRG state of `fc`, solved with `fc.maxm` and `fc.nsweeps`; `fc` is the head and should be short).
- `nz`: the number of twisted meshes averaged. One mesh at $\Lambda = 4$ is a comb of lines; 4 meshes give a smooth curve.
- `keep`: the states kept per Wilson site, 800 by default, enough up to four spins. Five and six spins need 1600, seven spins 3200.
- `T`: the temperature. At $T > 0$ the result is dI/dV with the Fermi function of a normal tip at the same temperature; `tip_fermi=False` gives $A(\omega)$ itself.
- `m`: with `head="dmrg"`, the tail starts `m` Wilson sites before the end of the head. The bond basis there needs about 150 states on three spins (`m=4`, `fc.maxm` of at least 160) and 300 to 400 on seven to ten. The temperature has to be far below the energy of that site, since TASK is a $T = 0$ method.
- `symmetric`: with `head="dmrg"`, a head with an odd number of electrons is one member of a doublet; `"auto"` (the default) hands over the bond basis of both members, which keeps the tail symmetric under the flip of all spins. `False` uses one member only, a few percent off when the metal does not screen the moment.
- `nkry`: the Krylov blocks of TASK on the head, which gives the spectrum above that energy. The default 90 gives it to about 1% on three spins; 30 is about three times faster and about 11% off there.
- `join`: how the tail and TASK meet. `"fade"` (the default) mixes the two over a factor Λ on each side of the energy of the hand-over; `"cut"` switches at that energy, which leaves an error of a few percent there. The fade needs a converged TASK: at `nkry=30` it is no better than the cut around that energy.
- `b`, `omega0`: the log-Gaussian broadening, as in `zaverage`.

The examples are `examples/kondo/kondo_resonance_nrg` and `examples/kondo/spin_chain_nrg_head_tail`.

15.15 A spin chain on a superconducting island

We now take a chain of spins 1/2 whose last spin sits on a small superconducting island, and the island leaks into a normal metal. Everything is one linear chain: first the spins, then the island, then the Wilson chain of the metal. The Hamiltonian is

$$\begin{aligned} H = {} & J_s \sum_i \mathbf{S}_i \cdot \mathbf{S}_{i+1} + J_K \, \mathbf{S}_{N_s-1} \cdot \mathbf{s}_{a_0} + B \sum_i S_i^z \\ & + \sum_{k \in I} \left[\epsilon_I n_k + \Delta \left(c_{k\uparrow}^\dagger c_{k\downarrow}^\dagger + \text{h.c.} \right) \right] + t_I \sum_{k \in I} \left(c_k^\dagger c_{k+1} + \text{h.c.} \right) \\ & + t_c \left(c_{a_1}^\dagger f_0 + \text{h.c.} \right) + \sum_m \epsilon_m f_m^\dagger f_m + \sum_m t_m \left(f_m^\dagger f_{m+1} + \text{h.c.} \right), \end{aligned}$$

with $\mathbf{s}_{a_0}$ the spin of the electrons on the first island site a_0, a_1 the last island site, and f_m the Wilson chain of a flat band of half-width D. The pairing Δ is put by hand on the island and is not self-consistent. The metal has no gap, so there is no true subgap state: a peak inside the island gap is a resonance, and its width is the leak into the metal.

Each spin is one electron on its own site, so the total S^z fixes the fermion parity. The two parities differ by the energy of one quasiparticle, the lowest Bogoliubov level of island plus metal, and not by Δ. This level goes to zero with the length of the Wilson chain. At $\epsilon = 0$ it alternates with the parity of the number of island and Wilson sites, and it is much smaller when that number is odd. `ysr.island_energies(fc)` gives the sector energies, the splitting `E_parity`, and its value at $J_K = 0$ (`qp_gap`) as a reference. The STM spectrum on a spin uses the

composite tunneling vertex of A magnetic atom on a superconductor, and the electron enters at the first island site.

```
import numpy as np
from dmrgpy import ysr
from dmrgpy.ysrtk import paritystate
fc = ysr.island_chain(4, 2, 8, Js=0.3, JK=0.6, Delta=0.25, t_island=0.5,
                      t_c=0.1, itensor_version=3) # 4 spins, 2 island sites, 8 Wilson sites
fc.maxm, fc.nsweeps = 60, 12
gs, info = paritystate.parity_ground_state(fc) # ground state of definite parity
es = np.linspace(-1.0, 1.0, 201)
(e, didv) = ysr.get_stm_spectrum(gs, es, delta=0.03) # tip on the end spin
print(fc.island_gap, info["E_parity"]) # 0.559 and about 0.06
```

The spectrum has peaks from the island levels above `fc.island_gap`, and resonances of the end spin inside it. The resonances move to lower bias as t_c grows: from ± 0.38 at $t_c = 0.1$ to ± 0.25 at $t_c = 0.4$. `ysr.island_spin_correlations(fc)` gives $\langle \mathbf{S}_a \cdot \mathbf{S}_b \rangle$ over the spins and the first island site, which says how much of the end spin the island screens.

At small Δ the pairing is the only term that changes the number of electrons, and a sector solve can stop at the wrong parity. Compare with `mode="ED"` on a short chain, and raise `nsweeps`.

- `ns`, `ni`, `n`: the number of spins, island sites and Wilson sites.
- `Js`, `JK`: the Heisenberg and the Kondo exchange, antiferromagnetic when positive.
- `Delta`, `t_island`, `eps_island`: the pairing, hopping and onsite energy of the island.
- `t_c`: the hopping from the island into the metal. On the default mesh, `t_c*np.sqrt(ysr.lambda_factor(Lambda))` stands for the continuum t_c, with $\Gamma = \pi \rho t_c^2$.
- `B`: a field on the spins only.
- `Lambda`, `D`, `mesh`, `z`: the Wilson chain, as in `kondo_chain`.
- `nkry` (in `get_stm_spectrum`): raise it for sharp lines on a long chain. On 16 sites the default 100 leaves the lines about 1.5% of the peak away from 300.

The example is `examples/kondo/spin_chain_on_superconducting_island`.

16 Finite temperature

On this page we compute thermal averages,

$$\langle O \rangle_T = \frac{\mathrm{Tr}\,[e^{-\beta H} O]}{\mathrm{Tr}\, e^{-\beta H}}, \qquad \beta = 1/T.$$

There are three methods: purification, METTS and exact diagonalization. METTS is built on imaginary-time evolution, which has its own section.

16.1 Purification

Each physical site gets an ancilla site. The product of singlets between each site and its ancilla is a pure state whose reduced density matrix on the physical chain is $\rho \propto \mathbb{1}$, the $T = \infty$ state. Evolving it in imaginary time with the physical H,

$$|\Psi(\beta)\rangle = e^{-\beta H/2} |\Psi(0)\rangle,$$

and tracing out the ancillas gives $\rho(T) \propto e^{-\beta H}$.

```
from dmrgpy import thermal
n = 6
tc = thermal.Thermal_Spin_Chain(["S=1/2"]*n, T=0.5)
h = sum(tc.Sx[i]*tc.Sx[i+1] + tc.Sy[i]*tc.Sy[i+1] + tc.Sz[i]*tc.Sz[i+1]
```

```
        for i in range(n-1))
tc.set_hamiltonian(h)
tc.get_gs()                                    # builds |Psi(beta)>
E = tc.MBChain.vev(h).real                     # thermal energy
czz = tc.MBChain.vev(tc.Sz[0]*tc.Sz[1]).real   # thermal <Sz_0 Sz_1>
```

`tc.Sx`, `tc.Sy`, `tc.Sz` are the operators of the physical sites, and `tc.MBChain` is the doubled chain that holds the purified state. Static expectation values on `tc.MBChain` are thermal averages. A dynamical correlator on `tc.MBChain` is not the thermal one, so use METTS or ED for those. `T=0` gives the ordinary ground state. The imaginary-time evolution is accurate to about 10^{-6} relative at the defaults. The keywords are:

- `tc.anneal_step`: the size of each imaginary-time step, in units of the spectral width (default 2). Smaller is more accurate and slower.
- `tc.anneal_order`: the order of the expansion of each step (default 8).
- `tc.anneal_method`: how $e^{-\beta H/2}$ is applied. `"taylor"` is the expansion above, `"implicit"` the implicit steps of Imaginary-time evolution, whose number grows as $\log\beta$ and not as βW, with W the spectral width. The default, `None`, takes the implicit steps on `itensor_version=3` at $T \leq 0.05$ (in units of the largest coupling) when they need far fewer solves than the expansion needs steps, that is on long chains, and the expansion otherwise. Setting `anneal_step` or `anneal_order` keeps the expansion. The implicit steps run on `itensor_version=3`, `"python"` and `"julia_live"`.
- `tc.anneal_options`: the keywords of `evolve_imaginary_time` for `"implicit"`, except `shift`. The default is `order=8` with `nsteps=2`: the $T = \infty$ state holds every energy of the spectrum, and the trapezoid rule is not accurate enough for it.
- `tc.MBChain.maxm`: the bond dimension of the purified state. It needs to be larger at intermediate temperatures.

See `examples/finite_temperature/thermal_purification_VS_exact`.

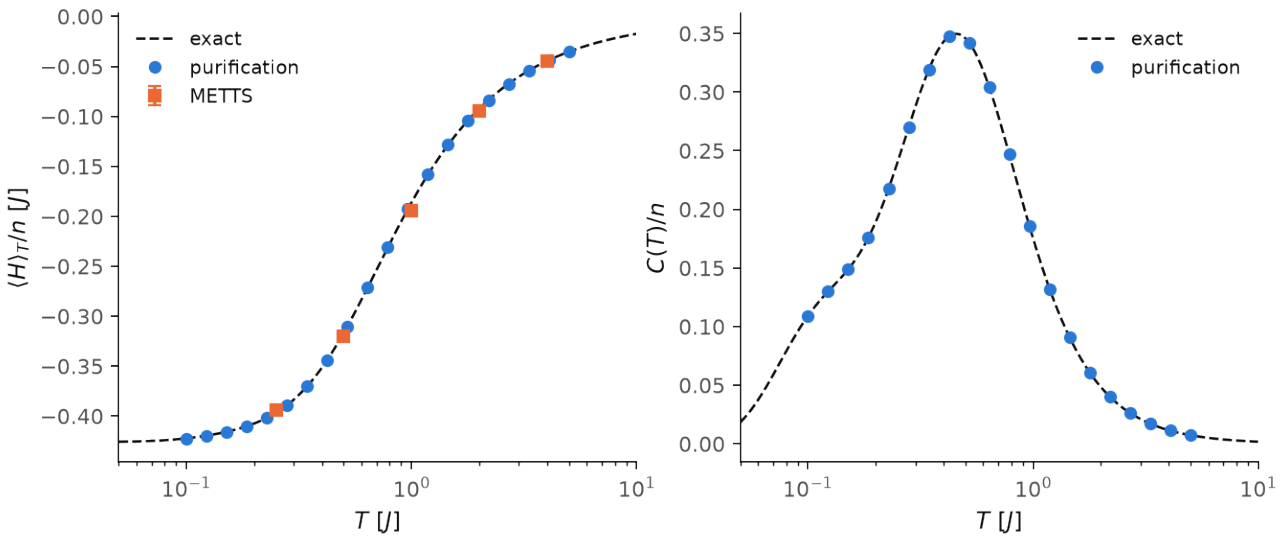


Figure 20: Thermal energy and specific heat of the Heisenberg chain

The figure shows the thermal energy and the specific heat per site of a spin-1/2 Heisenberg chain with $n = 10$. The specific heat is the energy variance, $C(T) = (\langle H^2\rangle_T - \langle H\rangle_T^2)/T^2$, measured in the purified state, so no numerical derivative is needed. The reference is the full spectrum of the chain (dashed lines). Purification agrees with it to 10^{-5} at every temperature. $C(T)$ has a broad maximum near $T \approx 0.46J$, the scale of the antiferromagnetic correlations. The METTS points are discussed in METTS.

The code of this figure is in `docs/figures/finite-temperature_specific_heat/`.

16.2 Imaginary-time evolution

Imaginary-time evolution projects a state onto the low energies,

$$|\psi(\tau)\rangle \propto e^{-\tau H}|\psi\rangle = \sum_n e^{-E_n\tau}|n\rangle\langle n|\psi\rangle.$$

At long τ it gives the ground state, and at $\tau = \beta/2$ it is the step of METTS below. Each eigenstate decays as $e^{-(E_n-E_0)\tau}$, so the energy scale that matters falls as τ grows, and the time steps can grow with it (Zima, Stoudenmire, White, Parcollet and Kaye, arXiv:2606.02930). The grid has panels $[0,\tau_0]$, $[\tau_0,2\tau_0]$, $[2\tau_0,4\tau_0]$, … up to τ, with a few equal steps in each, so the number of steps grows as $\log\tau$ and not as τ. A long step needs an implicit rule, the trapezoid rule

$$\Big(1+\tfrac{\Delta\tau}{2}(H-E_0)\Big)|\psi_k\rangle = \Big(1-\tfrac{\Delta\tau}{2}(H-E_0)\Big)|\psi_{k-1}\rangle,$$

which is stable for any step size. Each step needs one linear solve. Here `h` is the Hamiltonian of the chain and `wf` the initial state, for example a Néel state:

```
wft = sc.evolve_imaginary_time(wf, 100.)      # normalized e^{-100 H}|wf>
taus, e = sc.evolve_imaginary_time_and_measure(h, wf, 100.)
```

The second call returns the grid times and $\langle\psi(\tau)|h|\psi(\tau)\rangle$ on them. With an ED state, `mode="ED"` gives the exact evolution on the same grid. In `examples/finite_temperature/imaginary_time_log_grid` the Néel state of a 12-site Heisenberg chain reaches $\tau = 1000$ in 72 steps, and its energy ends 5×10^{-12} above E_0. The error of the trapezoid rule falls as $1/n^2$ with the number n of steps per panel. It is largest at intermediate τ, where the energy is off by about 2×10^{-3} (in units of the exchange) at $n=4$ in that example. The keywords are:

- `nsteps`: the steps per panel (default 4).
- `tau0`: the end of the first panel (default 0.01 over the largest coupling of H). The first steps must resolve the highest energies the initial state contains.
- `order`: 2 for the trapezoid rule, 4, 6, … for the Gauss-Legendre rules, more accurate per step, with `order/2` complex linear solves each.
- `shift`: the energy subtracted from H, by default the ground-state energy. It must be the ground-state energy: with a lower shift the long steps stop projecting.

It works with `itensor_version=3` (at least 3 sites), `"python"` and `"julia_live"`, for a Hermitian H.

16.3 METTS

Minimally entangled typical thermal states (Stoudenmire and White, arXiv:1002.1305) sample product states $|i\rangle$ with a Markov chain. Each one is evolved to $|\phi_i\rangle \propto e^{-\beta H/2}|i\rangle$ and then collapsed to a new product state. The plain average of $\langle\phi_i|O|\phi_i\rangle$ over the chain is the thermal average. The states stay weakly entangled, so no ancillas are needed. Here `sc` is the Heisenberg chain of Ground state:

```
T = 0.5
mean, err = sc.metts_vev(sc.Sz[0]*sc.Sz[1], T, nsamples=300, nwarmup=30)
res = sc.metts_vev([sc.Sz[0], sc.Sz[1], h], T)    # list of (mean, err)
```

Being a Monte Carlo method, it returns a mean and a statistical error. The METTS energies in the figure of Purification, with 300 samples per temperature, agree with the exact energy within two error bars. The error assumes independent samples, so it is too small when samples are correlated. Measuring several operators in one call is much cheaper than one call per operator, since they share the samples. The keywords are:

- `nsamples`, `nwarmup`: the number of samples, and the number of first steps discarded for equilibration.
- `dbeta_half_step`: the imaginary-time step (default 0.05).
- `basis_ops`: the two bases used in turn for the collapse (default `("Sz","Sx")`). Alternating them reduces autocorrelation.
- `njobs`: runs independent Markov chains in parallel processes. Only for `itensor_version="python"`.

- `itime`: how each sample is evolved to $\beta/2$. By default the cheaper of two ways at the same accuracy: TDVP steps of `dbeta_half_step` at high temperature, and below $T \approx 0.12$ (in units of the largest coupling) the implicit steps of Imaginary-time evolution, whose number grows as $\log\beta$ instead of β. `itime="tdvp"` or `itime="implicit"` forces one, and `itime_options` takes the keywords of `evolve_imaginary_time`, with 8 steps per panel by default here.

METTS works with `itensor_version=3` (at least 3 sites), `"python"` and `"julia_live"`. It is not available on `itensor_version=2`, where it raises an error. It also raises if the basis operators are all diagonal and H conserves a charge, such as the particle number, since the samples then never leave one charge sector. The default `basis_ops` avoids this.

The same sampling gives real-time thermal correlators $C_{AB}(t) = \langle A(t)B\rangle_T$ (arXiv:2405.18484). Each sample $|\psi_i\rangle$ gives $\langle\psi_i|e^{iHt}Ae^{-iHt}B|\psi_i\rangle$ by real-time evolution, and the average is $C_{AB}(t)$:

```
ts, c, err = sc.metts_dynamical_correlator((sc.Sz[0], sc.Sz[0]), T,
                 nt=100, dt=0.1, nsamples=200, nwarmup=30)
```

The result is in the time domain, on $t = 0, \Delta t, \ldots, (n_t - 1)\Delta t$. For a spectrum, apply a window and a Fourier transform yourself.

16.4 Exact diagonalization

For small chains we sum over the eigenstates, $\langle O\rangle_T = \sum_n p_n \langle n|O|n\rangle$ with $p_n = e^{-E_n/T}/Z$. Here `sc` is a short chain, `n` of about 10 sites:

```
sc.vev(sc.Sz[0]*sc.Sz[1], mode="ED", T=0.5)
sc.get_correlation_matrix(T=0.5, operators=[sc.Sz[i] for i in range(n)])
sc.get_dynamical_correlator(mode="ED", submode="ED", T=0.5,
                            name=(sc.Sz[0], sc.Sz[0]))
```

The default `vev` on a DMRG chain raises at `T>0`, so `mode="ED"` is needed. The sum uses the full spectrum when the Hilbert space has at most about 2000 states. Above that, only the `n=` lowest states are used, and an error is raised if the last one still has a large Boltzmann weight. The ED dynamical correlator is the exact reference for `metts_dynamical_correlator` (see `examples/finite_temperature/dynamical_metts_VS_ED`).

17 Open systems

On this page we compute the steady state and the slowest relaxation modes of a chain coupled to an environment, described by a Lindblad master equation,

$$\frac{d\rho}{dt} = \mathcal{L}\rho = -i[H,\rho] + \sum_\mu \Big(L_\mu \rho L_\mu^\dagger - \tfrac{1}{2}\{L_\mu^\dagger L_\mu, \rho\}\Big).$$

The jump operators L_μ describe what the environment does: decay, pumping, dephasing, dissipative hopping.

17.1 Steady state and dissipative gap

$\mathcal{L}$ is linear in ρ, so it has eigenvalues λ_k with $\mathrm{Re}\,\lambda_k \le 0$. The steady state is the eigenvector with $\lambda = 0$. The slowest mode sets the relaxation time through the dissipative gap $\Delta = -\mathrm{Re}\,\lambda_1$, meaning that a generic state approaches the steady state as $e^{-\Delta t}$. The way we do this is by writing ρ as a matrix product state on a chain of twice the length, one site for the ket and one for the bra index of each physical site. Then $\mathcal{L}$ becomes an operator on that chain, and we take its slowest eigenvalues from time evolution, as in Excited states.

As a reference, let us take an XXZ chain pumped up at the left end, pumped down at the right end, and dephased on every site:

```
from dmrgpy import spinchain
from dmrgpy.lindblad import Lindblad_Chain
n = 6
```

```
sc = spinchain.Spin_Chain(["S=1/2" for i in range(n)], itensor_version="python")
h = sum(sc.Sx[i]*sc.Sx[i+1] + sc.Sy[i]*sc.Sy[i+1] + 0.5*sc.Sz[i]*sc.Sz[i+1]
        for i in range(n-1))
sc.set_hamiltonian(h)
sp = sc.Sx[0] + 1j*sc.Sy[0]          # pumps spin up at the left end
sm = sc.Sx[n-1] - 1j*sc.Sy[n-1]      # pumps spin down at the right end
jumps = [0.6*sp, 0.6*sm] + [0.4*sc.Sz[i] for i in range(n)]
lc = Lindblad_Chain(sc, jumps)
lam, res, obs = lc.get_spectrum()           # lam[0] ~ 0 is the steady state
gap = -lam[1].real                          # dissipative gap
mz = [lc.steady_state_vev(sc.Sz[i]) for i in range(n)]
```

`lam` is sorted by real part, the steady state first. `res[k]` is how much λ_k moves when the last snapshot is left out; only the eigenvalues that do not move are converged. `mz` is the magnetization profile of the steady state, a linear ramp from $+0.18$ to -0.18 that carries a spin current through the chain. Here the gap is 0.0927 against 0.0926 from ED, and the profile agrees with ED to 3×10^{-4}. The keywords are those of `get_krylov_spectrum`, with one difference:

- `dt`: the time step (default 2.0). The evolution must run for several times $1/\Delta$, so `nk*dt` has to be large when the gap is small.
- `substeps`: the number of time steps per snapshot to start from (default 1). A strongly dissipative $\mathcal{L}$ makes long steps unstable, so a step that does not conserve the trace is redone with twice as many. Only the cost changes, not `dt`.
- `mode="ED"`: the dense $\mathcal{L}$, as a reference on a small chain.

When $\mathcal{L}$ conserves a quantity, for example the particle number for number-conserving jumps, every sector has its own steady state, and several eigenvalues come back at $\lambda \approx 0$. To stay in one sector we start from product states of that sector:

- `start=(psi, phi)`: one local basis state per site (the occupation, for bosons). The evolution starts from $|\psi\rangle\langle\psi|$ and from the traceless $|\psi\rangle\langle\phi|$, and the sector of ψ is kept, for example `start=([1,1,0],[0,1,1])` for two bosons on three sites. Modes of other sectors, which enter through small numerical errors, are left out of `lam`.

  Backends.

  `Lindblad_Chain` works for spin and boson chains on `itensor_version="python"` and `3`. Fermions are not supported.

18 Infinite chains

An infinite chain repeats a unit cell of `n_uc` sites forever in both directions. It describes the thermodynamic limit directly, with no edges and no finite-size extrapolation. The state is a uniform matrix product state, and the results are densities per site and correlators at any distance.

18.1 Building the Hamiltonian

Each operator exists in three copies per site `i` of the unit cell: `SzC[i]` in the central cell, `SzR[i]` in the next cell and `SzL[i]` in the previous cell. A term is written once, and it is repeated in every cell:

```
from dmrgpy import infinitechain
ic = infinitechain.Infinite_Spin_Chain(["1/2"])          # n_uc = 1
h = ic.SxC[0]*ic.SxR[0] + ic.SyC[0]*ic.SyR[0] + ic.SzC[0]*ic.SzR[0]
ic.set_hamiltonian(h)
e = ic.gs_energy()      # energy per site
```

A two-site cell describes for example a dimerized chain:

```
ic = infinitechain.Infinite_Spin_Chain(["1/2", "1/2"])
h = (1.0*(ic.SxC[0]*ic.SxC[1] + ic.SyC[0]*ic.SyC[1] + ic.SzC[0]*ic.SzC[1])
```

```
    + 0.4*(ic.SxC[1]*ic.SxR[0] + ic.SyC[1]*ic.SyR[0] + ic.SzC[1]*ic.SzR[0]))
ic.set_hamiltonian(h)
```

For longer couplings, `ic.get_operator(name, i, group=c)` gives site `i` of the cell at distance `c`, so `c=-1,0,1` are `L`, `C`, `R`. For example a J1-J2 chain on a one-site cell:

```
h = 0
for op in ("Sx", "Sy", "Sz"):
    o0 = ic.get_operator(op, 0)
    h = h + o0*ic.get_operator(op, 0, group=1)        # J1
    h = h + 0.4*o0*ic.get_operator(op, 0, group=2)    # J2
```

At $J_2 = 0.4$ this chain is dimerized, so its ground state has a period of two sites. A one-site cell cannot hold it and the solver does not converge. Use the same couplings on a two-site cell for this phase.

Fermionic chains use `Infinite_Many_Body_Chain` with a list of site codes: `0` for a spinless fermion (`C`, `Cdag`, `N`) and `1` for a spinful site (`Cup`, `Cdagup`, `Cdn`, `Cdagdn`, `Nup`, `Ndn`). The Jordan-Wigner string is added automatically, also for hoppings that skip sites. A term with an odd number of fermion operators raises an error.

18.2 Two solvers

`ic.gs_method` selects the algorithm:

- `"vumps"` (the default): variational uniform MPS (arXiv:1701.07035). It finds the best state of bond dimension `maxm` directly in the thermodynamic limit. It works for any unit cell and any range of coupling.
- `"idmrg"`: infinite DMRG, which grows a finite chain from the center until the energy per site converges. Only `n_uc` of 1 or 2.

Both run on `itensor_version="python"` and `3`. On `"julia_live"`, `"vumps"` runs for `n_uc` of 1 or 2 with couplings between neighboring cells, for the energy, `vev` and `correlator`; everything else raises an error there, and so does `itensor_version=2`. The keywords are:

- `ic.maxm`, `ic.cutoff`: the bond dimension (default 30) and the truncation cutoff. Increase `maxm` until the results stop changing.
- `ic.maxiter`, `ic.etol`: the maximal number of iterations and the energy tolerance.
- `ic.vumps_nrestarts`: the number of random starts of VUMPS (default 4).
- `ic.noise`, `ic.noise_iters`: for iDMRG. A particle-conserving model has the empty and the full state as exact eigenstates, and iDMRG can get stuck in them. The noise (default 10^{-4}) breaks this, and is applied only while the state is a product state.

After the solve, `ic.converged` says whether the tolerance was reached. If it was not, `gs_energy()` also gives a `RuntimeWarning` with the reason. VUMPS starts from a random state, so on a hard model call `gs_energy()` a few times and keep the lowest energy. A simple product state, such as the Neel state, gives an upper bound to compare with. For a gapless model on a one-site cell, iDMRG can end in a dimerized state (see Static correlators). Then the energy per site alternates between iterations, so `converged` stays `False` while the energy is already accurate to several digits. Entanglement entropies are not available for infinite chains.

The left panel shows the error of the iDMRG energy per site of the spin-1/2 Heisenberg chain, against the Bethe-ansatz value $e_\infty = 1/4 - \ln 2$, as a function of `maxm`. The chain is gapless, so the error falls as a power of `maxm`, from 3×10^{-4} at `maxm=8` to 1.6×10^{-6} at `maxm=64`. On the one-site cell it stops there. A two-site cell gives 5×10^{-7} at `maxm=96` (square). Here `converged` stays `False` up to `maxm=64`, because the one-site state is dimerized, while the energy is already accurate. The right panel is described in Static correlators.

The code of this figure is in `docs/figures/infinite-chains_energy/`.

18.3 Static correlators

```
ic.vev("Sz", 0)                    # <Sz> on site 0 of the cell
ic.correlator("Sz", 0, "Sz", r)    # <Sz_0 Sz_r>, r in sites, r >= 0
ic.correlator("Cdag", 0, "C", r)   # fermionic chain, with the string
```

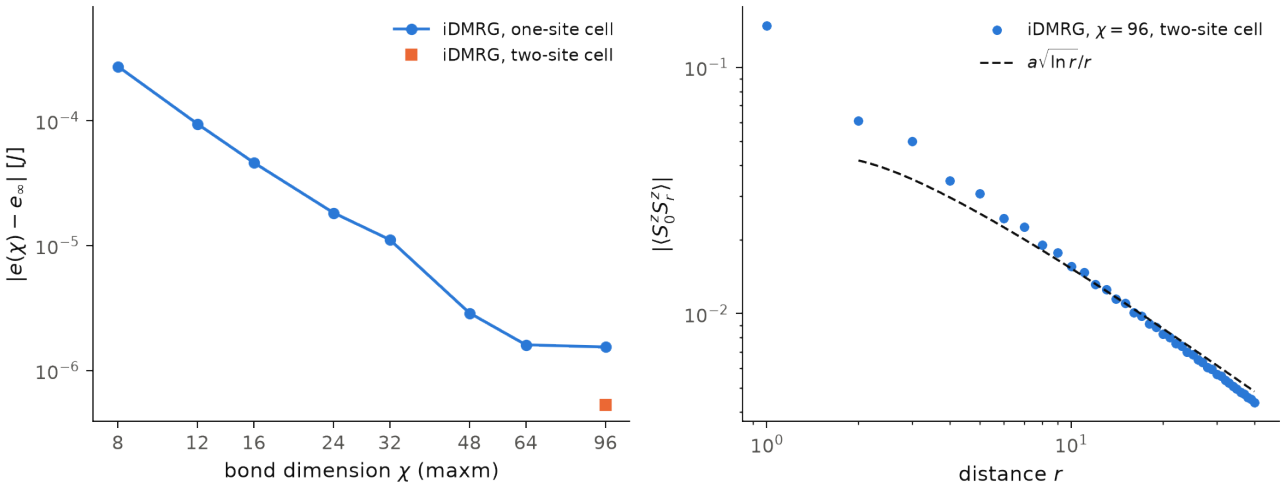


Figure 21: Energy and spin correlator of the infinite Heisenberg chain

They work with both solvers and both backends. A pair with an odd number of fermion operators raises an error. A good check of `maxm` needs no reference value: the sum of the bond correlators of one unit cell must be equal to `n_uc` times the energy per site.

With iDMRG at finite `maxm`, a gapless chain can end in a dimerized state: the bonds alternate in strength. This comes from the finite `maxm`, and the alternation becomes smaller when `maxm` grows. A one-site cell cannot hold this state. The energy per site then alternates between iterations, so `converged` stays `False`, and `ic.correlator("Sz", 0, "Sz", 1)` gives only one of the two bonds. In this case `gs_energy`, `vev` and `correlator` give a `RuntimeWarning` with the entanglement entropies of the two bonds. On a two-site cell both bonds are measured, and their mean is the energy per site.

When VUMPS did not converge, `vev`, `correlator` and the excitations also give a `RuntimeWarning`. Such a state is not a consistent uniform MPS, and its correlators can be wrong by 10 per cent while its energy is accurate.

The right panel of the figure in Two solvers shows $|\langle S_0^z S_r^z \rangle|$ of the Heisenberg chain on a two-site cell at `maxm=96`, averaged over the two sites. There is no edge, so the correlator decays as the power law $a\sqrt{\ln r}/r$ of a Luttinger liquid (dashed, a fitted at $r = 8$ to 20). At $r = 1$ it gives $3\langle S_0^z S_1^z \rangle = -0.443147$, which is e_∞. At the largest r the data fall a little below the power law, because a finite `maxm` gives a finite correlation length.

18.4 Excitations

In an infinite chain the excitations form bands $E(k)$. We use the quasiparticle ansatz: one tensor B inserted on top of the ground state, in a plane wave of momentum k. This gives the dispersion of the lowest modes, as measured by inelastic neutron scattering:

```
ic.excitation_energies(k, n=1)    # n lowest energies at momentum k
ic.excitation_gap()               # minimum over k
```

k is in radians per unit cell. It needs `gs_method="vumps"`, `n_uc` of 1 or 2, and couplings to the neighboring cell only.

18.5 Dynamical structure factor

Each quasiparticle branch a contributes a delta peak to $S(k, \omega)$ with weight $w_a = |\langle k, a | O_k | \mathrm{GS} \rangle|^2$:

$$S(k, \omega) = \sum_a w_a \, \delta(\omega - E_a(k)).$$

```
e, w = ic.spectral_weights("Sz", k, n=2)
e, w, total = ic.spectral_weights("Sz", k, n=2, return_total=True)
ks, es, S = ic.dynamical_structure_factor("Sz", delta=0.05)  # map in (k, w)
```

The peaks are exact in momentum and energy. They do not contain the continuum of two or more particles. `total` is the static structure factor $\sum_r e^{ikr}(\langle O_0 O_r\rangle - \langle O\rangle^2)$, so `w.sum()/total` is the fraction of the weight in the computed branches. When it is close to 1, a single-mode picture is valid at that k. In `dynamical_structure_factor`, `delta` is a Lorentzian broadening for plotting. These two methods need `gs_method="vumps"` and a parity-even operator, on `itensor_version="python"` or `3`. See `examples/idmrg/dynamical_structure_factor_tfim`.

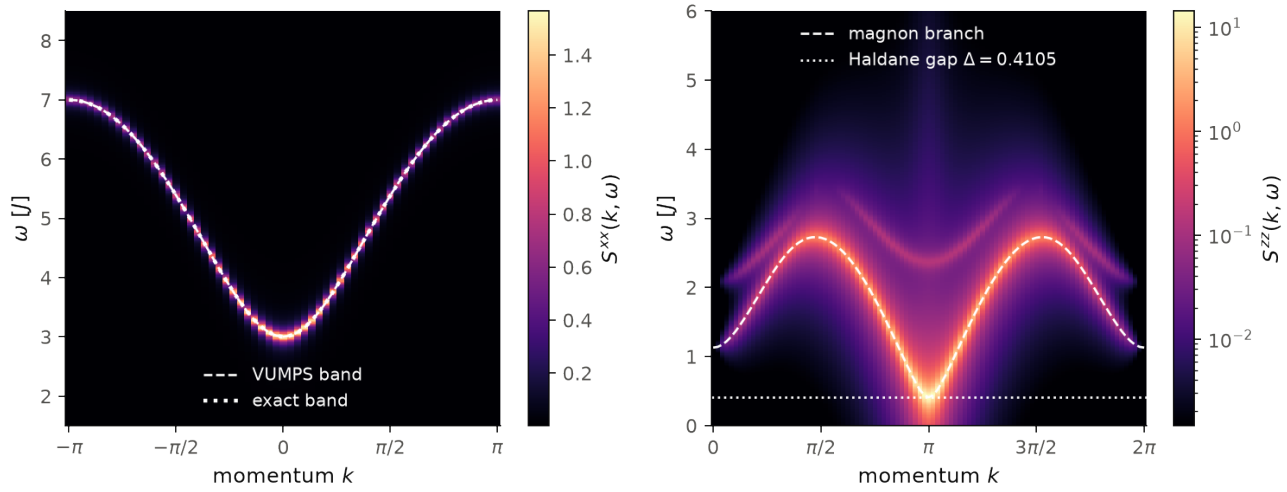


Figure 22: Excitation bands and structure factors of infinite chains

The left panel is the transverse-field Ising chain $H = -\sum \sigma^x\sigma^x - g\sum \sigma^z$ at $g = 2.5$, in the paramagnetic phase, with `maxm=4`. The map is $S^{xx}(k,\omega)$. The VUMPS band (dashed) lies on the exact free-fermion band $2\sqrt{1+g^2-2g\cos k}$ (dotted) within 10^{-8}. The right panel is the spin-1 Heisenberg chain with `maxm=16`, and the map is $S^{zz}(k,\omega)$ on a log color scale. The magnon branch has its minimum at $k = \pi$. There it gives the Haldane gap 0.4098, against the reference 0.4105 (dotted line). Near $k = 0$ the weight goes to zero, because $\sum_j S^z_j$ is conserved. The weak features above the branch come from higher states of the ansatz. The ansatz does not describe the two-magnon continuum, which starts at 2Δ.

The code of this figure is in `docs/figures/infinite-chains_excitations/`.

18.6 Local gap estimate

`ic.local_excitation_gap()` takes the converged two-site block of iDMRG and returns the gap between its two lowest states. It is cheap, but less accurate than `excitation_gap()` (10 per cent too high on the XX chain). On fermionic models it can be a charge excitation and shifts with a uniform chemical potential. Use it only as a rough check. It needs `gs_method="idmrg"`. `window=w`, with `itensor_version="python"` and `n_uc=1`, adds `w` sites on each side of the block and improves the estimate.

18.7 Dynamical correlators

Two methods compute $\langle O_i(t) O_j\rangle$ and its spectrum in an infinite chain:

```
es, ys = ic.kpm_finite("Sz", 0, "Sz", 0, n_window=16,
             window_chain_kwargs=dict(maxm=30, nsweeps=15),
             delta=0.3, es=np.linspace(-1, 5, 200))
ic.gs_method = "idmrg"
ks, es, Skw = ic.td_dynamical_correlator("Sz", 0, "Sz", n_window=14,
```

```
dt=0.05, nt=200, maxdim=60, x_values=range(-6, 7),
ks=np.linspace(-np.pi, np.pi, 41), delta=0.1)
```

- `kpm_finite`: the KPM dynamical correlator on a finite open window of `n_window` unit cells. It has finite-size and edge effects, so converge it in `n_window`. A small `delta` needs many moments, and then the edges matter even for a long window. The window runs on the same backend as the infinite chain.
- `td_dynamical_correlator`: real-time evolution of a window whose two ends are attached to the converged infinite environment (arXiv:1804.09163), so it has no edges. One evolution gives every distance `x` and every momentum `k`. `n_window` and `x_values` must be converged separately. It needs `gs_method="idmrg"`. `connected=True` (default) subtracts $\langle O_i \rangle \langle O_j \rangle$. It returns the one-sided Fourier transform, so take the real part of `Skw` for the spectral function.

See `examples/idmrg/td_dynamical_correlator` for both.

18.8 Operators on the state, overlaps and sums

Three methods act on the converged state itself and return the result as a new chain, so that `vev` and `correlator` measure it. Here `ic` and `ic2` are two converged chains on the same unit cell. They run on `itensor_version="python"` with both solvers, and on `itensor_version=3` with `gs_method="vumps"`:

```
flipped = ic.apply_operator(2*ic.SxC[0])    # Pauli X on every site
flipped.vev("Sz", 0)                        # -<Sz> of ic
flipped.eta                                 # 1 for a unitary operator
f = ic.overlap(flipped)                     # fidelity per site, 0 to 1
f2 = ic.overlap(ic2)                        # between two solved chains
try:
    cat = ic.summps(ic2)                    # |ic> + |ic2>
except RuntimeError:
    pass                                    # two normalized states, see below
```

- `apply_operator(op)`: `op` is an operator on the central cell, built from `ic.SxC`, `ic.SzC[0]*ic.SzC[1]` and so on, applied in every cell at once. It must stay inside the cell (an `SxR` factor raises an error), since it is repeated cell by cell. The state is normalized afterwards, and `eta` is the factor by which the operator changed its norm per cell. A gate that entangles neighboring sites grows the bond dimension, and the result is refused when truncating it back loses weight; pass `maxdim=` larger than `ic.maxm`. The gauge fixing behind this holds only for a state solved at a modest `maxm` (4 worked, 8 did not, on a free-fermion cell), so apply such a gate to a small state.
- `overlap(other)`: the fidelity per site of two states on the same unit cell, $|\langle \phi | \psi \rangle|^{1/N}$ in the limit of N sites. It is 1 for the same physical state in any gauge, and below 1 otherwise. Two ground states on the two sides of a phase transition have a fidelity that drops at the transition, the infinite-chain version of Fidelity susceptibility.
- `summps(other)`: the direct sum of two states. Tiled to infinity, the sum keeps only the branch of larger norm per site, so two normalized ground states tie and raise an error: a cat state of two ordered states has no uniform MPS representation.

The returned chain is not a ground state, so `gs_energy()` and the excitations raise on it; `set_hamiltonian()` makes it an ordinary chain again. See `examples/idmrg/apply_mpo_to_infinite_chain` and `examples/idmrg/imps_overlap`.

19 Cookbook

The first part writes out the Hamiltonian of the common models, ready for `set_hamiltonian`. The second part gives short recipes for calculations that combine several pages of the guide. Every chain is built as in Models, `n` is its number of sites, and the operators are those of its class. On a fermionic chain, `h + h.get_dagger()` adds the Hermitian conjugate of every hopping and pairing term, so we write each such term once.

19.1 Models

19.1.1 XXZ chain

$$H = J \sum_i \left(S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + \Delta\, S_i^z S_{i+1}^z \right)$$

```
sc = spinchain.Spin_Chain(["S=1/2"]*n)
Delta = 0.5
h = sum(sc.Sx[i]*sc.Sx[i+1] + sc.Sy[i]*sc.Sy[i+1] + Delta*sc.Sz[i]*sc.Sz[i+1]
        for i in range(n-1))
```

$\Delta = 1$ is the Heisenberg chain, `sc.SS(i, i+1)`; $\Delta = 0$ is the XX chain, which maps to free fermions; $|\Delta| > 1$ is gapped.

19.1.2 Transverse-field Ising chain

$$H = -J \sum_i S_i^z S_{i+1}^z - h \sum_i S_i^x$$

```
sc = spinchain.Spin_Chain(["S=1/2"]*n)
J, hx = 1.0, 0.5
h = -J*sum(sc.Sz[i]*sc.Sz[i+1] for i in range(n-1)) - hx*sum(sc.Sx)
```

With spin-1/2 operators the critical point is at $h = J/2$; with Pauli matrices ($4S^zS^z$, $2S^x$) it is at $h = J$.

19.1.3 J1-J2 chain

$$H = J_1 \sum_i \mathbf{S}_i \cdot \mathbf{S}_{i+1} + J_2 \sum_i \mathbf{S}_i \cdot \mathbf{S}_{i+2}$$

```
sc = spinchain.Spin_Chain(["S=1/2"]*n)
J1, J2 = 1.0, 0.4
h = J1*sum(sc.SS(i, i+1) for i in range(n-1)) + J2*sum(sc.SS(i, i+2) for i in range(n-2))
```

For $J_2/J_1 > 0.2411$ the chain is dimerized and gapped.

19.1.4 Dzyaloshinskii-Moriya interaction

$$H = \sum_i \mathbf{S}_i \cdot \mathbf{S}_{i+1} + D \sum_i \left(S_i^x S_{i+1}^y - S_i^y S_{i+1}^x \right)$$

```
sc = spinchain.Spin_Chain(["S=1/2"]*n)
D = 0.3
h = sum(sc.SS(i, i+1) + D*(sc.Sx[i]*sc.Sy[i+1] - sc.Sy[i]*sc.Sx[i+1]) for i in range(n-1))
```

19.1.5 Spin-1 chain with anisotropy and biquadratic exchange

$$H = \sum_i \left[\mathbf{S}_i \cdot \mathbf{S}_{i+1} + \beta \left(\mathbf{S}_i \cdot \mathbf{S}_{i+1} \right)^2 \right] + D \sum_i (S_i^z)^2$$

```
s1 = spinchain.Spin_Chain(["S=1"]*n)
D, beta = 0.2, -0.3
h = sum(s1.SS(i, i+1) + beta*s1.SS(i, i+1)*s1.SS(i, i+1) for i in range(n-1))
h = h + D*sum(s1.Sz[i]*s1.Sz[i] for i in range(n))
```

$\beta = D = 0$ is the Haldane chain, $\beta = -1/3$ the AKLT point, and a large $D > 0$ gives the trivial large-D phase.

19.1.6 Hubbard chain

$$H = -t \sum_{i\sigma} \left(c_{i\sigma}^\dagger c_{i+1\sigma} + \text{h.c.} \right) + U \sum_i \left(n_{i\uparrow} - \tfrac{1}{2} \right) \left(n_{i\downarrow} - \tfrac{1}{2} \right)$$

```
fc = fermionchain.Spinful_Fermionic_Chain(n)
t, U = 1.0, 4.0
h = -t*sum(fc.Cdagup[i]*fc.Cup[i+1] + fc.Cdagdn[i]*fc.Cdn[i+1] for i in range(n-1))
h = h + h.get_dagger()
h = h + U*sum((fc.Nup[i]-0.5)*(fc.Ndn[i]-0.5) for i in range(n))
```

The shifted interaction puts the chemical potential at zero, so the ground state is at half filling. With $U\sum_i n_{i\uparrow}n_{i\downarrow}$ instead, fix the filling with a sector.

19.1.7 Interacting spinless fermions

$$H = -t\sum_i \left(c_i^\dagger c_{i+1} + \text{h.c.}\right) + V\sum_i \left(n_i - \tfrac{1}{2}\right)\left(n_{i+1} - \tfrac{1}{2}\right)$$

```
fc = fermionchain.Fermionic_Chain(n)
t, V = 1.0, 2.0
h = -t*sum(fc.Cdag[i]*fc.C[i+1] for i in range(n-1))
h = h + h.get_dagger()
h = h + V*sum((fc.N[i]-0.5)*(fc.N[i+1]-0.5) for i in range(n-1))
```

This is the XXZ chain in fermionic language, with $t = J/2$ and $V = J\Delta$. For $V > 2t$ the chain is a charge density wave.

19.1.8 Kitaev chain

$$H = \sum_i \left(-t\, c_i^\dagger c_{i+1} + \Delta\, c_i c_{i+1} + \text{h.c.}\right) - \mu\sum_i n_i$$

```
fc = fermionchain.Fermionic_Chain(n)
t, Delta, mu = 1.0, 0.5, 0.3
h = sum(-t*fc.Cdag[i]*fc.C[i+1] + Delta*fc.C[i]*fc.C[i+1] for i in range(n-1))
h = h + h.get_dagger()
h = h - mu*sum(fc.N)
```

For $|\mu| < 2t$ the open chain has Majorana end modes and its two lowest states, of opposite fermion parity, are split only by $e^{-n/\xi}$. DMRG then returns a superposition of the two, and the parity of the state is not ± 1 (-0.06 on 8 sites in our run). Take the two lowest states from `get_excited_states(n=2)` and measure the parity of each, or use `mode="ED"` on a short chain.

19.1.9 SSH chain

$$H = \sum_i t_i \left(c_i^\dagger c_{i+1} + \text{h.c.}\right), \qquad t_i = t_1, t_2, t_1, t_2, \ldots$$

```
fc = fermionchain.Fermionic_Chain(n)
t1, t2 = 1.0, 0.5
h = sum((t1 if i % 2 == 0 else t2)*fc.Cdag[i]*fc.C[i+1] for i in range(n-1))
h = h + h.get_dagger()
```

The chain is topological, with a zero mode at each end, when the weak bond is at the edge; see the Zak phase in Topological invariants.

19.1.10 Bose-Hubbard chain

$$H = -t\sum_i \left(a_i^\dagger a_{i+1} + \text{h.c.}\right) + \frac{U}{2}\sum_i n_i(n_i - 1) - \mu\sum_i n_i$$

```
bc = bosonchain.Bosonic_Chain(n, maxnb=[4]*n)     # at most 3 bosons per site
t, U, mu = 1.0, 2.0, 1.0
h = -t*sum(bc.Adag[i]*bc.A[i+1] for i in range(n-1))
h = h + h.get_dagger()
h = h + 0.5*U*sum(bc.N[i]*(bc.N[i]-1) for i in range(n)) - mu*sum(bc.N)
```

`maxnb` cuts the local space; check that $\langle n_i \rangle$ and the energy do not change when it is increased.

19.1.11 $\mathbb{Z}_3$ clock model

$$H = -J \sum_i \left(\sigma_i^\dagger \sigma_{i+1} + \text{h.c.} \right) - f \sum_i \left(\tau_i + \tau_i^\dagger \right)$$

```
pc = parafermionchain.Parafermionic_Chain(n, Z=3)
J, f = 1.0, 0.7
h = -J*sum(pc.Sigd[i]*pc.Sig[i+1] for i in range(n-1)) - f*sum(pc.Tau)
h = h + h.get_dagger()
```

`Sig`, `Sigd`, `Tau`, `Taud` are the clock and shift operators and their adjoints. The model is self-dual at $f = J$, where it is critical.

19.1.12 Kondo lattice

$$H = -t \sum_{i\sigma} \left(c_{i\sigma}^\dagger c_{i+1\sigma} + \text{h.c.} \right) + J_K \sum_i \mathbf{S}_i \cdot \mathbf{s}_i$$

The local moments are spin sites between the fermion sites of a mixed chain, each coupled to the two conduction sites next to it:

```
mc = mixedchain.Mixed_Spin_Fermion_Chain(["F", "1/2", "F", "1/2", "F"])   # c S c S c
t, JK = 1.0, 0.8
h = -t*(mc.Cdagup[0]*mc.Cup[2] + mc.Cdagdn[0]*mc.Cdn[2]
        + mc.Cdagup[2]*mc.Cup[4] + mc.Cdagdn[2]*mc.Cdn[4])
h = h + h.get_dagger()
h = h + JK*(mc.SS(0, 1) + mc.SS(1, 2) + mc.SS(2, 3) + mc.SS(3, 4))
```

`mc.SS(i, j)` is $\mathbf{S}_i \cdot \mathbf{S}_j$ with the fermion spin on a fermion site and the local moment on a spin site. See `examples/fermion_models/mixed_spin_fermion_chain`.

19.2 Recipes

19.2.1 Central charge of the critical Ising chain

For $H = \sum_i S_i^z S_{i+1}^z + \frac{1}{2} \sum_i S_i^x$, at the critical field, the entanglement entropy of the open chain follows the Calabrese-Cardy formula with $c = 1/2$ (Entanglement):

```
sc.maxm = 200          # critical chains need a large bond dimension
wf = sc.get_gs()
c = wf.get_CFT_central_charge()
```

19.2.2 Haldane gap of the spin-1 chain

The open chain has four low states from the two spin-1/2 edge states. The bulk gap is the fifth state (Excited states):

```
s1 = spinchain.Spin_Chain(["S=1"]*n)
s1.set_hamiltonian(sum(s1.SS(i, i+1) for i in range(n-1)))
es = s1.get_excited(n=6)
gap = es[4] - es[0]
```

19.2.3 Gaps of a Hubbard chain

Here `fc` is the Hubbard chain above. `get_gap()` is the gap to the first excited state, and `get_charge_gap(d=...)` is $E_0(N+d) - E_0(N)$ (Excited states):

```
fc.get_gap()
fc.get_charge_gap(d=1)
fc.get_charge_gap(d=2)
```

19.2.4 Fidelity susceptibility across the Ising transition

See Fidelity susceptibility.

```
import numpy as np
from dmrgpy import fidelity
h0 = 4*sum(sc.Sz[i]*sc.Sz[i+1] for i in range(n-1))   # Ising coupling
h1 = 2*sum(sc.Sx[i] for i in range(n))                # transverse field
lambdas = np.linspace(0.5, 1.5, 11)
chis = [fidelity.get_fidelity(sc, h0, h1, lam) for lam in lambdas]
```

19.2.5 Dynamical structure factor $S(q,\omega)$

See Dynamical correlators. We compute it from the site-resolved correlators and a Fourier transform, $S(q,\omega) = \frac{1}{n}\sum_{ij} e^{iq(i-j)} S_{ij}(\omega)$:

```
es = np.linspace(-0.5, 3, 200)
q = np.pi
Sqw = 0
for i in range(n):
    for j in range(n):
        x, y = sc.get_dynamical_correlator(submode="KPM", es=es, delta=0.1,
                                           name=(sc.Sz[i], sc.Sz[j]))
        Sqw = Sqw + np.exp(1j*q*(i-j))*y/n
```

19.2.6 Extrapolation to the thermodynamic limit

See Post-processing tools. Here we extrapolate the energy per site computed on several lengths `Ls`:

```
from dmrgpy import extrapolate
e_inf = extrapolate.size_extrapolation(Ls, e_per_site, n=1)
```

20 Post-processing tools

- `extrapolate.size_extrapolation(L, y, n=1)`: fits $y(L)$ with a polynomial of order `n` in $1/L$ and returns the value at $L \to \infty$. Use it for energy densities, gaps and order parameters from finite chains.

The figure shows the ground-state energy per site of open Heisenberg chains, from $L = 16$ to $L = 128$, as a function of $1/L$. On an open chain the edges add an energy of order one, so $E_0/L = e_\infty + a/L + b/L^2$. The quadratic fit (`n=2`) gives $e_\infty = -0.443143$, which differs from the Bethe-ansatz value $1/4 - \ln 2 = -0.443147$ by 4×10^{-6}. The linear fit (`n=1`) misses the curvature and is more than 15 times worse. The inset zooms on $1/L = 0$.

The code of this figure is in `docs/figures/post-processing_extrapolation/`.

- `functionfit.fit(f, x0)`: minimizes a function `f` with several random starts of the Powell method. Use it for fits such as the Calabrese-Cardy formula.
- `analyticcontinuation.imag2real(zs, us)`: Pade continuation of a function known at complex frequencies `zs` to the real axis, for example the output of `submode="CVMimag"` (Choosing a correlator method). The farther `zs` is from the real axis, the more the result is smoothed.
- `reconstruct.reconstruct_distribution`: maximum-entropy reconstruction of a positive spectral function from its moments, used by `submode="maxent"` and `get_distribution_maxent`.

21 Performance

On this page we make a calculation faster without changing its result: one BLAS thread per process, several points of a scan in parallel, and a GPU for the pure-Python backend.

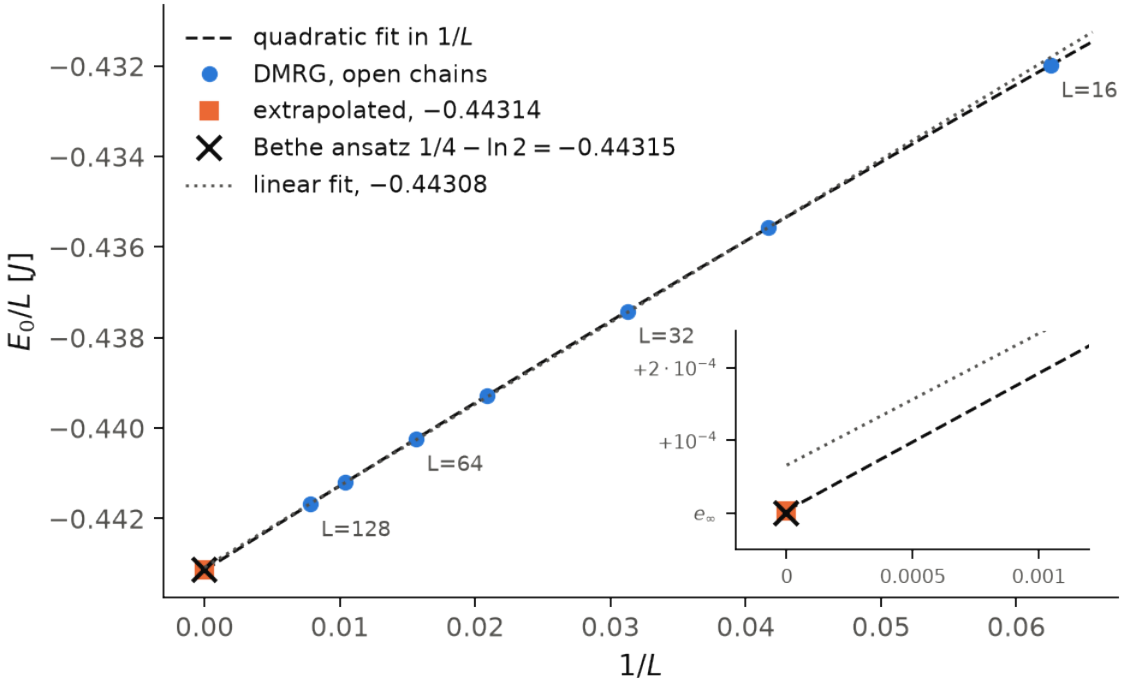


Figure 23: Size extrapolation of the energy per site

21.1 BLAS threads

DMRG makes many small matrix operations, and on small matrices several BLAS threads are slower than one. It matters most when the machine is shared, for example on a cluster node or with several runs in parallel, where a calculation can become 3 to 17 times slower. Pin one thread before Python imports numpy:

```
MKL_NUM_THREADS=1 OMP_NUM_THREADS=1 OPENBLAS_NUM_THREADS=1 python3 myscript.py
```

Inside a script, `blasthreads.limit(1)` does the same for one block. It needs the package `threadpoolctl`, and without it does nothing and gives a warning:

```
from dmrgpy import blasthreads
with blasthreads.limit(1):
    e0 = sc.gs_energy()
```

dmrgpy never changes the number of threads by itself.

21.2 Parallel scans

The points of a parameter scan are independent, so they can run in parallel processes. `parallel.pcall(f, args)` applies `f` to every entry of `args` and returns the list of results, in order:

```
from dmrgpy import parallel
parallel.cores = 4                  # processes; 1 runs in serial

def energy(b):
    sc = spinchain.Spin_Chain(["S=1/2"]*n)
    sc.set_hamiltonian(h0 + b*h1)
    return sc.gs_energy()

es = parallel.pcall(energy, bs)     # one process per point
```

`f` builds its own chain, since a chain holds the state of one solve and is not shared between processes. Pin one BLAS

thread per process as above, or the processes compete for the cores. `parallel.cores` is 1 by default, and the processes need the package `multiprocess`; without it `pcall` runs in serial. See `examples/utilities/parallel`.

21.3 GPU

The `itensor_version="python"` backend can keep its tensors on a GPU. It needs `jax` with CUDA support and a GPU with fast double precision, as in data-centre cards. A consumer card with slow double precision does not give a speedup. The switch is global and is set before building a chain:

```
from dmrgpy.pyitensor import backend
backend.set_backend("jax")        # "numpy", the default, goes back to the CPU
print(backend.device_info())      # for example "jax: cuda:0"
```

The rest of the script does not change, and the results agree with the CPU ones to about 10^{-11}. A GPU helps only when the bond dimension is large: below about 120 it is slower than the CPU, and at 240 it is about 5 times faster. So it pays for real-time evolution, where the bond dimension grows, and rarely for a 1D ground state. Infinite chains and non-Hermitian DMRG run but do not get faster. For a script that does one calculation and exits, `backend.set_pad_bonds(maxm)` fixes all tensor shapes so that each kernel is compiled only once. With padding on, the default `backend.set_jit("auto")` also fuses the kernels. For a KPM correlator, set `kpmmaxm` equal to `maxm`. See `docs/gpu_cpu_performance.md` for the measurements.