Aalto University Department of Applied Physics

PHYS-EV0007 — Machine Learning from and for Quantum Science

Project: Hamiltonian learning of a spin chain
via dynamical impurity tomography

Based on N. Karjalainen et al., arXiv:2510.18613 (2025)

Anouar Moustaj
anouar.moustaj@aalto.fi

Topic nr 3
Project nr 2

Project description

Understand and apply the Hamiltonian-learning methodology of to a *minimal* quantum-magnet problem. Concretely: simulate a small 1D Heisenberg-type spin chain with matrix-product-state / DMRG methods, compute its dynamical response to a single nearby quantum impurity ("dynamical impurity tomography"), build a small training dataset by sweeping the microscopic exchange couplings, and train a neural network to *learn back* those couplings from the impurity spectra.

Learning outcomes

- Understand matrix-product-state / DMRG simulations of 1D spin chains, and learn to use the `DMRGPy` library for ground-state and dynamical-correlator calculations.
- Understand what "dynamical impurity tomography" means in practice: how the local dynamical response near an impurity encodes the parent Hamiltonian.
- Gain working knowledge of the Hamiltonian-learning inverse problem: dataset construction, neural-network architecture choice, training / validation, and robustness to noise.

Presentation

- Explain the formalism used in the paper:
  - Heisenberg-type spin Hamiltonians, dynamical spin correlators, dynamical impurity tomography, and Hamiltonian learning as an inverse problem.
- Explain how DMRG / MPS methods are used as the forward solver in the training-set construction.
- Explain the implementation of the Hamiltonian learning.
- Explain the advantages and limitations of the method.

Simulations

1. **Simulating the spin chain with DMRGPy.** Using the DMRGPy package. Construct a small isotropic Heisenberg chain of length N (large enough to be non-trivial but reasonable on your machine) to with nearest-neighbor exchange J_1 and next-nearest-neighbor exchange J_2,

$$H_{\text{chain}} \;=\; J_1 \sum_{i=1}^{N-1} \vec{S}_i \cdot \vec{S}_{i+1} \;+\; J_2 \sum_{i=1}^{N-2} \vec{S}_i \cdot \vec{S}_{i+2}, \tag{1}$$

with $S = \frac{1}{2}$ on each site. Fix $J_1 = 1$ as the relevant energy scale, and use DMRGPy for simulations.

   a. Compute the ground-state energy and the bond-resolved spin–spin correlator as a heatmap $C_{ij} = \langle \vec{S}_i \cdot \vec{S}_j \rangle$ for a representative J_2. Plot your results

   b. Compute the dynamical spin structure factor

$$S^{\alpha\alpha}(q,\omega) \;=\; \frac{1}{N} \sum_{i,j} e^{-\,iq(i-j)} \, \langle \Omega |\, S_i^{\alpha} \delta(\omega - H_{\text{chain}} - E_\Omega) S_j^{\alpha} \,| \Omega \rangle \,, \qquad \alpha \in \{x, y, z\}, \tag{2}$$

   on a frequency grid $\omega \in [0, \omega_{\max}]$ with the DMRGPy `get_dynamical_correlator` routine. Plot $S^{zz}(q,\omega)$ as a heatmap on the (q,ω) plane for two or three J_2 points and comments on the qualitative differences ((anti)ferromagnetic vs. frustrated regimes).

2. **Dynamical impurity tomography.** Attach a single extra $S = \frac{1}{2}$ impurity, coupled to a chosen site i_0 of the chain with Heisenberg exchange J_{imp},

$$H \;=\; H_{\text{chain}} \;+\; J_{\text{imp}} \, \vec{S}_{\text{imp}} \cdot \vec{S}_{i_0}. \tag{3}$$

   a. Compute the local dynamical correlator $\chi(j,\omega) = \langle \Omega |\, S_{\text{imp}}^{\alpha} \delta(\omega - H - E_\Omega) S_{\text{j}}^{\alpha} \,| \Omega \rangle$. Plot your results and comment on your obsevations.

   b. Study how $\chi(j,\omega)$ varies when J_2 are changed at fixed J_{imp}. Plot your results and confirm visually that the signal is sensitive to both couplings and is therefore an informative probe.

3. **Building the training dataset.** Sweep J_2 on a small grid of physically reasonable values (motivate your choice). For each point, use DMRGPy to compute the impurity tomography signal of the previous step and store it as a feature vector (spectrum discretized on a fixed ω-grid, optionally concatenated over a few probe sites).

   a. Save the full dataset {(spectrum, J_2)} to disk in a simple format (e.g. `.npz`).

   b. Estimate the wall-clock time per DMRGPy call and the total dataset-generation time; plan your grid resolution accordingly.

   c. Visualize a few representative spectra across the grid to confirm that the feature space is smooth in J_2.

4. **Training the neural network to learn J_2.** Train a small feed-forward neural to map spectra to exchange couplings,

$$f_\theta \colon \chi(\omega) \;\mapsto\; J_2. \tag{4}$$

   a. Split the dataset into train / validation / test. Standardize inputs and outputs.

   b. Train with a mean-squared-error loss and Adam; report learning curves.

   c. Evaluate on the test set; plot predicted vs. true J_2 and the Pearson correlation function (the fidelity).

d. Re-train and re-test after adding Gaussian noise (e.g. 1%, 5%, 10%) to the spectra to mimic an experimental linewidth / measurement noise. Plot the test error as a function of the noise level.

5. **Performance benchmarking.**

   a. Measure the wall-clock time of the DMRGPy forward solve as a function of system size N and bond dimension, and the wall-clock time of the neural-network training as a function of dataset size.

   b. Report the final prediction error on the J_2 couplings as a function of training-set size; identify the regime in which more data stops helping.

   c. Generate a small set of "out-of-distribution" test Hamiltonians with J_2 just outside the training grid, and comment on the model's extrapolation behaviour.

   d. Comment on the bottleneck of the full pipeline (forward solver vs. training) and on what would change if you wanted to learn a third coupling (e.g. an anisotropy Δ or a Dzyaloshinskii–Moriya term D).

Deliverables

- A concise report in the form of a presentation.
- A repo with organized code, well-documented, and a notebook containing a reproducible end-to-end run (DMRGPy dataset generation → training → test-set evaluation) for a small system size.

Working practices and tools

You are strongly encouraged to recycle existing material. You are expected to **read, reuse, and adapt** existing reference implementations pipelines rather than re-derive or re-implement every algorithmic detail from scratch; the pedagogical goal of this project is to *understand and apply* the method, not to reproduce boilerplate. Any code you borrow must be clearly attributed in your repository (e.g. in a `README.md` or as in-source comments) and integrated cleanly with your own contributions. *Use of LLMs (ChatGPT, Claude, Gemini, . . .) is permitted and encouraged for onboarding.* In particular, LLMs are very effective at:

- explaining unfamiliar programming syntax and idioms;
- summarising sections of the paper or cross-referencing related literature;
- debugging installation issues, bookkeeping, and programming recommendations;
- generating boilerplate (plotting scripts, parameter sweeps, unit tests, diagnostic helpers).

You remain, however, fully responsible for the correctness, clarity, and scientific content of what you submit: LLM output must be checked, understood, and, where appropriate, cited. Treat an LLM as a capable but occasionally wrong collaborator, not as an oracle.

Programming language: Python.

Packages required: DMRGPy, `numpy`, `scipy`, `matplotlib`, PyTorch or tensorflow, scikit-learn).

References

- N. Karjalainen et al., arXiv:2510.18613 (2025).
- N. Karjalainen, GitHub repo for the paper (2025).
- J. L. Lado, DMRGPy — Python library for DMRG simulations of quasi-1D spin and fermionic systems.