




QMCTorch: a PyTorch Implementation of Real-Space Quantum Monte Carlo Simulations of Molecular Systems

Nicolas Renaud ¹

¹ Netherlands eScience Center, Science Park 402, 1098 XH Amsterdam, The Netherlands

DOI: [10.21105/joss.05472](https://doi.org/10.21105/joss.05472)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Jarvist Moore Frost](#)  

Reviewers:

- [@tonnylou44853](#)
- [@scemama](#)
- [@AbdAmmar](#)

Submitted: 28 April 2023

Published: 14 November 2023

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

Quantum Monte-Carlo (QMC) simulations allow to compute the electronic structure of quantum systems with high accuracy and can be parallelized over large compute resources. QMC relies on the variational principle and optimize a wave function ansatz to minimize the total energy of the quantum system. QMCTorch expresses this optimization process as a machine learning problem where the wave function ansatz is encoded in a physically-motivated neural network. The use of PyTorch as a backend to perform the optimization, allows leveraging automatic differentiation and GPU computing to accelerate the development and deployment of QMC simulations. QMCTorch supports the use of both Gaussian and Slater type orbitals via interface to popular quantum chemistry packages pyscf and ADF.

Statement of need

QMCTorch is a Python package using PyTorch ([Paszke et al., 2019](#)) as a backend to perform Quantum Monte-Carlo (QMC) simulations, namely Variational Monte-Carlo, of molecular systems. Many software such as QMCPack ([Kim et al., 2018](#)), QMC=Chem ([Scemama et al., 2013](#)), CHAMP ([C. Filippi, 2019](#)) provide high-quality implementation of advanced QMC methodologies in low-level languages (C++/Fortran). Python implementations of QMC such as PAUXY ([Fionn Malone, n.d.](#)) and PyQMC ([Wheeler et al., 2023](#)) have also been proposed to facilitate the use and development of QMC techniques. Large efforts have been made to leverage recent development of deep learning techniques for QMC simulations with for example the creation of neural-network based wave-function ansatz ([Choo et al., 2020](#); [Han et al., 2019](#); [Hermann et al., 2020](#); [Inui et al., 2021](#); [Kessler et al., 2021](#); [Lin et al., 2023](#); [Pfau et al., 2020](#); [Schätzle et al., 2021](#); [Yang et al., 2020](#)) that have lead to very interesting results. QMCTorch allows to perform QMC simulations using physically motivated neural network architectures that closely follow the wave function ansatz used by QMC practitioners. Its architecture allows to rapidly explore new functional forms of some key elements of the wave function ansatz. Users do not need to derive analytical expressions for the gradients of the total energy w.r.t. the variational parameters, that are simply obtained via automatic differentiation. This includes for example the parameters of the atomic orbitals that can be variationally optimized and the atomic coordinates that allows QMCTorch to perform geometry optimization of molecular structures. In addition, the GPU capabilities offered by PyTorch combined with the parallelization over multiple computing nodes obtained via Horovod ([Sergeev & Balso, 2018](#)), allow to deploy the simulations on large heterogenous computing architectures. In summary, QMCTorch provides QMC practitioners a framework to rapidly prototype new ideas and to test them using modern computing resources.

Wave Function Ansatz

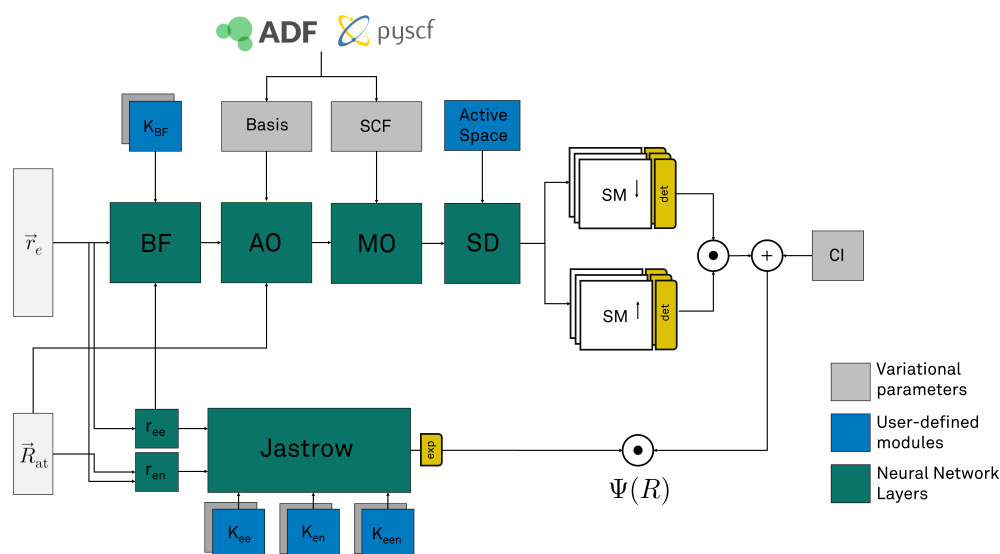


Figure 1: General architecture of the neural network used by QMCTorch to encode the wave function ansatz. The neural network computes and assembles the different elements of the wave function ansatz and can be used to compute the electronic density required for the sampling and the total energy of the system required for the wave function optimization.

The neural network used to encode the wave-function ansatz used in QMCTorch is shown in Fig. 1. As common in QMC simulations, the wave function is given by the product of a Jastrow factor, $J(r)$, that accounts for electronic correlations and a sum of Slater determinants, $D_n^\dagger(r_\uparrow)$, built over the molecular orbitals of the spin up and down electrons:

$$\Psi(r) = J(r) \sum_n c_n D_n^\dagger(r_\uparrow) D_n^\dagger(r_\downarrow).$$

Jastrow Factor The Jastrow layer computes the sum of three components: an electron-electron term K_{ee} ; an electron-nuclei term K_{en} ; and a three body electron-electron-nuclei term K_{een} . The sum is then exponentiated to give the Jastrow factor: $J(r_{ee}, r_{en}) = \exp(K_{ee}(r_{ee}) + K_{en}(r_{en}) + K_{een}(r_{ee}, r_{en}))$ where r_{ee} and r_{en} are the electron-electron and electron-nuclei distances. Several well-known Jastrow factor functional forms, as for example the electron-electron Pade-Jastrow: $K(r_{ee}) = \frac{\omega_0 r_{ee}}{1 + \omega r_{ee}}$, where ω is a variational parameter, are already implemented and available for use. Users can also define their own functional forms for the different kernel functions, K , and explore their effects on the resulting optimization.

Backflow Transformation The backflow transformation layer, BF, creates quasi-particles by mixing the electronic positions of the electrons: $\mathbf{q}_i = \mathbf{r}_i + \sum_{j \neq i} K_{BF}(r_{ij})(\mathbf{r}_i - \mathbf{r}_j)$ (Holzmann & Moroni, 2019; Schmidt et al., 1981). Well-known transformations such as: $K_{BF} = \frac{\mu}{r_{ij}}$ where μ is a variational parameter, are already implemented and ready to use. Users can also easily specify the kernel of the backflow transformation, K_{BF} to explore its impact on the wave function optimization.

Atomic Orbitals The Atomic Orbital layer AO computes the values of the different atomic orbitals of the system at all the positions \mathbf{q}_e . Both Slater type orbitals (STOs) and Gaussian type orbitals (GTOs) are supported. The initial parameters of the AOs are extracted from popular quantum chemistry codes, pyscf (Sun et al., 2017) and ADF (Velde et al., 2001). During the optimization, the parameters of the AOs (exponents, coefficients) are variational parameters that can be optimized to minimize the total energy. Since GTOs can introduce

a significant amount of noise in the QMC simulations, QMCTorch offers the possibility to fit GTOs to single exponent STOs.

Molecular Orbitals The Molecular Orbital layer, MO, computes the values of all the MOs at the positions of the quasi particles. The MO layer is a simple linear transformation defined by $MO = AO \cdot W_{SCF}^T$, where W_{SCF}^T is the matrix of the MOs coefficients on the AOs. The initial values of these coefficients are obtained from a Hartree-Fock (HF) or Density Functional Theory (DFT) calculation of the system via pycscf or ADF. These coefficients are then variational parameters that can be optimized to minimize the total energy of the system.

Slater Determinants The Slater determinants layer, SD, extracts the spin up/down matrices of the different electronic configurations specified by the user. Users can freely define the number of electrons as well as the number and types of excitations they want to include in the definition of their wave function ansatz. The SD layer will extract the corresponding matrices, multiply their determinants and sum all the terms. The CI coefficients of the sum can be freely initialized and optimized to minimize the total energy.

The Jastrow factor and the sum of Slater determinants are then multiplied to yield the final value of the wave function calculated for the electronic and atomic positions $\Psi(R)$ with $R = \{r_e, R_{at}\}$. Note that the backflow transformation and Jastrow factor are optional and can be individually removed from the definition of the wave function ansatz.

Sampling, Cost Function & Optimization

QMC simulations use samples of the electronic density to approximate the total energy of the system. In QMCTorch, Markov-Chain Monte-Carlo (MCMC) techniques, namely Metropolis-Hasting and Hamiltonian Monte-Carlo, are used to obtain those samples. Each sample, R_i , contains the positions of all the electrons contained in the system. MCMC techniques require the calculation of the density for a given positions of the electrons: $\Pi(R_i) = |\Psi(R_i)|^2$ that can simply be obtained by squaring the result of a forward pass of the network described above.

The value of local energy of the system is then computed at each sampling point and these values are summed up to compute the total energy of the system: $E = \sum_i \frac{H\Psi(R_i)}{\Psi(R_i)}$, where H is the Hamiltonian of the molecular system: $H = -\frac{1}{2} \sum_i \Delta_i + V_{ee} + V_{en}$, with Δ_i the Laplacian w.r.t the i -th electron, V_{ee} the coulomb potential between the electrons and V_{en} the electron-nuclei potential. In QMCTorch, the calculation of the Laplacian of the Slater determinants can be performed using automatic differentiation but analytical expressions have also been implemented as they are computationally more robust and less expensive (Claudia Filippi et al., 2016). The gradients of the total energy w.r.t the variational parameters of the wave function, i.e. $\frac{\partial E}{\partial \theta_i}$ are simply obtained via automatic differentiation. Thanks to this automatic differentiation, users can define new kernels for the backflow transformation and Jastrow factor without having to derive analytical expressions of the energy gradients.

Any optimizer included in PyTorch (or compatible with it) can then be used to optimize the wave function. This gives users access to a wide range of optimization techniques that they can freely explore for their own use cases. Users can also decide to freeze certain variational parameters or define different learning rates for different layers. Note that the positions of atoms are also variational parameters, and therefore one can perform geometry optimization using QMCTorch. At the end of the optimization, all the information relative to the simulations is dumped in a dedicated HDF5 file to enhance reproducibility of the simulations.

Example

```
from torch import optim
from qmctorch.scf import Molecule
```

```
from qmctorch.wavefunction import SlaterJastrow
from qmctorch.solver import Solver
from qmctorch.sampler import Metropolis

# define the molecule
mol = Molecule(atom='H 0 0 -0.69; H 0 0 0.69', calculator='pyscf', basis='sto-3g')

# define the wave function ansatz
wf = SlaterJastrow(mol, configs='single_double(2,2)').gto2sto()

# define a Metropolis Hasting Sampler
sampler = Metropolis(nwalkers=5000, nstep=200, nelec=wf.nelec, init=mol.domain('atomic'))

# define the optimizer
opt = optim.Adam(wf.parmaters(), lr=1E-3)

# define the solver
solver = Solver(wf=wf, sampler=sampler, optimizer=opt)

# optimize the wave function
obs = solver.run(50)
```

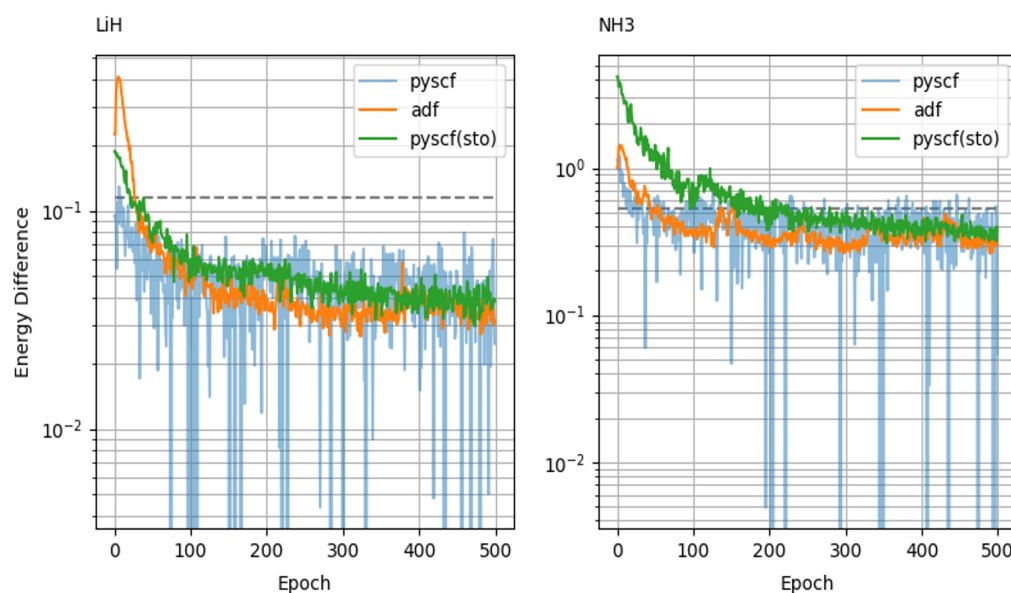


Figure 2: Result of the optimization of the wave function of LiH and NH3 using atomic atomic orbitals provided by pyscf, ADF and also a STO fit of the pyscf atomic orbitals. The vertical axis shows the difference between the variational energy and the true ground state energy. The horizontal dashed line indicate the Hartree-Fock energy.

The snippet of code above shows a typical example of QMCTorch script. A `Molecule` object is first created by specifying the atomic positions and the calculator required to run the HF or DFT calculations (here `pyscf` using a `sto-3g` basis set). This molecule is then used to create a `SlaterJastrow` wave function ansatz. Other options, such as the required Jastrow kernel, active space, and the use of GPUs can also be specified here. A sampler and optimizer are then defined that are then used with the wave function to instantiate the solver. This solver can then be used to optimize the variational parameters, that is done here through 50 epochs.

Fig. 2 shows typical optimization runs for two different molecular structures, LiH and NH₃ using atomic orbitals provided by pycsf, ADF and also a STO fit of the pycsf atomic orbitals. As seen in this figure, the variance of the local energy values obtained with the GTOs provided by pycsf is a limiting factor for the optimization. A simple STO fit of these atomic orbitals leads to variance comparable to those obtained with the STOs of ADF.

Acknowledgements

We acknowledge contributions from Felipe Zapata, Matthijs de Witt and guidance from Claudia Filippi. The development of the code was done during the project “A Light in the Dark: quantum Monte-Carlo meets energy conversion” from the Joint CSER and eScience program for Energy Research (JCER2017) funded by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO) and the Netherlands eScience Center, project number CSER.JCER.022

References

- Choo, K., Mezzacapo, A., & Carleo, G. (2020). Fermionic neural-network states for ab-initio electronic structure. *Nature Communications*, 11(1), 2368. <https://doi.org/10.1038/s41467-020-15724-9>
- Filippi, C. (2019). *CHAMP*. <https://github.com/filippi-claudia/champ>
- Filippi, Claudia, Assaraf, R., & Moroni, S. (2016). Simple formalism for efficient derivatives and multi-determinant expansions in quantum monte carlo. *The Journal of Chemical Physics*, 144(19), 194105. <https://doi.org/10.1063/1.4948778>
- Fionn Malone, J. M. F., Joohno Lee. (n.d.). *PAUXY*. <https://github.com/pauxy-qmc/pauxy>
- Han, J., Zhang, L., & E, W. (2019). Solving many-electron schrödinger equation using deep neural networks. *Journal of Computational Physics*, 399, 108929. <https://doi.org/10.1016/j.jcp.2019.108929>
- Hermann, J., Schätzle, Z., & Noé, F. (2020). Deep-neural-network solution of the electronic schrödinger equation. *Nature Chemistry*, 12(10), 891–897. <https://doi.org/10.1038/s41557-020-0544-y>
- Holzmam, M., & Moroni, S. (2019). Orbital-dependent backflow wave functions for real-space quantum monte carlo. *Physical Review B*, 99(8). <https://doi.org/10.1103/physrevb.99.085121>
- Inui, K., Kato, Y., & Motome, Y. (2021). Determinant-free fermionic wave function using feed-forward neural networks. *Phys. Rev. Res.*, 3, 043126. <https://doi.org/10.1103/PhysRevResearch.3.043126>
- Kessler, J., Calcavecchia, F., & Kühne, T. D. (2021). Artificial neural networks as trial wave functions for quantum monte carlo. *Advanced Theory and Simulations*, 4(4), 2000269. <https://doi.org/10.1002/adts.202000269>
- Kim, J., Baczewski, A. D., Beaudet, T. D., Benali, A., Bennett, M. C., Berrill, M. A., Blunt, N. S., Borda, E. J. L., Casula, M., Ceperley, D. M., Chiesa, S., Clark, B. K., Clay, R. C., Delaney, K. T., Dewing, M., Esler, K. P., Hao, H., Heinonen, O., Kent, P. R. C., ... Zhao, L. (2018). QMCPACK: An open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules and solids. *Journal of Physics: Condensed Matter*, 30(19), 195901. <https://doi.org/10.1088/1361-648X/aab9c3>
- Lin, J., Goldshlager, G., & Lin, L. (2023). Explicitly antisymmetrized neural network layers for variational monte carlo simulation. *Journal of Computational Physics*, 474, 111765. <https://doi.org/10.1016/j.jcp.2022.111765>

- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). *PyTorch: An imperative style, high-performance deep learning library*. <https://arxiv.org/abs/1912.01703>
- Pfau, D., Spencer, J. S., G. Matthews, A. G. de, & Foulkes, W. M. C. (2020). Ab-initio solution of the many-electron schrödinger equation with deep neural networks. *Phys. Rev. Research*, 2, 033429. <https://doi.org/10.1103/PhysRevResearch.2.033429>
- Scemama, A., Caffarel, M., Oseret, E., & Jalby, W. (2013). QMC=chem: A quantum monte carlo program for large-scale simulations in chemistry at the petascale level and beyond. In M. Dayde, O. Marques, & K. Nakajima (Eds.), *High performance computing for computational science - VECPAR 2012* (pp. 118–127). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-38718-0_14
- Schätzle, Z., Hermann, J., & Noé, F. (2021). Convergence to the fixed-node limit in deep variational Monte Carlo. *The Journal of Chemical Physics*, 154(12), 124108. <https://doi.org/10.1063/5.0032836>
- Schmidt, K. E., Lee, M. A., Kalos, M. H., & Chester, G. V. (1981). Structure of the ground state of a fermion fluid. *Phys. Rev. Lett.*, 47, 807–810. <https://doi.org/10.1103/PhysRevLett.47.807>
- Sergeev, A., & Balso, M. D. (2018). *Horovod: Fast and easy distributed deep learning in TensorFlow*. <https://arxiv.org/abs/1802.05799>
- Sun, Q., Berkelbach, T. C., Blunt, N. S., Booth, G. H., Guo, S., Li, Z., Liu, J., McClain, J., Sayfutyarova, E. R., Sharma, S., Wouters, S., & Chan, G. K.-L. (2017). *The python-based simulations of chemistry framework (PySCF)*. <https://arxiv.org/abs/1701.08223>
- Velde, G. te, Bickelhaupt, F. M., Baerends, E. J., Fonseca Guerra, C., Gisbergen, S. J. A. van, Snijders, J. G., & Ziegler, T. (2001). Chemistry with ADF. *J. Comput. Chem.*, 22(9), 931–967. <https://doi.org/10.1002/jcc.1056>
- Wheeler, W. A., Pathak, S., Kleiner, K. G., Yuan, S., Rodrigues, J. o. N. B., Lorsung, C., Krongchon, K., Chang, Y., Zhou, Y., Busemeyer, B., Williams, K. T., Muñoz, A., Chow, C. Y., & Wagner, L. K. (2023). PyQMC: An all-python real-space quantum monte carlo module in PySCF. *The Journal of Chemical Physics*, 158(11), 114801. <https://doi.org/10.1063/5.0139024>
- Yang, P.-J., Sugiyama, M., Tsuda, K., & Yanai, T. (2020). Artificial neural networks applied as molecular wave function solvers. *Journal of Chemical Theory and Computation*, 16(6), 3513–3529. <https://doi.org/10.1021/acs.jctc.9b01132>