The Journal of Open Source Software

torchquad: Numerical Integration in Arbitrary Dimensions with PyTorch

Pablo Gómez[*1], **Håvard Hem Toftevaag**[1], **and Gabriele Meoni**[1]

1 Advanced Concepts Team, European Space Agency, Noordwijk, The Netherlands

DOI: 10.21105/joss.03439

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @felipefr
- @scarlehoff

Submitted: 15 June 2021
Published: 31 August 2021

Summary

torchquad is a Python module for n-dimensional numerical integration optimized for graphics processing units (GPUs). Various deterministic and stochastic integration methods, such as Newton-Cotes formulas and Monte Carlo integration methods like VEGAS Enhanced (Lepage, 2020), are available for computationally efficient integration for arbitrary dimensionality n_{d}. As it is implemented using PyTorch (Paszke et al., 2019), one of the most popular machine learning frameworks, torchquad provides fully automatic differentiation throughout the integration, which is essential for many machine learning applications.

Statement of Need

Multidimensional integration is needed in many fields, such as physics (ranging from particle physics (Kersevan & Richter-Was, 2013) to astrophysics (Izzo & Gómez, 2021)), applied finance (Campolieti & Makarov, 2007), medical statistics (Ray et al., 2011), and machine learning (Atay & Hutt, 2006). Most of the conventional Python packages for multidimensional integration, such as quadpy (Schlömer et al., 2021) and nquad (Virtanen et al., 2020), only target and are optimized for central processing units (CPUs). However, as many numerical integration methods are embarrassingly parallel, GPUs can offer superior computational performance in their computation. Furthermore, numerical integration methods typically suffer from the so-called *curse of dimensionality* (Wu et al., 2020). This phenomenon refers to the fact that the computational complexity of the integration grows exponentially with the number of dimensions (Bellman, 2003). Reducing the error of the integration value requires increasing the number of function evaluation points N exponentially, which significantly increases the runtime of the computation, especially for higher dimensions. Previous work has demonstrated that this problem can be mitigated by leveraging the *single instruction, multiple data* parallelization of GPUs (Wu et al., 2020).

Although GPU-based implementations for multidimensional numerical integration in Python exist, some of these packages do not allow fully automatic differentiation (Borowka et al., 2019), which is crucial for many machine learning applications (Baydin et al., 2018). Recently, to fill this gap, the packages VegasFlow (Carrazza & Cruz-Martinez, 2020) and ZMCintegral (Wu et al., 2020) were developed. Both of these implementations are, however, based on TensorFlow (Abadi et al., 2016), and there are currently no packages available that enable more than one-dimensional integration in PyTorch. Additionally, the available GPU-based Python packages that allow fully automatic differentiation rely solely on Monte Carlo methods (Carrazza & Cruz-Martinez, 2020; Wu et al., 2020). Even though such methods offer good speed–accuracy trade-offs for problems of high dimensionality n_{d}, the

[*]corresponding author

efficiency of deterministic methods, such as the Newton-Cotes formulas, is often superior for lower dimensionality (Lepage, 1978).

In summary, to the authors' knowledge, torchquad is the first PyTorch-based module for n-dimensional numerical integration. Furthermore, it incorporates several deterministic and stochastic methods, including Newton-Cotes formulas and VEGAS Enhanced, which allow obtaining high-accuracy estimates for varying dimensionality at configurable computational cost as controlled by the maximum number of function evaluations N. It is, to the authors' knowledge, also the first GPU-capable implementation of VEGAS Enhanced (Lepage, 2020), which improves on its predecessor VEGAS by introducing an adaptive stratified sampling strategy.

Finally, being PyTorch-based, torchquad is fully differentiable, extending its applicability to use cases such as those in machine learning. In these applications, it is typically necessary to compute the gradient of some parameters with regard to input variables to perform updates of the trainable parameters in the machine learning model. With torchquad, e.g., the employed loss function can contain integrals without breaking the automatic differentiation required for training.

Implemented Integration Methods

torchquad features fully vectorized implementations of various deterministic and stochastic methods to perform n-dimensional integration over cubical domains. In particular, the following deterministic integration methods are available in torchquad (version 0.2.1):

- Trapezoid Rule (Sag & Szekeres, 1964)
- Simpson's Rule (Sag & Szekeres, 1964)
- Boole's Rule (Ubale, 2012)

The stochastic integration methods implemented in torchquad so far are:

- Classic Monte Carlo Integrator (Caflisch, 1998)
- VEGAS Enhanced (VEGAS+) integration method (Lepage, 2020)

The functionality and the convergence of all the methods are ensured through automatic unit testing, which relies on an extensible set of different test functions. Both single and double precision are supported to allow different trade-offs between accuracy and memory utilization. Even though it is optimized for GPUs, torchquad can also be employed without a GPU without any functional limitations.

Installation & Contribution

The torchquad package is implemented in Python 3.8 and is openly available under a GPL-3 license. Installation with either pip (PyPi)[1] or conda[2] is available. Our public GitHub repository[3] provides users with direct access to the main development branch. Users wishing to contribute to torchquad can submit issues or pull requests to our GitHub repository following the contribution guidelines outlined there.

[1] torchquad package on PyPi, https://pypi.org/project/torchquad/
[2] torchquad package on conda, https://anaconda.org/conda-forge/torchquad
[3] torchquad GitHub repository, https://github.com/esa/torchquad

Tutorials

The `torchquad` documentation, hosted on `Read the Docs`,[4] provides some examples of the use of `torchquad` for one-dimensional and multidimensional integration utilizing a variety of the implemented methods.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., … Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, 265–283. ISBN: 9781931971331

Atay, F. M., & Hutt, A. (2006). Neural Fields with Distributed Transmission Speeds and Long-Range Feedback Delays. *SIAM Journal on Applied Dynamical Systems*, *5*(4), 670–698. https://doi.org/10.1137/050629367

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic Differentiation in Machine Learning: a Survey. *Journal of Machine Learning Research*, *18*(153), 1–43. http://jmlr.org/papers/v18/17-468.html

Bellman, R. E. (2003). *Dynamic Programming (Dover Books on Computer Science)*. Dover Publications. ISBN: 0486428095

Borowka, S., Heinrich, G., Jahn, S., Jones, S. P., Kerner, M., & Schlenk, J. (2019). A GPU compatible quasi-Monte Carlo integrator interfaced to pySecDec. *Computer Physics Communications*, *240*, 120–137. https://doi.org/10.1016/j.cpc.2019.02.015

Caflisch, R. E. (1998). Monte Carlo and quasi-Monte Carlo methods. *Acta Numerica*, *7*, 1–49. https://doi.org/10.1017/S0962492900002804

Campolieti, G., & Makarov, R. (2007). Pricing path-dependent options on state dependent volatility models with a Bessel bridge. *International Journal of Theoretical and Applied Finance*, *10*(01), 51–88. https://doi.org/10.1142/s0219024907004081

Carrazza, S., & Cruz-Martinez, J. M. (2020). VegasFlow: accelerating Monte Carlo simulation across multiple hardware platforms. *Computer Physics Communications*, *254*, 107376. https://doi.org/10.1016/j.cpc.2020.107376

Izzo, D., & Gómez, P. (2021). *Geodesy of irregular small bodies via neural density fields: geodesyNets*. http://arxiv.org/abs/2105.13031

Kersevan, B. P., & Richter-Was, E. (2013). The Monte Carlo event generator AcerMC versions 2.0 to 3.8 with interfaces to PYTHIA 6.4, HERWIG 6.5 and ARIADNE 4.1. *Computer Physics Communications*, *184*(3), 919–985. https://doi.org/10.1016/j.cpc.2012.10.032

Lepage, G. P. (1978). A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, *27*(2), 192–203. https://doi.org/10.1016/0021-9991(78)90004-9

Lepage, G. P. (2020). *Adaptive Multidimensional Integration: VEGAS Enhanced*. https://doi.org/10.1016/j.jcp.2021.110386

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, 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. In H. Wallach, H. Larochelle, A.

[4]`torchquad` documentation on `Read the Docs`, https://torchquad.readthedocs.io/

Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Ray, J., Marzouk, Y. M., & Najm, H. N. (2011). A Bayesian approach for estimating bioterror attacks from patient data. *Statistics in Medicine*, *30*(2), 101–126. https://doi.org/10.1002/sim.4090

Sag, T. W., & Szekeres, G. (1964). Numerical Evaluation of High-Dimensional Integrals. *Mathematics of Computation*, *18*(86), 245–253. https://doi.org/10.2307/2003298

Schlömer, N., Papior, N., Arnold, D., & Zetter, R. (2021). *Nschloe/quadpy v0.16.6* (Version v0.16.6) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.4519699

Ubale, P. V. (2012). Numerical Solution of Boole's rule in Numerical Integration By Using General Quadrature Formula. *Bulletin of Society for Mathematical Services and Standards*, *2*(1), 1–5. https://doi.org/10.18052/www.scipress.com/BSMaSS.2.1

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wu, H.-Z., Zhang, J.-J., Pang, L.-G., & Wang, Q. (2020). ZMCintegral: A package for multi-dimensional Monte Carlo integration on multi-GPUs. *Computer Physics Communications*, *248*, 106962. https://doi.org/10.1016/j.cpc.2019.106962