The Journal of Open Source Software

PyArmadillo: a streamlined linear algebra library for Python

Jason Rumengan[1, 2], **Terry Yue Zhuo**[3], **and Conrad Sanderson**[1, 4]

1 Data61/CSIRO, Australia **2** Queensland University of Technology, Australia **3** University of New South Wales, Australia **4** Griffith University, Australia

DOI: 10.21105/joss.03051

Software

- Review
- Repository
- Archive

Editor: Matthew Sottile

Reviewers:

- @JaroslavHron
- @uellue

Submitted: 10 February 2021
Published: 15 October 2021

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

Summary

PyArmadillo is a linear algebra library for the Python language, with the aim of closely mirroring the programming interface of the widely used Armadillo C++ library, which in turn is deliberately similar to Matlab. PyArmadillo hence facilitates algorithm prototyping with Matlab-like syntax directly in Python, and relatively straightforward conversion of PyArmadillo-based Python code into performant Armadillo-based C++ code. The converted code can be used for purposes such as speeding up Python-based programs in conjunction with pybind11 (Jakob et al., 2017), or the integration of algorithms originally prototyped in Python into larger C++ codebases.

PyArmadillo provides objects for matrices and cubes, as well as over 200 associated functions for manipulating data stored in the objects. Integer, floating point and complex numbers are supported. Various matrix factorisations are provided through integration with LAPACK (Anderson et al., 1999), or one of its high performance drop-in replacements such as Intel MKL (Intel, 2016) or OpenBLAS (Zhang et al., 2016).

Statement of Need

Armadillo is a popular linear algebra and scientific computing library for the C++ language (Sanderson & Curtin, 2018, 2016) that has three main characteristics: (i) a high-level programming interface deliberately similar to Matlab, (ii) an expression evaluator (based on template meta-programming) that automatically combines several operations to increase speed and efficiency, and (iii) an efficient mapper between mathematical expressions and low-level BLAS/LAPACK functions (Psarras et al., 2021). Matlab is widely used in both industrial and academic contexts, providing a programming interface that allows mathematical expressions to be written in a concise and natural manner (Linge & Langtangen, 2016), especially in comparison to directly using low-level libraries such as LAPACK (Anderson et al., 1999). In industrial settings, algorithms are often first prototyped in Matlab, before conversion into another language, such as C++, for the purpose of integration into products. The similarity of the programming interfaces between Armadillo and Matlab facilitates direct prototyping in C++, as well as the conversion of research code into production environments. Armadillo is also often used for implementing performance critical parts of software packages running under the R environment for statistical computing (R Core Team, 2020), via the RcppArmadillo bridge (Eddelbuettel & Sanderson, 2014).

Over the past few years, Python has become popular for data science and machine learning. This partly stems from a rich ecosystem of supporting frameworks and packages, as well as lack of licensing costs in comparison to Matlab. Python allows relatively quick prototyping of algorithms, aided by its dynamically typed nature and the interpreted execution of user code,

avoiding time-consuming compilation into machine code. However, for the joint purpose of algorithm prototyping and deployment, the flexibility of Python comes with two main issues: (i) slow execution speed due to the interpreted nature of the language, (ii) difficulty with integration of code written in Python into larger programs and/or frameworks written in another language. The first issue can be somewhat addressed through conversion of Python-based code into the low-level Cython language (Behnel et al., 2020). However, since Cython is closely tied with Python, conversion of Python code into C++ may be preferred as it also addresses the second issue, as well as providing a higher-level of abstraction.

PyArmadillo is aimed at: (i) users that prefer compact Matlab-like syntax rather than the somewhat more verbose syntax provided by NumPy/SciPy (Harris et al., 2020; Virtanen et al., 2020), and (ii) users that would like a straightforward conversion path to performant C++ code. More specifically, PyArmadillo aims to closely mirror the programming interface of the Armadillo library, thereby facilitating the prototyping of algorithms with Matlab-like syntax directly in Python. Furthermore, PyArmadillo-based Python code can be easily converted into high-performance Armadillo-based C++ code. Due to the similarity of the programming interfaces, the risk of introducing bugs in the conversion process is considerably reduced. Moreover, conversion into C++ based code allows taking advantage of expression optimisation performed at compile-time by Armadillo, resulting in further speedups. The resulting code can be used in larger C++ programs, or used as a replacement of performance critical parts within a Python program with the aid of the pybind11 interface layer (Jakob et al., 2017).

Functionality

PyArmadillo provides matrix objects for several distinct element types: integers, single- and double-precision floating point numbers, as well as complex numbers. In addition to matrices, PyArmadillo also has support for cubes (3 dimensional arrays), where each cube can be treated as an ordered set of matrices. Multi-dimensional arrays beyond 3 dimensions are explicitly beyond the scope of PyArmadillo. Over 200 functions are provided for manipulating data stored in the objects, covering the following areas: fundamental arithmetic operations, contiguous and non-contiguous submatrix views, diagonal views, element-wise functions, scalar/vector/matrix valued functions of matrices, generation of various vectors/matrices, statistics, signal processing, storage of matrices in files, matrix decompositions/factorisations, matrix inverses, and equation solvers. See the online documentation at https://pyarma.sourceforge.io/docs.html for details. PyArmadillo matrices and cubes are convertible to/from NumPy arrays, allowing users to tap into the wider Python data science ecosystem, including plotting tools such as Matplotlib (Hunter, 2007).

Implementation

PyArmadillo relies on pybind11 (Jakob et al., 2017) for interfacing C++ and Python, as well as on Armadillo for the underlying C++ implementation of matrix objects and associated functions. Due to its expressiveness and relatively straightforward use, pybind11 was selected over other interfacing approaches such as Boost.Python (Abrahams & Grosse-Kunstleve, 2003) and manually writing C++ extensions for Python. In turn, Armadillo interfaces with low-level routines in BLAS and LAPACK (Anderson et al., 1999), where BLAS is used for matrix multiplication, and LAPACK is used for various matrix decompositions/factorisations and equation solvers. As the low-level routines in BLAS and LAPACK are considered as the *de facto* standard for numerical linear algebra, it is possible to use high performance drop-in replacements such as Intel MKL (Intel, 2016) and OpenBLAS (Zhang et al., 2016).

PyArmadillo is open-source software, distributed under the Apache 2.0 license (Apache Software Foundation, 2004), making it useful in both open-source and proprietary (closed-source)

contexts (St. Laurent, 2008). It can be obtained at https://pyarma.sourceforge.io or via the Python Package Index in precompiled form.

Acknowledgements

We would like to thank our colleagues at Data61/CSIRO (Dan Pagendam, Dan Gladish, Andrew Bolt, Piotr Szul) for providing feedback and testing.

References

Abrahams, D., & Grosse-Kunstleve, R. W. (2003). Building hybrid systems with Boost.Python. *C/C++ Users Journal*, *21*(7). https://www.osti.gov/biblio/815409

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., & Sorensen, D. (1999). *LAPACK users' guide*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898719604

Apache Software Foundation. (2004). *Apache license 2.0*. http://www.apache.org/licenses/LICENSE-2.0

Behnel, S., Bradshaw, R., Dalcín, L., Florisson, M., Makarov, V., & Seljebotn, D. S. (2020). *Cython*. https://cython.org

Eddelbuettel, D., & Sanderson, C. (2014). RcppArmadillo: Accelerating R with high-performance C++ linear algebra. *Computational Statistics & Data Analysis*, *71*, 1054–1063. https://doi.org/10.1016/j.csda.2013.02.005

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., R'ıo, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Intel. (2016). *Math Kernel Library (MKL)*. http://software.intel.com/en-us/intel-mkl/

Jakob, W., Rhinelander, J., & Moldovan, D. (2017). pybind11 – seamless operability between C++11 and Python. In *GitHub repository*. GitHub. https://github.com/pybind/pybind11

Linge, S., & Langtangen, H. P. (2016). *Programming for computations - MATLAB/Octave*. Springer. https://doi.org/10.1007/978-3-319-32452-4

Psarras, C., Barthels, H., & Bientinesi, P. (2021). The linear algebra mapping problem: Current state of linear algebra languages and libraries. *arXiv:1911.09421v2*. https://arxiv.org/abs/1911.09421

R Core Team. (2020). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org

Sanderson, C., & Curtin, R. (2018). A user-friendly hybrid sparse matrix class in C++. *Lecture Notes in Computer Science (LNCS), Vol. 10931*, 422–430. https://doi.org/10.1007/978-3-319-96418-8_50

Sanderson, C., & Curtin, R. (2016). Armadillo: A template-based C++ library for linear algebra. *Journal of Open Source Software*, *1*, 26. https://doi.org/10.21105/joss.00026

St. Laurent, A. (2008). *Understanding open source and free software licensing*. O'Reilly Media.

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

Zhang, X., Wang, Q., & Saar, W. (2016). OpenBLAS: An optimized BLAS library. In *GitHub repository*. GitHub. https://github.com/xianyi/openblas