The Journal of Open Source Software

BetterFFTW: A High-Performance, User-Friendly Wrapper for FFTW in Python

Sebastian Griego [1]

1 San Diego State University

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

BetterFFTW is a GPL-3.0–licensed Python library that offers a high-level interface to the FFTW C library by wrapping pyFFTW. It provides drop-in replacements for NumPy and SciPy FFT functions with optimized defaults. By abstracting away FFTW's often complex configuration (such as planner flag management and thread selection), BetterFFTW makes high-performance FFT computation accessible to non-expert users.

Statement of Need

Fourier transforms are essential for many scientific and engineering applications. While Python's NumPy and SciPy FFT functions are easy to use, they typically lack the performance benefits afforded by FFTW—a library renowned for its speed and efficiency when compared with traditional algorithms (Frigo & Johnson, 2005). Although pyFFTW exposes FFTW's capabilities to Python, its default settings are often suboptimal. BetterFFTW fills this gap by automating plan caching, thread management, and planner flag selection. This approach ensures that while the initial call may incur FFTW's planning overhead, subsequent FFTs benefit from high-amortized efficiency—particularly for multi-dimensional transforms where speedups can exceed 10×.

Implementation

BetterFFTW is implemented as a thin, lightweight layer over pyFFTW (Contributors, n.d.). It mirrors the APIs of NumPy's and SciPy's FFT modules and leverages Python's uarray protocol for seamless integration. FFT plans are cached based on transform parameters. The library inspects input array sizes and dimensionality to automatically determine optimal thread counts, mitigating the overhead of excessive threading on small arrays. When testing indicates that NumPy's FFT may be faster (especially for one-off or very small transforms), BetterFFTW will automatically revert to the default implementation. Users benefit from improved defaults without manual tuning.

Performance

Benchmarks show that BetterFFTW is, on average, nearly 4× faster than NumPy's FFT functions for common use cases. For multi-dimensional transforms and scenarios where the same transform is applied repeatedly, speedups can exceed 10× once the planning overhead is amortized. These improvements arise from FFTW's efficient algorithms and the automatic reuse of highly optimized FFT plans. BetterFFTW's performance approach is similar to that of FFTW itself, and while a one-off transform may bear a planning cost, the benefits become

evident in iterative computations—exactly the context where FFTW (and thus BetterFFTW) excels (Contributors, n.d.; Frigo & Johnson, 2005).

Acknowledgements

The development of BetterFFTW was inspired by FFTW's renowned efficiency and by pyFFTW's efforts to bring FFTW to Python. Many thanks to Matteo Frigo and Steven G. Johnson for creating FFTW (Frigo & Johnson, 2005) and to the maintainers of pyFFTW (Contributors, n.d.). BetterFFTW also benefits from the robust ecosystems of NumPy and SciPy. Feedback from early adopters has been invaluable for refining its default settings and usability.

References

Contributors, pyFFTW. (n.d.). *pyFFTW: A pythonic interface to FFTW*. https://github.com/pyFFTW/pyFFTW.

Frigo, M., & Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE*, *93*(2), 216–231. https://doi.org/10.1109/JPROC.2005.846062