# pylevin: efficient numerical integration of integrals containing up to three Bessel functions

**Robert Reischke** [1¶]

**1** Argelander Institut fuer Astronomie ¶ Corresponding author

**DOI:** 10.xxxxxx/draft

**Software**

- Review
- Repository
- Archive

**Editor:** Neea Rusch

**Reviewers:**

- @mithun218
- @KumarSaurabh1992

**Submitted:** 24 February 2025
**Published:** unpublished

## Summary

Bessel functions naturally occur in physical systems with some degree of rotational symmetry. Theoretical predictions of observables therefore often involve integrals over those functions which are not solvable analytically and have to be treated numerically instead. However, standard integration techniques like quadrature generally fail to solve these types of integrals efficiently and reliably due to the very fast oscillations of the Bessel functions. Providing general tools to quickly compute these types of integrals is therefore paramount. `pylevin` can calculate the following types of frequently encountered integrals

$$I_{\ell_1\ell_2\ell_3}(k_1,k_2,k_3) = \int_a^b \mathrm{d}x\, f(x) \prod_{i=1}^{N} \mathcal{J}_{\ell_i}(k_i x)\,, \quad N = 1,2,3\,,$$

here $\mathcal{J}_\ell(x)$ denotes a spherical or cylindrical Bessel function of order $\ell$ and $f(x)$ can be any non-oscillatory function, i.e. with frequencies much lower than the one of the product of Bessel functions.

## Statement of need

Typical approaches numerically estimate integrals over highly-oscillatory integrands are based on Fast Fourier Transforms (FFTLog) (Fang et al., 2020; Grasshorn Gebhardt & Jeong, 2018; Schöneberg et al., 2018) and asymptotic expansions (Iserles & Nørsett, 2005; Levin, 1996). In `pylevin`, we implement one of the former methods, in particular, the adaptive Levin collocation (Chen et al., 2022; Leonard et al., 2023; Levin, 1996). Extending and improving the work done in Zieser & Merkel (2016), `pylevin` can solve integrals of the type $I_{\ell_1\ell_2\ell_3}(k_1,k_2,k_3)$ (see summary).

The main code is implemented in `C++` and wrapped into `python` using `pybind`. Due to the way `pylevin` implements Levin's method it makes extensive use of precomputed quantities allowing updating the function $f(x)$ and making successive calls of the integration routine an order of magnitude faster than the first call. An aspect that is particularly important for situations where the same type of integral needs to be evaluated many times for slightly different $f(x)$. This is for example the case in inference when running Markov Chain Monte-Carlo.

In contrast to other implementations for highly oscillatory integrals, `pylevin` is very flexible, as it is not hardcoded and tailored to one particular application but is completely agnostic regarding the integrand. Furthermore, it implements integrals over three Bessel functions for the first time. These are for example required in many cosmological applications for higher-order statistics. Due to its implementation in a statically typed compiled language, it is also extremely fast, while making use of the convenience and white-spread use of `python` via `pybind`.

## Examples

The way `pylevin` works is that one first defines an integrand, $f(x)$, the integral type (spherical or cylindrical Bessel functions and $N$) and if the interpolation of the integrand should be carried: out logarithmically

```
x = np.geomspace(1e-5,100,100)
f_of_x = x**3 + (x**2 +x)
integral_type = 0
number_omp_threads = 1
interploate_logx = True
interploate_logy = True
lp_single = levin.pylevin(integral_type,
                          x,
                          f_of_x[:, None],
                          logx,
                          logy,
                          number_omp_threads)
```

Note that the broadcasting of `f_of_x`is required as one can in principle pass many different integrands at the same time and the code always expects this dimension. We can then define the values $k$ and $\ell$ at which we want to evaluate the integral which are all one-dimensional arrays of the same shape. Additionally, we also have to allocate the memory for the result which is stored in-place:

```
k = np.geomspace(1e-3,1e4,1000)
ell = (5*np.ones_like(k)).astype(int)
result_levin = np.zeros((len(k), 1))
lp_single.levin_integrate_bessel_single(x[0]*np.ones_like(k),
                                        x[-1]*np.ones_like(k),
                                        k,
                                        ell,
                                        False,
                                        result_levin)
```

If we would have passed more integrands before, the results must have the corresponding shape in the second dimension. For more detailed examples we refer to the example notebook on github and to the API.

We now demonstrate the performance of `pylevin` on a single core on an Apple M3 and compare it to `scipy.integrate.quad`, an adaptive quadrature. The relative accuracy required for both methods is set to $10^{-3}$. We use the following two integrals as an example:

$$I_2 = \int_{10^{-5}}^{100} \mathrm{d}x\ (x^3 + x^2 + x) j_{10}(kx) j_5(kx)\ ,$$

$$I_3 = \int_{10^{-5}}^{100} \mathrm{d}x\ (x^3 + x^2 + x) j_{10}(kx) j_5(kx) j_{15}(kx)\ ,$$

The result of $I_2$ is shown on the left and for $I_3$ on the right in Figure 1. In order for the quadrature to converge over an extended $k$-range, the number of maximum sub-intervals was increased to $10^3$ $(2 \times 10^3)$ for $I_2$ $(I_3)$. The grey-shaded area indicates where the quadrature fails to reach convergence even after this change. It is therefore clear that `pylevin` is more accurate and around three to four orders of magnitudes faster than standard integration routines.

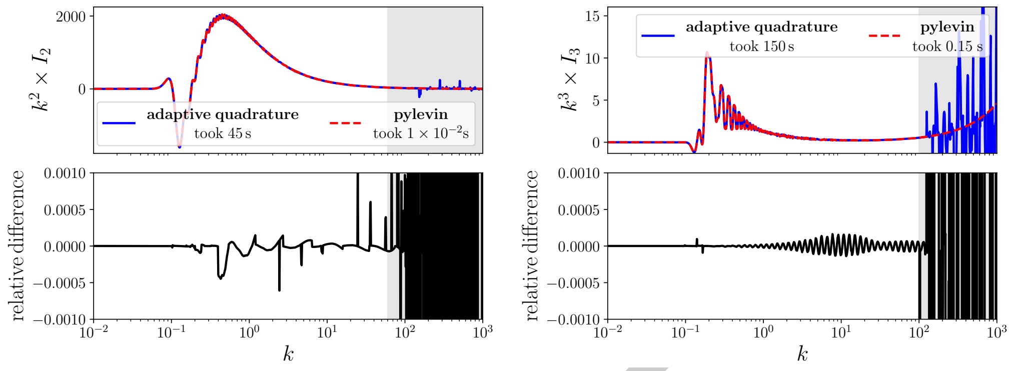

**Figure 1:** Speed and accuracy comparison of `pylevin` (shown in dashed red) against a standard adaptive quadrature (shown in solid blue). The runtime for the two methods is given in the legend. For the adaptive quadrature the maximum number of sub-intervals was set to 1000 (default is 50). The grey shaded region indicates when the quadrature starts to fail. The bottom panel shows the relative difference between the two methods. **Left**: Result of the integral $I_2$. **Right**: Result of the integral $I_3$.

## Comparison with various codes

Additionally to the benchmark, `pylevin` is compared with more specialised, codes who mostly solve integrals over single Bessel functions. All computing times presented here are averages over several runs. The comparison was done with an M3 processor with 8 cores (4 performance cores).

### `hankel`

We calculate the following Hankel transformation with `pylevin` and Ogata's method (Ogata (2005)) as implemented in the `hankel` package (Murray & Poulin (2019)).

$$\mathrm{integral}(k) = \int_0^\infty \frac{x^2}{x^2+1} J_0(kx)\, \mathrm{d}x \;,$$

for 500 values of $k$ logarithmically-spaced between 1 and $10^4$. The result is depicted on the left side of Figure 2. It can be seen that both methods agree very well and are roughly equally fast. While the Hankel transformation formally goes from zero to infinity, $a = 10^{-5}$ and $b = 10^8$ was used for `pylevin`. This choice of course depends on the specific integrand.

### `hankl`

Here, we follow the cosmology example provided in the `hankl` documentation (Karamanis & Beutler (2021)) to compute the monopole of the galaxy power spectrum:

$$\xi_0(s) = \int_0^\infty (b^2 + fb/3 + f^2/5)\, P_\mathrm{lin}(k) J_0(ks) k^2 \,\mathrm{d}k \;,$$

where $b$ is the galaxy bias, $f$ the logarithmic growth rate and $P_\mathrm{lin}(k)$ is the linear matter power spectrum, which is calculated using `camb` (Lewis & Bridle (2002)) at six redshifts. Since `hankl` is FFT based, it requires $k$ to be discretised, the FFT-dual will then be calculated at the inverse grid points. For this comparison, we use $2^{10}$ logarithmically-spaced points between $k = 10^{-4}$ and $k = 1$ for the transformation to converge. For `pylevin`, the number of points where the transformation is evaluated is arbitrary. Here we use 100 points, which is more than enough to resolve all features in $\xi_0$. The results are shown on the right of Figure 2 and good agreement can be found between the two methods with `hankl` being roughly twice as fast as `pylevin`.

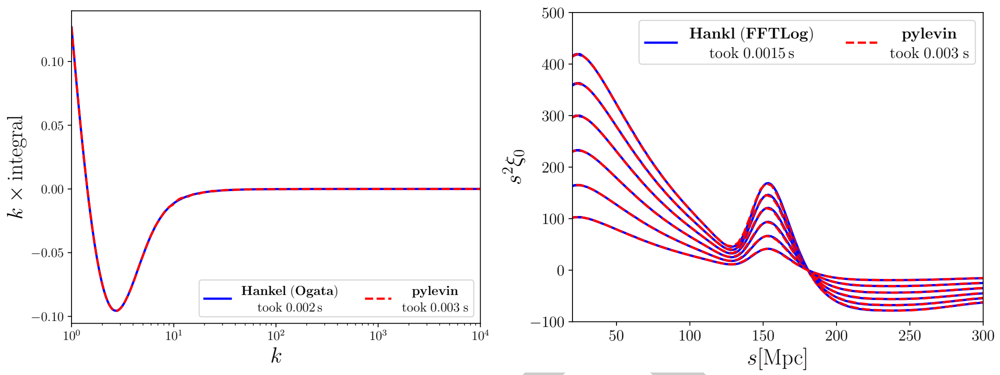

**Figure 2:** Comparison of `pylevin` with two methods to calculate a Hankel transformation. Dashed red is `pylevin` while solid blue is the alternative method. **Left**: Integral($k$) evaluated with the Ogata method using the `hankel` package. **Right**: Integral for the galaxy power spectrum monopole evaluated using the **hankl** package. Different lines refer to different redshifts.

## pyfftlog

For `pyfftlog` (Hamilton (2000)), we use the following transformation:

$$\mathrm{FT}(k) = \int_0^\infty r^5 \mathrm{e}^{-r^2/2} J_4(kr)\,\mathrm{d}r\;.$$

For `pyfftlog`, $2^8$ logarithmically-spaced points between $10^{-4}$ and $10^4$ for $r$ and hence also for $k$. `pylevin` is evaluated for the same number of points, this value could, however, be reduced due to the featureless transformation, thus increasing the speed. In the left panel of Figure 3, the result of this exercise is shown. Good agreement between the two methods is found, with both taken the same amount of time. The large relative error at large values of $k$ is due to the small value of the integral, and hence purely numerical noise.

## pyCCL

Here, we compare the implementation of the non-Limber projection for the angular power spectrum:

$$C_\ell = \frac{2}{\pi}\int \mathrm{d}\chi_1 W(\chi_1)\int \mathrm{d}\chi_2 W(\chi_2)\int k^2\mathrm{d}k\; P_\mathrm{m}(k,\chi_1,\chi_2) j_\ell(k\chi_1) j_\ell(k\chi_2)\;,$$

for $W$, we assume a Gaussian shell in redshift with width $\sigma_z = 0.01$ centred at $z = 0.6$, $\chi(z)$ is the comoving distance. The matter power spectrum, $P_\mathrm{m}$, is again calculated with `camb`. The results from `pylevin` is compared to `CCL` (Chisari et al. (2019)) which implements an FFTLog algorithm (Fang et al., 2020; Leonard et al., 2023). This implementation first solves the two integrals over $\chi_{1,2}$ using FFTLog and then carries out the remaining integral over $k$ and assumes that the $k$ and $\chi_{1,2}$ dependence in the matter power spectrum is separable. To be consistent, we follow the same approach and solve the $\chi_{1,2}$ integration using `pylevin` and then calculate the remaining $k$ integration with the composite Simpson's rule implemented in `scipy`. The right side of **??** shows that the two methods agree very well with each other and that the method implemented in `CCL` is about a factor of 2 faster. If the power spectrum, however, would not be separable on small $k$, as it can be the case in modified gravity scenarios, the `CCL` method would need to split the integral up into sub-intervals where the separability holds, slowing down the computation by a factor equal to the number of sub-intervals. This assumption is not done in `pylevin`.

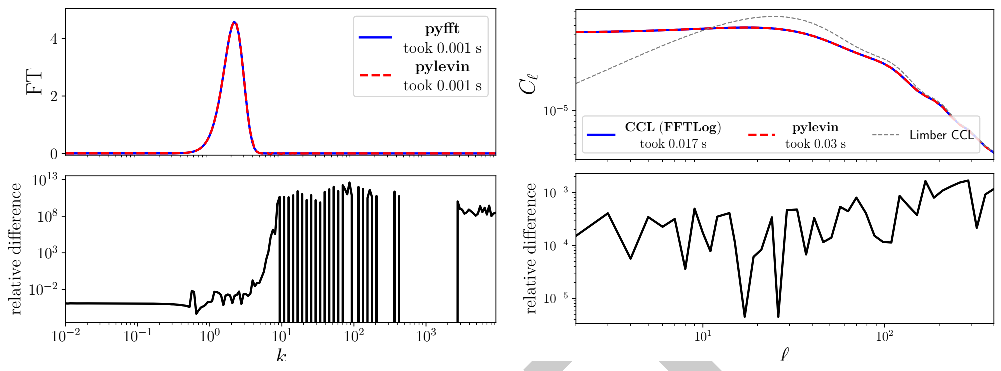

**Figure 3:** Comparison of `pylevin` with two other methods, the colour scheme is the same as in Figure 2. **Left**: Transformation defined in $\mathrm{FT}(k)$ with the `pyfftlog` package. **Right**: angular power spectra, $C(\ell)$ computed with `pyCCL`. We show the relative difference between the two methods in the lower panel.

## References

Chen, S., Serkh, K., & Bremer, J. (2022). On the adaptive Levin method. *arXiv e-Prints*, arXiv:2211.13400. https://doi.org/10.48550/arXiv.2211.13400

Chisari, N. E., Alonso, D., Krause, E., Leonard, C. D., Bull, P., Neveu, J., Villarreal, A. S., Singh, S., McClintock, T., Ellison, J., Du, Z., Zuntz, J., Mead, A., Joudaki, S., Lorenz, C. S., Tröster, T., Sanchez, J., Lanusse, F., Ishak, M., … LSST Dark Energy Science Collaboration. (2019). Core Cosmology Library: Precision Cosmological Predictions for LSST. *The Astrophysical Journal Supplement Series*, *242*, 2. https://doi.org/10.3847/1538-4365/ab1658

Fang, X., Eifler, T., & Krause, E. (2020). 2D-FFTLog: efficient computation of real-space covariance matrices for galaxy clustering and weak lensing. *MNRAS*, *497*(3), 2699–2714. https://doi.org/10.1093/mnras/staa1726

Grasshorn Gebhardt, H. S., & Jeong, D. (2018). Fast and accurate computation of projected two-point functions. *PRD*, *97*(2), 023504. https://doi.org/10.1103/PhysRevD.97.023504

Hamilton, A. J. S. (2000). Uncorrelated modes of the non-linear power spectrum. *Monthly Notices of the Royal Astronomical Society*, *312*(2), 257–284. https://doi.org/10.1046/j.1365-8711.2000.03071.x

Iserles, A., & Nørsett, S. P. (2005). Efficient quadrature of highly oscillatory integrals using derivatives. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *461*(2057), 1383–1399. https://doi.org/10.1098/rspa.2004.1401

Karamanis, M., & Beutler, F. (2021). hankl: A lightweight Python implementation of the FFTLog algorithm for Cosmology. *arXiv e-Prints*, arXiv:2106.06331. https://doi.org/10.48550/arXiv.2106.06331

Leonard, C. D., Ferreira, T., Fang, X., Reischke, R., Schoeneberg, N., Tröster, T., Alonso, D., Campagne, J.-E., Lanusse, F., Slosar, A., & Ishak, M. (2023). The N5K Challenge: Non-Limber Integration for LSST Cosmology. *The Open Journal of Astrophysics*, *6*, 8. https://doi.org/10.21105/astro.2212.04291

Levin, D. (1996). Fast integration of rapidly oscillatory functions. *Journal of Computational and Applied Mathematics*, *67*(1), 95–101. https://doi.org/10.1016/0377-0427(94)00118-9

Lewis, A., & Bridle, S. (2002). Cosmological parameters from CMB and other data: A Monte Carlo approach. *Physical Review D*, *D66*, 103511. https://doi.org/10.1103/PhysRevD.66.103511

Murray, S., & Poulin, F. (2019). hankel: A Python library for performing simple and accurate Hankel transformations. *The Journal of Open Source Software*, *4*(37), 1397. https://doi.org/10.21105/joss.01397

Ogata, H. (2005). A numerical integration formula based on the bessel functions, publications of the research institute for mathematical sciences. *Publications of the Research Institute for Mathematical Sciences*, *41*, 949–970. https://doi.org/10.2977/prims/1145474602

Schöneberg, N., Simonović, M., Lesgourgues, J., & Zaldarriaga, M. (2018). Beyond the traditional line-of-sight approach of cosmological angular statistics. *JCAP*, *2018*(10), 047. https://doi.org/10.1088/1475-7516/2018/10/047

Zieser, B., & Merkel, P. M. (2016). The cross-correlation between 3D cosmic shear and the integrated Sachs-Wolfe effect. *459*(2), 1586–1595. https://doi.org/10.1093/mnras/stw665