User Manual for FFTLog-and-Beyond

Xiao Fang

1 Versions

- Version 1.0: 2019/11/22

2 Problem Description

The integral we are solving is

$$F(y) = \int_0^\infty \frac{dx}{x} f(x) j_\ell^{(n)}(xy) \ , \tag{1}$$

where $f(x)$ is an input array, j_ℓ is the order-ℓ spherical Bessel function of the first kind, the superscript $^{(n)}$ denotes the order of derivative. This type of integrals are numerically challenging due to the rapidly oscillitory nature of the spherical Bessel functions, especially when the input $f(x)$ data array correspond to sampling array x over a large range (*i.e.*, over several orders of magnitude).

3 Efficient Computation with FFTLog-and-Beyond

The essential idea is to expand $f(x)$ into a series of power-laws and solve each component integral analytically. We require a logarithmic sampling of x with linear spacing in $\ln(x)$ equal to $\Delta_{\ln x}$, *i.e.*, $x_q = x_0 \exp(q\Delta_{\ln x})$ with x_0 being the smallest value in the x array. The power-law decomposition then means

$$f(x_q) = \frac{1}{N} \sum_{m=-N/2}^{N/2} c_m x_0^\nu \left(\frac{x_q}{x_0}\right)^{\nu + i\eta_m} \ , \tag{2}$$

where N is the sample size of the input function, $\eta_m = 2\pi m/(N\Delta_{\ln x})$, and ν is the bias index. The Fourier coefficients satisfy $c_m^* = c_{-m}$ since function $f(x)$ is real, and are computed by discrete Fourier transforming the "biased" input function $f(x)/x^\nu$ as

$$c_m = W_m \sum_{q=0}^{N-1} \frac{f(x_q)}{x_q^\nu} e^{-2\pi i m q/N} \ , \tag{3}$$

where W_m is a window function which smooths the edges of the c_m array and takes the form of Eq. (C.1) in McEwen et al (2016, arXiv: 1603.04826). This filtering is found to reduce the ringing effects.

Each term is now analytically solvable. For $n = 0$, *i.e.*no derivative,

$$\begin{aligned} F(y) &= \frac{1}{Ny^\nu} \sum_{m=-N/2}^{N/2} c_m x_0^{-i\eta_m} y^{-i\eta_m} \int_0^\infty \frac{dx}{x} x^{\nu + i\eta_m} j_\ell(x) \\ &= \frac{\sqrt{\pi}}{4Ny^\nu} \sum_{m=-N/2}^{N/2} c_m x_0^{-i\eta_m} y^{-i\eta_m} g_\ell(\nu + i\eta_m) \ , \end{aligned} \tag{4}$$

where the first equality uses change of variable $xy \to x$. Function $g_\ell(z)$ is given by

$$g_\ell(z) = 2^z \frac{\Gamma\left(\frac{\ell+z}{2}\right)}{\Gamma\left(\frac{3+\ell-z}{2}\right)} \ , \quad -\ell < \Re(z) < 2 \ , \tag{5}$$

giving the range of bias index $-\ell < \nu < 2$.

Finally, assuming that y is logarithmically sampled with the same linear spacing $\Delta_{\ln y} = \Delta_{\ln x}$ in $\ln y$, we can write the last summation in Eq. (4) as

$$F(y_p) = \frac{\sqrt{\pi}}{4y_p^\nu} \mathrm{IFFT}\left[c_m^* (x_0 y_0)^{i\eta_m} g_\ell(\nu - i\eta_m)\right] \ , \tag{6}$$

where y_p ($p = 0, 1, \cdots, N-1$) is the p-th element in the y array. IFFT stands for the Inverse Fast Fourier Transform. In summary, this method performs two FFT operations, one in computing c_m, one in the final summation over m. Thus, the total time complexity is $\mathcal{O}(N \log N)$. So far we have described the principle of the FFTLog algorithm.

For $n > 0$, following the same procedure of power-law decomposition, we have

$$F_n(y) = \frac{1}{Ny^\nu} \sum_{m=-N/2}^{N/2} c_m x_0^{-i\eta_m} y^{-i\eta_m} \int_0^\infty \frac{dx}{x} x^{\nu+i\eta_m} j_\ell^{(n)}(x) \ . \tag{7}$$

Again, the integral for each m has an analytic solution, which can be shown with integration by parts. We write the solution in the same form with the FFTLog, *i.e.*,

$$F_n(y) = \frac{\sqrt{\pi}}{4Ny^\nu} \sum_{m=-N/2}^{N/2} c_m x_0^{-i\eta_m} y^{-i\eta_m} \tilde{g}_\ell(n, \nu + i\eta_m) \ , \tag{8}$$

and its discrete version assuming $\Delta_{\ln y} = \Delta_{\ln x}$,

$$F_n(y_p) = \frac{\sqrt{\pi}}{4y_p^\nu} \mathrm{IFFT}\left[c_m^*(x_0 y_0)^{i\eta_m} \tilde{g}_\ell(n, \nu - i\eta_m)\right] \ , \tag{9}$$

where $\tilde{g}_\ell(n,z) = 4\pi^{-1/2}\int_0^\infty dx\, x^{z-1} j_\ell^{(n)}(x)$. For $n = 0$, $\tilde{g}_\ell(0,z) = g_\ell(z)$, and for $n = 1, 2$, it is given by

$$\tilde{g}_\ell(1,z) = -2^{z-1}(z-1)\frac{\Gamma\left(\frac{\ell+z-1}{2}\right)}{\Gamma\left(\frac{4+\ell-z}{2}\right)} \ , \ \left(\begin{array}{ll} 0 < \Re(z) < 2 \ , & \text{for } \ell = 0 \\ 1-\ell < \Re(z) < 2 \ , & \text{for } \ell \geq 1 \end{array} \right) \ , \tag{10}$$

$$\tilde{g}_\ell(2,z) = 2^{z-2}(z-1)(z-2)\frac{\Gamma\left(\frac{\ell+z-2}{2}\right)}{\Gamma\left(\frac{5+\ell-z}{2}\right)} \ , \ \left(\begin{array}{ll} -\ell < \Re(z) < 2 \ , & \text{for } \ell = 0,1 \\ 2-\ell < \Re(z) < 2 \ , & \text{for } \ell \geq 2 \end{array} \right) \ . \tag{11}$$

We choose $\nu = 1$ for all ℓ's. With this generalized FFTLog algorithm, the integral containing one derivative of a spherical Bessel function also takes 2 FFT operations to compute.

Most generally, for $n \in \mathcal{N}$,

$$\int_0^\infty dx\, x^{\alpha-1} j_\ell^{(n)}(x) = (-1)^n \frac{\sqrt{\pi}}{4} 2^{\alpha-n} \frac{\Gamma(\alpha)}{\Gamma(\alpha-n)} \frac{\Gamma(\frac{\ell+\alpha-n}{2})}{\Gamma(\frac{3+n+\ell-\alpha}{2})} \ , \ \left(\begin{array}{ll} -\ell < \Re(\alpha) < 2 \ , & \text{for } \ell < n \\ n-\ell < \Re(\alpha) < 2 \ , & \text{for } \ell \geq n \end{array} \right) . \tag{12}$$

4 Using the Code

This repository contains independent python module and C module for computing Eq. (1).

4.1 Python Version

The main code is all in `python/fftlog.py`, along with `python/test.py` which provides an example of calling the module to compute the integrals with $f(x)$ being the power spectrum in `Pk_test`, and $n = 0, 1, 2$, respectively. We also have a Hankel function, defined as replacing j_ℓ as J_n in Eq. (1). If you find them useful in your research, please cite Fang et al (2019).

4.2 C Version

The code, sitting in `cfftlog` folder, uses `FFTW` library. The main code is in `cfftlog.c`, with auxiliary functions (*e.g.*, extrapolation, window function, g_ℓ functions) defined in `utils.c` and `utils_complex.c`.

`cfftlog.c` provides different ways to run the computation. `cfftlog` provides the simplest usage. `cfftlog_ells` function enables to compute the same integral with an array of ℓ values. This is more optimal than calling single function `cfftlog` many times since `FFTW` plans have to be re-created and destroyed over and over again, and same for the `FFTW_COMPLEX` arrays used for the FFTs.

`cfftlog_ells_increment` does the same thing as `cfftlog_ells`, but increment the results (instead of refreshing). This saves memory for some applications.

`test.c` and `test_ells.c` show examples of calling those functions.