pyfftlog Documentation

Release 0.2.1.dev13+gd09623c

Dieter Werthmüller

31 January 2022

User Manual

Version: 0.2.1.dev13+gd09623c ~ Date: 31 January 2022

CHAPTER 1

pyfftlog - A python version of FFTLog

This is a python version of the logarithmic FFT code *FFTLog* as presented in Appendix B of Hamilton (2000) and published at casa.colorado.edu/~ajsh/FFTLog.

A simple *f2py*-wrapper (*fftlog*) can be found on github.com/prisae/fftlog. Tests have shown that *fftlog* is a bit faster than *pyfftlog*, but *pyfftlog* is easier to implement, as you only need *NumPy* and *SciPy*, without the need to compile anything.

I hope that *FFTLog* will make it into *SciPy* in the future, which will make this project redundant. (If you have the bandwidth and are willing to chip in have a look at SciPy PR #7310.)

Be aware that *pyfftlog* has not been tested extensively. It works fine for the test from the original code, and my use case, which is *pyfftlog.fftl* with *mu=0.5* (sine-transform), *q=0* (unbiased), *k=1*, *kropt=1*, and *tdir=1* (forward). Please let me know if you encounter any issues.

- **Documentation**: https://pyfftlog.readthedocs.io
- **Source Code**: https://github.com/prisae/pyfftlog

1.1 Description of FFTLog from the FFTLog-Website

FFTLog is a set of fortran subroutines that compute the fast Fourier or Hankel (= Fourier-Bessel) transform of a periodic sequence of logarithmically spaced points.

FFTLog can be regarded as a natural analogue to the standard Fast Fourier Transform (FFT), in the sense that, just as the normal FFT gives the exact (to machine precision) Fourier transform of a linearly spaced periodic sequence, so also FFTLog gives the exact Fourier or Hankel transform, of arbitrary order m, of a logarithmically spaced periodic sequence.

FFTLog shares with the normal FFT the problems of ringing (response to sudden steps) and aliasing (periodic folding of frequencies), but under appropriate circumstances FFTLog may approximate the results of a continuous Fourier or Hankel transform.

The FFTLog algorithm was originally proposed by Talman (1978).

For the full documentation, see casa.colorado.edu/~ajsh/FFTLog.

1.2 Installation

You can install pyfftlog either via **conda**:

```
conda install -c conda-forge pyfftlog
```

or via **pip**:

```
pip install pyfftlog
```

1.3 License, Citation, and Credits

All releases have a Zenodo-DOI, which can be found on 10.5281/zenodo.3830364.

Be kind and give credits by citing Hamilton (2000). See the references-section in the manual for full references.

1.3.1 Manual and API

pyfftlog – Python version of FFTLog

This is a Python version of the FFTLog Fortran code by Andrew Hamilton, [Hami00].

The function `scipy.special.loggamma` replaces the file *cdgamma.f* in the original code, and the functions `scipy.fftpack.rfft()` and `scipy.fftpack.irfft()` replace the files *drffti.f*, *drfftf.f*, and *drfftb.f* in the original code.

The original documentation has just been adjusted where necessary, and put into a more pythonic format (e.g. using *Parameters* and *Returns* in the documentation').

What follows is the original documentation from the file 'fftlog.f':

THE FFTLog CODE

FFTLog computes the discrete Fast Fourier Transform or Fast Hankel Transform (of arbitrary real index) of a periodic logarithmic sequence.

- Version of 13 Mar 2000.
- For more information about FFTLog, see http://casa.colorado.edu/~ajsh/FFTLog.
- Andrew J S Hamilton March 1999.
- Refs: [Talm78].

FFTLog computes a discrete version of the Hankel Transform (= Fourier-Bessel Transform) with a power law bias $(kr)^q$

$$\tilde{a}(k) = \int_0^\infty a(r)(kr)^q J_\mu(kr) k\, dr\,, \tag{1.1}$$

$$a(r) = \int_0^\infty \tilde{a}(k)(kr)^{-q} J_\mu(kr) r\, dk\,, \tag{1.2}$$

where J_μ is the Bessel function of order μ. The index μ may be any real number, positive or negative.

The input array a_j is a periodic sequence of length n, uniformly logarithmically spaced with spacing $dlnr$

$$a_j = a(r_j) \quad \text{at} \quad r_j = r_c \exp[(j - j_c) dlnr] \tag{1.3}$$

centred about the point r_c. The central index $j_c = (n+1)/2$ is 1/2 integral if n is even. Similarly, the output array $\tilde{a}_j$ is a periodic sequence of length n, also uniformly logarithmically spaced with spacing $dlnr$

$$\tilde{a}_j = \tilde{a}(k_j) \quad \text{at} \quad k_j = k_c \exp[(j - j_c)dlnr] \tag{1.4}$$

centred about the point k_c.

The centre points r_c and k_c of the periodic intervals may be chosen arbitrarily; but it would be normal to choose the product

$$kr = k_c r_c = k_j r_{(n+1-j)} = k_{(n+1-j)} r_j \tag{1.5}$$

to be about 1 (or 2, or pi, to taste).

The FFTLog algorithm is (see [Hami00]):

1. FFT the input array a_j to obtain the Fourier coefficients c_m ;
2. Multiply c_m by $u_m = (kr)^{-i2m\pi/(ndlnr)} U_\mu[q+i2m\pi/(ndlnr)]$ where $U_\mu(x) = 2^x \Gamma[(\mu+1+x)/2]/\Gamma[(\mu+1-x)/2]$ to obtain $c_m u_m$;
3. FFT $c_m u_m$ back to obtain the discrete Hankel transform $\tilde{a}_j$.

The Fourier sine and cosine transforms

$$\tilde{A}(k) = \sqrt{2/\pi} \int_0^\infty A(r) \sin(kr)\, dr\,, \tag{1.6}$$

$$\tilde{A}(k) = \sqrt{2/\pi} \int_0^\infty A(r) \cos(kr)\, dr\,, \tag{1.7}$$

may be regarded as special cases of the Hankel transform with $\mu = 1/2$ and $-1/2$ since

$$\sqrt{2/\pi} \sin(x) = \sqrt{(x)} J_{1/2}(x)\,, \tag{1.8}$$

$$\sqrt{2/\pi} \cos(x) = \sqrt{(x)} J_{-1/2}(x)\,. \tag{1.9}$$

The Fourier transforms may be done by making the substitutions

$$A(r) = a(r) r^{q-1/2} \quad \text{and} \quad \tilde{A}(k) = \tilde{a}(k) k^{-q-1/2} \tag{1.10}$$

and Hankel transforming $a(r)$ with a power law bias $(kr)^q$

$$\tilde{a}(k) = \int_0^\infty a(r)(kr)^q J_{\pm 1/2}(kr) k\, dr\,. \tag{1.11}$$

Different choices of power law bias q lead to different discrete Fourier transforms of $A(r)$, because the assumption of periodicity of $a(r) = A(r) r^{-q+(1/2)}$ is different for different q.

If $A(r)$ is a power law, $A(r)$ proportional to $r^{q-(1/2)}$, then applying a bias q yields a discrete Fourier transform $\tilde{A}(k)$ that is exactly equal to the continuous Fourier transform, because then $a(r)$ is a constant, which is a periodic function.

The Hankel transform

$$\tilde{A}(k) = \int_0^\infty A(r) J_\mu(kr) k\, dr \tag{1.12}$$

may be done by making the substitutions

$$A(r) = a(r)r^q \quad \text{and} \quad \tilde{A}(k) = \tilde{a}(k)k^{-q} \tag{1.13}$$

and Hankel transforming $a(r)$ with a power law bias $(kr)^q$

$$\tilde{a}(k) = \int_0^\infty a(r)(kr)^q J_\mu(kr)k\,dr\,. \tag{1.14}$$

Different choices of power law bias q lead to different discrete Hankel transforms of $A(r)$, because the assumption of periodicity of $a(r) = A(r)r^{-q}$ is different for different q.

If $A(r)$ is a power law, $A(r)$ proportional to r^q, then applying a bias q yields a discrete Hankel transform $\tilde{A}(k)$ that is exactly equal to the continuous Hankel transform, because then $a(r)$ is a constant, which is a periodic function.

There are five routines:

Comments in the subroutines contain further details.

1. **subroutine 'fhti(n,mu,q,dlnr,kr,kropt,wsave,ok)'** is an initialization routine.
2. **subroutine 'fftl(n,a,rk,dir,wsave)'** computes the discrete Fourier sine or cosine transform of a logarithmically spaced periodic sequence. This is a driver routine that calls *fhtq*.
3. **subroutine 'fht(n,a,dir,wsave)'** computes the discrete Hankel transform of a logarithmically spaced periodic sequence. This is a driver routine that calls *fhtq*.
4. **subroutine 'fhtq(n,a,dir,wsave)'** computes the biased discrete Hankel transform of a logarithmically spaced periodic sequence. **This is the basic FFTLog routine.**
5. **real*8 function 'krgood(mu,q,dlnr,kr)'** takes an input *kr* and returns the nearest low-ringing *kr*. This is an optional routine called by *fhti*.

END of the original documentation from the file 'fftlog.f'

pyfftlog.pyfftlog.**fhti**(*n*, *mu*, *dlnr*, *q=0*, *kr=1*, *kropt=0*)

Initialize the working array xsave used by fftl, fht, and fhtq.

fhti initializes the working array xsave used by fftl, fht, and fhtq. fhti need be called once, whereafter fftl, fht, or fhtq may be called many times, as long as n, mu, q, dlnr, and kr remain unchanged. fhti should be called each time n, mu, q, dlnr, or kr is changed. The work array xsave should not be changed between calls to fftl, fht, or fhtq.

Parameters

n [int] Number of points in the array to be transformed; n may be any positive integer, but the FFT routines run fastest if n is a product of small primes 2, 3, 5.

mu [float] Index of J_mu in Hankel transform; mu may be any real number, positive or negative.

dlnr [float] Separation between natural log of points; dlnr may be positive or negative.

q [float, optional] Exponent of power law bias; q may be any real number, positive or negative. If in doubt, use q = 0, for which case the Hankel transform is orthogonal, i.e. self-inverse, provided also that, for n even, kr is low-ringing. Non-zero q may yield better approximations to the continuous Hankel transform for some functions. Defaults to 0 (unbiased).

kr [float, optional] k_c r_c where c is central point of array = k_j r_(n+1-j) = k_(n+1-j) r_j . Normally one would choose kr to be about 1 (default) (or 2, or pi, to taste).

kropt [int, optional; {0, 1, 2, 3}]

- 0 to use input kr as is (default);
- 1 to change kr to nearest low-ringing kr, quietly;

- 2 to change kr to nearest low-ringing kr, verbosely;
- 3 for option to change kr interactively.

Returns

kr [float, optional] kr, adjusted depending on kropt.

xsave [array] Working array used by fftl, fht, and fhtq. Dimension: - for q = 0 (unbiased transform): n+3 - for q != 0 (biased transform): 1.5*n+4 If odd, last element is not needed.

pyfftlog.pyfftlog.**fftl**(*a*, *xsave*, *rk=1*, *tdir=1*)

Logarithmic fast Fourier transform FFTLog.

This is a driver routine that calls *fhtq()*.

fftl computes a discrete version of the Fourier sine (if mu = 1/2) or cosine (if mu = -1/2) transform

$$\tilde{A}(k) = \sqrt{2/\pi} \int_0^\infty A(r) \sin(kr)\, dr\,,$$

$$\tilde{A}(k) = \sqrt{2/\pi} \int_0^\infty A(r) \cos(kr)\, dr\,,$$

by making the substitutions

$$A(r) = a(r) r^{q-1/2} \quad \text{and} \quad \tilde{A}(k) = \tilde{a}(k) k^{-q-1/2}$$

and applying a biased Hankel transform to $a(r)$.

The steps are: 1. $a(r) = A(r) r^{[} - dir(q - 0.5)]$ 2. call *fhtq* to transform $a(r) \to \tilde{a}(k)$ 3. $\tilde{A}(k) = \tilde{a}(k) k^{[} - dir(q + 0.5)]$

fhti must be called before the first call to *fftl*, with *mu=1/2* for a sine transform, or *mu=-1/2* for a cosine transform.

A call to *fftl* with *dir=1* followed by a call to *fftl* with *dir=-1* (and rk unchanged), or vice versa, leaves the array a unchanged.

Parameters

a [array] Array A(r) to transform: a(j) is A(r_j) at r_j = r_c exp[(j-jc) dlnr], where jc = (n+1)/2 = central index of array.

xsave [array] Working array set up by fhti.

rk [float, optional] r_c/k_c = r_j/k_j (a constant, the same constant for any j); rk is not (necessarily) the same quantity as kr. rk is used only to multiply the output array by sqrt(rk)^dir, so if you want to do the normalization later, or you don't care about the normalization, you can set rk = 1. Defaults to 1.

tdir [int, optional; {1, -1}]

- 1 for forward transform (default),
- -1 for backward transform.

A backward transform (dir = -1) is the same as a forward transform with q -> -q and rk -> 1/rk, for any kr if n is odd, for low-ringing kr if n is even.

Returns

a [array] Transformed array Ã(k): a(j) is Ã(k_j) at k_j = k_c exp[(j-jc) dlnr].

pyfftlog.pyfftlog.**fht**(*a*, *xsave*, *tdir=1*)

Fast Hankel transform FHT.

This is a driver routine that calls *fhtq()*.

fht computes a discrete version of the Hankel transform

$$\tilde{A}(k) = \int_0^\infty A(r) J_\mu(kr) k\, dr$$

by making the substitutions

$$A(r) = a(r) r^q \quad \text{and} \quad \tilde{A}(k) = \tilde{a}(k) k^{-q}$$

and applying a biased Hankel transform to $a(r)$.

The steps are: 1. $a(r) = A(r) r^{-dirq}$ 2. call *fhtq* to transform $a(r) \to \tilde{a}(k)$ 3. $\tilde{A}(k) = \tilde{a}(k) k^{-dirq}$

fhti must be called before the first call to *fht*.

A call to *fht* with *dir=1* followed by a call to *fht* with *dir=-1*, or vice versa, leaves the array a unchanged.

Parameters

a [array] Array A(r) to transform: a(j) is A(r_j) at r_j = r_c exp[(j-jc) dlnr], where jc = (n+1)/2 = central index of array.

xsave [array] Working array set up by fhti.

tdir [int, optional; {1, -1}]

- 1 for forward transform (default),
- -1 for backward transform.

A backward transform (dir = -1) is the same as a forward transform with q -> -q, for any kr if n is odd, for low-ringing kr if n is even.

Returns

a [array] Transformed array Ã(k): a(j) is Ã(k_j) at k_j = k_c exp[(j-jc) dlnr].

pyfftlog.pyfftlog.**fhtq**(*a*, *xsave*, *tdir=1*)

Kernel routine of FFTLog.

This is the basic FFTLog routine.

fhtq computes a discrete version of the biased Hankel transform

$$\tilde{a}(k) = \int_0^\infty a(r) (kr)^q J_\mu(kr) k\, dr\,.$$

fhti must be called before the first call to *fhtq*.

A call to *fhtq* with *dir=1* followed by a call to *fhtq* with *dir=-1*, or vice versa, leaves the array a unchanged.

Parameters

a [array] Periodic array a(r) to transform: a(j) is a(r_j) at r_j = r_c exp[(j-jc) dlnr] where jc = (n+1)/2 = central index of array.

xsave [array] Working array set up by fhti.

tdir [int, optional; {1, -1}]

- 1 for forward transform (default),
- -1 for backward transform.

A backward transform (dir = -1) is the same as a forward transform with q -> -q, for any kr if n is odd, for low-ringing kr if n is even.

Returns

a [array] Transformed periodic array ã(k): a(j) is ã(k_j) at k_j = k_c exp[(j-jc) dlnr].

pyfftlog.pyfftlog.**krgood**(*mu*, *q*, *dlnr*, *kr*)

Return optimal kr.

Use of this routine is optional.

Choosing kr so that

$$(kr)^{-ipi/dlnr} U_\mu(q + ipi/dlnr)$$

is real may reduce ringing of the discrete Hankel transform, because it makes the transition of this function across the period boundary smoother.

Parameters

mu [float] index of J_mu in Hankel transform; mu may be any real number, positive or negative.

q [float] exponent of power law bias; q may be any real number, positive or negative. If in doubt, use q = 0, for which case the Hankel transform is orthogonal, i.e. self-inverse, provided also that, for n even, kr is low-ringing. Non-zero q may yield better approximations to the continuous Hankel transform for some functions.

dlnr [float] separation between natural log of points; dlnr may be positive or negative.

kr [float, optional] k_c r_c where c is central point of array = k_j r_(n+1-j) = k_(n+1-j) r_j . Normally one would choose kr to be about 1 (default) (or 2, or pi, to taste).

Returns

krgood [float] low-ringing value of kr nearest to input kr. ln(krgood) is always within dlnr/2 of ln(kr).

1.3.2 Examples

FFTLog-Test

This example is a translation of *fftlogtest.f* from the Fortran package *FFTLog*, which was presented in Appendix B of [Hami00] and published at http://casa.colorado.edu/~ajsh/FFTLog. It serves as an example for the python package *pyfftlog* (which is a Python version of *FFTLog*), in the same manner as the original file *fftlogtest.f* serves as an example for Fortran package *FFTLog*.

What follows is the original documentation from the file 'fftlogtest.f':

This is fftlogtest.f

This is a simple test program to illustrate how *FFTLog* works. The test transform is:

$$\int_0^\infty r^{\mu+1} \exp\left(-\frac{r^2}{2}\right) J_\mu(k,r)\, k\, \mathrm{d}r = k^{\mu+1} \exp\left(-\frac{k^2}{2}\right) \tag{1.15}$$

Disclaimer

FFTLog does NOT claim to provide the most accurate possible solution of the continuous transform (which is the stated aim of some other codes). Rather, *FFTLog* claims to solve the exact discrete transform of a logarithmically-spaced periodic sequence. If the periodic interval is wide enough, the resolution high enough, and the function well enough behaved outside the periodic interval, then *FFTLog* may yield a satisfactory approximation to the continuous transform.

Observe:

1. How the result improves as the periodic interval is enlarged. With the normal FFT, one is not used to ranges orders of magnitude wide, but this is how *FFTLog* prefers it.

2. How the result improves as the resolution is increased. Because the function is rather smooth, modest resolution actually works quite well here.
3. That the central part of the transform is more reliable than the outer parts. Experience suggests that a good general strategy is to double the periodic interval over which the input function is defined, and then to discard the outer half of the transform.
4. That the best bias exponent seems to be $q = 0$.
5. That for the critical index $\mu = -1$, the result seems to be offset by a constant from the 'correct' answer.
6. That the result grows progressively worse as mu decreases below -1.

The analytic integral above fails for $\mu \leq -1$, but *FFTLog* still returns answers. Namely, *FFTLog* returns the analytic continuation of the discrete transform. Because of ambiguity in the path of integration around poles, this analytic continuation is liable to differ, for $\mu \leq -1$, by a constant from the 'correct' continuation given by the above equation.

FFTLog begins to have serious difficulties with aliasing as μ decreases below -1, because then $r^{\mu+1}\exp(-r^2/2)$ is far from resembling a periodic function. You might have thought that it would help to introduce a bias exponent $q = \mu$, or perhaps $q = \mu + 1$, or more, to make the function $a(r) = A(r)r^{-q}$ input to *fhtq* more nearly periodic. In practice a nonzero q makes things worse.

A symmetry argument lends support to the notion that the best exponent here should be $q = 0$, as empirically appears to be true. The symmetry argument is that the function $r^{\mu+1}\exp(-r^2/2)$ happens to be the same as its transform $k^{\mu+1}\exp(-k^2/2)$. If the best bias exponent were q in the forward transform, then the best exponent would be $-q$ that in the backward transform; but the two transforms happen to be the same in this case, suggesting $q = -q$, hence $q = 0$.

This example illustrates that you cannot always tell just by looking at a function what the best bias exponent q should be. You also have to look at its transform. The best exponent q is, in a sense, the one that makes both the function and its transform look most nearly periodic.

```
import pyfftlog
import numpy as np
import matplotlib.pyplot as plt
```

Define the parameters you wish to use

The presets are the *Reasonable choices of parameters* from *fftlogtest.f*.

```
# Range of periodic interval
logrmin = -4
logrmax = 4

# Number of points (Max 4096)
n = 64

# Order mu of Bessel function
mu = 0

# Bias exponent: q = 0 is unbiased
q = 0

# Sensible approximate choice of k_c r_c
kr = 1

# Tell fhti to change kr to low-ringing value
# WARNING: kropt = 3 will fail, as interaction is not supported
kropt = 1

# Forward transform (changed from dir to tdir, as dir is a python fct)
tdir = 1
```

Computation related to the logarithmic spacing

```
# Central point log10(r_c) of periodic interval
logrc = (logrmin + logrmax)/2

print(f"Central point of periodic interval at log10(r_c) = {logrc}")

# Central index (1/2 integral if n is even)
nc = (n + 1)/2.0

# Log-spacing of points
dlogr = (logrmax - logrmin)/n
dlnr = dlogr*np.log(10.0)
```

Out:

```
Central point of periodic interval at log10(r_c) = 0.0
```

Compute input function: $r^{\mu+1}\exp\left(-\frac{r^2}{2}\right)$

```
r = 10**(logrc + (np.arange(1, n+1) - nc)*dlogr)
ar = r**(mu + 1)*np.exp(-r**2/2.0)
```

Initialize FFTLog transform - note fhti resets *kr*

```
kr, xsave = pyfftlog.fhti(n, mu, dlnr, q, kr, kropt)
print(f"pyfftlog.fhti: new kr = {kr}")
```

Out:

```
pyfftlog.fhti: new kr = 0.9535389675791917
```

Call *pyfftlog.fht* (or *pyfftlog.fhtl*)

```
logkc = np.log10(kr) - logrc
print(f"Central point in k-space at log10(k_c) = {logkc}")

# rk = r_c/k_c
rk = 10**(logrc - logkc)

# Transform
# ak = pyfftlog.fftl(ar.copy(), xsave, rk, tdir)
ak = pyfftlog.fht(ar.copy(), xsave, tdir)
```

Out:

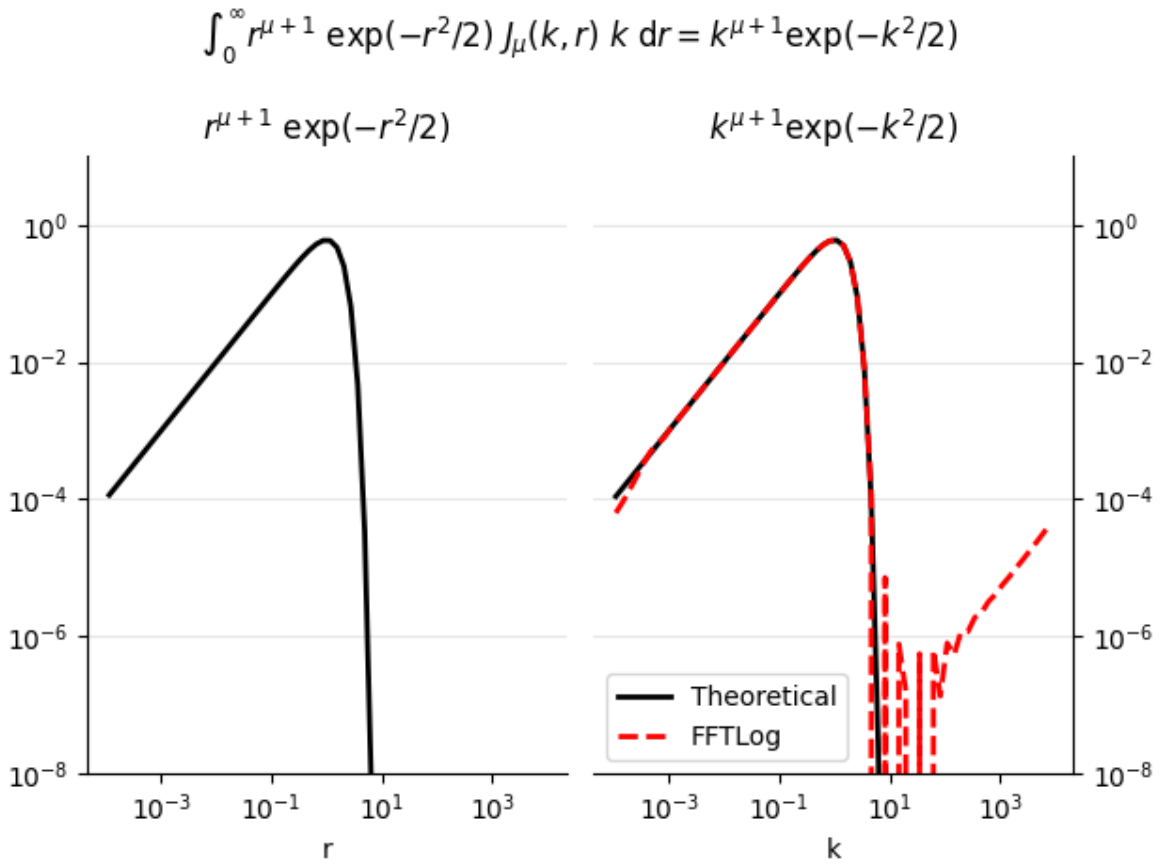
```
Central point in k-space at log10(k_c) = -0.020661554260541743
```

Compute Output function: $k^{\mu+1}\exp\left(-\frac{k^2}{2}\right)$

```
k = 10**(logkc + (np.arange(1, n+1) - nc)*dlogr)
theo = k**(mu + 1)*np.exp(-k**2/2.0)
```

Plot result

```
plt.figure()

# Input
ax1 = plt.subplot(121)
plt.title(r'$r^{\mu+1}\ \exp(-r^2/2)$')
plt.xlabel('r')

plt.loglog(r, ar, 'k', lw=2)

plt.grid(axis='y', c='0.9')

# Transformed result
ax2 = plt.subplot(122, sharey=ax1)
plt.title(r'$k^{\mu+1} \exp(-k^2/2)$')
plt.xlabel('k')

plt.loglog(k, theo, 'k', lw=2, label='Theoretical')
plt.loglog(k, ak, 'r--', lw=2, label='FFTLog')

plt.legend()
plt.ylim([1e-8, 1e1])

ax2.yaxis.tick_right()
ax2.yaxis.set_label_position("right")
plt.grid(axis='y', c='0.9')

# Switch off spines
ax1.spines['top'].set_visible(False)
ax1.spines['right'].set_visible(False)
ax2.spines['top'].set_visible(False)
ax2.spines['left'].set_visible(False)
plt.tight_layout(rect=[0, 0, 1, .9])

# Main title
plt.suptitle(r"$\int_0^\infty r^{\mu+1}\ \exp(-r^2/2)\ J_\mu(k,r)\ " +
             r"k\ {\rm d}r = k^{\mu+1} \exp(-k^2/2)$", y=0.98)

plt.show()
```



Print values

```
print('            k                    a(k)        k^(mu+1) exp(-k^2/2)')
print('-------------------------------------------------------------------')
for i in range(n):
    print(f"{k[i]:18.6e} {ak[i]:18.6e} {theo[i]:18.6e}")
```

Out:

```
            k                    a(k)        k^(mu+1) exp(-k^2/2)
-------------------------------------------------------------------
      1.101130e-04       6.332603e-05       1.101130e-04
      1.468380e-04       9.168618e-05       1.468380e-04
      1.958116e-04       1.374282e-04       1.958116e-04
      2.611190e-04       2.131954e-04       2.611190e-04
      3.482078e-04       3.318802e-04       3.482077e-04
      4.643425e-04       4.923984e-04       4.643425e-04
      6.192107e-04       6.460278e-04       6.192106e-04
      8.257307e-04       7.968931e-04       8.257304e-04
      1.101130e-03       1.113736e-03       1.101129e-03
      1.468380e-03       1.464233e-03       1.468378e-03
      1.958116e-03       1.959475e-03       1.958112e-03
      2.611190e-03       2.610678e-03       2.611181e-03
      3.482078e-03       3.482260e-03       3.482056e-03
      4.643425e-03       4.643299e-03       4.643375e-03
      6.192107e-03       6.191999e-03       6.191988e-03
      8.257307e-03       8.257056e-03       8.257026e-03
      1.101130e-02       1.101057e-02       1.101063e-02
      1.468380e-02       1.468230e-02       1.468222e-02
```

(continues on next page)

(continued from previous page)

```
      1.958116e-02       1.957729e-02       1.957741e-02
      2.611190e-02       2.610314e-02       2.610300e-02
      3.482078e-02       3.479950e-02       3.479967e-02
      4.643425e-02       4.638444e-02       4.638422e-02
      6.192107e-02       6.180220e-02       6.180247e-02
      8.257307e-02       8.229239e-02       8.229205e-02
      1.101130e-01       1.094470e-01       1.094474e-01
      1.468380e-01       1.452640e-01       1.452635e-01
      1.958116e-01       1.920928e-01       1.920934e-01
      2.611190e-01       2.523680e-01       2.523671e-01
      3.482078e-01       3.277241e-01       3.277250e-01
      4.643425e-01       4.168889e-01       4.168871e-01
      6.192107e-01       5.111853e-01       5.111866e-01
      8.257307e-01       5.871956e-01       5.871927e-01
      1.101130e+00       6.005500e-01       6.005516e-01
      1.468380e+00       4.996049e-01       4.996187e-01
      1.958116e+00       2.879340e-01       2.879045e-01
      2.611190e+00       8.632888e-02       8.634968e-02
      3.482078e+00       8.102022e-03       8.108819e-03
      4.643425e+00       1.180344e-04       9.656964e-05
      6.192107e+00      -1.553139e-05       2.923736e-08
      8.257307e+00       7.225353e-06       1.291402e-14
      1.101130e+01      -2.588950e-06       5.164519e-26
      1.468380e+01       7.719794e-07       2.222595e-46
      1.958116e+01       1.586977e-07       1.078537e-82
      2.611190e+01      -1.874092e-07      2.285953e-147
      3.482078e+01       5.576689e-07      1.793712e-262
      4.643425e+01      -1.317041e-07       0.000000e+00
      6.192107e+01       6.415736e-07       0.000000e+00
      8.257307e+01       1.351283e-07       0.000000e+00
      1.101130e+02       7.997181e-07       0.000000e+00
      1.468380e+02       5.394094e-07       0.000000e+00
      1.958116e+02       1.165867e-06       0.000000e+00
      2.611190e+02       1.176786e-06       0.000000e+00
      3.482078e+02       1.889416e-06       0.000000e+00
      4.643425e+02       2.248731e-06       0.000000e+00
      6.192107e+02       3.228937e-06       0.000000e+00
      8.257307e+02       4.113223e-06       0.000000e+00
      1.101130e+03       5.651921e-06       0.000000e+00
      1.468380e+03       7.408687e-06       0.000000e+00
      1.958116e+03       1.001142e-05       0.000000e+00
      2.611190e+03       1.330606e-05       0.000000e+00
      3.482078e+03       1.792186e-05       0.000000e+00
      4.643425e+03       2.410633e-05       0.000000e+00
      6.192107e+03       3.277422e-05       0.000000e+00
      8.257307e+03       4.510046e-05       0.000000e+00
```

Total running time of the script: (0 minutes 0.595 seconds)

Geophysical Electromagnetic modelling

In this example we use *pyfftlog* to obtain time-domain EM data from frequency-domain data and vice versa. We do this by using analytical halfspace solution in both domains, and comparing the transformed responses to the true result. The analytical halfspace solutions are computed using *empymod* (see https://empymod.github.io).

```
import empymod
import pyfftlog
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import InterpolatedUnivariateSpline as iuSpline
```

Model and Survey parameters

```
# Impulse response (in the time domain)
signal = 0

# x-directed electric source and receiver point-dipoles
ab = 11

# We use the same range of times (s) and frequencies (Hz)
ftpts = np.logspace(-4, 4, 301)

# Source and receiver
src = [0, 0, 100]     # At the origin, 100 m below surface
rec = [6000, 0, 200]  # At an inline offset of 6 km, 200 m below surface

# Resistivity
depth = [0]       # Interface at z = 0, default for empymod.analytical
res = [2e14, 1]   # Horizontal resistivity [air, subsurface]
aniso = [1, 2]    # Anisotropy [air, subsurface]

# Collect parameters
analytical = {
    'src': src,
    'rec': rec,
    'res': res[1],
    'aniso': aniso[1],
    'solution': 'dhs',  # Diffusive half-space solution
    'verb': 2,
    'ab': ab,
}

dipole = {
    'src': src,
    'rec': rec,
    'depth': depth,
    'res': res,
    'aniso': aniso,
    'ht': 'dlf',
    'verb': 2,
    'ab': ab,
}
```

Analytical solutions

```
# Frequency Domain
f_ana = empymod.analytical(**analytical, freqtime=ftpts)

# Time Domain
t_ana = empymod.analytical(**analytical, freqtime=ftpts, signal=signal)
```

Out:

```
:: empymod END; runtime = 0:00:00.002148 ::


:: empymod END; runtime = 0:00:00.001233 ::
```

FFTLog

```
# FFTLog parameters
pts_per_dec = 5     # Increase if not precise enough
add_dec = [-2, 2]   # e.g. [-2, 2] to add 2 decades on each side
q = 0               # -1 - +1; can improve results

# Compute minimum and maximum required inputs
rmin = np.log10(1/ftpts.max()) + add_dec[0]
rmax = np.log10(1/ftpts.min()) + add_dec[1]
n = np.int(rmax - rmin)*pts_per_dec

# Pre-allocate output
f_resp = np.zeros(ftpts.shape, dtype=complex)

# Loop over Sine, Cosine transform.
for mu in [0.5, -0.5]:

    # Central point log10(r_c) of periodic interval
    logrc = (rmin + rmax)/2

    # Central index (1/2 integral if n is even)
    nc = (n + 1)/2.

    # Log spacing of points
    dlogr = (rmax - rmin)/n
    dlnr = dlogr*np.log(10.)

    # Compute required input x-values
    pts_req = 10**(logrc + (np.arange(1, n+1) - nc)*dlogr)/2/np.pi

    # Initialize FFTLog
    kr, xsave = pyfftlog.fhti(n, mu, dlnr, q, kr=1, kropt=1)

    # Compute pts_out with adjusted kr
    logkc = np.log10(kr) - logrc
    pts_out = 10**(logkc + (np.arange(1, n+1) - nc)*dlogr)

    # rk = r_c/k_r; adjust for Fourier transform scaling
    rk = 10**(logrc - logkc)*np.pi/2

    # Compute required times/frequencies with the analytical solution
    t2f_t_resp = empymod.analytical(**analytical, freqtime=pts_req,
                                    signal=signal)
    f2t_f_resp = empymod.analytical(**analytical, freqtime=pts_req)

    # Carry out FFTLog
    t2f_f_coarse = pyfftlog.fftl(t2f_t_resp, xsave.copy(), rk, 1)
    if mu > 0:
        f2t_t_coarse = pyfftlog.fftl(f2t_f_resp.imag, xsave.copy(), rk, 1)
    else:
        f2t_t_coarse = pyfftlog.fftl(f2t_f_resp.real, xsave.copy(), rk, 1)

    # Interpolate for required frequencies/times
    t2f_f_spline = iuSpline(np.log(pts_out), t2f_f_coarse)
    f2t_t_spline = iuSpline(np.log(pts_out), f2t_t_coarse)

    if mu > 0:
        f_resp += -1j*t2f_f_spline(np.log(ftpts))/np.pi/2
        t_resp_sin = -f2t_t_spline(np.log(ftpts))/np.pi*2
    else:
        f_resp += t2f_f_spline(np.log(ftpts))/np.pi/2
```

(continues on next page)

(continued from previous page)

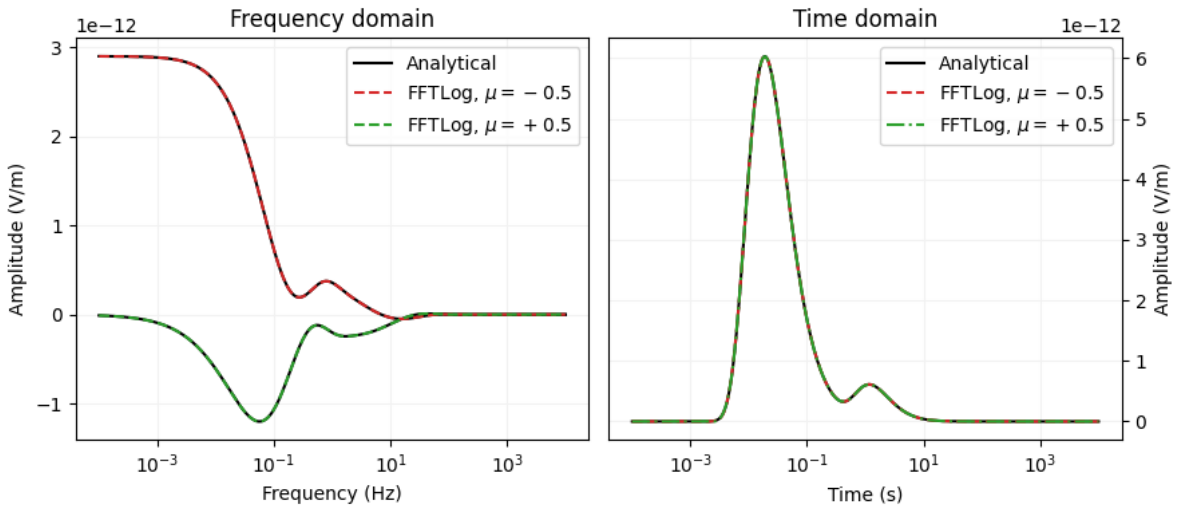
```
        t_resp_cos = f2t_t_spline(np.log(ftpts))/np.pi*2
```

Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/pyfftlog/checkouts/latest/
↪examples/geophysical_em.py:86: DeprecationWarning: `np.int` is a deprecated␣
↪alias for the builtin `int`. To silence this warning, use `int` by itself. Doing␣
↪this will not modify any behavior and is safe. When replacing `np.int`, you may␣
↪wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish␣
↪to review your current use, check the release note link for additional␣
↪information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  n = np.int(rmax - rmin)*pts_per_dec

:: empymod END; runtime = 0:00:00.000758 ::


:: empymod END; runtime = 0:00:00.001016 ::


:: empymod END; runtime = 0:00:00.000837 ::


:: empymod END; runtime = 0:00:00.000978 ::
```

Comparison

```
fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(9, 4))

# TIME DOMAIN
ax0.set_title(r'Frequency domain')
ax0.set_xlabel('Frequency (Hz)')
ax0.set_ylabel('Amplitude (V/m)')
ax0.semilogx(ftpts, f_ana.real, 'k-', label='Analytical')
ax0.semilogx(ftpts, f_ana.imag, 'k-')
ax0.semilogx(ftpts, f_resp.real, 'C3--', label=r'FFTLog, $\mu=-0.5$')
ax0.semilogx(ftpts, f_resp.imag, 'C2--', label=r'FFTLog, $\mu=+0.5$')
ax0.legend(loc='best')
ax0.grid(which='both', c='.95')

# TIME DOMAIN
ax1.set_title(r'Time domain')
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('Amplitude (V/m)')
ax1.semilogx(ftpts, t_ana, 'k', label='Analytical')
ax1.semilogx(ftpts, t_resp_cos, 'C3--', label=r'FFTLog, $\mu=-0.5$')
ax1.semilogx(ftpts, t_resp_sin, 'C2-.', label=r'FFTLog, $\mu=+0.5$')
ax1.legend(loc='best')
ax1.yaxis.set_label_position("right")
ax1.yaxis.tick_right()
ax1.grid(which='both', c='.95')

fig.tight_layout()
fig.show()
```



Total running time of the script: (0 minutes 0.561 seconds)

Examples contributed by users

Sine Transform

Contributed by @ShazAlvi.

This is a simple test program to illustrate how the sine (or cosine as it works basically the same way) Fourier transform works using *FFTLog*. The test provides as input as sine function and performs the sine Fourier transform. The input function is then recovered by performing an inverse Fourier transform. The inverse is performed using the following integral,

$$F(t) = \sqrt{\frac{\pi}{2}} \int_0^\infty A(f)\ \sin(ft)\ \mathrm{d}f\ . \tag{1.16}$$

```
import pyfftlog
import numpy as np
import scipy.integrate
import matplotlib.pyplot as plt
```

Define the parameters you wish to use

The presets are the *Reasonable choices of parameters* from *fftlogtest.f*.

```
# Range of periodic interval
logtmin = -3
# logtmax = 0.798 #2pi
# 5(2pi) #Longer range in r gives you a better reconstruction. 10\pi will give
# you a better reconstruction than 2\pi.
logtmax = 1.497
# Number of points (Max 4096)
# 1000 points give you a fairly smooth distribution of af in frequency, f.
# However you can get a good, working fit for 300 points as well.
n = 1000

# Order mu of Bessel function
mu = 0.5  # Choose -0.5 for cosine fourier transform
```

(continues on next page)

(continued from previous page)

```
# Bias exponent: q = 0 is unbiased
# The unbiased transforms give better results as far as I checked.
q = 0
# Sensible approximate choice of f_c t_c
# The output and the reconstruction is sensitive to the choice of this value
# This value is found by trial and error. In this example, the input function
# is a simple sine function which is not smooth in frequency space (as it
# only has one frequency) because of this reason a better value of this
# quantity is not found by the function fhti. For functions smooth
# in both time and frequency domain, the fhti should return the best
# value of the f_c t_c.

ft = 0.016

# Tell fhti to change ft to low-ringing value
# WARNING: kropt = 3 will fail, as interaction is not supported
ftopt = 1

# Forward transform (changed from dir to tdir, as dir is a python fct)
tdir = 1
```

Computation related to the logarithmic spacing

```
# Central point log10(t_c) of periodic interval
logtc = (logtmin + logtmax)/2

print(f"Central point of periodic interval at log10(t_c) = {logtc}")

# Central index (1/2 integral if n is even)
nc = (n + 1)/2.0

# Log-spacing of points
dlogt = (logtmax - logtmin)/n

dlnr = dlogt*np.log(10.0)
```

Out:

```
Central point of periodic interval at log10(t_c) = -0.7515
```

Compute input function: $\sin(t)$

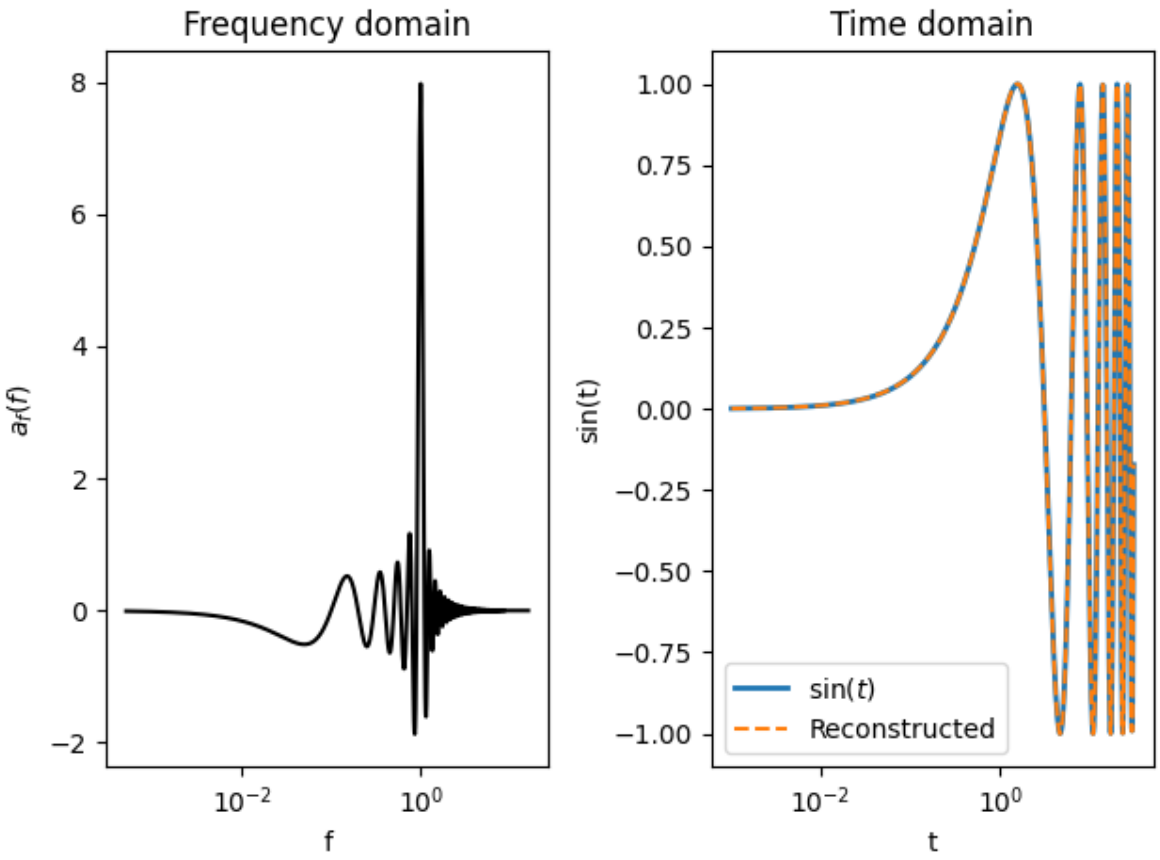
```
t = 10**(logtc + (np.arange(1, n+1) - nc)*dlogt)
a_t = np.sin(t)
```

Initialize FFTLog transform - note *fhti* resets *ft*

```
ft, xsave = pyfftlog.fhti(n, mu, dlnr, q, ft, ftopt)
```

Call *pyfftlog.fhtl*

```
logfc = np.log10(ft) - logtc

# Fourier sine Transform
a_f = pyfftlog.fftl(a_t.copy(), xsave, np.sqrt(2/np.pi), tdir)
# Notice that np.sqrt(2/np.pi) is the normalization factor for the transform
# Reconstruct the input function by taking the inverse fourier transform as
# given in the description
f = 10**(logfc + (np.arange(1, n+1) - nc)*dlogt)
# Array to store the reconstructed function for each value of t
Recon_Fun = np.zeros((len(t)))
for i in range(len(t)):
    Recon_Fun[i] = (np.sqrt(2/np.pi)**-1) * \
       scipy.integrate.trapz(f, a_f*np.sin(t[i]*f))

# Plotting the input function and the reconstructed input function and also
# the distribution of the a(f) vs f.
plt.figure()

ax1 = plt.subplot(121)
plt.title(r'Frequency domain')
plt.xlabel('f')
plt.ylabel(r'$a_f(f)$')
plt.semilogx(f, a_f, 'k')

ax2 = plt.subplot(122)
plt.title('Time domain')
plt.xlabel("t")
plt.ylabel("sin(t)")
plt.semilogx(t, a_t, lw=2, label=r'$\sin(t)$')
plt.semilogx(t, -Recon_Fun, '--', label='Reconstructed')
plt.legend()

plt.tight_layout()
plt.show()
```



Total running time of the script: (0 minutes 0.286 seconds)

1.3.3 References

1.3.4 Changelog

v0.2.0 : First packaged release

2020-05-16

First packaged release on PyPi and conda-forge. This includes:

- Re-structuring the repo.
- Add a proper documentation, https://pyfftlog.readthedocs.io, convert plain-text math into LaTeX, and add a reference section.
- Add example notebook as sphinx-gallery to docs.
- Add tests and CI on Travis, https://travis-ci.org/github/prisae/pyfftlog.
- Link to Zenodo, https://zenodo.org/record/3830366.
- PEP8 checking and coveralls (https://coveralls.io/github/prisae/pyfftlog).
- Add the relevant badges to README.

v0.1.1 : Bugfix uneven values

2019-08-16

- Small bugfix for uneven values.

v0.1.0 : Initial upload to GitHub

2016-12-09

- Initially working version uploaded to GitHub.

Bibliography

[Hami00] Hamilton, A. J. S., 2000, Uncorrelated modes of the non-linear power spectrum: Monthly Notices of the Royal Astronomical Society, 312, pages 257–284; DOI: 10.1046/j.1365-8711.2000.03071.x; Website of FFTLog: casa.colorado.edu/~ajsh/FFTLog.

[Talm78] Talman, J. D., 1978, Numerical Fourier and Bessel transforms in logarithmic variables: Journal of Computational Physics, 29, pages 35–48; DOI: 10.1016/0021-9991(78)90107-9.

Python Module Index

p

Index

F

K

P