scipy.fft Cheat Sheet

realpython.com

`scipy.fft` Cheat Sheet

To use the `scipy.fft` module, you need to remember a lot of math and function naming conventions. This cheat sheet contains some helpful tips in a condensed format, which should make using the library easier.

New to the `scipy.fft` module? Check out Fourier Transforms With scipy.fft: Python Signal Processing.

Note: Functions with names beginning with *i* are the inverse of the functions that share the name without the *i*.

`fft()`, `fft2()`, `fftn()`

Compute the fast Fourier transform on one-, two-, and *n*-dimensional input. Work on real or complex input.

`rfft()`, `rfft2()`, `rfftn()`

Compute the fast Fourier transform on one-, two-, and *n*-dimensional real input. Faster than the fft*() functions.

`hfft()`, `hfft2()`, `hfftn()`

Compute the fast Fourier transform on one-, two-, and *n*-dimensional input that has Hermitian symmetry. Produce real-valued output.

`dct()`, `dctn()`

Compute the discrete cosine transform on one- and *n*-dimensional input. Real-valued input and output. Often a better choice than `fft()` or `rfft()`.

`dst()`, `dstn()`

Compute the discrete sine transform on one- and *n*-dimensional input. Real-valued input and output.

`fftshift()`

By default, the `fft` and `fftfreq` functions return all the positive components, followed by all the negative components. This function swaps the two halves so that the zero-frequency component is in the center: `[0, 1, 2, -3, -2, -1]` → `[-3, -2, -1, 0, 1, 2, 3]`.

`fftfreq()`, `rfftfreq()`

Get the frequencies of each bin returned by the FFT functions, such as the X-axis.

`next_fast_len()`

The FFT functions work fastest at certain lengths of input. This function returns the next largest one for use with zero-padding.

`set_workers()`

Context manager to set the number of parallel worker threads used to compute the FFT.

`get_workers()`

Returns the number of workers used in the current context.

Note: I've left out backend-switching functions since, at the time of writing, SciPy ships with only one available backend.

DFT vs DCT vs DST

There are three main transforms that `scipy.fft` deals with: the discrete Fourier transform, the discrete cosine transform, and the discrete sine transform. This section is a quick reminder of what each transform is.

Discrete Fourier transform (DFT):

- Converts between a signal's time-domain representation and its frequency spectrum
- Uses both cosine and sine waves to decompose the signal into frequency components
- Handles complex input and output
- Assumes a function repeats to infinity

Discrete cosine transform (DCT):

- Does the same job as the DFT
- Uses only cosines to decompose the function
- Both input and output are real-valued
- Performs faster than the DFT
- Assumes a function has even symmetry (symmetry about the Y-axis), which extends to infinity

Discrete sine transform (DST):

- Does the same job as the DFT and the DST
- Uses sines to decompose the function
- Both input and output are real-valued
- Performs faster than the DFT
- Assumes a function has odd symmetry (symmetry about the origin) that extends to infinity
- May introduce high-frequency components if input data doesn't have odd symmetry