The General Fourier Family Transform (GFT) Library - Python Module

Introduction

The General Fourier Family Transform (GFT) describes all the transforms that use a Fourier-style complex sinusoidal kernel. The most common of these are the Fourier transform itself, and the short-time Fourier transform. As efficient algorithms are widely available to calculate these transforms, this library focuses on a more recent addition, the S-transform (ST). An ST produces local frequency spectra like a short-time Fourier transform, but offers progressive resolution, like a wavelet transform. That is, as the frequency increases, the ST trades off frequency resolution for temporal (or spatial) resolution, so higher frequencies can be better localized. This smooth tradeoff is responsible for many of the desirable properties of wavelets, and is considered by many to offer a better time-frequency representation of a signal.

This package implements the fast frequency domain GFT algorithm published by Brown *et. al.* (IEEE Transactions on Signal Processing, 58, 281-290, 2010), in both 1 dimension (signals) and two dimensions (images). If you publish work using the GFT you should cite this paper. If you feel like writing some code you can easily use the 1D transform to perform the GFT of any number of dimensions.

The fast GFT is an O(N log N) algorithm, which is the same order as the fast Fourier transform. In general, the GFT implemented in this library will take roughly twice as long to calculate as the FFT of the same signal.

Copyright and License

Requirements

The GFT C library requires a recent version of the FFTW library (http://www.fftw.org/ - tested with version 3.1.2). If FFTW is not installed in the standard location you may need to modify the setup.py script. The Python wrapper also requires Numpy (http://numpy.scipy.org/ - tested with version 1.4.1) and the examples require matplotlib (http://matplotlib.sourceforge.net/). You will also need a Python interpreter (http://python.org/ - tested with versions 2.4, 2.5 and 2.6) and a C build environment compatible with Python. The Python module uses Cython (http://www.cython.org/) to provide fast

access to the GFT C functions. Cython is not required to build this module but if you wish to modify the Python wrappers you will need Cython to rebuild them.

Installation

Installation is simple. Unpack the archive, change into the resulting directory, and, if you're using Linux, OS X or another UNIX-like system, type:

```
sudo python setup.py install
```

If you are using Windows, type:

```
sudo python setup.py install
```

Testing

Start a Python interpreter (not in the source directory) and type:

```
import PyGFT
```

If this works, chances are you have successfully installed the GFT Python library. To make doubly sure, enter:

```
import PyGFT.examples
PyGFT.examples.run()
```

Note that the examples only work if you have matplotlib installed. The examples.py file also functions as a tutorial for using the GFT library under Python and should be present in the same directory as this README file.

Interface

One Dimensional GFT

```
gft1d(sig,windowType='gaussian')
```

This function calculates the 1D GFT.

`sig` is a numpy array containing the signal to transform

windowType is an optional string parameter, indicating the type of window to use. Currently valid options are `'gaussian'` for Gaussian windows or `'box'` for boxcar windows. The default is Gaussian, but if you pass an invalid value the library will default to boxcars.

The function returns a one-dimensional complex64 numpy array containing the (packed) GFT of `sig` with the DC component at the centre.

Two Dimensional GFT

```
gft2d(image,windowType='gaussian')
```

This function calculates the 2D GFT, in wavelet standard form (all rows transformed, then all columns, yielding a full decomposition).

`image` is a 2D numpy array containing the image to transform

windowType is an optional string parameter, indicating the type of window to use. Currently valid options are `'gaussian'` for Gaussian windows or `'box'` for boxcar windows. The default is Gaussian, but if you pass an invalid value the library will default to boxcars.

The function returns a two-dimensional complex64 numpy array containing the (packed) GFT of `image` with the DC component at the centre. The layout of this array is similar to standard representations of the wavelet transform, except that there are additional quadrants corresponding to the negative frequencies.

Interpolators

Since the discrete GFT frequently does not produce a uniformly sampled result, interpolation of some form is required for a spectrogram-style display. These interpolation functions transform a GFT spectrum into a regularly sampled grid for display.

```
gft1dInterpolateNN(SIG,M=None)
```

Implements basic nearest-neighbor interpolation of a discrete GFT spectrum. `SIG` is the GFT spectrum and `M` is the desired output size. The output will be a uniformly sampled `MxM` grid that can be easily displayed. If `M` is `None` (the default), or equal to the length of `SIG`, pure interpolation is performed. If `M` is less than the length of `SIG`, downsampling will occur. Although some detail is lost, this option is useful to fit the spectrum to a particular size (to fill the screen,

for example) and, since the uniformly sampled spectrum is frequently very large, may be required due to memory restrictions.

Utility Functions

Several utility functions are also included, which you may find useful.

`phase(num)`

Takes as input a complex numpy array and returns a real array containing the phase at each point.

`shift(a,nx,ny=None)`

Takes as input an array of one or two dimensions and a shifts it along the *x* dimension by `nx` and along the *y* dimension by `ny`. For a 1D array `ny` is ignored.

`partitions(N)`

Takes as input the length of a 1D signal and returns the partitions used by the GFT. See examples.py for the return format and usage.

`gaussianWindows(N)`

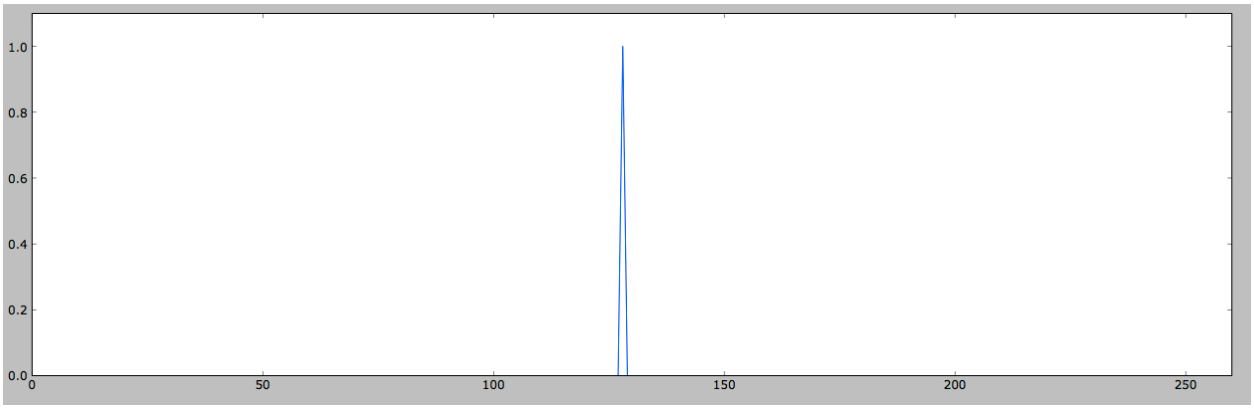
Takes as input the length of a 1D signal and returns the Gaussian windows used by the GFT. These windows are in the Fourier domain.

Future

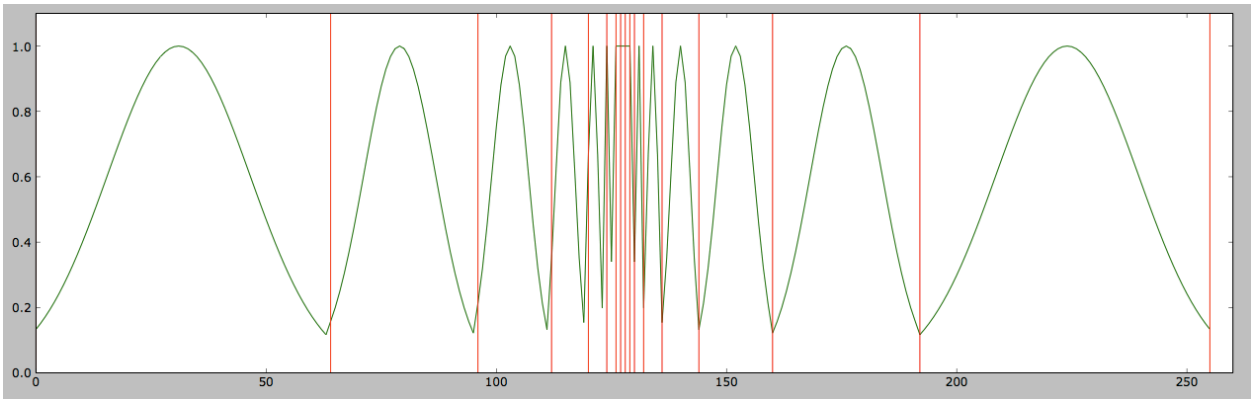
In this release the GFT only works with Gaussian or boxcar windows and a strict octave pattern of frequency partitions. Future releases will expose the full capability of the underlying C library, including the ability to specify custom windows and partitions. For the moment, if you need this functionality you will have to use the C library directly.

A GFT Example

Consider a simple signal, consisting of a single spike:

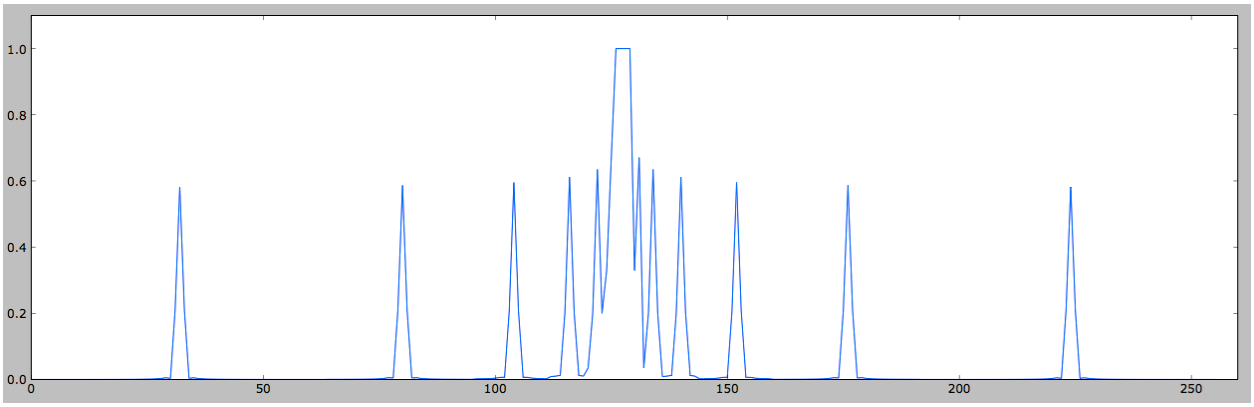


Since the frequency representation of a spike is a constant, the GFT spectrum should have a representation of the spike in each frequency band. To perform the GFT we must first choose windows and partitions. Gaussian windows and standard GFT partition layout are shown, windows in green and partitions in red, shifted so the 0 frequency is at the centre (zero frequency is normally the first element):

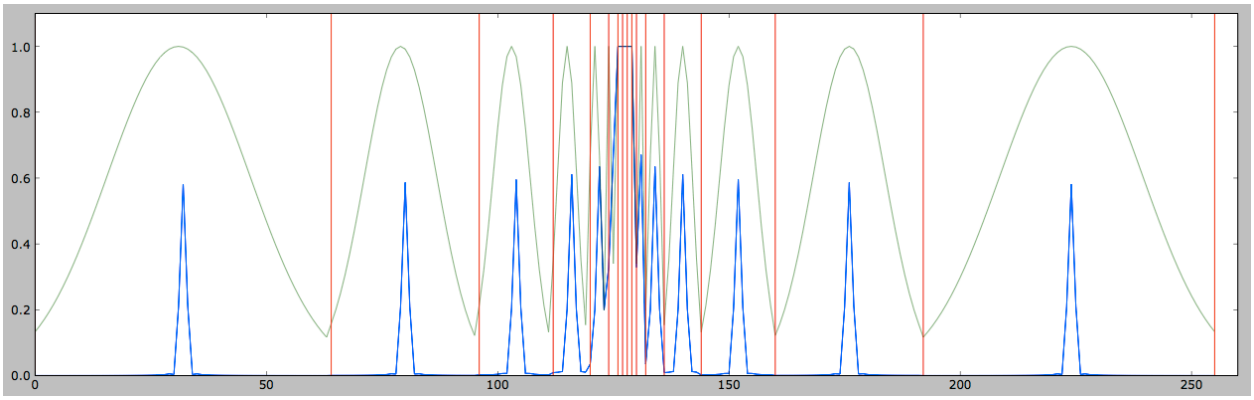


Note how, in the standard GFT partition scheme, the partition bandwidth doubles with each successively higher frequency band.

The GFT of the signal using the standard partition scheme and Gaussian windows looks like this:



Finally, with superimposed windows (transparent green) and partitions (red):



Note that the layout of the GFT spectrum is the same as a discrete wavelet spectrum except that negative frequencies are included. A more traditional spectrogram-like representation can be generated by interpolating the non-uniform sampling of the GFT partitions into uniformly sampled time and frequency axes: