X-FFT

December 7, 2014

1 X-FFT - Fast Fourier Transform

```
In [8]: from IPython.display import HTML
```

```
In [9]: HTML('http://www.ams.org/journals/mcom/1965-19-090/S0025-5718-1965-0178586-1/')
```

```
Out[9]: <IPython.core.display.HTML at 0x7f27b419fa90>
```

There are several FFT implementations in python. We will use the NumPy and SciPy ones. The fastest is PyFFTW.

```
In [9]: import numpy as np
        import scipy as scp
        %pylab
```

```
Using matplotlib backend: Qt4Agg
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: help(np.fft)
```

```
Help on package numpy.fft in numpy:

NAME
    numpy.fft

FILE
    /home/jpsilva/anaconda/lib/python2.7/site-packages/numpy/fft/__init__.py

DESCRIPTION
    Discrete Fourier Transform (:mod:`numpy.fft`)
    =============================================

    .. currentmodule:: numpy.fft

    Standard FFTs
    -------------

    .. autosummary::
       :toctree: generated/

       fft       Discrete Fourier transform.
       ifft      Inverse discrete Fourier transform.
       fft2      Discrete Fourier transform in two dimensions.
       ifft2     Inverse discrete Fourier transform in two dimensions.
```

```
   fftn      Discrete Fourier transform in N-dimensions.
   ifftn     Inverse discrete Fourier transform in N dimensions.

Real FFTs
---------

.. autosummary::
   :toctree: generated/

   rfft      Real discrete Fourier transform.
   irfft     Inverse real discrete Fourier transform.
   rfft2     Real discrete Fourier transform in two dimensions.
   irfft2    Inverse real discrete Fourier transform in two dimensions.
   rfftn     Real discrete Fourier transform in N dimensions.
   irfftn    Inverse real discrete Fourier transform in N dimensions.

Hermitian FFTs
--------------

.. autosummary::
   :toctree: generated/

   hfft      Hermitian discrete Fourier transform.
   ihfft     Inverse Hermitian discrete Fourier transform.

Helper routines
---------------

.. autosummary::
   :toctree: generated/

   fftfreq   Discrete Fourier Transform sample frequencies.
   rfftfreq  DFT sample frequencies (for usage with rfft, irfft).
   fftshift  Shift zero-frequency component to center of spectrum.
   ifftshift Inverse of fftshift.


Background information
----------------------

Fourier analysis is fundamentally a method for expressing a function as a
sum of periodic components, and for recovering the function from those
components.  When both the function and its Fourier transform are
replaced with discretized counterparts, it is called the discrete Fourier
transform (DFT).  The DFT has become a mainstay of numerical computing in
part because of a very fast algorithm for computing it, called the Fast
Fourier Transform (FFT), which was known to Gauss (1805) and was brought
to light in its current form by Cooley and Tukey [CT]_.  Press et al. [NR]_
provide an accessible introduction to Fourier analysis and its
applications.

Because the discrete Fourier transform separates its input into
components that contribute at discrete frequencies, it has a great number
of applications in digital signal processing, e.g., for filtering, and in
```

```
this context the discretized input to the transform is customarily
referred to as a *signal*, which exists in the *time domain*.  The output
is called a *spectrum* or *transform* and exists in the *frequency
domain*.

Implementation details
----------------------

There are many ways to define the DFT, varying in the sign of the
exponent, normalization, etc.  In this implementation, the DFT is defined
as

.. math::
   A_k =  \sum_{m=0}^{n-1} a_m \exp\left\{-2\pi i{mk \over n}\right\}
   \qquad k = 0,\ldots,n-1.

The DFT is in general defined for complex inputs and outputs, and a
single-frequency component at linear frequency :math:`f` is
represented by a complex exponential
:math:`a_m = \exp\{2\pi i\,f m\Delta t\}`, where :math:`\Delta t`
is the sampling interval.

The values in the result follow so-called "standard" order: If ``A =
fft(a, n)``, then ``A[0]`` contains the zero-frequency term (the mean of
the signal), which is always purely real for real inputs. Then ``A[1:n/2]``
contains the positive-frequency terms, and ``A[n/2+1:]`` contains the
negative-frequency terms, in order of decreasingly negative frequency.
For an even number of input points, ``A[n/2]`` represents both positive and
negative Nyquist frequency, and is also purely real for real input.  For
an odd number of input points, ``A[(n-1)/2]`` contains the largest positive
frequency, while ``A[(n+1)/2]`` contains the largest negative frequency.
The routine ``np.fft.fftfreq(n)`` returns an array giving the frequencies
of corresponding elements in the output.  The routine
``np.fft.fftshift(A)`` shifts transforms and their frequencies to put the
zero-frequency components in the middle, and ``np.fft.ifftshift(A)`` undoes
that shift.

When the input `a` is a time-domain signal and ``A = fft(a)``, ``np.abs(A)``
is its amplitude spectrum and ``np.abs(A)**2`` is its power spectrum.
The phase spectrum is obtained by ``np.angle(A)``.

The inverse DFT is defined as

.. math::
   a_m = \frac{1}{n}\sum_{k=0}^{n-1}A_k\exp\left\{2\pi i{mk\over n}\right\}
   \qquad m = 0,\ldots,n-1.

It differs from the forward transform by the sign of the exponential
argument and the normalization by :math:`1/n`.

Real and Hermitian transforms
-----------------------------

When the input is purely real, its transform is Hermitian, i.e., the
```

component at frequency :math:`f_k` is the complex conjugate of the
component at frequency :math:`-f_k`, which means that for real
inputs there is no information in the negative frequency components that
is not already available from the positive frequency components.
The family of `rfft` functions is
designed to operate on real inputs, and exploits this symmetry by
computing only the positive frequency components, up to and including the
Nyquist frequency. Thus, ``n`` input points produce ``n/2+1`` complex
output points. The inverses of this family assumes the same symmetry of
its input, and for an output of ``n`` points uses ``n/2+1`` input points.

Correspondingly, when the spectrum is purely real, the signal is
Hermitian. The `hfft` family of functions exploits this symmetry by
using ``n/2+1`` complex points in the input (time) domain for ``n`` real
points in the frequency domain.

In higher dimensions, FFTs are used, e.g., for image analysis and
filtering. The computational efficiency of the FFT means that it can
also be a faster way to compute large convolutions, using the property
that a convolution in the time domain is equivalent to a point-by-point
multiplication in the frequency domain.

Higher dimensions

In two dimensions, the DFT is defined as

```
.. math::
   A_{kl} =  \sum_{m=0}^{M-1} \sum_{n=0}^{N-1}
   a_{mn}\exp\left\{-2\pi i \left({mk\over M}+{nl\over N}\right)\right\}
   \qquad k = 0, \ldots, M-1;\quad l = 0, \ldots, N-1,
```

which extends in the obvious way to higher dimensions, and the inverses
in higher dimensions also extend in the same way.

References

.. [CT] Cooley, James W., and John W. Tukey, 1965, "An algorithm for the
machine calculation of complex Fourier series," *Math. Comput.*
19: 297-301.

.. [NR] Press, W., Teukolsky, S., Vetterline, W.T., and Flannery, B.P.,
2007, *Numerical Recipes: The Art of Scientific Computing*, ch.
12-13. Cambridge Univ. Press, Cambridge, UK.

Examples

For examples, see the various functions.

PACKAGE CONTENTS
    fftpack
    fftpack_lite

```
    helper
    info
    setup

DATA
    absolute_import = _Feature((2, 5, 0, 'alpha', 1), (3, 0, 0, 'alpha', 0...
    division = _Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192...
    print_function = _Feature((2, 6, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0)...
    using_mklfft = True
```

In [3]: help(scp.fft)

```
Help on function fft in module mklfft.fftpack:

fft(a, n=None, axis=-1)
    Compute the one-dimensional discrete Fourier Transform.

    This function computes the one-dimensional *n*-point discrete Fourier
    Transform (DFT) with the efficient Fast Fourier Transform (FFT)
    algorithm [CT].

    Parameters
    ----------
    a : array_like
        Input array, can be complex.
    n : int, optional
        Length of the transformed axis of the output.
        If `n` is smaller than the length of the input, the input is cropped.
        If it is larger, the input is padded with zeros.  If `n` is not given,
        the length of the input along the axis specified by `axis` is used.
    axis : int, optional
        Axis over which to compute the FFT.  If not given, the last axis is
        used.

    Returns
    -------
    out : complex ndarray
        The truncated or zero-padded input, transformed along the axis
        indicated by `axis`, or the last one if `axis` is not specified.

    Raises
    ------
    IndexError
        if `axes` is larger than the last axis of `a`.

    See Also
    --------
    numpy.fft : for definition of the DFT and conventions used.
    ifft : The inverse of `fft`.
    fft2 : The two-dimensional FFT.
    fftn : The *n*-dimensional FFT.
    rfftn : The *n*-dimensional FFT of real input.
    fftfreq : Frequency bins for given FFT parameters.

    Notes
```

```
-----
FFT (Fast Fourier Transform) refers to a way the discrete Fourier
Transform (DFT) can be calculated efficiently, by using symmetries in the
calculated terms.  The symmetry is highest when `n` is a power of 2, and
the transform is therefore most efficient for these sizes.

The DFT is defined, with the conventions used in this implementation, in
the documentation for the `numpy.fft` module.

References
----------
.. [CT] Cooley, James W., and John W. Tukey, 1965, "An algorithm for the
        machine calculation of complex Fourier series," *Math. Comput.*
        19: 297-301.

Examples
--------
>>> np.fft.fft(np.exp(2j * np.pi * np.arange(8) / 8))
array([ -3.44505240e-16 +1.14383329e-17j,
         8.00000000e+00 -5.71092652e-15j,
         2.33482938e-16 +1.22460635e-16j,
         1.64863782e-15 +1.77635684e-15j,
         9.95839695e-17 +2.33482938e-16j,
         0.00000000e+00 +1.66837030e-15j,
         1.14383329e-17 +1.22460635e-16j,
         -1.64863782e-15 +1.77635684e-15j])

>>> import matplotlib.pyplot as plt
>>> t = np.arange(256)
>>> sp = np.fft.fft(np.sin(t))
>>> freq = np.fft.fftfreq(t.shape[-1])
>>> plt.plot(freq, sp.real, freq, sp.imag)
[<matplotlib.lines.Line2D object at 0x...>, <matplotlib.lines.Line2D object at 0x...>]
>>> plt.show()

In this example, real input has an FFT which is Hermitian, i.e., symmetric
in the real part and anti-symmetric in the imaginary part, as described in
the `numpy.fft` documentation.
```

Let's obtain the FFT of

$$e^{\frac{i2\pi k}{8}}$$

for $k = 1, ..., 8$

```
In [15]: np.fft.fft(np.exp(2j * np.pi * np.arange(8) / 8))
```

```
Out[15]: array([ -3.44505240e-16 +1.14383329e-17j,
                  8.00000000e+00 -8.52057261e-16j,
                  2.33482938e-16 +1.22460635e-16j,
                  0.00000000e+00 +1.22460635e-16j,
                  9.95839695e-17 +2.33482938e-16j,
                 -8.88178420e-16 +1.17293449e-16j,
                  1.14383329e-17 +1.22460635e-16j,   0.00000000e+00 +1.22460635e-16j])
```

```
In [16]: x = np.exp(2j * np.pi * np.arange(9) / 8)
         xt = np.fft.fft(x)
```

```
In [17]: plot(xt)
```

```
Out[17]: [&lt;matplotlib.lines.Line2D at 0x7f4897a5dc50&gt;]
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

```
In []:
```

1.0.1 Optional

- Install FFTW sudo apt-get install fftw3 libfftw3-dev
- Install PyFFTW pip install PyFFTW

```
In [1]: import pyfftw
```

```
In []:
```

```
In []:
```

```
In [7]: %load_ext version_information
        %version_information numpy,scipy,pyfftw
```

```
The version_information extension is already loaded. To reload it, use:
  %reload_ext version_information
```

Out[7]:

Software	Version
Python	2.7.8 —Anaconda 2.1.0 (64-bit)— (default, Aug 21 2014, 18:22:21) [GCC 4.4.7 20120313 (Red Hat 4.4.7-1)
IPython	2.3.1
OS	posix [linux2]
numpy	1.9.1
scipy	0.14.0
pyfftw	0.9.2
Fri Dec 05 14:20:07 2014 CET	

In [2]:
```
from IPython.core.display import HTML
def css_styling():
    styles = open("./styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[2]: `<IPython.core.display.HTML at 0x7fcfb8ffd550>`

In []: