用 Numpy 做 Fourier 变换

XU Xiang-hua

November 2, 2012

Contents

1 Fourier 变换

Fourier 变换是一种线性的积分变换。没有加限定语时，Fourier 变换就指连续 Fourier 变换，

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}\mathrm{d}t \tag{1}$$

在实际应用中，原函数 f(t) 通常是随时间变化的函数（t 指时间），变换后，ω 具有频率的性质（圆频率，ω=2π f , f 是频率），因此函数的自变量从时间变成了频率，这就是常说的『从时域变为频域』。因为通常随时间的变化规律是杂乱的，难以看出规律的，而当进行 Fourier 变换后，从信号的频率变化上就容易找出规律。所以 Fourier 变换在信号处理中非常有用。

Fourier 变换是可逆的，其逆变换为：

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{i\omega t}\mathrm{d}\omega \tag{2}$$

2 Fourier 级数

当原函数 f(x) 为周期函数时，Fourier 变换变为 Fourier 级数，

$$f(x) = \sum_{n=-\infty}^{\infty} F_n e^{inx} \tag{3}$$

其中，F_n 按下式计算：

$$F_n = \frac{1}{T}\int_0^T f(x) e^{-ik\frac{2\pi}{T}t} \mathrm{d}t \tag{4}$$

其中 T 为 f(x) 的周期。

3 离散 Fourier 变换

在实际应用中，原函数通常是在有限的离散点上定义的，比如采集某个设备的温度，每隔一定的时间间隔采集一个信号，建立的是离散的温度与离散的时间之间的离散对应关系。在这种情况下，就要用到『离散 Fourier 变换』。

离散 Fourier 变换为：

$$x_n = \sum_{k=0}^{N-1} X_k e^{-i\frac{2\pi}{N}kn} \qquad n = 0, 1, ...N-1 \tag{5}$$

其中 X 是原信号，比如采集温度信号时，按相同的时间间隔采集，X 就是各个时间点上的温度。

那么变换后的 x 是什么呢？x 是长度为 N 的复数数组。其性质为：

- x_0 就是 X 的和，如果除以长度，就是 X 的平均值，可以看作是 X 的直流成分；
- x_i 和 $x_{\mathrm{N-i}}$ 共轭，即二者的实部相同，虚部反号；
- x_i 表示周期为 N/i (实际上还要乘以时间步长) 的波动成分，具体的，实部表示余弦波成分，虚部表示正弦波成分。

下面用 numpy.fft 进行实例说明。fft（fast Fourier tranform，快速 Fourier 变换）是一种高效的离散 Fourier 变换算法，应用非常普遍。它做的事情就是前面的离散 Fourier 变换。

3.1 例 1.

在一个周期 [0,2π] 的正弦波中均匀取出 8 个点，做 fft，看能否识别出这个波。

```
import numpy as np
from math import pi
x = np.arange(0,2*pi,pi/4)
y = np.sin(x)
yf = np.fft.fft(y)/len(y)
print yf
```

```
[  1.43029718e-18 +0.00000000e+00j  -4.44089210e-16 -5.00000000e-01j
   1.53080850e-17 -1.38777878e-17j   3.87727691e-17 -1.11022302e-16j
   2.91858728e-17 +0.00000000e+00j   0.00000000e+00 -1.11022302e-16j
   1.53080850e-17 +1.38777878e-17j   3.44084101e-16 +5.00000000e-01j]
```

结果中的绝大部分都是 0（由于计算误差，得到的是很小的数），非 0 的是第一项和倒数第一项的虚部，这表示存在周期为 N 的正弦波动，幅度是 -0.5（实际的幅度是 1，相差一个倍数）。可以看出，识别地很准确。

3.2 例 2.

1 个周期 $[0,2\pi]$ 的正弦波叠加一个幅度减半、周期减半的余弦波，均匀取 8 个点，看能否用 fft 识别。

```
import numpy as np
from math import pi
x=np.arange(0,2*pi,pi/4)
y=np.sin(x)+0.5*np.cos(2*x)
yf=np.fft.fft(y)/len(y)
print yf
```

```
[ -4.16333634e-17 +0.00000000e+00j  -4.42658913e-16 -5.00000000e-01j
   2.50000000e-01 -6.93889390e-17j   4.02030662e-17 -1.11022302e-16j
   4.16333634e-17 +0.00000000e+00j   1.43029718e-18 -1.11022302e-16j
   2.50000000e-01 +6.93889390e-17j   3.45514398e-16 +5.00000000e-01j]
```

可以看出 fft 识别出了两个波动，一个是整周期的正弦，幅度 -0.5，一个是减半周期的余弦，幅度 0.25 。虽然只有 8 个点（对于减半周期的余弦，每个周期只有 4 个点），却能够准确识别出 2 个叠加的波动，其实有巧合的成分，因为正弦和余弦波是 Fourier 变换的基本元素。如果原波动中的余弦波变为相同周期的三角波，也会被识别为余弦波动。因此为了更准确地进行识别和分析，测量点不能太稀疏。

4 关于 numpy.fft

前面已经用 numpy.fft.fft 对离散 Fourier 变换进行了演示。现在再具体介绍一下 numpy.fft 。

4.1 fft

numpy.fft 包含了系列的离散 Fourier 变换函数，包括一维、二维，正变换、逆变换，实变换等，还有几个辅助函数。

fft 是最常用的一维『标准』离散 Fourier 变换，其输入是一维数组（还有可选参数 n，axis，可参见帮助），输出则是一维复数数组。输出的结果按照所谓的『标准』顺序排列，如 A=fft(a,n) ，则：

- A[0] 是直流成分
- A[1:n/2] 包含的是正频率的项，按照频率的升序排列
- A[n/2+1:] 包含负频率的项，按照频率的降序排列。前面已经提到过正频率和负频率的项相互共轭
- 如果 n 是偶数，则 A[n/2] 包含了正负 Nyquist 频率；如果 n 为奇数，则 A[(n-1)/2] 包含了最大的正频率项，A[(n+1)/2] 包含了最大的负频率项

标准输出格式便于理解，但不利于数据的处理和展示。为此 numpy 提供了两个辅助函数，fftfreq 和 fftshift。

4.2 fftfreq

fftfreq(n, d=1.0) 返回的是 fft 输出的标准格式相对应的频率数组，输出为

```
f = [0, 1, ..., n/2-1, -n/2, ..., -1] / (d*n)         if n is even
f = [0, 1, ..., (n-1)/2, -(n-1)/2, ..., -1] / (d*n)   if n is odd
```

d 是采样的（时间）间隔。

结合 fftfreq 能够方便地将 fft 的结果绘图。看例子：

```
import numpy as np
from math import pi
import matplotlib.pyplot as plt
n=32
x=np.arange(0,2*pi,2*pi/n)
y=np.sin(x)+0.5*np.cos(2*x)+2*np.sin(3*x)+3*np.cos(4*x)
yf=np.fft.fft(y)/n
f=np.fft.fftfreq(n,d=1./n)
fig=plt.figure(figsize=[6,4.5],dpi=120)
plt.subplot(211)
plt.bar(f-0.3, yf.real,width=0.6)
plt.xlabel('f')
plt.ylabel('cos')
plt.xlim(-16,16)
plt.subplot(212)
plt.bar(f-0.3,yf.imag,width=0.6)
plt.xlabel('f')
plt.ylabel('sin')
plt.xlim(-16,16)
plt.savefig('img/fft_fftfreq.png')
```

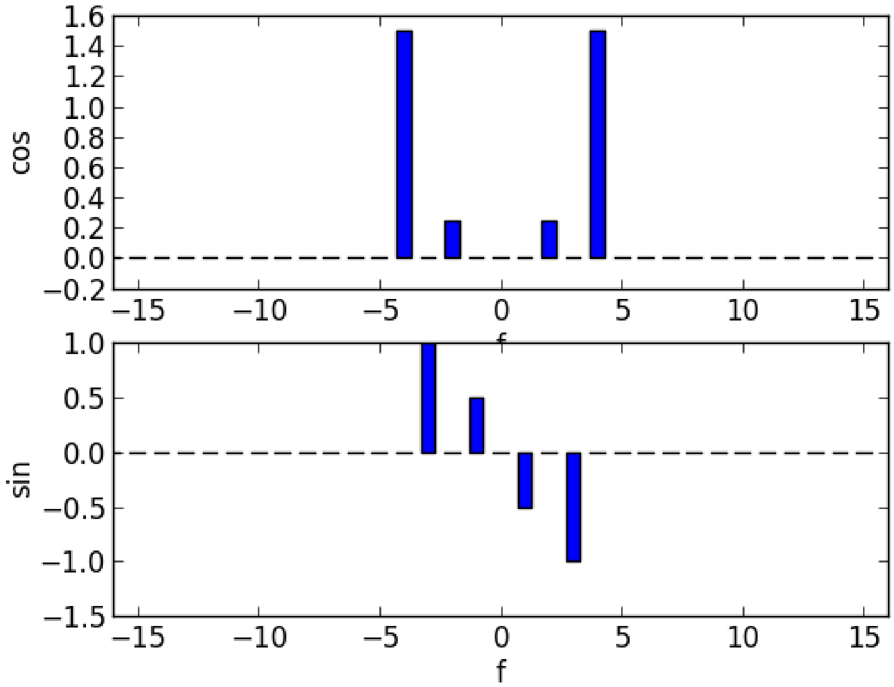

Figure1: 用 fftpfreq 绘制 fft 结果

4.3 fftshift

fftshift(x, axis=None) 把 fft 的输出进行重新排序，将结果按照频率升序排列，0 频率置于中间。也可以用于结果的显示。

通过下面的例子能够看到 fftshift 的作用。

```
import numpy as np
from math import pi
x = np.arange(0,2*pi,pi/4)
y = np.cos(x)+np.sin(2*x)
yf = np.fft.fft(y)
print yf
print np.fft.fftshift(yf)
```

```
 [ -7.66951701e-17 +0.00000000e+00j   4.00000000e+00 -1.66533454e-15j
   -5.89438645e-16 -4.00000000e+00j   8.88178420e-16 +1.44328993e-15j
    3.67394040e-16 +0.00000000e+00j  -8.88178420e-16 +1.66533454e-15j
   -5.89438645e-16 +4.00000000e+00j   4.00000000e+00 -1.44328993e-15j]
 [  3.67394040e-16 +0.00000000e+00j  -8.88178420e-16 +1.66533454e-15j
   -5.89438645e-16 +4.00000000e+00j   4.00000000e+00 -1.44328993e-15j
```

```
 -7.66951701e-17 +0.00000000e+00j   4.00000000e+00 -1.66533454e-15j
 -5.89438645e-16 -4.00000000e+00j   8.88178420e-16 +1.44328993e-15j]
```

5 Fourier 变换和 Fourier 级数的例子

下面对锯齿波进行 fft，然后利用结果得到的 Fourier 级数查看逼近结果。

```
import numpy as np
from math import pi
import matplotlib.pyplot as plt

def zigzeg(n):
    x=np.arange(0,1.,1./n)
    y=1-x
    return x,y

def fourier_series(f,n,m=1):
    N = len(f)*m
    y = np.zeros(N)
    x = np.arange(0, N, 1.)/len(f) * 2*pi
    for k,p in enumerate(f[:n]):
        if k != 0:
            p *= 2
        y += np.real(p) * np.cos(k*x)
        y -= np.imag(p) * np.sin(k*x)
    return x, y

n = 256
x,y = zigzeg(n)
yf = np.fft.fft(y)/n
plt.figure(figsize=[6,4.5], dpi=120)
plt.bar(range(20), np.abs(yf[:20]), width=0.4)
plt.xlabel('freq')
plt.ylabel('amplitude')
plt.savefig('./img/zigzeg_fft_freq.png')
plt.figure(figsize=[8,6], dpi=120)
plt.plot(y,linewidth=2,label='zigzeg')
for i in xrange(1,18,4):
    x,y = fourier_series(yf, i+1, 2)
    plt.plot(y,linewidth=1,label='N=%d'%i)
plt.legend(loc='best')
plt.savefig('img/zigzeg_fouier_series.png')
```

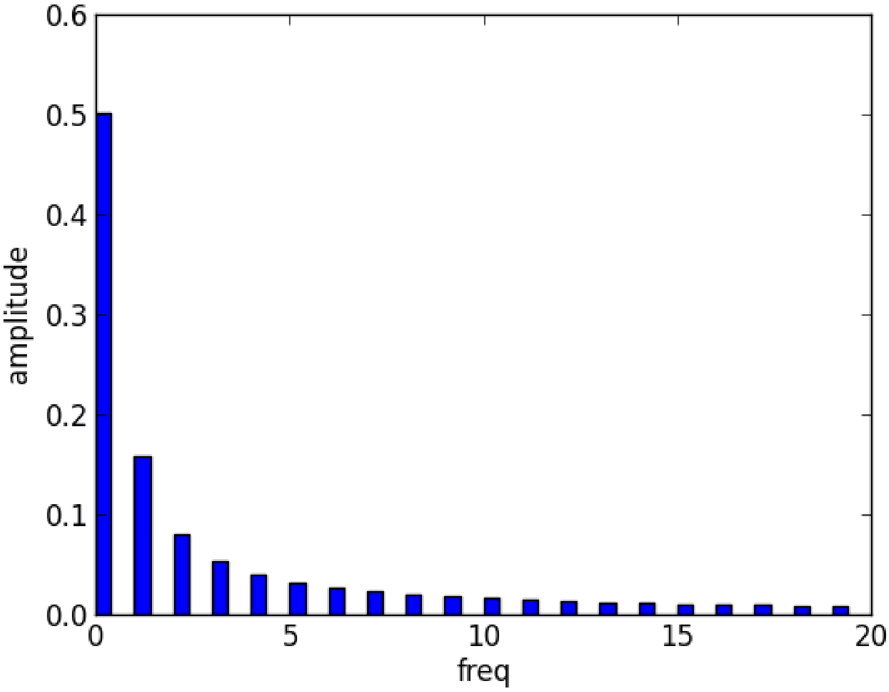

Figure2: 锯齿波经过 Fourier 变换得到的前 20 阶振幅

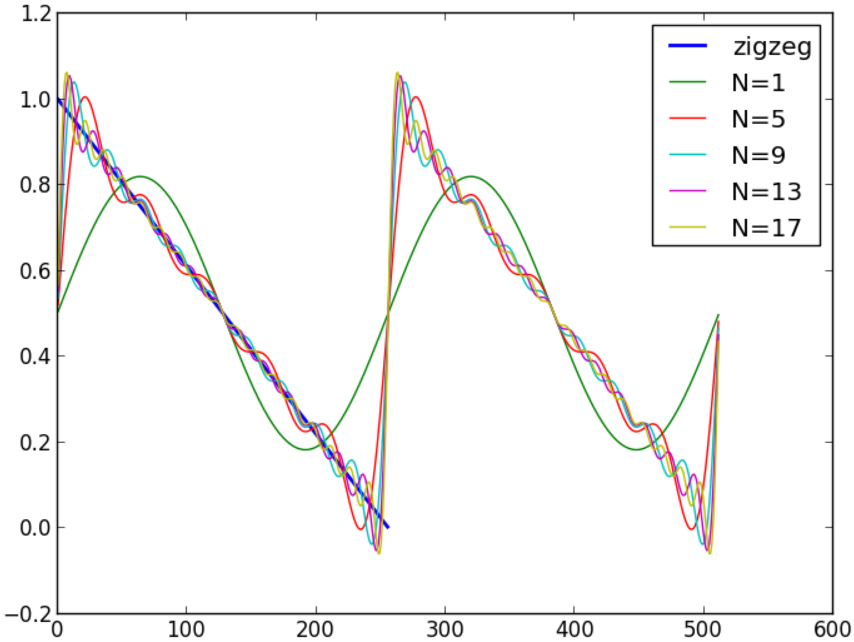

Figure3: Fourier 级数逼近的锯齿波