

FFT与锁相放大器

- 傅里叶级数到傅里叶变换
- 信号的采样与离散信号
- DFT与FFT
- 相关性检测
- PSD(phase-sensitive-detect)

熟悉又陌生的数学公式：

$$\begin{aligned}\int dx f(x) &= \Delta x \sum_n f(x_n) \\ \sum_n \delta(t - nT) &= \frac{1}{T} \sum_n e^{jn\frac{2\pi}{T}t}\end{aligned}\tag{1}$$

前者为积分离散化，由高等数学积分的定义立得。在固体物理中，它常用于对布里渊区内波矢 \vec{k} 的积分，即 $\sum_{\vec{k} \in BZ} f(\vec{k}) \approx (\frac{L}{2\pi})^D \int_{BZ} f(\vec{k}) d\vec{k}$ ，例如当 $f(\vec{k}) = \frac{\hbar^2 k^2}{2m}$ 时，表示布里渊区内所有态(波矢 \vec{k} ，考虑自旋时一个波矢对应2个态)的动能之和。

第二个等式的证明如下：

$$\begin{aligned}\frac{1}{T} \sum_{n=0}^N e^{jn\frac{2\pi}{T}t} &= \frac{1}{T} \times 1 \times \frac{1 - e^{j(N+1)\frac{2\pi}{T}t}}{1 - e^{j\frac{2\pi}{T}t}} \\ &= \begin{cases} 0, & \frac{2\pi t}{T} \neq n2\pi \\ \frac{N+1}{T}, & \frac{2\pi t}{T} = n2\pi \end{cases} \\ \sum_{n=0}^N \delta(t - nT) &= \frac{1}{T} \sum_{n=0}^N \delta(n - \frac{t}{T}) \\ &= \begin{cases} 0, & \frac{t}{T} \neq n \\ \frac{N+1}{T}, & \frac{t}{T} = n \end{cases}\end{aligned}$$

在固体物理中，第二个等式在声子振动等理论中经常用到，可以表示周期性晶格的傅里叶级数：

$$\begin{aligned}\sum_n e^{i\vec{q} \cdot \vec{R}_n} &= N \sum \vec{G} \delta_{\vec{q}, \vec{G}} \\ \sum_{n=1}^N e^{inaq} &= e^{iaq} \frac{1 - e^{iNaq}}{1 - e^{iaq}} = \begin{cases} 0, & q \neq G \\ N, & q = G \end{cases}\end{aligned}$$

如果波矢 \vec{q} (或 \vec{k})被限制在第一布里渊区内：

$$\sum_n e^{i\vec{q} \cdot \vec{R}_n} = N \delta_{\vec{q}, 0}$$

傅里叶级数到傅里叶变换

正交函数系： $\{f_0, f_1, f_2, f_3, \dots, f_n\}$

$= \{\cos 0, \cos(\omega_0 x), \sin(\omega_0 x), \cos(2\omega_0 x), \sin(2\omega_0 x), \dots, \cos(n\omega_0 x), \sin(n\omega_0 x)\}$

满足 $\langle \cos m\omega_0 x, \sin n\omega_0 x \rangle = 0$,

$$\begin{aligned} \langle \sin n\omega_0 x, \sin m\omega_0 x \rangle &= 0 \quad (n \neq m), \\ \langle \cos n\omega_0 x, \cos m\omega_0 x \rangle &= 0 \quad (n \neq m) \end{aligned}$$

其中 $\langle f, g \rangle = \int f(x)g(x)dx$ 为函数内积，即三角函数集中的元素彼此内积为零，是正交函数系。任意一个周期为 T 的信号 $f(t) = f(t + T)$ 可以由三角函数作为基底展开，展开的级数称为**傅里叶级数**。

傅里叶级数：

对于周期为 T 的函数 $f_T(x)$ ， $\omega_0 = \frac{2\pi}{T}$

$$\begin{aligned} f_T(x) &= \frac{a_0}{2} + \sum_{n=1}^{+\infty} a_n \cos(n\omega_0 x) + \sum_{n=1}^{+\infty} b_n \sin(n\omega_0 x) \\ &= \frac{a_0}{2} + \sum_{n=1}^{+\infty} \frac{1}{2} (a_n - ib_n) e^{in\omega_0 x} + \sum_{n=-\infty}^{-1} \frac{1}{2} (a_{-n} + ib_{-n}) e^{in\omega_0 x} \\ &= \sum_{n=-\infty}^{+\infty} d_n e^{in\omega_0 x} \end{aligned}$$

$$d_n = \begin{cases} \frac{1}{2}(a_n - ib_n), & n > 0 \\ \frac{1}{2}a_0, & n = 0 \\ \frac{1}{2}(a_{-n} + ib_{-n}), & n < 0 \end{cases} \quad (2)$$

其中

$$\begin{aligned} a_0 &= \frac{1}{T} \int_0^T f(x) dx \\ a_n &= \frac{2}{T} \int_0^T \cos(n\omega_0 x) f(x) dx \\ b_n &= \frac{2}{T} \int_0^T \sin(n\omega_0 x) f(x) dx \end{aligned}$$

我们用复数表示，形式更加简洁。易证对于 $n \in (-\infty, +\infty)$ ，均有：

$$d_n = \frac{1}{T} \int_0^T f_T(x) e^{-in\omega_0 x} dx$$

$$\begin{aligned} f_T(x) &= \sum_{n=-\infty}^{+\infty} d_n e^{in\omega_0 x} \\ &= \sum_{n=-\infty}^{+\infty} \left[\frac{1}{T} \int_0^T f_T(x') e^{-in\omega_0 x'} dx' \right] e^{in\omega_0 x} \\ &= \sum_{n=-\infty}^{+\infty} \left[\frac{\omega_0}{2\pi} \int_0^T f_T(x') e^{-in2\pi\omega x'} dx' \right] e^{in2\pi\omega x} \\ &= \sum_{n=-\infty}^{+\infty} \left[\omega \int_0^T f_T(x') e^{-in2\pi\omega x'} dx' \right] e^{in2\pi\omega x} \\ T \rightarrow +\infty : &= \int_{-\infty}^{+\infty} \left[\int_{-\infty}^{+\infty} f(x') e^{-in2\pi\omega x'} dx' \right] e^{in2\pi\omega x} d(n\omega) \\ &= \int_{-\infty}^{+\infty} \left[\int_{-\infty}^{+\infty} f(x') e^{-i\omega' x'} dx' \right] e^{i\omega' x} d\frac{\omega'}{2\pi} \\ &= \frac{1}{2\pi} \int_{-\infty}^{+\infty} \left[\int_{-\infty}^{+\infty} f(x') e^{-i\omega' x'} dx' \right] e^{i\omega x} d\omega \\ &= \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega) e^{i\omega x} d\omega \end{aligned}$$

其中

$$\begin{aligned} F(\omega) &= \int_{-\infty}^{+\infty} f(x') e^{-i\omega x'} dx' \\ f(x) &= \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega) e^{i\omega x} d\omega \end{aligned}$$

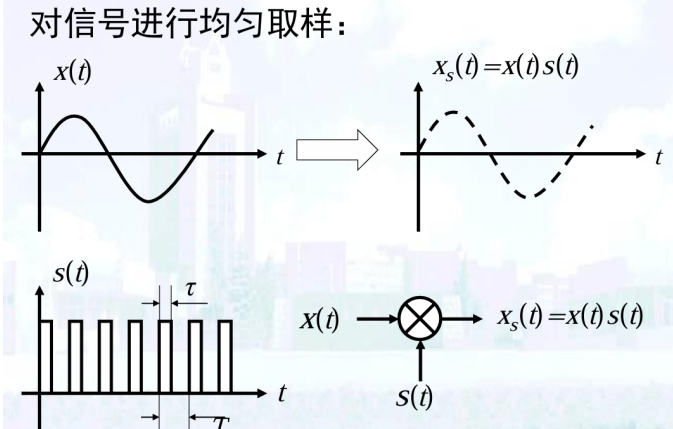
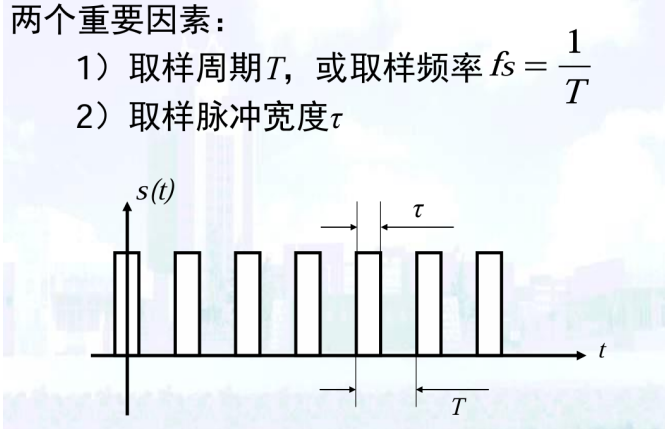
此即为非周期信号的傅里叶级数——**傅里叶变换**及其逆变换。推导过程使用了(1)中的求和化积分公式，并多次进行了变量代换。当 $T \rightarrow +\infty$ 时， $\omega_0 = \frac{2\pi}{T} = 2\pi\omega \rightarrow 0$ ， $n\omega$ 视为被积分的自变量。

信号的采样与离散信号

尽管傅里叶级数和傅里叶变换看上去优美而简洁，但是对于计算机来说，时域采得的信号永远是离散的。前述傅里叶级数与傅里叶变换都是对连续变化的信号所算得的，为了解决实际的信号处理问题，我们需要了解离散的信号对傅里叶级数和傅里叶变换的影响，即我们需要了解**离散傅里叶级数(DFS)**和**离散傅里叶变换(DFT)**。

信号的采样：

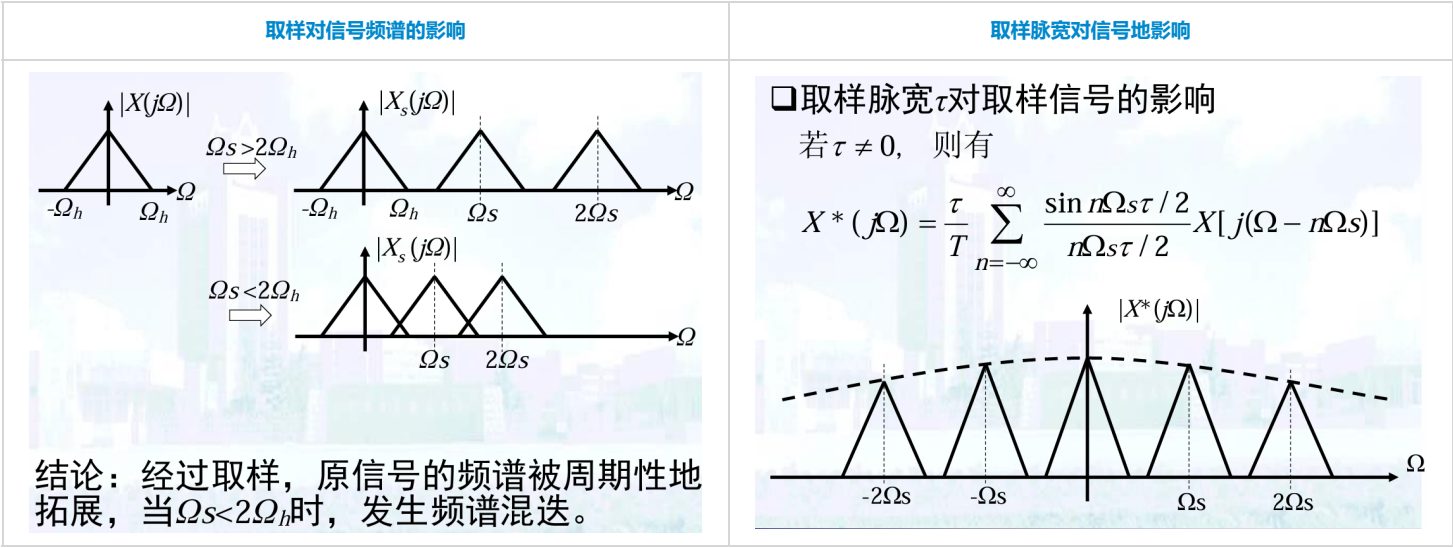
如图所示，取样可是看作是用离散的取样间隔为 T_s 的取样函数 $s(t) = \sum_{n=-\infty}^{+\infty} \delta(t - nT_s)$ 乘以连续的信号 $x(t)$ 的到的离散的输出 $x_s(t)$ 。这里不考虑取样函数的脉冲宽度，视为 δ 函数。先下一个暴论：**时域离散，频域就会有周期性；频域离散，时域就会有周期性。**

$x_s(t) = x(t)s(t)$ 信号均匀取样	取样函数 $s(t)$ 的参数
<p>对信号进行均匀取样：</p> 	<p>两个重要因素：</p> <ol style="list-style-type: none"> 1) 取样周期T，或取样频率$f_s = \frac{1}{T}$ 2) 取样脉冲宽度τ 

由前述公式(1)中的采样函数的傅里叶级数展开可得：

$$\begin{aligned}x_s(t) &= s(t)x(t) = \sum_{n=-\infty}^{+\infty} \delta(t - nT_s)x(t) = x(t)\frac{1}{T_s} \sum_{n=-\infty}^{+\infty} e^{jn\frac{2\pi}{T_s}t} \\&\Rightarrow \\X_s(\omega) &= \int_{-\infty}^{+\infty} x_s(t)e^{-j\omega t}dt \\&= \int_{-\infty}^{+\infty} \frac{1}{T_s} \sum_{n=-\infty}^{+\infty} e^{jn\omega_s t} x(t)e^{-j\omega t}dt \\&= \frac{1}{T_s} \sum_{n=-\infty}^{+\infty} \int_{-\infty}^{+\infty} x(t)e^{j(n\omega_s - \omega)t}dt \\&= \frac{1}{T_s} \sum_{n=-\infty}^{+\infty} X(\omega - n\omega_s)\end{aligned}$$

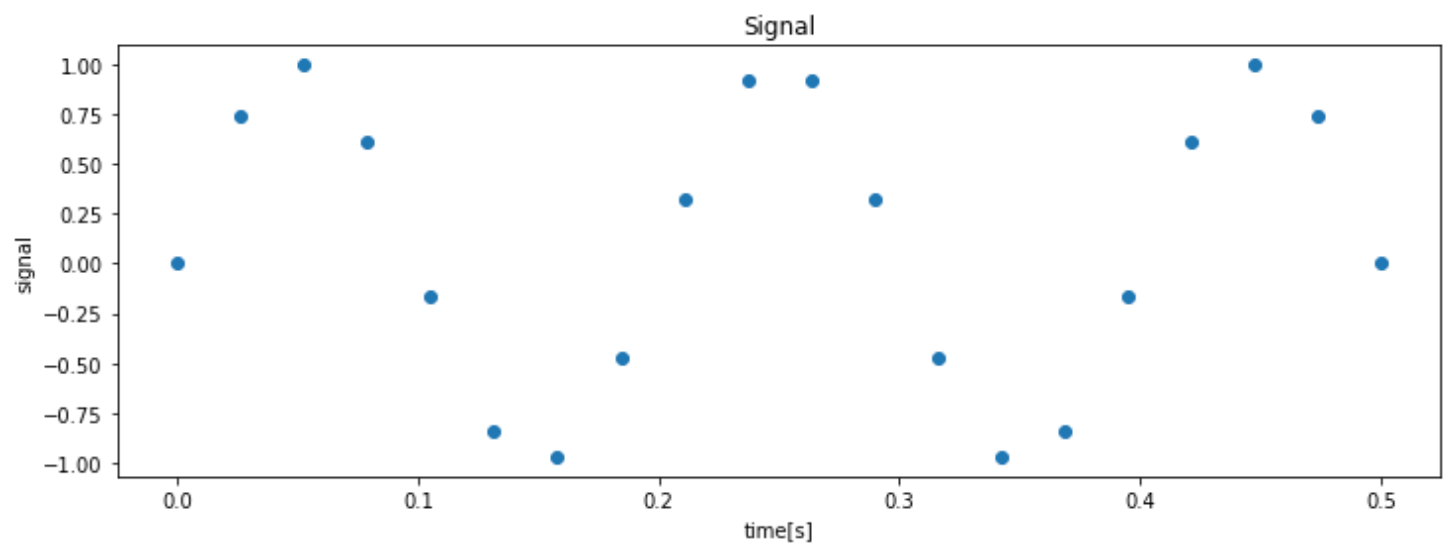
可以看到，采样后的离散信号 $x_s(t)$ 在频域的谱线 $X_s(\omega)$ 就相当于把原信号的频谱 $X(\omega)$ 不断地以采样角频率 $\omega_s = \frac{2\pi}{T_s}$ 的间隔进行复制平移。效果如图。



DFT与FFT

离散傅里叶变换

假设在 T_0 时间内有一串采样数为 N 、采样间隔为 $T_s = T_0/N$ 的信号输入，如下图所示：



在该图中, $T_0 = 0.5s$ 、 $N = 20$ 、 $T_s = 0.025s$ 、 $f = 1/T = 5Hz$ 、 $f_s = 40Hz$ 。我们所有的其实只有两组无量纲的**有序数组**

```
x = array([0.          , 0.02631579, 0.05263158, 0.07894737, 0.10526316,
          0.13157895, 0.15789474, 0.18421053, 0.21052632, 0.23684211,
          0.26315789, 0.28947368, 0.31578947, 0.34210526, 0.36842105,
          0.39473684, 0.42105263, 0.44736842, 0.47368421, 0.5])
signal = array([ 0.00000000e+00,  7.35723911e-01,  9.96584493e-01,  6.14212713e-01,
        -1.64594590e-01, -8.37166478e-01, -9.69400266e-01, -4.75947393e-01,
         3.24699469e-01,  9.15773327e-01,  9.15773327e-01,  3.24699469e-01,
        -4.75947393e-01, -9.69400266e-01, -8.37166478e-01, -1.64594590e-01,
         6.14212713e-01,  9.96584493e-01,  7.35723911e-01,  6.12323400e-16])
```

所以我们如何计算signal这个有序数组的傅里叶变换? 如何得到原信号的频率 f ?

梳理一下思路:

连续的原信号及其傅里叶变换, 我们无法得到:

$$x(t) \implies X(\omega) = \int_{-\infty}^{+\infty} x(t') e^{-i\omega t'} dt'$$

离散的signal, 输入计算机的有序数列:

$$x_s(t) = \sum_{n=0}^{N-1} \delta(t - nT_s) x(t)$$

我们所说的傅里叶变换, 并不是想得到 $X(\omega)$, 我们只用得到 $x_s(t)$ 的傅里叶变换即可, 这很简单:

$$\begin{aligned} X_s(\omega) &= \int_{-\infty}^{+\infty} x_s(t) e^{-j\omega t} dt \\ &= \int_{-\infty}^{+\infty} \sum_{n=0}^{N-1} \delta(t - nT_s) x(t) e^{-j\omega t} dt \\ &= \sum_{n=0}^{N-1} \int_{-\infty}^{+\infty} \delta(t - nT_s) x(t) e^{-j\omega t} dt \\ &= \sum_{n=0}^{N-1} x(nT_s) e^{-j\omega nT_s} \end{aligned}$$

此即离散傅里叶变换的公式。其中, N 为采样数, T_s 为采样间隔, $x(nT_s)$ 就是signal有序数列中的第 n 个数值 x_n 。 ω 可以是任意值, 可以视为DFT输出数组的索引。 T_s 是采样间隔, 起到**将索引 n 对应到时域的作用**, 可以视为量纲, 比如 $T_s = 0.025s$, 那么 T_s 就将索引对应到时间 s , 如果 $T_s = 0.025nm$, 那么DFT就是计算空间中某一模式的频率谱, T_s 将索引对应到空间间隔 nm

$$\begin{aligned} t &= x_l, \quad l = 0, 1, 2, \dots, N-1 \\ x &= x_l, \quad l = 0, 1, 2, \dots, N-1 \\ X &= \sum_{n=0}^{N-1} x_n e^{i \frac{2\pi}{N}} \end{aligned}$$

```

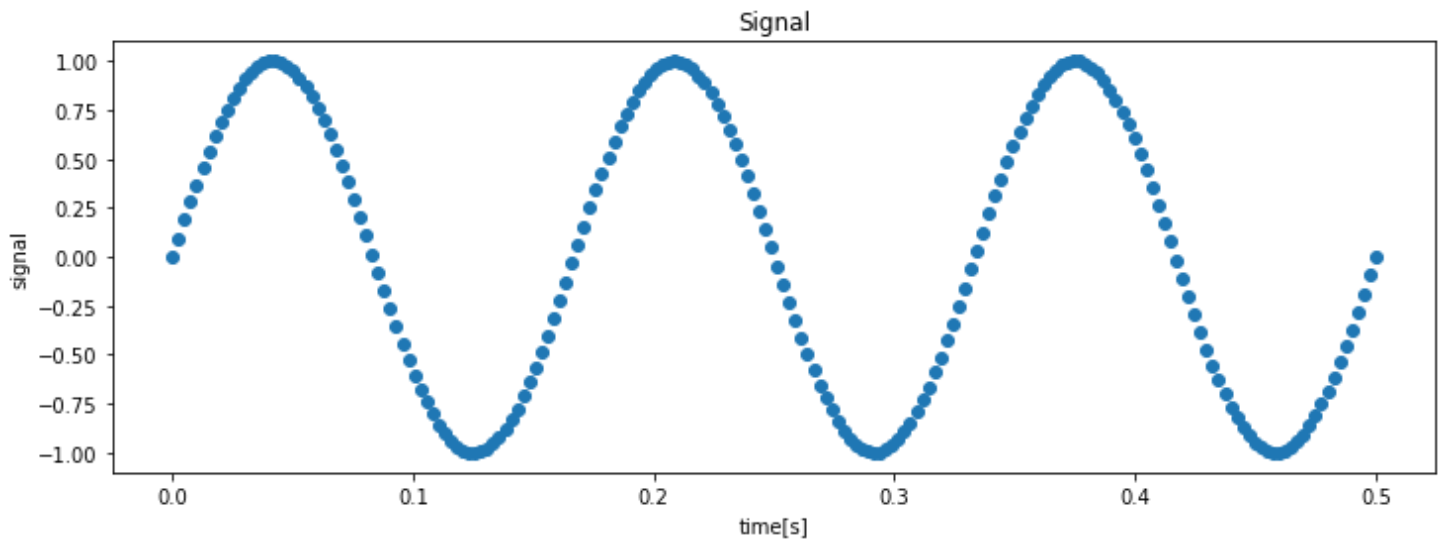
# 采样信号示意图
duration = 0.5 # 采样时间 T_0
sampling_rate = 400 # 采样率 N
T_s = 1/sampling_rate # 采样间隔 T_s
x = np.linspace(0, duration, int(sampling_rate*duration)) # 采样点时间 nT_s

index = np.linspace(0, int(sampling_rate*duration), int(sampling_rate*duration)) # 提取fft信号的索引 n
frequency = index*sampling_rate/(sampling_rate*duration) # 由索引、采样频率、采样数计算fft基信号的频率 f=n/(Nf_s)

signal = np.sin(2*np.pi*6*x) # 6Hz采样信号有序数组

plt.figure(figsize=(12,4))
plt.scatter(x, signal)
plt.title("Signal")
plt.xlabel("time[s]")
plt.ylabel("signal")

```



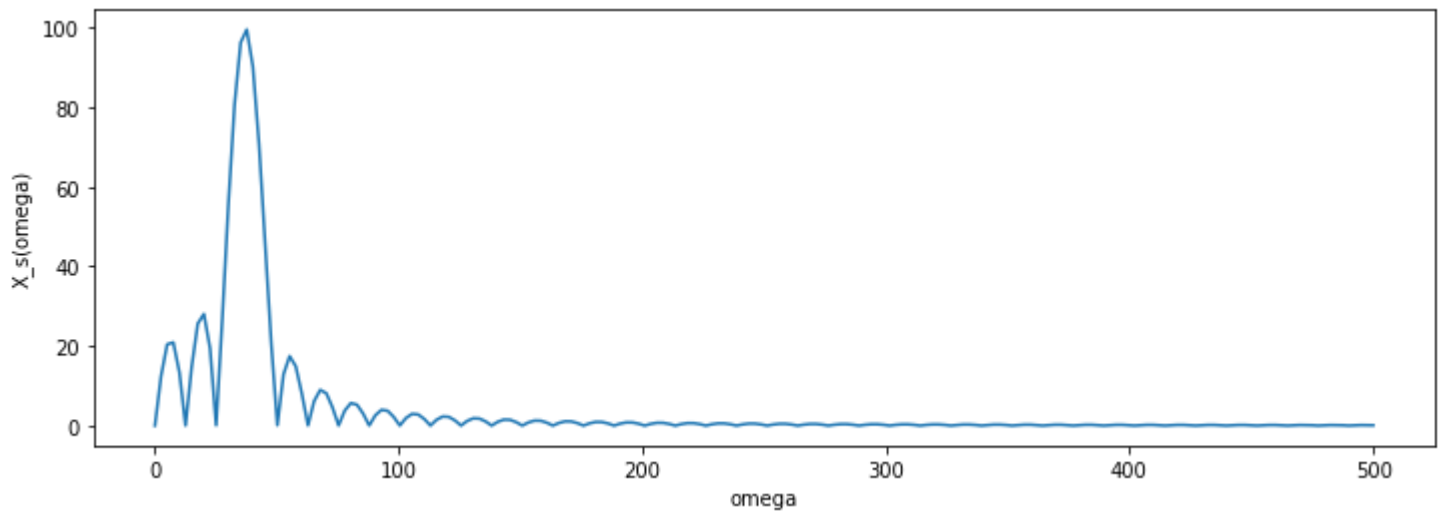
```

def DFT(x, signal, omega):
    result = 0
    for i in range(len(x)):
        result += signal[i]*np.e**(-1j*omega*x[i])
    return result

omega = np.linspace(0, 500, 200)
signal_fft = DFT(x, signal, omega)
plt.figure(figsize=(12,4))
plt.plot(omega, np.abs(signal_fft))

plt.xlabel('omega')
plt.ylabel('X_s(omega)')

```



峰值处并不是我们信号的频率6，所以横轴和纵轴是什么？

```
omega[np.where(abs(signal_fft)==max(abs(signal_fft)))]
```

```
array([37.68844221])
```

$$\omega = 37.688 = 2\pi \times 6!$$

此外值得一提的是，DFT在 $f = 6\text{Hz}$ 周围频率也出现了振荡，这完全是由于采样函数的周期不完整导致的。即使把采样数不断增大，该震荡也不会衰减，相反如果我们增大采样时间，让信号中存在更多的周期，DFT就会越发趋近于 δ 函数

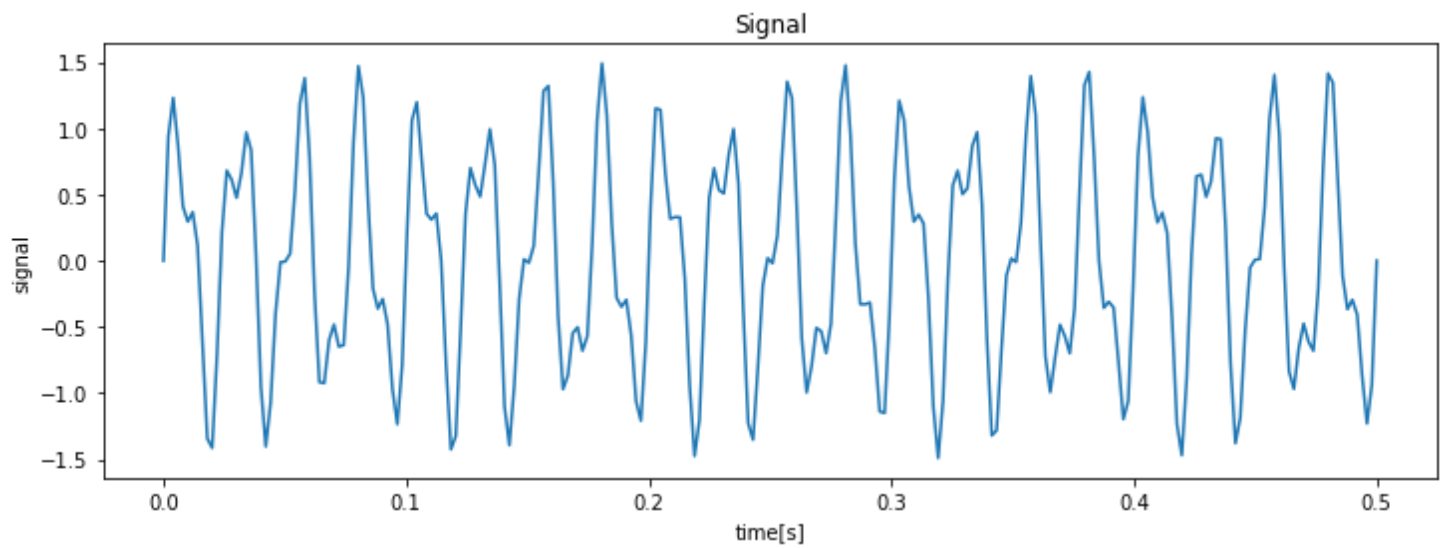
快速傅里叶变换

实例1.理解FFT:

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

# 原始信号生成
duration = 0.5 # 采样时间
sampling_rate = 500 # 采样率
T_s = 1/sampling_rate # 采样周期
x = np.linspace(0, duration, int(sampling_rate*duration)) # 采样点n
signal = np.sin(2*np.pi*40*x) + 0.5*np.sin(2*np.pi*90*x) # 40Hz和90Hz正弦信号叠加产生采样信号

plt.figure(figsize=(12,4)) #看看原始信号长啥样
plt.plot(x, signal)
plt.title("Signal")
plt.xlabel("time[s]")
plt.ylabel("signal")
```



```
# 对采样信号做FFT
signal_fft = np.fft.fft(signal)

# 输出FFT结果索引在(0,46)与(204,250)的数值
# 很明显都应该是复数，并且索引为i与索引为len(signal_fft)-i-1的结果应该复共轭
for i in range(0, 46):
    print("fft at index[" + i + "]: ", signal_fft[i],
          "fft at index[" + int(sampling_rate*duration)-i-1,
          "]: ", signal_fft[int(sampling_rate*duration)-i-1])
```



```

fft at index[ 0 ]: (-2.5757174171303632e-14+0j) fft at index[ 249 ]: (0.0007378440493890914+0.058712673131002424j)
fft at index[ 1 ]: (0.0007378440493880367-0.05871267313100256j) fft at index[ 248 ]: (0.002970571767752117+0.11817040791237154j)
fft at index[ 2 ]: (0.0029705717677523946-0.11817040791237177j) fft at index[ 247 ]: (0.0067571855805809156+0.17915499307596355j)
fft at index[ 3 ]: (0.006757185580578584-0.1791549930759646j) fft at index[ 246 ]: (0.01220094537758315+0.24252563266022126j)
fft at index[ 4 ]: (0.012200945377581984-0.2425256326602212j) fft at index[ 245 ]: (0.0194575128005659+0.3092683097589256j)
fft at index[ 5 ]: (0.019457512800569177-0.3092683097589273j) fft at index[ 244 ]: (0.028748065741912243+0.38056027225628464j)
fft at index[ 6 ]: (0.028748065741911688-0.38056027225628375j) fft at index[ 243 ]: (0.04037961498585274+0.45785934843090936j)
fft at index[ 7 ]: (0.04037961498585485-0.45785934843091014j) fft at index[ 242 ]: (0.05477636095257904+0.5430337267254692j)
fft at index[ 8 ]: (0.054776360952579095-0.5430337267254695j) fft at index[ 241 ]: (0.07252879511594501+0.6385587421251379j)
fft at index[ 9 ]: (0.07252879511594523-0.638558742125138j) fft at index[ 240 ]: (0.094472644112789+0.7478279817005156j)
fft at index[ 10 ]: (0.09447264411278895-0.7478279817005153j) fft at index[ 239 ]: (0.12182046887937473+0.8756674703694303j)
fft at index[ 11 ]: (0.12182046887937584-0.8756674703694309j) fft at index[ 238 ]: (0.15639135943100502+1.0292293827460266j)
fft at index[ 12 ]: (0.15639135943100513-1.0292293827460264j) fft at index[ 237 ]: (0.20103554104314386+1.2196409252203415j)
fft at index[ 13 ]: (0.20103554104314486-1.2196409252203404j) fft at index[ 236 ]: (0.26047811252905584+1.465278124054619j)
fft at index[ 14 ]: (0.2604781125290564-1.4652781240546187j) fft at index[ 235 ]: (0.34315923065485965+1.7989036846465671j)
fft at index[ 15 ]: (0.34315923065486287-1.7989036846465651j) fft at index[ 234 ]: (0.4657799525618445+2.2852981039883735j)
fft at index[ 16 ]: (0.4657799525618447-2.2852981039883744j) fft at index[ 233 ]: (0.6667019726967065+3.0732298840168855j)
fft at index[ 17 ]: (0.6667019726967064-3.0732298840168855j) fft at index[ 232 ]: (1.0579928313000497+4.597311040247976j)
fft at index[ 18 ]: (1.0579928313000502-4.597311040247976j) fft at index[ 231 ]: (2.166520671367053+8.900926688093772j)
fft at index[ 19 ]: (2.1665206713670515-8.900926688093772j) fft at index[ 230 ]: (30.754744295539357+119.78182060025081j)
fft at index[ 20 ]: (30.754744295539354-119.78182060025081j) fft at index[ 229 ]: (-2.8137723655564675-10.413849452877608j)
fft at index[ 21 ]: (-2.8137723655564675+10.413849452877608j) fft at index[ 228 ]: (-1.4014726174984466-4.939534267996505j)
fft at index[ 22 ]: (-1.4014726174984469+4.939534267996505j) fft at index[ 227 ]: (-0.9504532403016216-3.196380947646645j)
fft at index[ 23 ]: (-0.9504532403016221+3.1963809476466447j) fft at index[ 226 ]: (-0.7240688591126921-2.3275787782007535j)
fft at index[ 24 ]: (-0.7240688591126926+2.3275787782007535j) fft at index[ 225 ]: (-0.5844838814788077-1.7988564197715107j)
fft at index[ 25 ]: (-0.5844838814788074+1.7988564197715111j) fft at index[ 224 ]: (-0.48686445846704607-1.436727478865722j)
fft at index[ 26 ]: (-0.48686445846704574+1.436727478865722j) fft at index[ 223 ]: (-0.4121446219873226-1.1677453573532293j)
fft at index[ 27 ]: (-0.41214462198732293+1.1677453573532297j) fft at index[ 222 ]: (-0.3507192912564261-0.9552845890530399j)
fft at index[ 28 ]: (-0.3507192912564267+0.9552845890530396j) fft at index[ 221 ]: (-0.29709384410942835-0.7788275132110704j)
fft at index[ 29 ]: (-0.2970938441094283+0.7788275132110707j) fft at index[ 220 ]: (-0.24775329505697574-0.6257533934243871j)
fft at index[ 30 ]: (-0.24775329505697385+0.6257533934243895j) fft at index[ 219 ]: (-0.20017969558412507-0.48760365127917826j)
fft at index[ 31 ]: (-0.20017969558412496+0.4876036512791777j) fft at index[ 218 ]: (-0.15233244661159218-0.35817731400291203j)
fft at index[ 32 ]: (-0.15233244661159218+0.35817731400291186j) fft at index[ 217 ]: (-0.10232389981470963-0.23243869482422164j)
fft at index[ 33 ]: (-0.10232389981470941+0.23243869482422075j) fft at index[ 216 ]: (-0.04816526959146844-0.1057868789707071j)
fft at index[ 34 ]: (-0.04816526959146855+0.10578687897070715j) fft at index[ 215 ]: (0.012494733938757996+0.026552661214636675j)
fft at index[ 35 ]: (0.01249473393875572-0.02655266121463562j) fft at index[ 214 ]: (0.08271357312477606+0.17019066279430417j)
fft at index[ 36 ]: (0.08271357312477601-0.17019066279430384j) fft at index[ 213 ]: (0.16679237527045732+0.3324966878764439j)
fft at index[ 37 ]: (0.16679237527045757-0.33249668787644227j) fft at index[ 212 ]: (0.271222115137411+0.524135774470346j)
fft at index[ 38 ]: (0.2712221151374113-0.5241357744703461j) fft at index[ 211 ]: (0.4065187303377439+0.7619819692359101j)
fft at index[ 39 ]: (0.40651873033774394-0.7619819692359113j) fft at index[ 210 ]: (0.5911606691844523+1.0753172653046832j)
fft at index[ 40 ]: (0.5911606691844522-1.0753172653046832j) fft at index[ 209 ]: (0.8611585763428036+1.5208537057698406j)
fft at index[ 41 ]: (0.8611585763428049-1.5208537057698406j) fft at index[ 208 ]: (1.2976600102680287+2.226027051144519j)
fft at index[ 42 ]: (1.2976600102680282-2.226027051144519j) fft at index[ 207 ]: (2.130488170882151+3.5513413671909753j)
fft at index[ 43 ]: (2.130488170882148-3.5513413671909753j) fft at index[ 206 ]: (4.36780408718508+7.077602313022698j)
fft at index[ 44 ]: (4.36780408718508-7.077602313022698j) fft at index[ 205 ]: (31.330448621962667+49.368887367917246j)
fft at index[ 45 ]: (31.33044862196267-49.36888736791725j) fft at index[ 204 ]: (-7.521085690733225-11.528523666465613j)

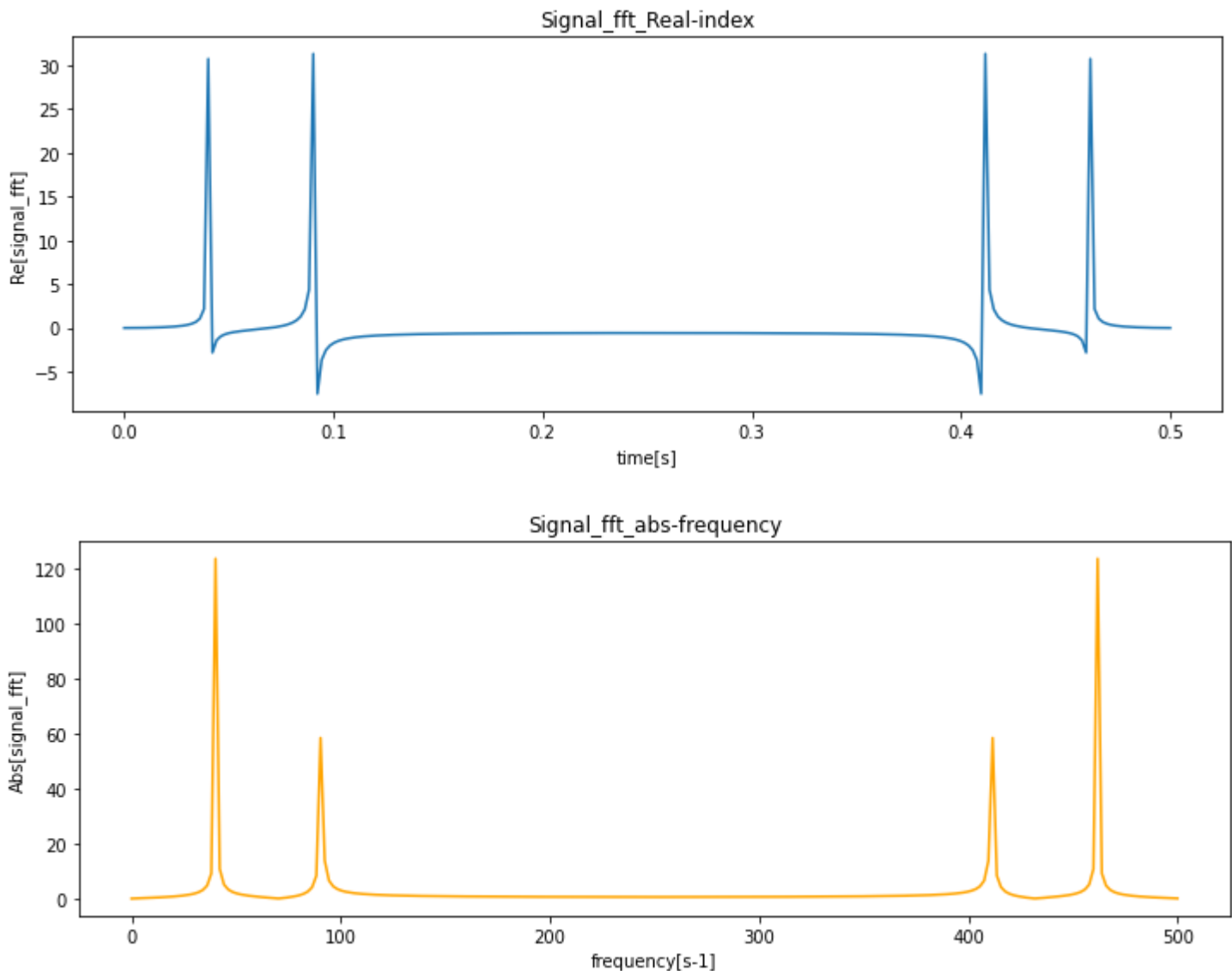
```

由上述数据可以看出，index = n 与 index = length-n 的fft数值是复共轭，即fft的N个值中，X[n]只有[0, 1, 2, ..., N/2-1]这些项的频率就够了，后续的点在频率信息上是冗余的。

$f = \frac{nf_s}{N}$ 其中 f 为基信号频率， n 为FFT后信号索引， f_s 为采样频率， N 为采样数。由该式可以将FFT结果的索引转为频率。

```
# 简单的画出fft
plt.figure(figsize=(12,4))
plt.plot(x, signal_fft) #这里画出来的横坐标“time[s]”实际上不是时间，而是按照频率分布，只是把时间的有序索引强行加给了频率分布索引
plt.title("Signal_fft_Real-index")
plt.xlabel("time[s]")
plt.ylabel("Re[signal_fft]")

# 画出想要的fft
signal_fft_abs = np.abs(signal_fft) # fft取模
index = np.linspace(0, int(sampling_rate*duration), int(sampling_rate*duration)) # 提取fft信号的索引
frequency = index*sampling_rate/(sampling_rate*duration) # 由索引、采样频率、采样数计算fft基信号的频率
plt.figure(figsize=(12,4))
plt.plot(frequency, signal_fft_abs,c= 'orange')
plt.title("Signal_fft_abs-frequency")
plt.xlabel("frequency[s-1]")
plt.ylabel("Abs[signal_fft]")
```



实例2.采样时间duration对采样信号及其FFT的影响:

```

# 使用plotly实现交互式图片
# Create figure
fig1 = go.Figure()
fig2 = go.Figure()

#参数设置
#duration = 0.5 # 采样时间
frequency_signal = 40 # 信号频率
sampling_rate = 500 # 采样率
duration_mini = 0.25 # 采样时间范围
duration_max = 0.75
pace = 1/(40*4) # 采样时间Slider步长

frequency_signal2 = 90 # 两个频率的波函数叠加

# 对duration在设定的范围内按照步长生成多张图片
for duration in np.arange(duration_mini, duration_max, pace):
    # 信号生成
    x = np.linspace(0, duration, int(sampling_rate*duration)) # 时间-索引
    index = np.linspace(0, int(sampling_rate*duration), int(sampling_rate*duration)) # 索引
    frequency = index*sampling_rate/(sampling_rate*duration) # FFT频率-索引
    signal = np.sin(frequency_signal * np.pi * 2 * x) #+ 0.5*np.sin(frequency_signal2 * np.pi * 2 * x) # 信号生成
    signal_fft = np.fft.fft(signal) # 信号FFT

    # 在图片中生成轨迹
    fig1.add_trace(
        go.Scatter(
            visible=False,
            line=dict(color="#00CED1", width=2),
            name="T_0 = " + str(duration) ,
            x = frequency, # 采样点n
            y=np.abs(signal_fft))
    )
    fig2.add_trace(
        go.Scatter(
            visible=False,
            line=dict(color="#00CED1", width=2),
            name="T_0 = " + str(duration),
            x = x, # 采样点n
            y=signal))
    )

# Make 10th trace visible
#fig.data[10].visible = True

# 创建滑条
steps = []
for i in range(len(fig1.data)):
    sampling_rate = dict( #这里sampling_rate应该写duration,不过发现这里写啥都无所谓
        method="update",
        args=[{"visible": [False] * len(fig1.data)},
              {"title": "Duration: " + str(duration_mini+pace*i) + "s" + " " + str((duration_mini+pace*i)*40) + "T"}], # layout attribute
    )
    sampling_rate["args"][0]["visible"][i] = True # Toggle i'th trace to "visible"
    steps.append(sampling_rate)

sliders = [dict(
    active=10,
    currentvalue={"prefix": "Slider: "},
    pad={"t": 50},
    steps=steps
)]

# 下面都是对图片的标签等的定义
fig1.update_layout(
    sliders=sliders
)
fig2.update_layout(
    sliders=sliders
)

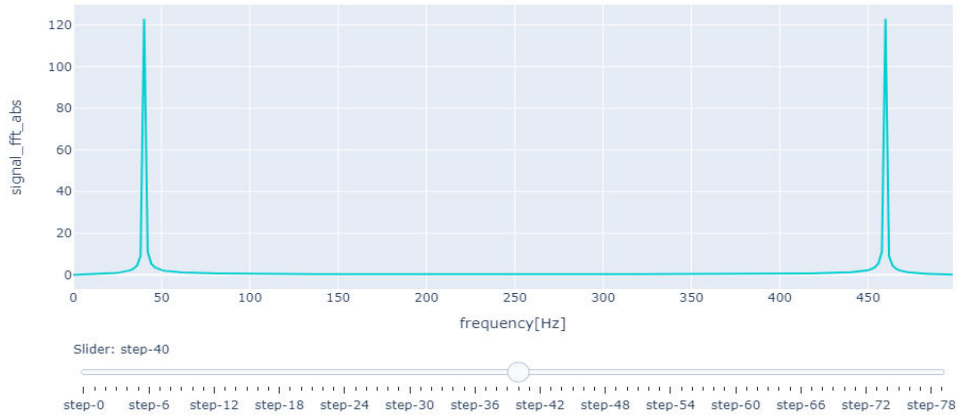
```

)

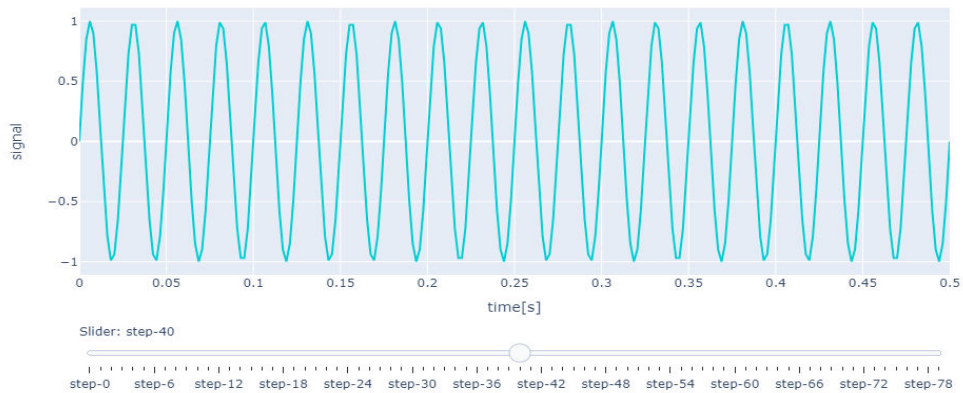
```
fig1.update_layout(title='Abs[signal_fft]', xaxis_title='frequency[Hz]', yaxis_title='signal_fft_abs')
fig2.update_layout(title='signal_fft', xaxis_title='time[s]', yaxis_title='signal')

fig1.show()
fig2.show()
```

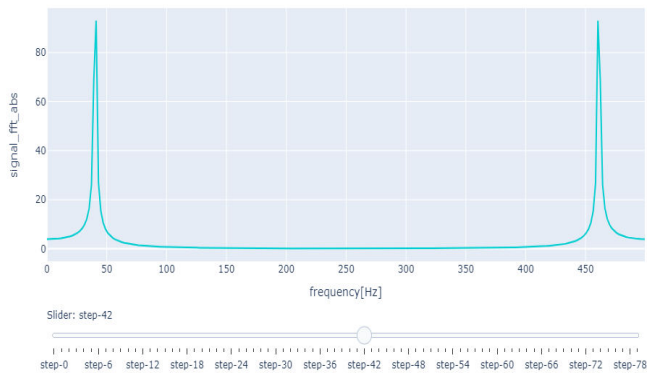
Duration: 0.5s 20.0T



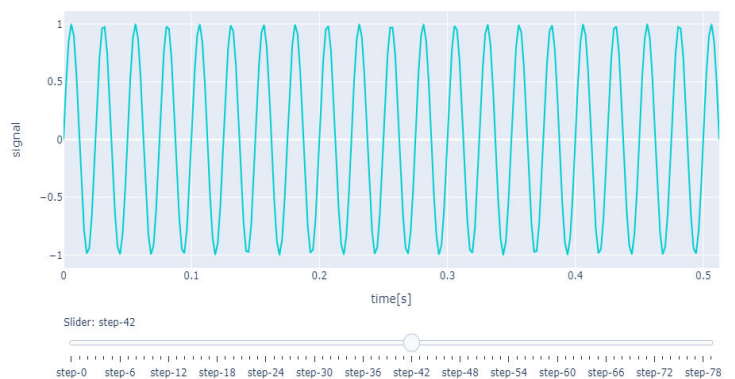
Duration: 0.5s 20.0T



Duration: 0.5125s 20.5T



Duration: 0.5125s 20.5T



可以观察到，在采样时间为整数倍的信号周期时，信号失真最小。在采样时间比整数倍的信号周期多半个周期时，信号失真最大。

实例3.采样频率对采样信号及其FFT的影响：

```

# Create figure
fig1 = go.Figure()
fig2 = go.Figure()
# Add traces, one for each slider step

#duration = 0.5 # 采样时间
frequency_signal = 40 # 信号频率
duration = 0.5 # 采样时间范围
sampling_rate_mini = 20 # 采样频率范围
sampling_rate_max = 500
pace = 10 # 采样时间Slider步长

frequency_signal2 = 90

for sampling_rate in np.arange(sampling_rate_mini, sampling_rate_max, pace):
    x = np.linspace(0, duration, int(sampling_rate*duration)) # 时间-索引
    index = np.linspace(0, int(sampling_rate*duration), int(sampling_rate*duration)) # 索引
    frequency = index*sampling_rate/(sampling_rate*duration) # FFT频率-索引
    signal = np.sin(frequency_signal * np.pi * 2 * x) + 0.5*np.sin(frequency_signal2 * np.pi * 2 * x) # 信号生成
    signal_fft = np.fft.fft(signal) # 信号FFT
    fig1.add_trace(
        go.Scatter(
            visible=False,
            line=dict(color="#00CED1", width=2),
            name="T_0 = " + str(duration) ,
            x = frequency, # 采样点n
            y=np.abs(signal_fft))
    fig2.add_trace(
        go.Scatter(
            visible=False,
            line=dict(color="#00CED1", width=2),
            name="T_0 = " + str(duration),
            x = x, # 采样点n
            y=signal))

# Make 10th trace visible
#fig.data[10].visible = True

# Create and add slider
steps = []
for i in range(len(fig1.data)):
    sampling_rate = dict(
        method="update",
        args=[{"visible": [False] * len(fig1.data)},
            {"title": "Samplig_rate: " + str(sampling_rate_mini+pace*i) + "Hz"+ " " + str((sampling_rate_mini+pace*i)/40) + "f"}], # layout attribute
    )
    sampling_rate["args"][0]["visible"][i] = True # Toggle i'th trace to "visible"
    steps.append(sampling_rate)

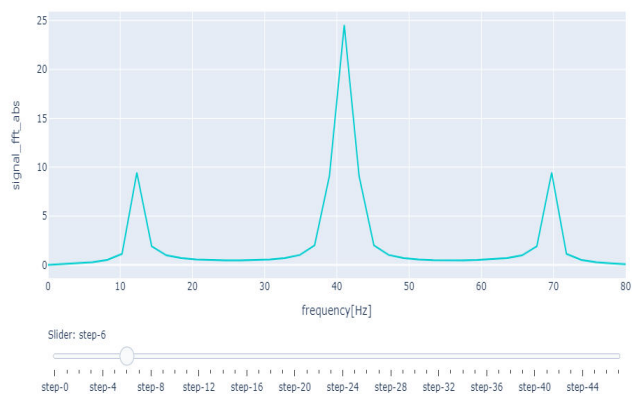
sliders = [dict(
    active=10,
    currentvalue={"prefix": "Slider: "},
    pad={"t": 50},
    steps=steps
)]

fig1.update_layout(
    sliders=sliders
)
fig2.update_layout(
    sliders=sliders
)
fig1.update_layout(title='Abs[signal_fft]', xaxis_title='frequency[Hz]', yaxis_title='signal_fft_abs')
fig2.update_layout(title='signal_fft', xaxis_title='time[s]', yaxis_title='signal')

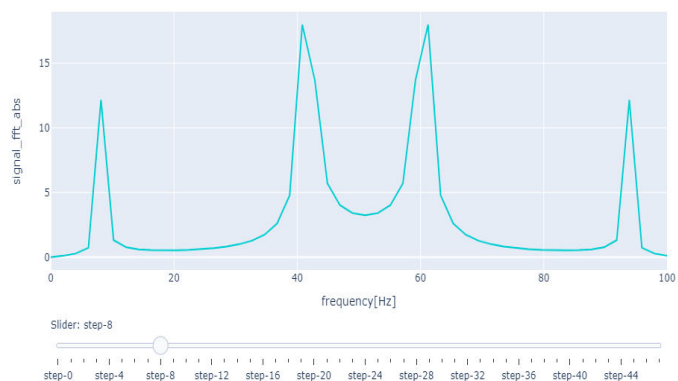
fig1.show()
fig2.show()

```

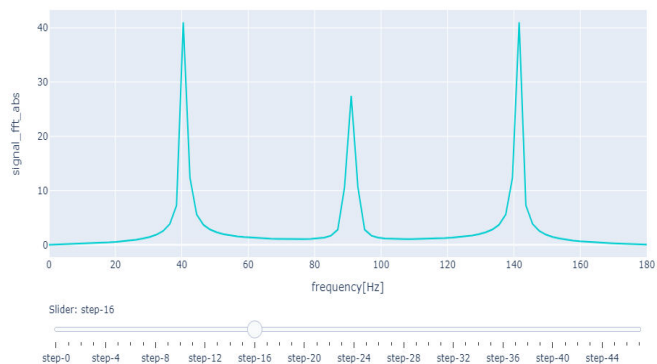
Samplig_rate: 80Hz 2.0f



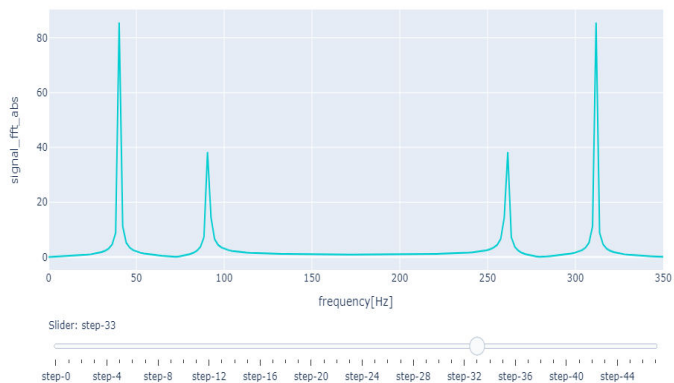
Samplig_rate: 100Hz 2.5f



Samplig_rate: 180Hz 4.5f



Samplig_rate: 350Hz 8.75f



可以看出，在采样频率至少大于两倍的信号频率时，才能测量出原信号的频率。对于有多个频率分量的信号，采样频率至少要大于两倍的信号中的最高频率。图中是40Hz和90Hz频率信号的混合。