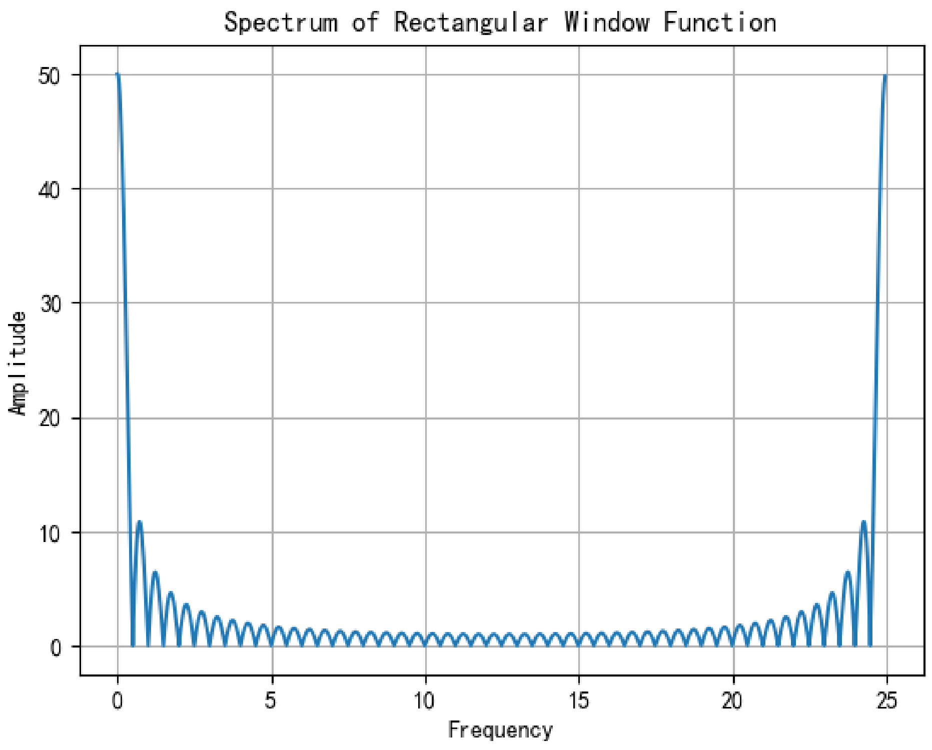
In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['KaiTi']
```

一、矩形窗函数的频谱

In [2]:
```python
# 定义矩形窗函数
def rect(t):
    return np.where(np.abs(t) <= 1, 1, 0)

# 创建时间序列
N=1000
t = np.linspace(-20, 20, N)
fs=1/(t[1]-t[0])

# 计算矩形窗函数的频域表示
spectrum =np.fft.fft(rect(t))

# 创建频率序列
freq = np.arange(0,N,1)*fs/N

# 绘制频谱
plt.figure()
plt.plot(freq, np.abs(spectrum))
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.title('Spectrum of Rectangular Window Function')
plt.grid(True)
plt.show()
```



从上图可以看出,基本还原出了矩形窗函数的傅里叶变换的形状,
所以我们的整体思路大致是没问题的.

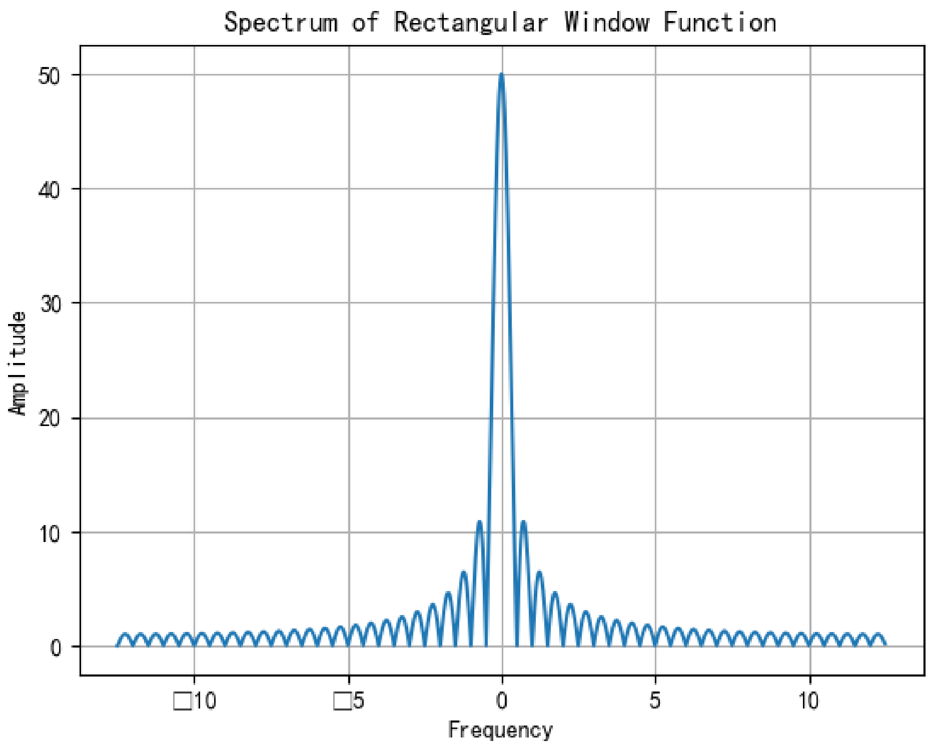
后话:
你可能还会看到fft包下面还有fftfreq和fftshift两个函数,fftfreq就是生成
上面讲的横坐标,fftshift是把$[\frac{f_s}{2}, f_s)$右边的图像挪到$[-\frac{f_s}{2}, 0)$上.
另外,在0点的峰值就是矩形窗的面积E乘以fs

In [3]:
```python
print("E*fs=",2*fs)
```

```
E*fs= 49.95000000000002
```

In [4]:
```python
N=1000
t = np.linspace(-20, 20, N)

# 计算矩形窗函数的频域表示
spectrum = np.fft.fftshift(np.fft.fft(rect(t)))

# 创建频率序列
freq = np.fft.fftshift(np.fft.fftfreq(N, t[1]-t[0]))

# 绘制频谱
plt.figure()
plt.plot(freq, np.abs(spectrum))
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.title('Spectrum of Rectangular Window Function')
plt.grid(True)
plt.show()
```

```
C:\ProgramData\anaconda3\envs\py310\lib\site-packages\IPython\core\pylabtools.py:152: UserWarning: Glyph 8722 (\N{MINUS SIGN}) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```



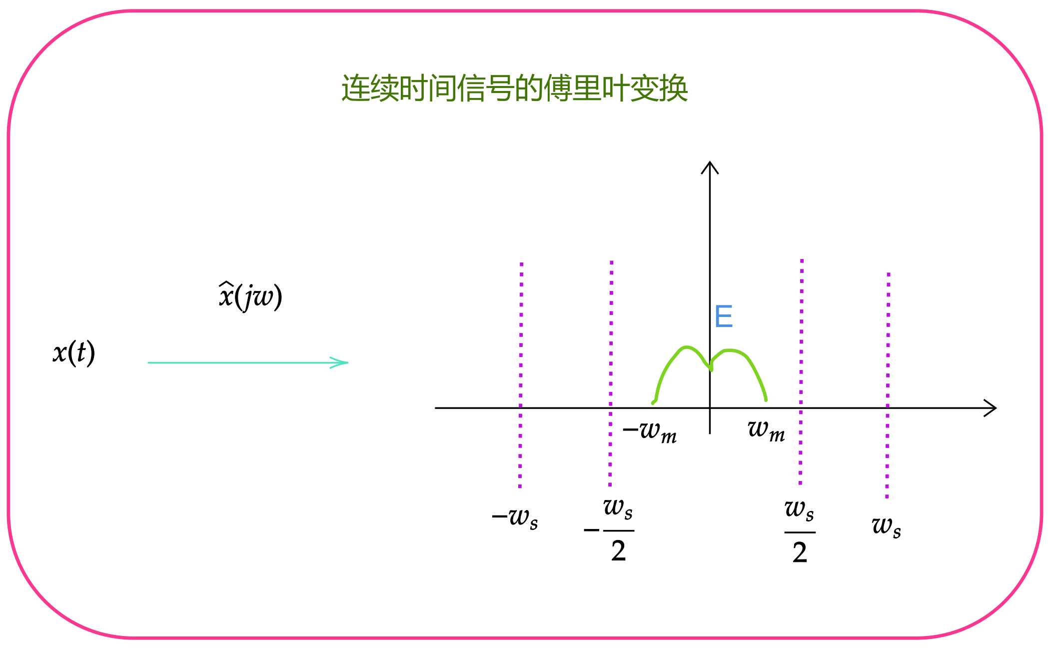
二、绘制$sinw_ct$的频谱

初始参数设定

In [5]:
```python
fc=50 # 信号频率
wc=2*np.pi*fc # 角频率

fs=400 # 采样频率
ws=2*np.pi*fs # 采样角频率

T0=2
T=1/fs
N=int(T0/T) # 采样点数

print(f"信号频率:fc={fc}Hz,wc={wc:.2f}rad/s")
print(f"采样频率:fs={fs}Hz,ws={ws:.2f}rad/s")
print(f"时间轴范围:[-1,1],总时间T0=2,采样时间间隔T:{T},采样点数N:{N}")
```

```
信号频率:fc=50Hz,wc=314.16rad/s
采样频率:fs=400Hz,ws=2513.27rad/s
时间轴范围:[-1,1],总时间T0=2,采样时间间隔T:0.0025,采样点数N:800
```

$$x(t)=\begin{cases} sinw_ct & 0\le t\le \frac{2*\pi}{w_c} \\ 0 & 其他 \end{cases}$$



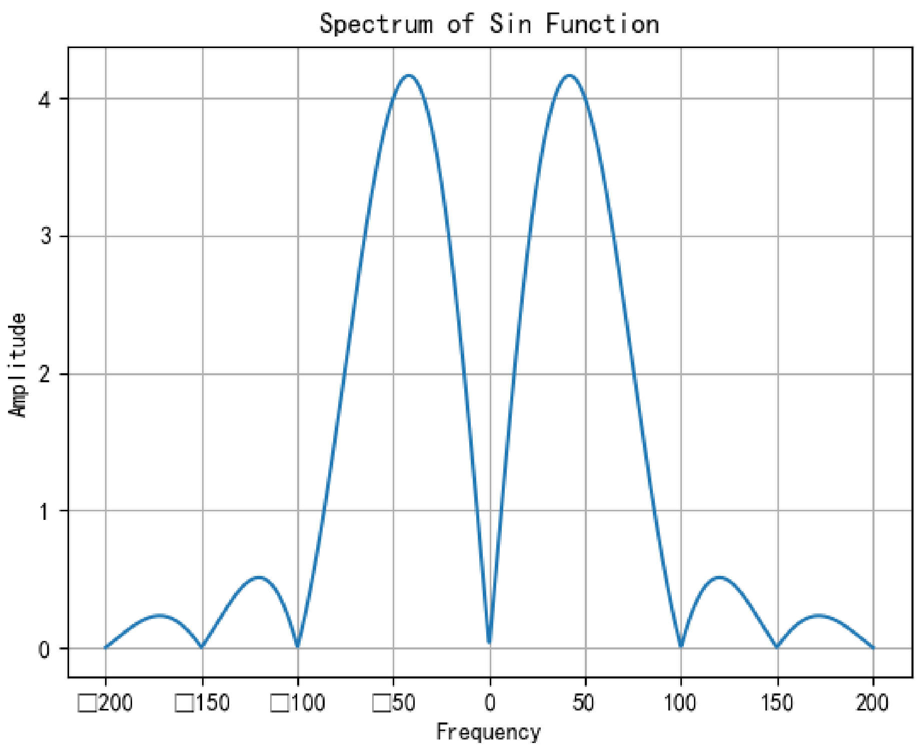
我们对$x(t)$手动进行傅里叶变换，直接求出$\left[-\frac{w_s}{2},\frac{w_s}{2}\right)$上的$\widehat{x}(jw)$表达式，

接着我们按照设定的采样频率，对$\widehat{x}(jw)$进行采样，我们预想这个采样的点和后面我们用fft计算出来的点是一样的，为了方便对比，我们把这些点画在$\left[-\frac{f_s}{2},\frac{f_s}{2}\right)$区间上。

手动计算$sinw_ct$的傅里叶变换：使用连续时间信号进行积分计算

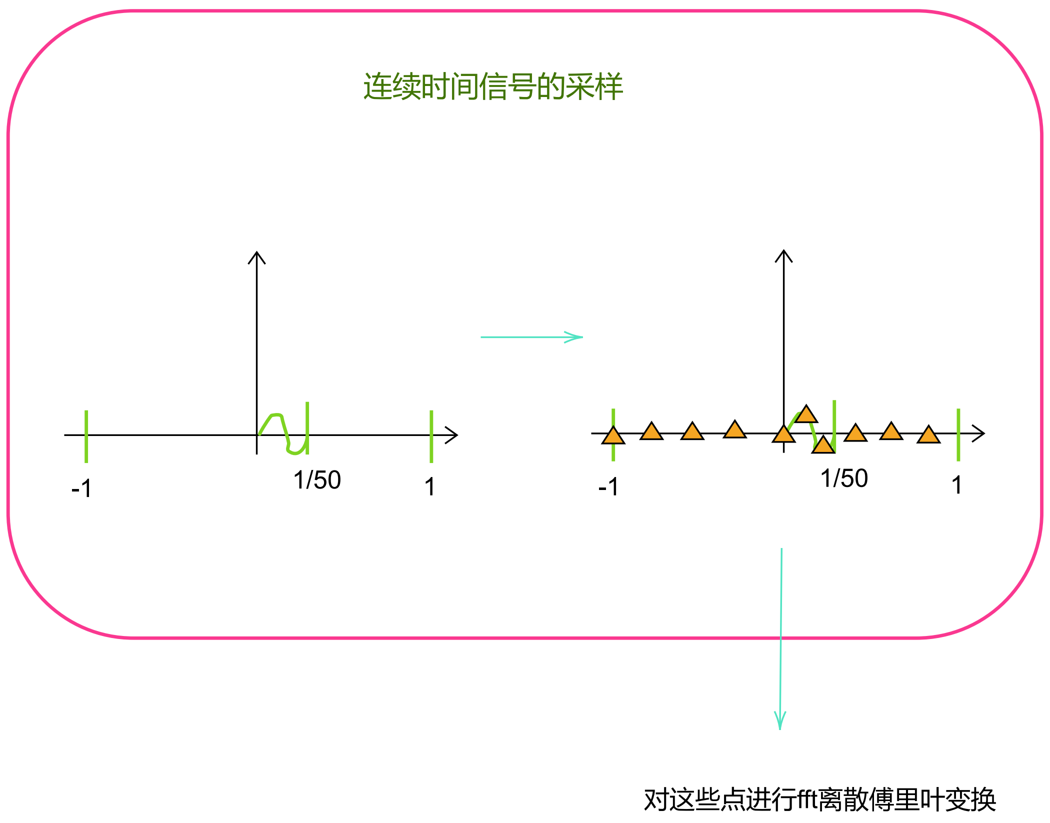
$$\int_{-1}^{1} sinw_cte^{-jwt}dt=\int_{0}^{1/50} sinw_cte^{-jwt}dt=\int_{0}^{1/50} sinw_ct[cos(wt)-jsin(wt)]dt$$
$$=\int_{0}^{1/50} sinw_ctcos(wt)dt-j\int_{0}^{1/50} sinw_ctsin(wt)dt$$
$$=\frac{1}{2}[-\frac{cos(w_c+w)t}{w_c+w}-\frac{cos(w_c-w)t}{w_c-w}]|_0^{1/50}+\frac{j}{2}[\frac{sin((w_c-w)t)}{w_c-w}-\frac{sin(w_c+w)t}{w_c+w}]|_0^{1/50}$$

In [6]:
```
def real_part(wc,w):
    t_1=-np.cos((wc+w)*(1/50))/(wc+w)
    t_2=-np.cos((wc-w)*(1/50))/(wc-w)
    upper=1/2*(t_1+t_2)
    t_3=-1/(wc+w)
    t_4=-1/(wc-w)
    lower=1/2*(t_3+t_4)
    return upper-lower
def imag_part(wc,w):
    t_1=np.sin((wc-w)*(1/50))/(wc-w)
    t_2=-np.sin((wc+w)*(1/50))/(wc+w)
    return 1/2*(t_1+t_2)
def spectrum(wc,w):
    real=real_part(wc,w)
    im=-imag_part(wc,w)
    return np.vectorize(complex)(real,im)

w=np.linspace(-ws/2,ws/2,N)
spect=spectrum(wc,w)*fs
freq=w/(2*np.pi)

plt.plot(freq,abs(spect))
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.title('Spectrum of Sin Function')
plt.grid(True)
plt.show()
```



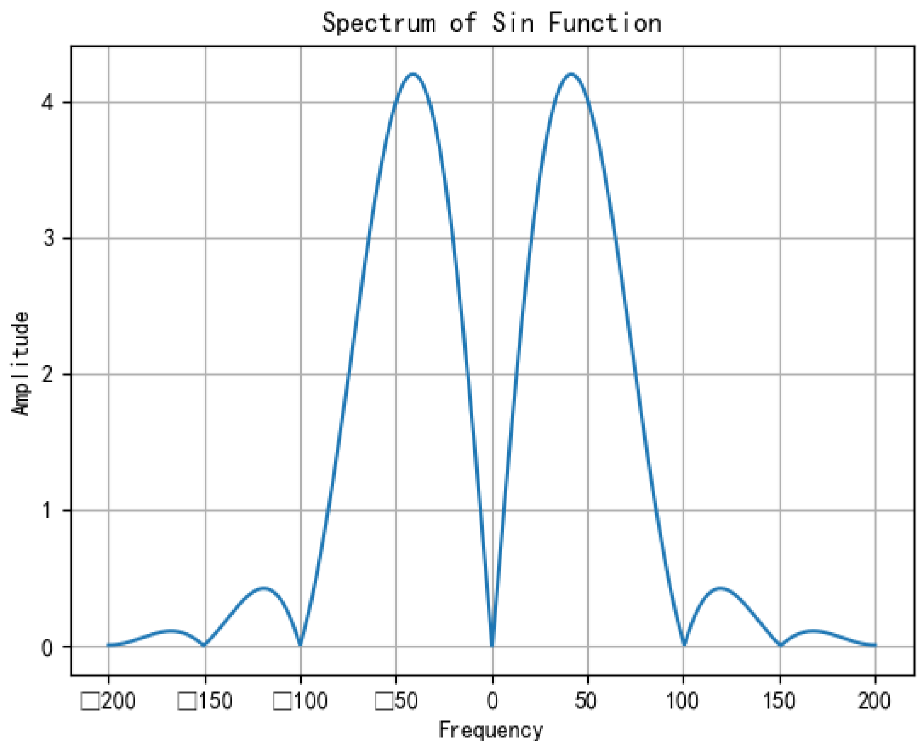
$$x(t) = \begin{cases} sinw_c t & 0 \le t \le \dfrac{2*\pi}{w_c} \\ 0 & 其他 \end{cases}$$

由于无法对 $(-\infty, \infty)$ 采样，所以缩小区间到$[-1, 1]$上，
注意采样的时候，一定要确保$[0, 1/50]$区间采到一定数量的点



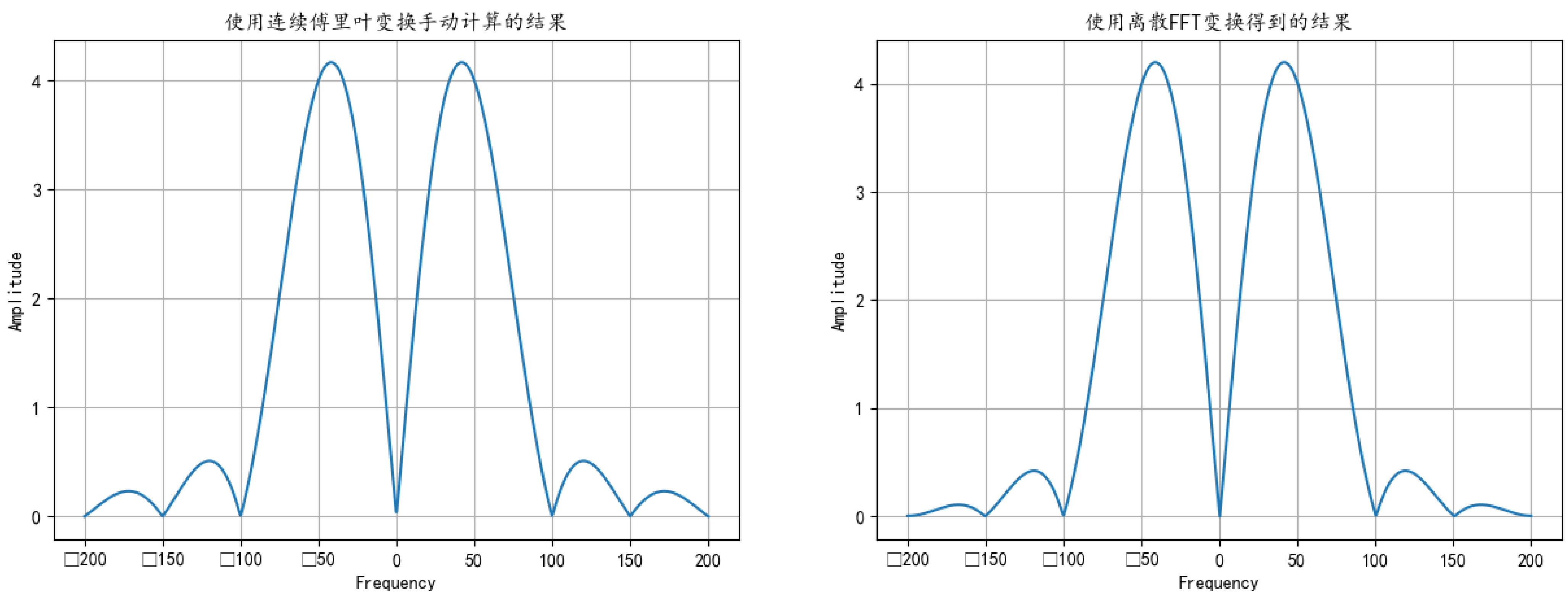
使用FFT包，对 $sinw_c t$ 采样计算

In [7]:
```
t=np.linspace(-1,1,N)

y=np.where((t >= 0) & (t <= 1/50), np.sin(wc*t), 0)

# 计算sinwct函数的频域表示
spect_2 = np.fft.fftshift(np.fft.fft(y))

# 绘制频谱
plt.figure()
plt.plot(freq,abs(spect_2))
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.title('Spectrum of Sin Function')
plt.grid(True)
plt.show()
```

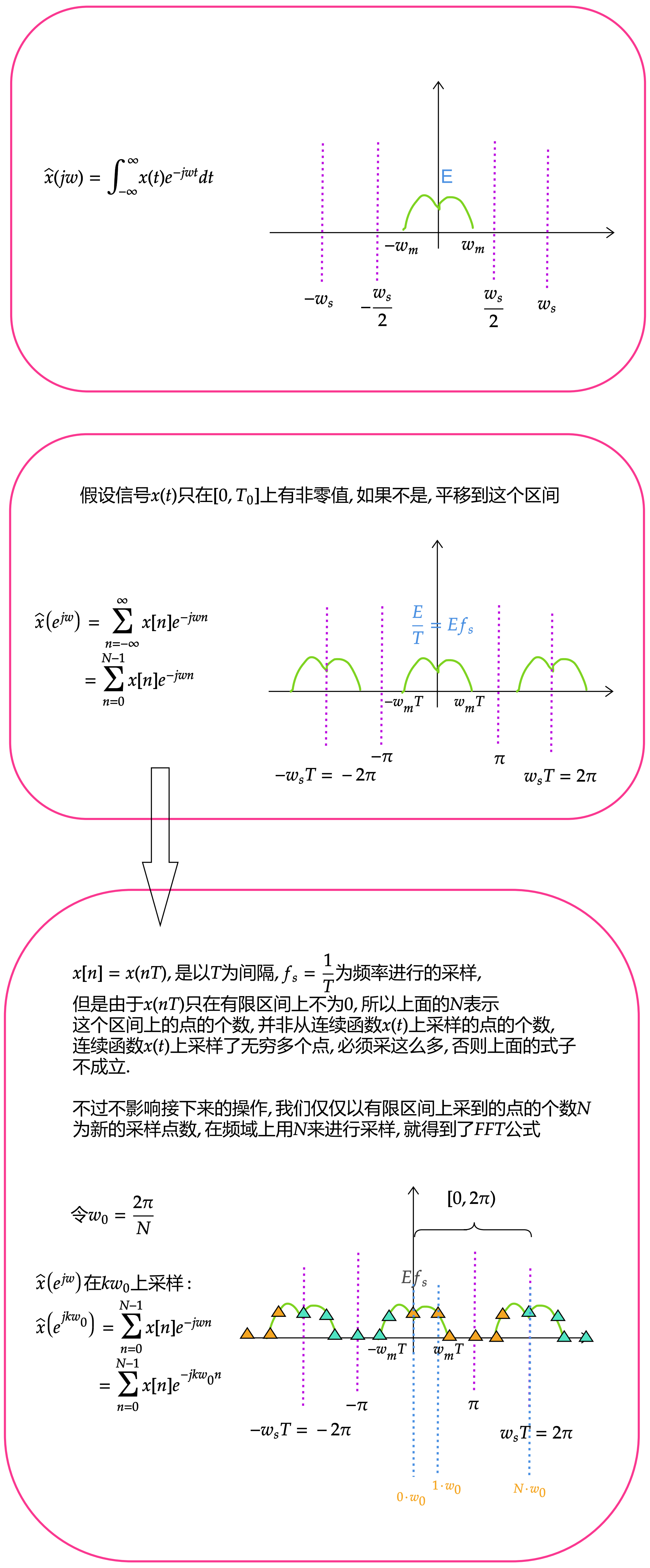


画在一个地方进行对比

In [8]:
```
plt.figure(figsize=(15,5))
plt.subplot(121)
plt.plot(freq,abs(spect))
plt.grid(True)
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.title('使用连续傅里叶变换手动计算的结果')

plt.subplot(122)
plt.plot(freq,abs(spect_2))
plt.grid(True)
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.title('使用离散FFT变换得到的结果')

plt.show()
```



然而真的需要两边多采那么多0吗？

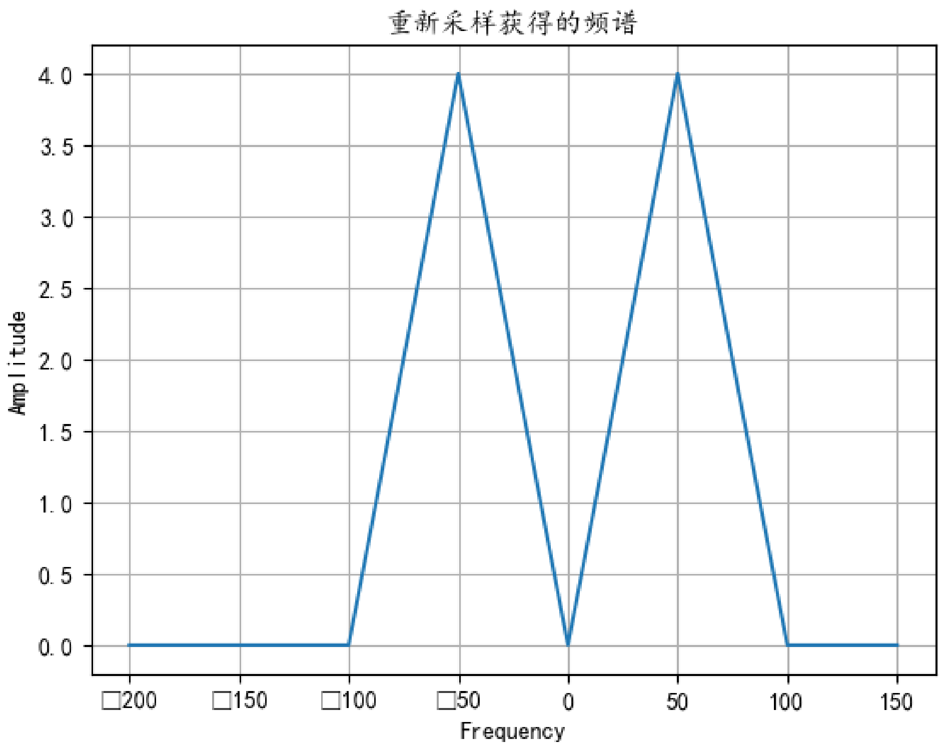


根据上面的推导,我们只需要得到给定采样频率下,$[0, \frac{1}{50})$上的点即可,数量由采样频率决定

In [9]:
```python
fc=50 # 信号频率
wc=2*np.pi*fc # 角频率

fs=400 # 采样频率
ws=2*np.pi*fs # 采样角频率

T0=2
T=1/fs

print(f"信号频率:fc={fc}Hz,wc={wc:.2f}rad/s")
print(f"采样频率:fs={fs}Hz,ws={ws:.2f}rad/s")
print(f"采样时间间隔T:{T}")
```

```
信号频率:fc=50Hz,wc=314.16rad/s
采样频率:fs=400Hz,ws=2513.27rad/s
采样时间间隔T:0.0025
```

In [10]:
```python
t3=np.arange(0,1/50,T)
y3=np.where((t3 >= 0) & (t3 <= 1/50), np.sin(wc*t3), 0)

spect_3 = np.fft.fftshift(np.fft.fft(y3))
freq_3 = np.fft.fftshift(np.fft.fftfreq(len(t3), T))

# 绘制频谱
plt.figure()
plt.plot(freq_3,abs(spect_3))
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.title('重新采样获得的频谱')
plt.grid(True)
plt.show()
```

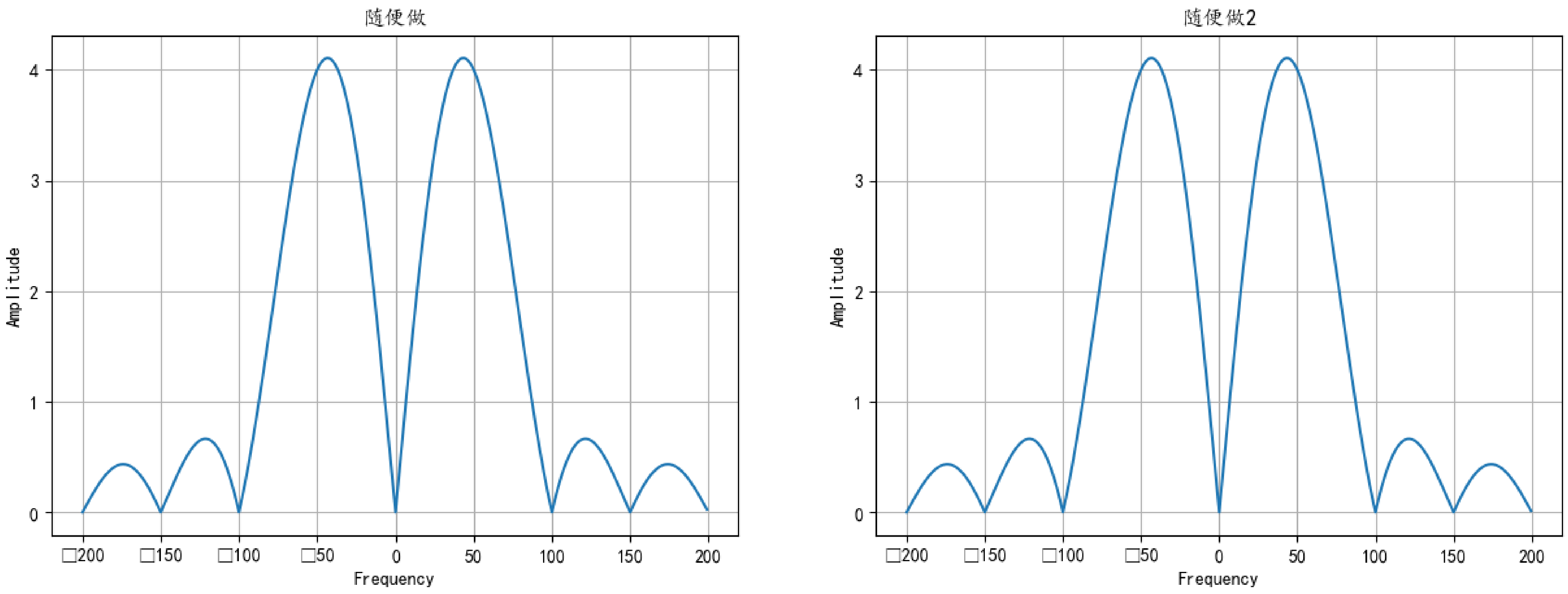


频域"分辨率"过低原因分析:

就是在频域采样的点太少,频谱曲线上拿到的点太少了,
上面我们是拿$[1,\frac{1}{50}]$中仅有的8个点进行计算的,所以频域中也只能
采到8个点,所以不能够呈现频谱曲线原本的样子.
解决思路:
首先采样频率我们是定死的,因为需要和前面的结果进行对比(时间轴上采样频率定死后,其背后
的频谱就已经固定下来了,分辨率也就定下来了,这是真正的分辨率),所有连续离散在频域上
的曲线要有严格的对应关系,而这个严格的对应关系是被时间域的采样频率定死的,
只要采样频率定下来,那么所有频域上在$[-\frac{f_s}{2},\frac{f_s}{2}]$中间的图像都是一样的.
既然采样频率定死了,那么时间轴上x(t)上离散的那些点也就固定下来了,从公式里可以看出来
我们不一定要拿所有的点出来计算,只需要拿出非零区间上面的点即可,但是上面我们这么做
发现拿的点数太少,导致频域分辨率过低,必须多拿一些,这正是我们第一次使用fft在[-1,1]区间
上做的.
接下来要做的事:
既然明白了原因和解决思路,我们后面随便拿一些点出来计算,记住保证采样频率一样即可,然后看看
是不是画出来的频谱图全都一样

In [11]:

```
tt=np.arange(-0.5,0.5,T)
yy=np.where((tt >= 0) & (tt <= 1/50), np.sin(wc*tt), 0)

spect_ = np.fft.fftshift(np.fft.fft(yy))
freq_ = np.fft.fftshift(np.fft.fftfreq(len(tt), T))

# 绘制频谱
plt.figure(figsize=(15,5))

plt.subplot(121)
plt.plot(freq_,abs(spect_))
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.title('随便做')
plt.grid(True)


ttt=np.arange(-0.1,2,T)  # 只要保证非零区间在就行
yyy=np.where((ttt >= 0) & (ttt <= 1/50), np.sin(wc*ttt), 0)

spect_1 = np.fft.fftshift(np.fft.fft(yyy))
freq_1 = np.fft.fftshift(np.fft.fftfreq(len(ttt), T))

# 绘制频谱
plt.subplot(122)
plt.plot(freq_1,abs(spect_1))
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.title('随便做2')
plt.grid(True)

plt.show()
```

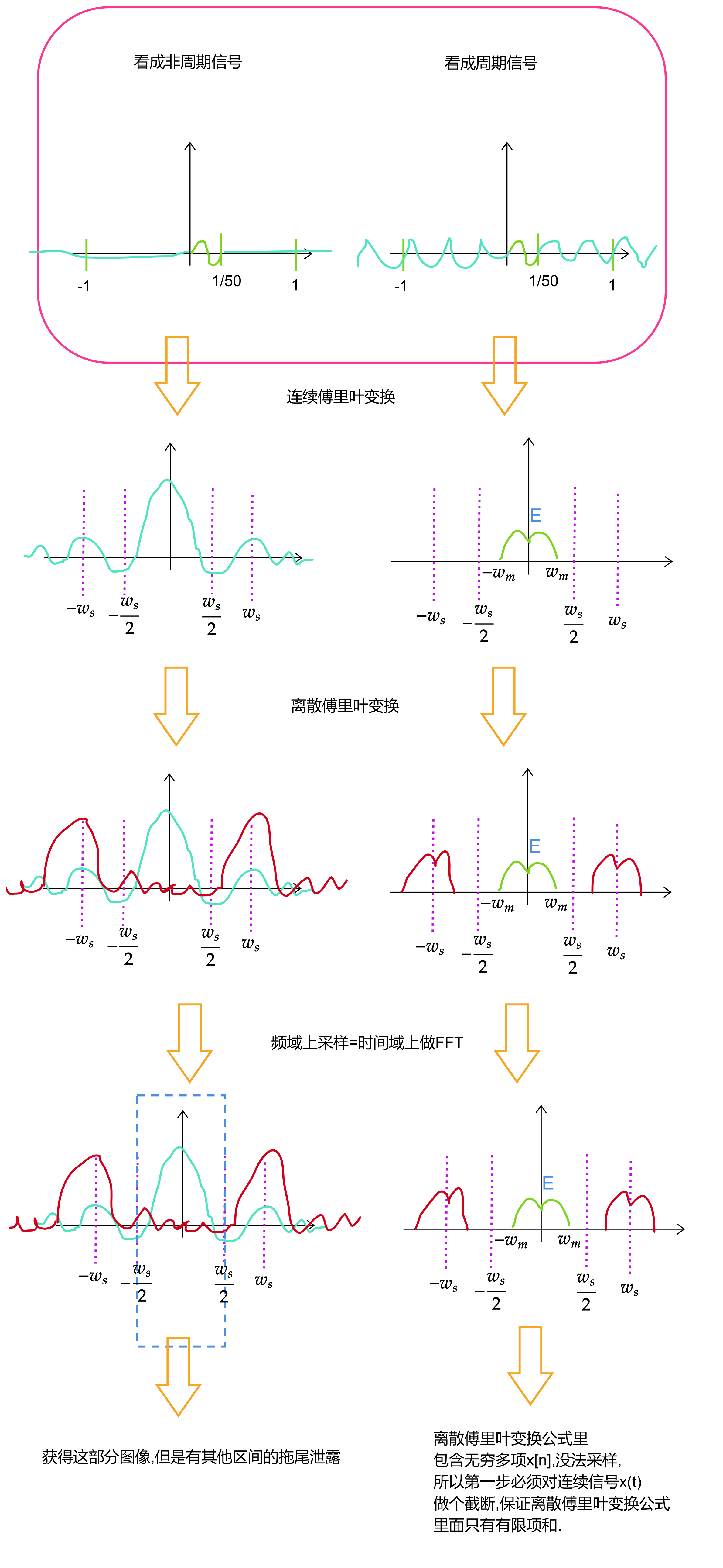


进一步分析:为什么会出现这种情况?

当然,前面从公式里看出确实会这样,公式里表明采样频率固定后,拿出来的点越多,
进行fft后就能越准确描绘出频谱特征,这确实是一方面,但是我们从更为本质的一方面
来分析出现这种情况的原因.
在fft中,我们取出一行,来与信号x[n]进行内积,得到了y[n]的一个数,那一行实际就是
e^{-jwn}的某个频率kw_0上的采样,就当它是$coskw_0$上的采样吧,然后与x[n]算
内积,实际就是它们原本的连续函数相乘求和,这种乘法也可以看成卷积,就是把cos看成
颠倒的即可,于是离散傅里叶变换其实就是拿不同频率的cos和函数进行卷积,所以拿的
卷积核越多,越是能把函数各个频率成分提取出来,对于卷积运算来说,函数放在哪个区间
是不影响提出来的特征本身的.
所以在时间轴上的采样频率定死的情况下,你拿出来的x[n]越多,你的高频卷积核越多,
抽的特征越多,频谱曲线绘制的更加完整.

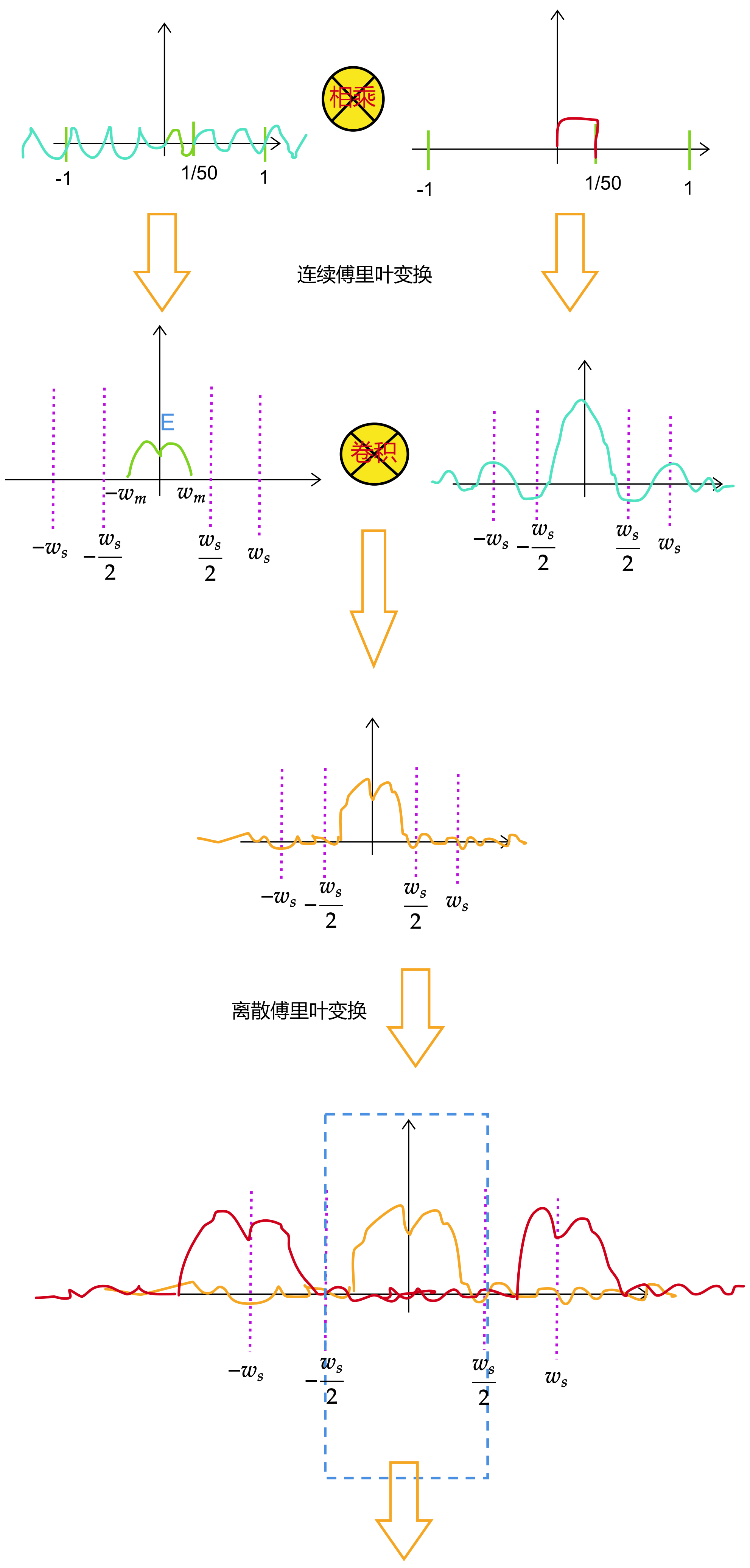
再进一步分析:为什么是采样函数的形状

我们是sint函数是有限区间上不为0的,实际也可以看成整个无穷区间上的sint与窗函数
乘积,这样傅里叶变换过去就是冲激函数与采样函数进行卷积,就成了上面采样函数的形状
注意:有的地方说的频率泄露,意思是把你的原始信号看成无穷区间上的周期信号,本身没看成
一个有限区间上不为0其他区间为0的信号,比如sint,认为它就是无穷区间上的信号,接着在一个
周期范围内采样进行傅里叶变换发现是上面采样函数的形状,不是冲激函数,认为它频谱泄露了,
这个主要取决于你看待原始信号的方式,如果看成无穷区间周期重复的sint,那么得到的就是用
冲激函数对采样函数进行卷积的频谱,这个频谱就是冲激函数的泄露形式,如果你的原始信号就是
只在一个区间不为0,其他区间为0,那么它的频谱就是fft得到的频谱,很准确,没泄露.
但是非周期信号,也就是有限区间不为0其他区间为0的信号,有个问题,就是它的频率无限长,没有
截止频率,导致不满足Nyquist采样定理,也就是在频域上,频域一定会发生混叠,不过没关系,由于
非周期信号的频谱一定也是个衰减信号,只要你采样频率够大,混叠的尾巴是个很小的量,没多大影响.



小结

1.看成非周期信号,仅在有限区间上不为0,此时你的频域频率没有截止,那么离散傅里叶变换在
$[-\frac{f_s}{2}, \frac{f_s}{2})$上的频谱是其真正频谱叠加其他区间频谱的拖尾,由于拖尾
也不是很大,所以形状看起来是不会有太大改变的;
2.看成周期信号与矩形窗函数相乘,那么离散傅里叶变换获得的是$[-\frac{f_s}{2}, \frac{f_s}{2})$上的频谱
和全区间上的带混叠的采样函数的卷积,然后fft就是上面的采样.
3.看成周期信号,手动延拓,再与其他的窗函数相乘,那么连续傅里叶变换获得的是$[-\frac{f_s}{2}, \frac{f_s}{2})$上的频谱
和窗函数傅里叶变换的卷积,卷积后的频谱相比原始信号的频谱,变成了无限长带拖尾,后面做离散傅里叶变换,就是
把这个带拖尾的频谱每个区间都叠加一次,所以其他区间的频谱因为拖尾会泄露到中间区间$[-\frac{f_s}{2}, \frac{f_s}{2})$上,
这是无法避免的,需要做的就是让拖尾带来的泄露尽可能小.



此时频域上采样=时间域上做FFT,因为此时时间域上的信号不是无限长了,其离散傅里叶变换仅有限项和.

但是这部分图像还是有其他区间的拖尾泄露,这是无法避免的,能改进的地方就是,一开始的窗函数不一定要选矩形窗,还可以选其他窗,让拖尾尽量不那么明显