在 [ ]:

```
# 獲取 thinkdsp.py

導入 操作系統

如果 不是 操作系統。路徑。存在('thinkdsp.py'):
    !wget  https://github.com _ _ _ _ com / AllenDowney / ThinkDSP / raw / master / code /
```

在 [2]:

```
 將numpy 導入為 np
導入 matplotlib。pyplot 作為 plt

從 thinkdsp 導入 裝飾
```

練習 1

在本章中，我聲稱高斯曲線的傅里葉變換也是高斯曲線。對於離散傅里葉變換，這種關係近似成立。

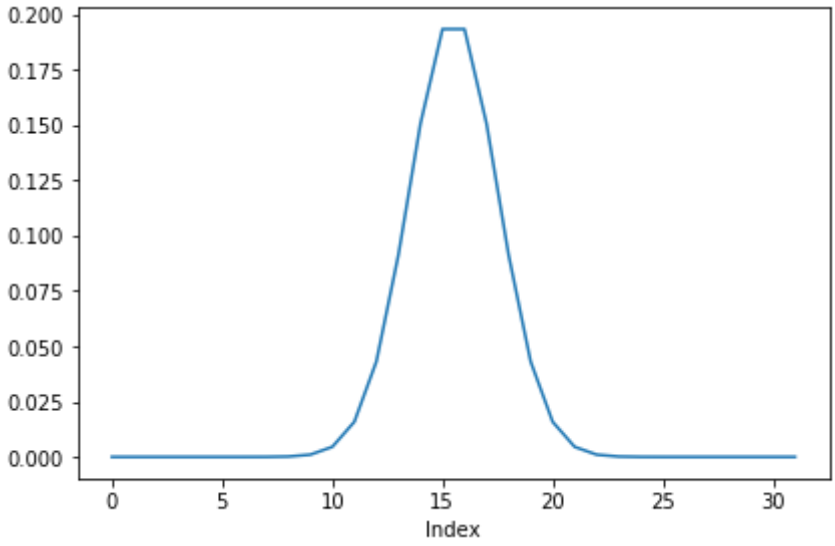
試試看幾個例子。隨著您的變化，傅立葉變換會發生什麼變化 `std` ？

Solution

我將從類似於書中示例的高斯開始。

在 [3]:

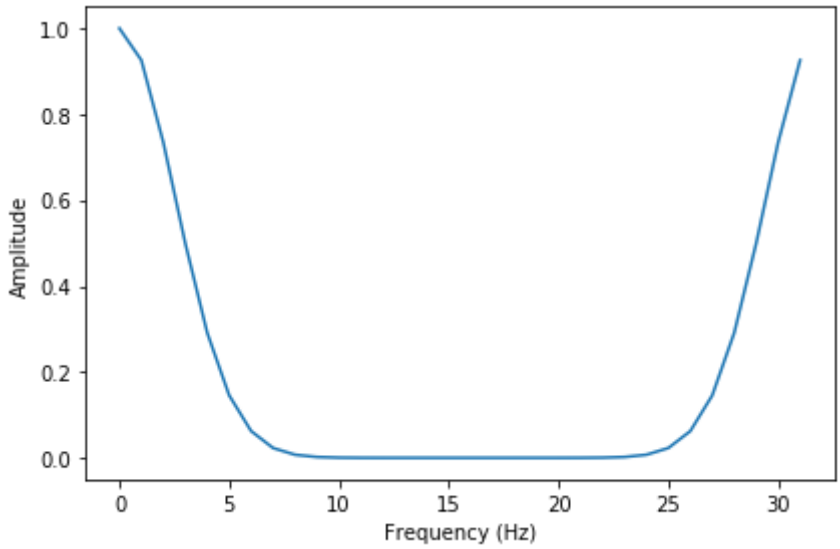
```
導入 scipy。信號

高斯 =  scipy。信號。高斯( M = 32 ,標準= 2 )
高斯 /= 總和(高斯)
plt _ 情節(高斯)
裝飾(xlabel = '索引')
```



這是 FFT 的樣子：

在 [4]:

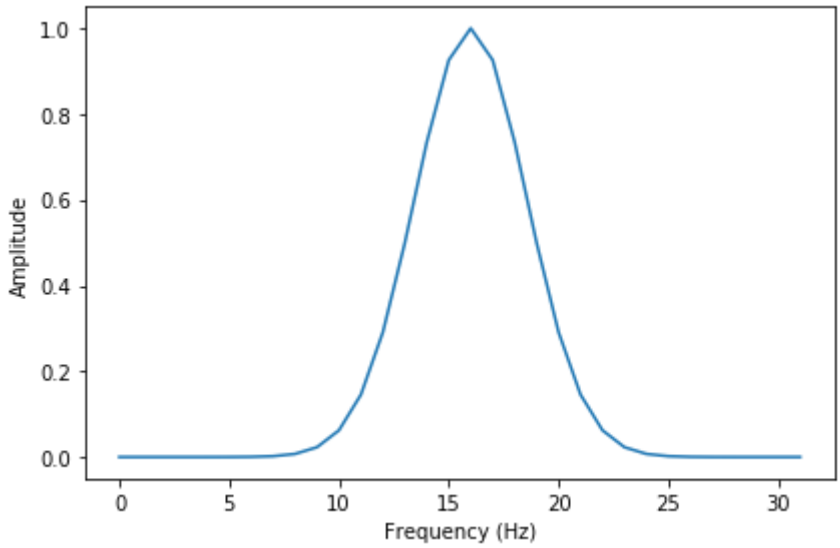
```
fft_gaussian  =  np。fft _ fft (高斯)
plt _ 情節 ( abs ( fft_gaussian ) )
裝飾( xlabel = '頻率 (Hz)' , ylabel = '幅度' )
```



如果我們將負頻率向左滾動，我們可以更清楚地看到它是高斯的，至少是近似的。

在 [5]:

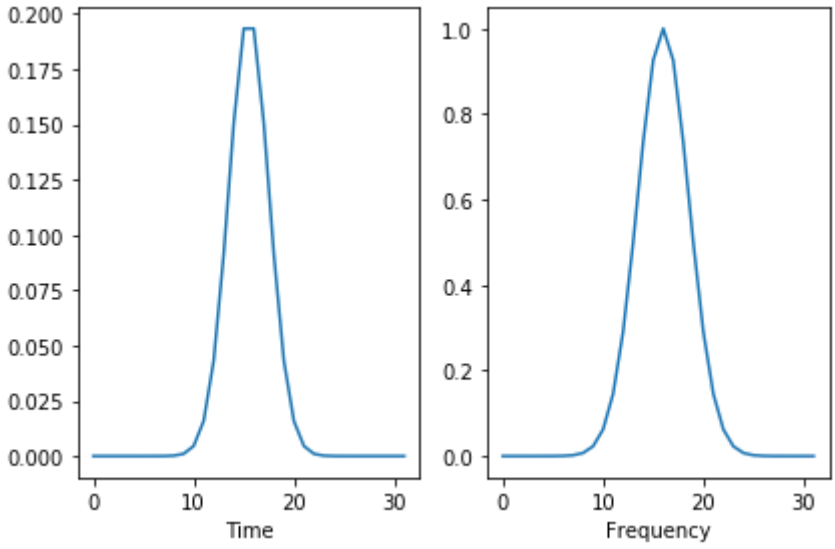
```
N  =  len (高斯)
fft_rolled  =  np。滾動( fft_gaussian , N // 2 )
plt _ 情節 ( abs ( fft_rolled ) )
裝飾( xlabel = '頻率 (Hz)' , ylabel = '幅度' )
```



此函數並排繪製高斯窗口及其 FFT。

在 [6]:

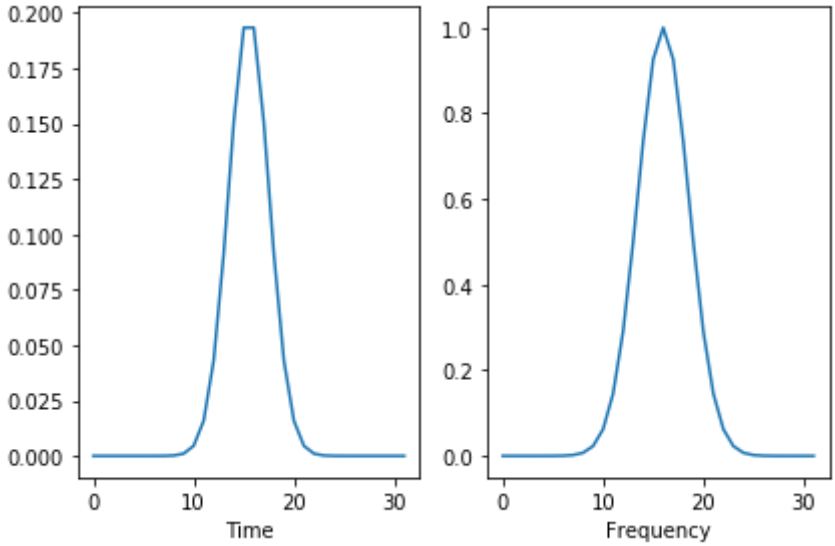
```
def  plot_gaussian (標準):
    米 =  32
    高斯 =  scipy。信號。高斯( M = M , std = std )
    高斯 /= 總和(高斯)

    plt _ 子圖( 1 , 2 , 1 )
    plt _ 情節 ( 高斯 )
    裝飾 ( xlabel = '時間' )

    fft_gaussian  =  np。fft _ fft (高斯)
    fft_rolled  =  np。滾動( fft_gaussian , M // 2 )

    plt _ 子圖( 1 , 2 , 2 )
    plt _ 情節 ( np.abs ( fft_rolled  ) ) _
    裝飾 ( xlabel = '頻率' )
    plt.show()

情節高斯 ( 2 )
```



現在我們可以進行交互，顯示發生的 `std` 變化。

在 [7]:

```
從 ipywidgets 導入 交互，交互，固定
 將ipywidgets 導入為 小部件

滑塊 = 小部件。FloatSlider（最小值= 0.1，最大值= 10，值= 2）
交互（plot_gaussian，std = slider）；
```



練習 2

如果你做了第 3 章中的練習，你就會看到漢明窗以及 NumPy 提供的其他一些窗對頻譜洩漏的影響。我們可以通過查看它們的 DFT 來了解這些窗口的效果。

除了我們在這個窗口中使用的高斯窗口之外，創建一個相同大小的漢明窗口。零填充窗口並繪製它們的 DFT。哪個窗口可以作為更好的低通濾波器？您可能會發現在對數上繪製 DFT 很有用——y規模。

嘗試幾個不同的窗口和幾個不同的大小。

Solution

按照本章中的示例，我將創建一個以 44.1 kHz 採樣的 1 秒波。

在 [8]:

```
從 thinkdsp 導入 SquareSignal

信號 =  SquareSignal (頻率= 440 )
波 = 信號。make_wave (持續時間= 1.0 ,幀率= 44100 )
```

我將創建幾個窗口。我選擇了高斯窗口的標準差以使其與其他窗口相似。

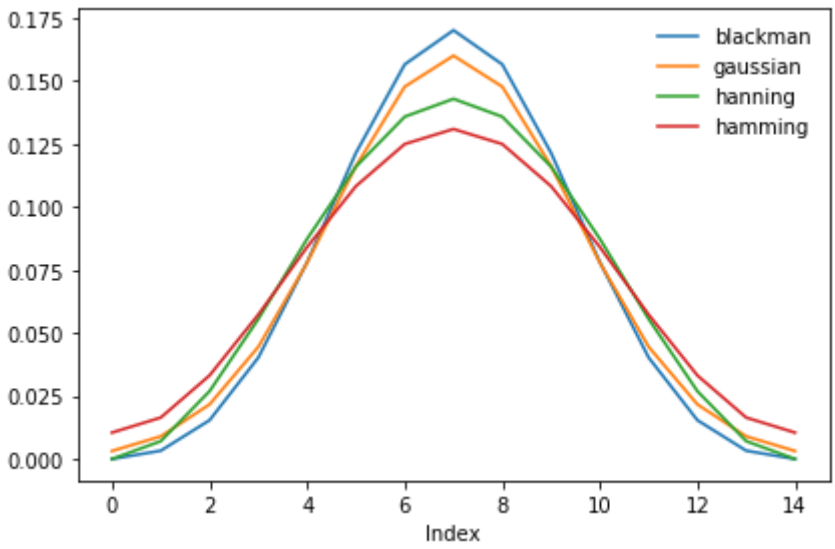
在 [9]:

```
米 =  15
標準 =  2.5

高斯 =  scipy。信號。高斯( M = M , std = std )
巴特利特 =  np。巴特利特( M )
布萊克曼 =  np。布萊克曼( M )
漢明 =  np。漢明( M )
漢寧 =  np。漢寧( M )

windows  = [布萊克曼，高斯，漢寧，漢明]
名稱 = [ '布萊克曼'，'高斯'， '漢寧'，'漢明' ]

對於 窗口 中的 窗口：
    窗口 /= 總和(窗口)
```

讓我們看看窗戶是什麼樣子的。

在 [10]:

```
對於 window，zip中的名稱 （windows，names）：
    plt _ 情節（窗口，標籤=名稱）

裝飾（xlabel = '索引'）
```



它們非常相似。讓我們看看他們的 DFT 長什麼樣：

在 [11]:

```
def  zero_pad (數組, n ):
    """用零擴展一個數組。

    數組：NumPy 數組
    n：結果的長度

    返回：新的 NumPy 數組
    """
    水庫 =  np。零( n )
    res [: len (數組)] = 數組
    返回 資源
```

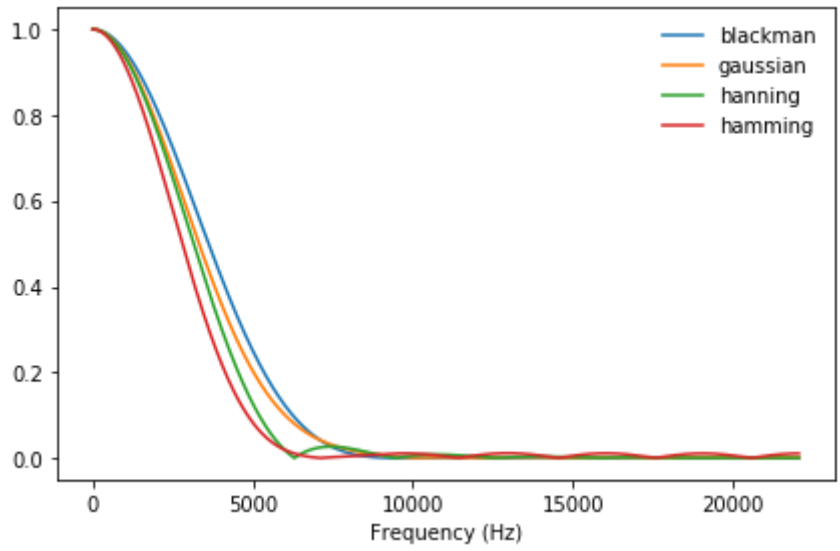
在 [12]:

```
def  plot_window_dfts ( 窗口，名稱 ) ：
    """
    """
    對於 window，zip中的名稱  ( windows，names ) ：
        填充 =   zero_pad (窗口, len (波))
        dft_window  =  np。fft _ rfft ( 填充 )
        plt _ 情節 ( abs ( dft_window ) ，標籤=名稱 )
```

也非常相似，但看起來 Hamming 下降最快，Blackman 下降最慢，而 Hanning 具有最明顯的旁瓣。

在 [13]:

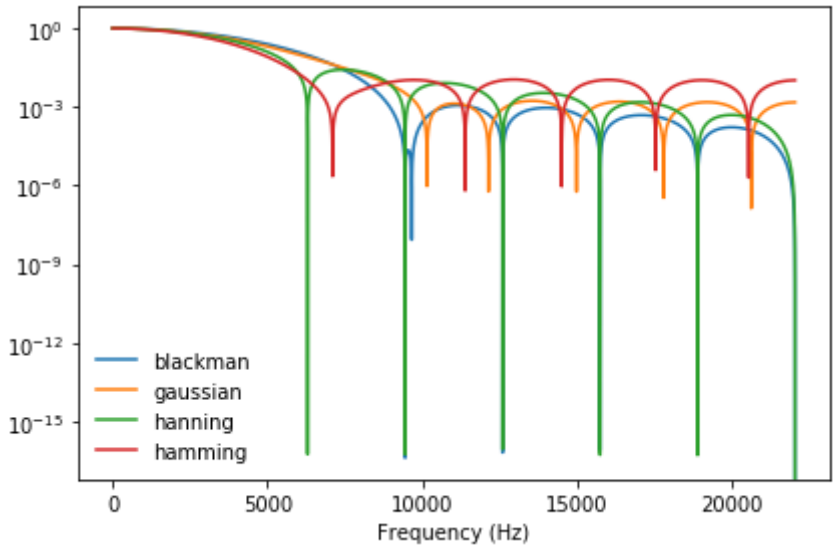
```
plot_window_dfts ( 窗口，名稱 )
裝飾 ( xlabel = '頻率 ( Hz ) ' )
```



這是 log-y 尺度上的相同圖。

在 [14]:

```
plot_window_dfts（窗口，名稱）
裝飾( xlabel = '頻率 (Hz)' , yscale = 'log' )
```



在對數尺度上，我們可以看到 Hamming 和 Hanning 一開始比其他兩個下降得更快。漢明窗和高斯窗似乎具有最持久的旁瓣。漢寧窗可能具有快速衰減和最小旁瓣的最佳組合。

在 [ ]:

```

```