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，我將捲積描述為信號的移位、縮放副本的總和。嚴格來說，這個操作是 *線性*卷積，它不假設信號是周期性的。

但是當我們將信號的 DFT 乘以傳遞函數時，該操作對應於 *循環*卷積，它假設信號是周期性的。結果，您可能會注意到輸出在開頭包含一個額外的註釋，該註釋從結尾處環繞。

幸運的是，這個問題有一個標準的解決方案。如果在計算 DFT 之前在信號末尾添加足夠多的零，則可以避免迴繞併計算線性卷積。

修改中的示例 `chap10soln.ipynb` 並確認零填充消除了輸出開頭的額外註釋。

*解決方案：*我會將兩個信號截斷為2^{16}元素，然後將它們零填充到2^{17}. 使用 2 的冪使 FFT 算法最有效。

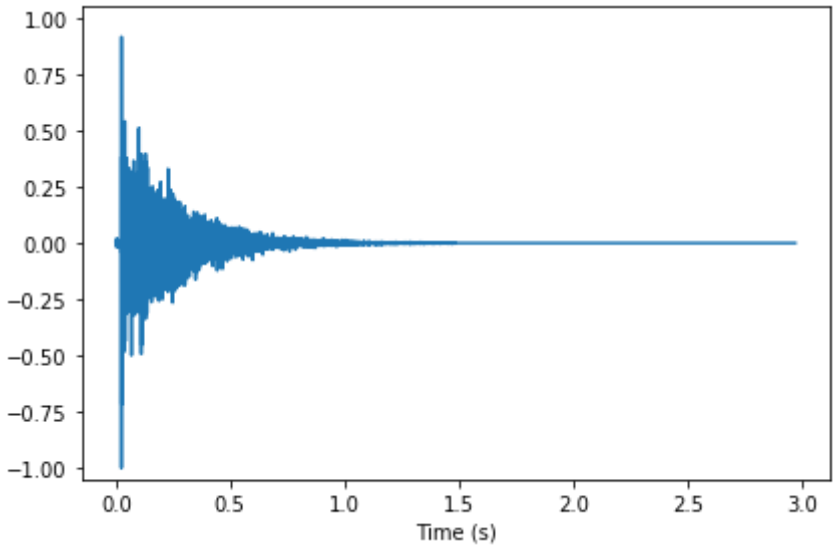
這是脈衝響應：

在 [3]:

```
如果 不是 操作系統。路徑。存在('180960__kleeb__gunshot.wav'):
    !wget  https://github.com _ _ _ _ com / AllenDowney / ThinkDSP / raw / master / code /
```

在 [4]:

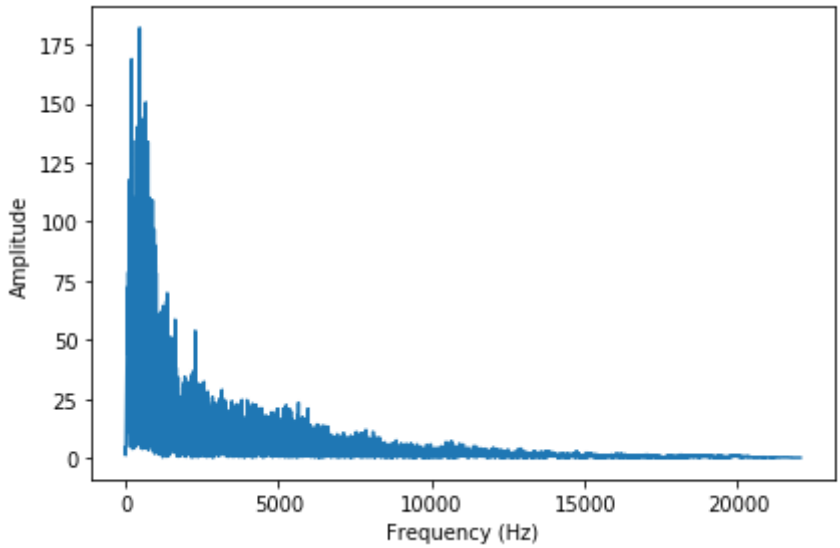
```
從 thinkdsp 導入 read_wave

響應 =  read_wave ( '180960__kleeb__gunshot.wav' )

開始 =  0.12
響應 = 響應。段 ( 開始=開始 )
回應。班次 ( -開始 )

回應。截斷( 2 ** 16 )
回應。zero_pad ( 2 ** 17 )

回應。規範化 ( )
回應。情節 ( )
裝飾( xlabel = '時間 (s)' )
```



及其光譜：

在 [5]:

```
轉移 = 響應。make_spectrum ()
轉移。情節 ( )
裝飾( xlabel = '頻率 (Hz)' , ylabel = '幅度' )
```



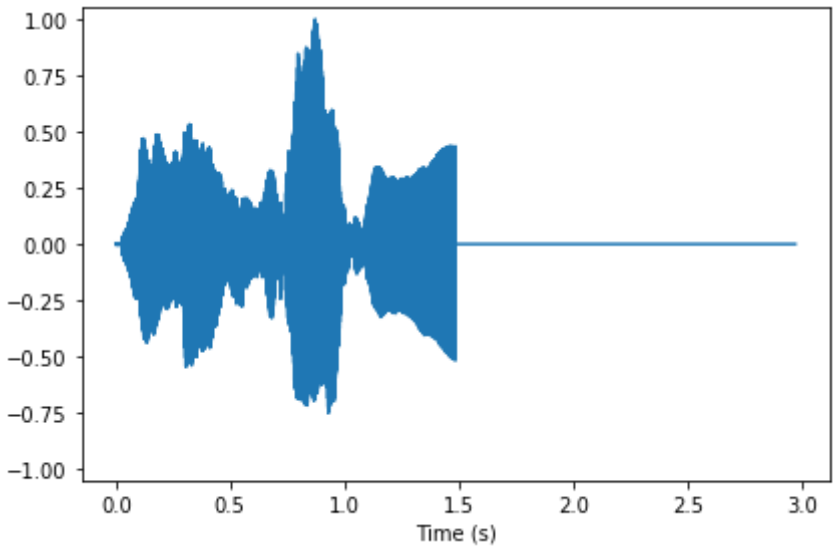
這是信號：

在 [6]:

```
如果 不是 操作系統。路徑。存在 ( '92002__jcveliz__violin-original.wav' ) :
    ! wget  https://github.com _ _ _ _ com / AllenDowney / ThinkDSP / raw / master / code /
```

在 [7]:

```
小提琴 =  read_wave ( '92002__jcveliz__violin-original.wav' )

開始 =  0.11
小提琴 = 小提琴。段 ( 開始=開始 )
小提琴。班次 ( -開始 )

小提琴。截斷( 2 ** 16 )
小提琴。zero_pad ( 2 ** 17 )

小提琴。規範化 ( )
小提琴。情節 ( )
裝飾( xlabel = '時間 (s)' )
```



及其光譜：

在 [8]:

```
頻譜 = 小提琴。make_spectrum ()
```

現在我們可以將信號的 DFT 乘以傳遞函數，然後轉換回波：

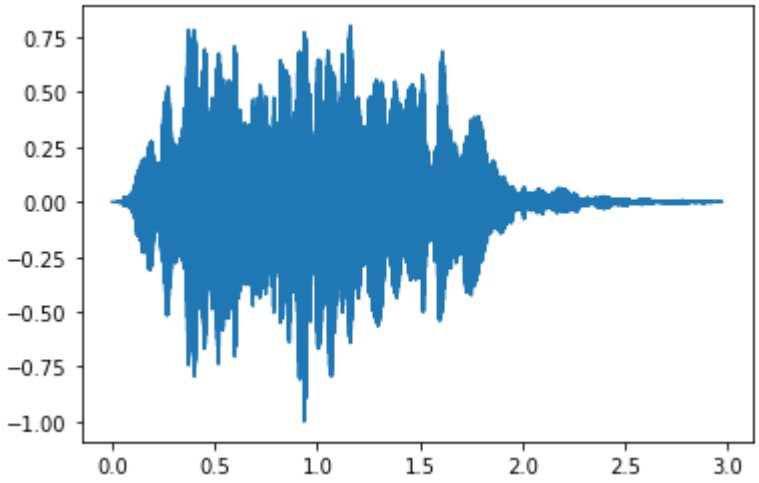
在 [9]:

```
輸出 = ( 頻譜 * 傳輸 )。make_wave ()
輸出。規範化 ( )
```

結果看起來不像是環繞：

在 [10]:

```
輸出。情節 ( )
```



而且我們在開頭沒有聽到額外的音符：

在 [11]:

```
輸出。make_audio ()
```

輸出[11]：

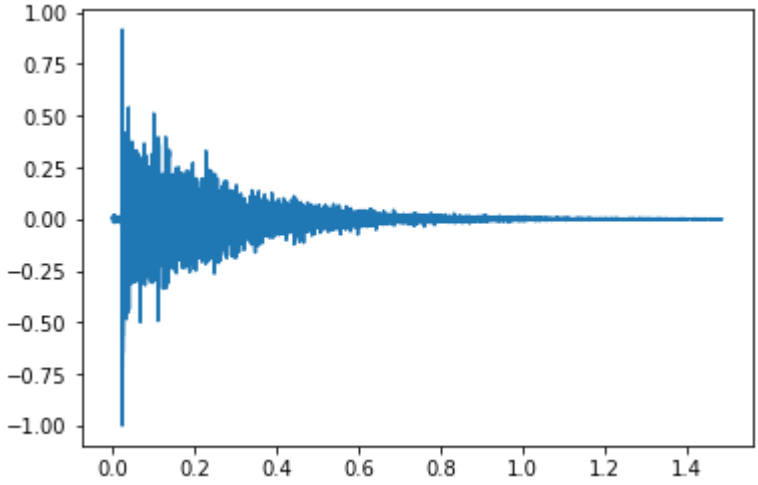
0:00 / 0:00

我們應該從 `np.convolve` 和得到相同的結果 `scipy.signal.fftconvolve` 。

首先，我將擺脫零填充：

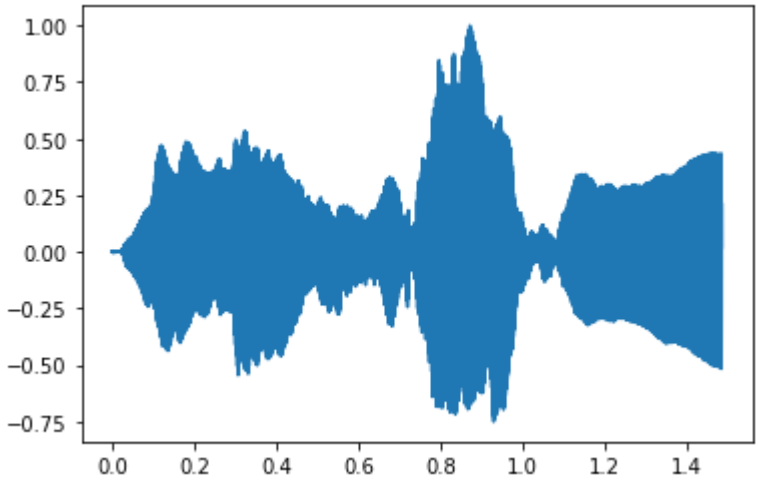
在 [12]:

```
回應。截斷( 2 ** 16 )
回應。情節 ( )
```



在 [13]:

```
小提琴。截斷( 2 ** 16 )
小提琴。情節 ( )
```



現在我們可以比較 `np.convolve` ：

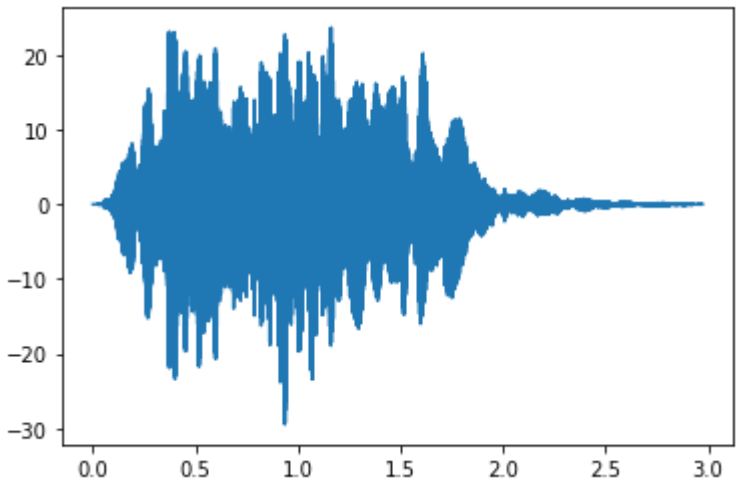
在 [14]:

```
輸出2  = 小提琴。卷積（響應）
```

結果類似：

在 [15]:

```
輸出2。情節（）
```



聽起來一樣：

在 [16]:

```
輸出2。make_audio ()
```

出[16]：

0:00 / 0:00

但結果的長度並不完全相同：

在 [17]:

```
len（輸出），len（輸出2）
```

出[17]：

```
(131072, 131071)
```

`scipy.signal.fftconvolve` 做同樣的事情，但顧名思義，它使用 FFT，因此速度要快得多：

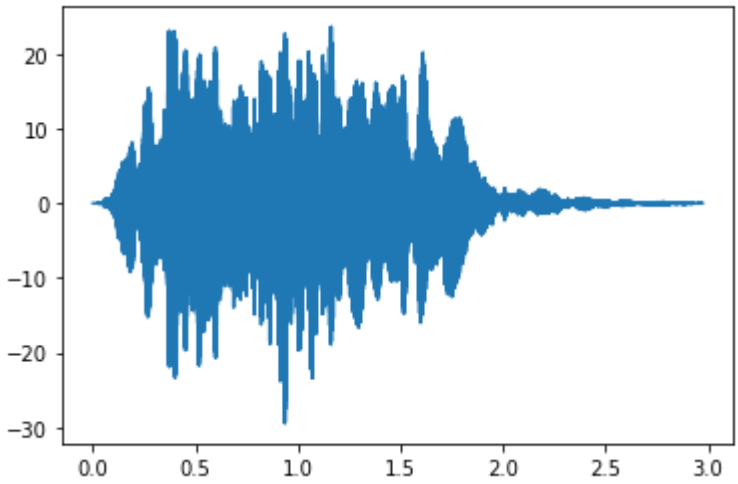
在 [18]:

```
從 thinkdsp 導入 Wave

導入 scipy。信號
ys  =  scipy。信號。fftconvolve（小提琴。ys，響應。ys）_ _
output3  =  Wave ( ys , framerate = violin.framerate ) _ _
```

結果看起來一樣。

在 [19]:

```
輸出 3。情節（ ）
```



聽起來一樣：

在 [20]:

```
輸出 3。make_audio ()
```

出[20]：

0:00 / 0:00

在浮點錯誤中，它們是相同的：

在 [21]:

```
輸出2。最大差異（輸出 3）
```

輸出[21]：

```
1.7763568394002505e-14
```

練習 2

Open AIR 庫為任何對可聽化和聲學脈衝響應數據感興趣的人提供“集中的.....在線資源”（http://www.openairlib.net (http://www.openairlib.net)）。瀏覽他們收集的脈衝響應數據並下載一個聽起來很有趣的數據。查找與您下載的脈衝響應具有相同採樣率的簡短錄音。

在測量脈衝響應的空間中模擬錄音的聲音，計算方法有兩種：將錄音與脈衝響應進行卷積，計算對應於脈衝響應的濾波器並乘以錄音的 DFT。

*解決方案：*我下載了聖奧爾本斯大教堂聖母教堂的脈衝響應http://www.openairlib.net/auralizationdb/content/lady-chapel-st-albans-cathedral (http://www.openairlib.net/auralizationdb/content/lady-chapel-st-albans-cathedral)

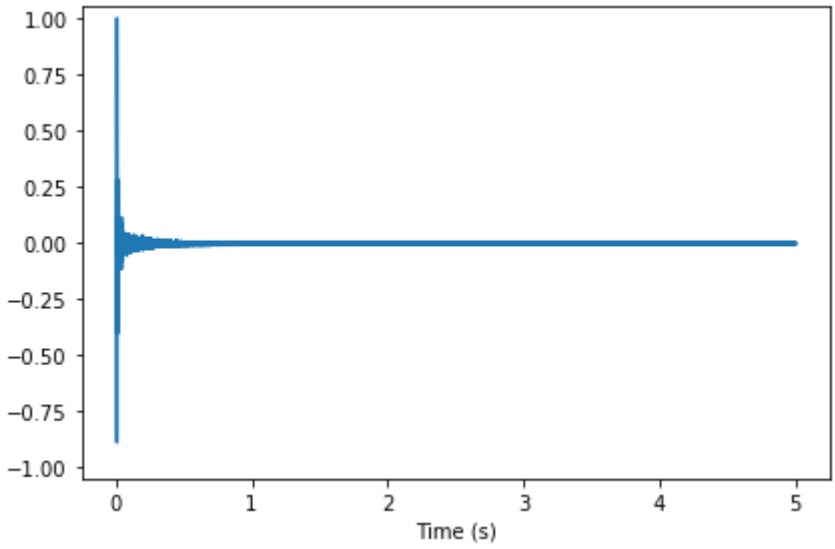
感謝約克大學的 Audiolab：Marcin Gorzel、Gavin Kearney、Aglaia Foteinou、Sorrel Hoare、Simon Shelley。

在 [22]:

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

在 [23]:

```
響應 =  read_wave（'stalbans_a_mono.wav'）

開始 =  0
持續時間 =  5
響應 = 響應。段（持續時間=持續時間）
回應。班次（-開始）

回應。規範化（）
回應。情節（）
裝飾（ xlabel = '時間 (s)'）
```



聽起來是這樣的：

在 [24]:

```
回應。make_audio ()
```

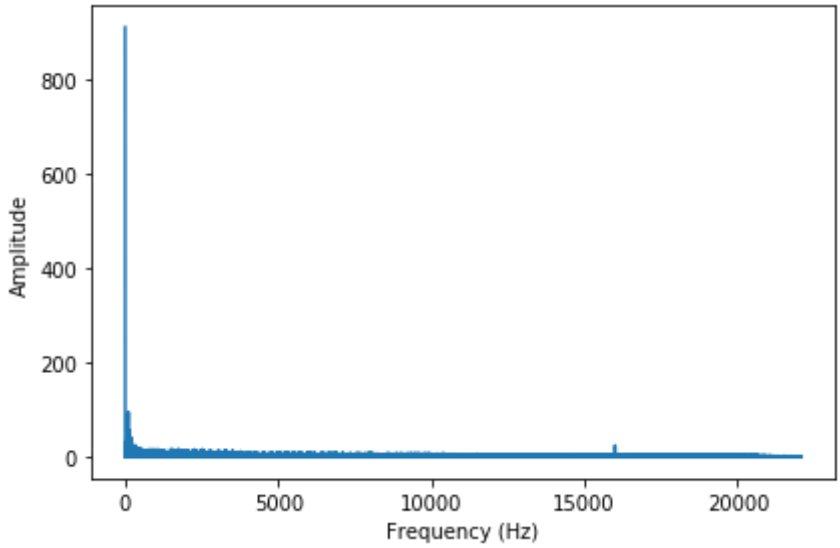
出[24]：

0:00 / 0:00

脈衝響應的 DFT 是傳遞函數：

在 [25]:

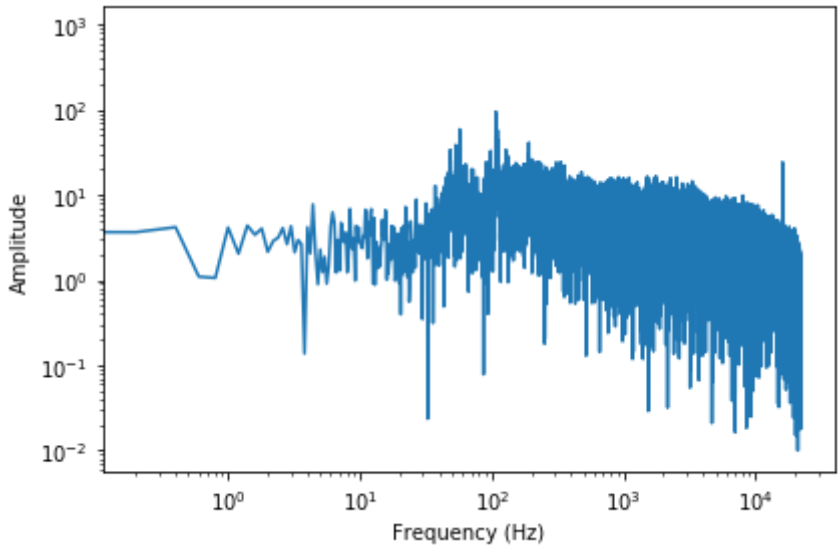
```
轉移 = 響應。make_spectrum ()
轉移。情節 ( )
裝飾( xlabel = '頻率 (Hz)' , ylabel = '幅度' )
```



這是對數刻度上的傳遞函數：

在 [26]:

```
轉移。情節 ( )
裝飾( xlabel = '頻率 (Hz)' , ylabel = '幅度' ,
         xscale = 'log' , yscale = 'log' )
```



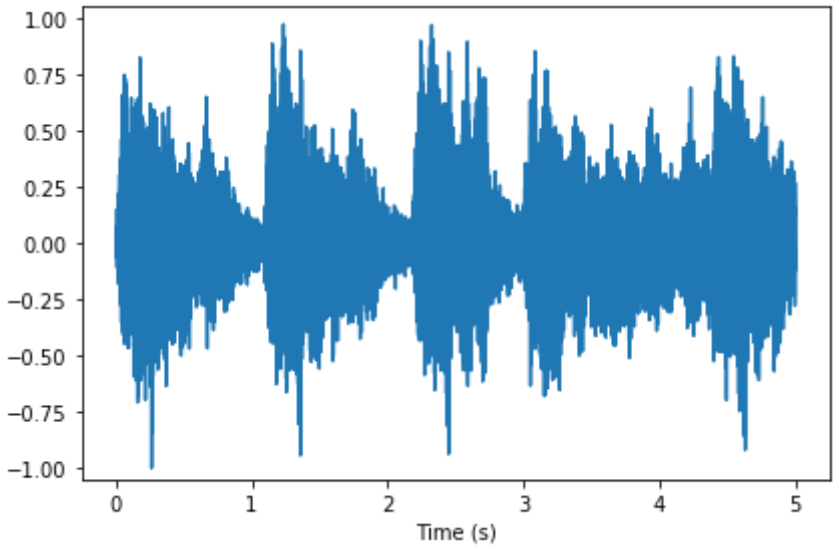
現在我們可以模擬在同一個房間播放並以相同方式錄製的錄音聽起來會是什麼樣子。這是我們之前使用的小號錄音：

在 [27]:

```
如果 不是 操作系統。路徑。存在 ( '170255__dublie__trumpet.wav' ) :
    ! wget  https://github.com _ _ _ _ com / AllenDowney / ThinkDSP / raw / master / code /
```

在 [28]:

```
wave  =  read_wave ( '170255__dublie__trumpet.wav' )

開始 =  0.0
波浪 = 波浪。段 ( 開始=開始 )
波。班次 ( -開始 )

波。截斷 ( len ( 響應 ) )
波。規範化 ( )
波。情節 ( )
裝飾( xlabel = '時間 (s)' )
```



這是轉換前的聲音：

在 [29]:

```
波。make_audio ()
```

出[29]：

0:00 / 0:00

現在我們計算小提琴錄音的 DFT。

在 [30]:

```
頻譜 = 波。make_spectrum ()
```

我將小提琴錄音修剪為與脈衝響應相同的長度：

在 [31]:

```
len(頻譜.hs), len(傳輸.hs)
```

輸出[31]：

```
(110251, 110251)
```

在 [32]:

```
頻譜.fs
```

輸出[32]：

```
數組([0.00000e+00, 2.00000e-01, 4.00000e-01, ..., 2.20496e+04,
       2.20498e+04, 2.20500e+04])
```

在 [33]:

```
轉移.fs
```

出[33]：

```
數組([0.00000e+00, 2.00000e-01, 4.00000e-01, ..., 2.20496e+04,
       2.20498e+04, 2.20500e+04])
```

現在我們可以在頻域中相乘並將變換回時域。

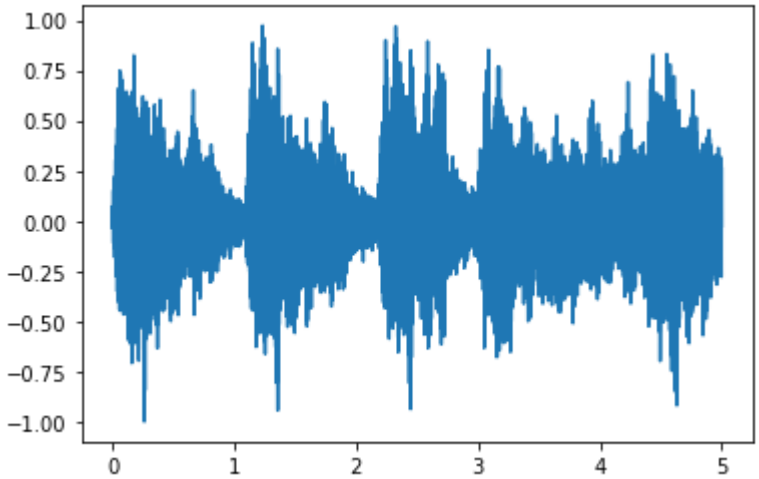
在 [34]:

```
輸出 = (頻譜 * 傳輸).make_wave()
輸出.規範化()
```

以下是原始記錄和轉換記錄的比較：

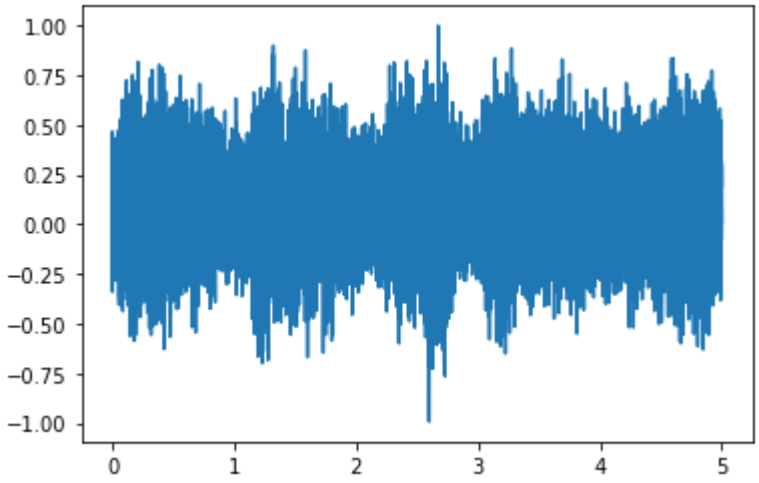
在 [35]:

```
波.情節()
```



在 [36]:

```
輸出。情節 ( )
```



聽起來是這樣的：

在 [37]:

```
輸出。make_audio ()
```

出[37]：

0:00 / 0:00

現在我們將此操作識別為卷積，我們可以使用 convolve 方法計算它：

在 [38]:

```
卷積2  = 波。卷積 ( 響應 )
卷積2。規範化 ( )
卷積2。make_audio ()
```

出[38]：

0:00 / 0:00

在 [ ]:

```

```