CS168 Spring Assignment [8]

SUNet ID(s): 16337266

Name(s): 徐原

Part 1

(a) 连接关系如下：

对于以上连接的分析和说明：

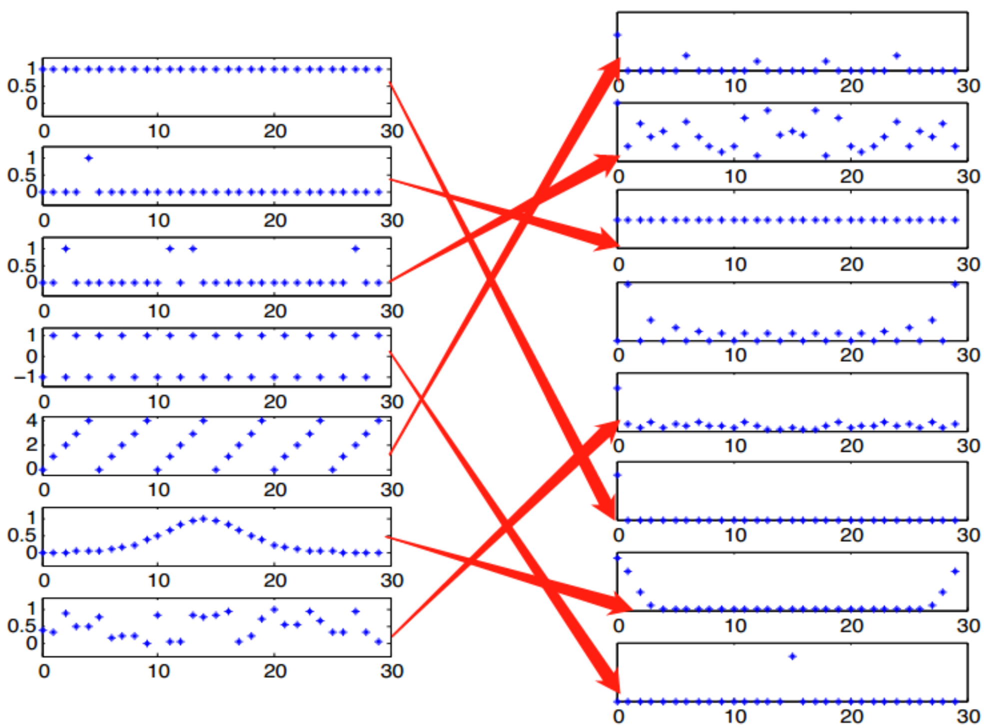

图 1: line connent left and right

向量 1：常数项的傅里叶变换是狄拉克函数。这里左边是一组恒定的水平点，而右边是狄拉克函数。

向量 2：狄拉克函数的傅里叶变换是常数函数。这里左边是狄拉克型函数，右边是常数。

向量 3：左向量表示放在一起的几个不同的频率。明显地，左图有四个 e 非零值，进

行傅立叶变换后，应该存在由 4 个不同频率组成的正弦波。

向量 4：左向量有两个取值, 分别为 1 和-1，可以看作是正弦波中的极大极小值，对应一个正弦波。

向量 5：左向量呈锯齿波状，可以分解为多个波的叠加，故对应带有多个非零值的右图。

向量 6：左向量是高斯分布状，傅里叶变换后，高斯分布形态不变，因此对应的仍是带有高斯分布性质的频域。

向量 7：左向量是分布在 0 和 1 之间的恒定值周围的噪声。对应的傅立叶变换将是零碎的点对应右图 5。

Part 2

(a) 在概率 q [150] 的情况下，结果总和为 250。

(b) 为了证明 $F(f * g) = Ff + \cdot Fg+$, 即证明 $F(f * g)[m] = (Ff + \cdot Fg+)[m]$ 对于任意的 0 m 2N –1。首先，我们将使用傅立叶变换的定义为长度为 n 的向量 v 重写等式的右侧。然后，因为对于 $N\ a\ 2N - 1\ f+[a]\ g+[a]$ 等于 0，我们可以将两个求和的上界改为 N-1。接着，接下来，让 $l = j + k$。将使用 l 使方程看起来更接近卷积的傅立叶变换。以上四个操作的数学表达式子为:

$$(Ff^+ \cdot Fg^+)[m] = \left(\sum_{j=0}^{2N-1} e^{\frac{2\pi imj}{2N}} f^+[j]\right)\left(\sum_{k=0}^{2N-1} e^{\frac{2\pi imk}{2N}} g^+[k]\right)$$

$$(Ff^+ \cdot Fg^+)[m] = \left(\sum_{j=0}^{N-1} e^{\frac{2\pi imj}{2N}} f^+[j]\right)\left(\sum_{k=0}^{N-1} e^{\frac{2\pi imk}{2N}} g^+[k]\right)$$

$$(Ff^+ \cdot Fg^+)[m] = \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} e^{\frac{2\pi im(j+k)}{2N}} f^+[j] g^+[k]$$

$$(Ff^+ \cdot Fg^+)[m] = \sum_{j=0}^{N-1}\sum_{l=j}^{j+N-1} e^{\frac{2\pi iml}{2N}} f^+[j] g^+[l-j]$$

$$(Ff^+ \cdot Fg^+)[m] = \sum_{j=0}^{N-1}\sum_{l=0}^{2N-1} e^{\frac{2\pi iml}{2N}} f^+[j]g^+[l-j]$$
$$(Ff^+ \cdot Fg^+)[m] = \sum_{l=0}^{2N-1} e^{\frac{2\pi iml}{2N}} \sum_{j=0}^{N-1} f^+[j]g^+[l-j]$$
$$(Ff^+ \cdot Fg^+)[m] = \sum_{l=0}^{2N-1} e^{\frac{2\pi iml}{2N}} \sum_{j=0}^{N-1} f[j]g[l-j]$$

$$(Ff^+ \cdot Fg^+)[m] = \sum_{j=0}^{N-1}\sum_{l=0}^{2N-1} e^{\frac{2\pi iml}{2N}} f^+[j]g^+[l-j]$$
$$(Ff^+ \cdot Fg^+)[m] = \sum_{l=0}^{2N-1} e^{\frac{2\pi iml}{2N}} \sum_{j=0}^{N-1} f^+[j]g^+[l-j]$$
$$(Ff^+ \cdot Fg^+)[m] = \sum_{l=0}^{2N-1} e^{\frac{2\pi iml}{2N}} \sum_{j=0}^{N-1} f[j]g[l-j]$$

至此，当 $l - j < 0$ $g + [1 - j] = 0$ 时，意味着整个项将等于 0. 因此，我们可以将内部求和中的 $l = j$ 移位到 $l = 0$ 而不影响求和。因为对于 N a $g + [a] = 0$，我们可以将相同求和的上界从 $j + N - 1$ 移位到 $2N - 1$。
对于等式的左边，计算 F（f * g）[m]。
结论：使用快速傅里叶变换实现 f 和 g 的卷积：F-1（Ff+ · Fg +），其中 f+ 和 g+ 是通过用零填充 f 和 g 获得的 2N 元组，并且 · 表示逐元素乘法。如果两个元组具有不同的长度，则较短的元组可以用额外的零填充。

(c) Code:

```
def pro_x_y(x, y):
  lengthOfx = len(x)
  lengthOfy = len(y)
  xr = x
  yr = y
  if lengthOfx < lengthOfy:
    xr += [0 for i in range(lengthOfy - lengthOfx)]
  elif lengthOfy < lengthOfx:
```

```python
    yr += [0 for i in range(lengthOfx - lengthOfy)]
  xr += [0 for i in range(len(xr))]
  yr += [0 for i in range(len(yr))]
  return xr, yr

def multiply(x, y):
  xr, yr = pro_x_y(x, y)
  x_fft = np.fft.fft(xr)
  y_fft = np.fft.fft(yr)
  x_y_mult = np.multiply(x_fft, y_fft)
  inv = np.fft.ifft(x_y_mult)
  values = []
  carry_over = 0
  for val in inv:
    print(val)
    curr = int(round(val.real, 0) + carry_over)
    if curr >= 10:
      carry_over = int(curr)//10
      curr %= 10
    else:
      carry_over = 0
    values.append(curr)
  while(values[-1] == 0):
    del values[-1]
  return values
x = [0,9,8,7,6,5,4,3,2,1,0,9,8,7,6,5,4,3,2,1]
y = [0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9]
print(multiply(x, y))
```

结果：0, 0, 9, 6, 2, 5, 3, 6, 2, 1, 1, 1, 1, 0, 8, 3, 6, 4, 7, 3, 2, 2, 4, 6, 0, 7, 5, 6, 9, 4, 2, 2, 5, 9, 7, 1, 2, 0, 7, 3, 1, 1, 3, 6, 2, 3, 9, 1, 2, 1
反过来，即：1, 2, 1, 9, 3, 2, 6, 3, 1, 1, 3, 7, 0, 2, 1, 7, 9, 5, 2, 2, 4, 9, 6, 5, 7, 0, 6, 4, 2, 2, 3, 7, 4, 6, 3, 8, 0, 1, 1, 1, 1, 2, 6, 3, 5, 2, 6, 9, 0, 0

(d) 分析：使用此方法计算卷积需要 O（nlogn）时间，其中 n 是最长数字中的位数。但是，使用整数乘法算法，将花费 O（nm）时间，其中 n 和 m 是每个数字中的数字位数因此，FFT 方法要快得多。

Part 3

(a) 我听到了"Laurel" 和"Yanny"

(b) Code:

```
file = 'laurel_yanny.wav'
sampleRate, data = wavfile.read(file)
print("sample rate: ", sampleRate)
print("shape of data: ", data.shape)

def plot_b(data):
  time = data.shape[0]
  x= [i for i in range(time)]
  plt.plot(x, data)
  plt.title("part3_b")
  plt.xlabel("Time")
  plt.ylabel("Phsyical Position")
  plt.savefig('p3_b.png', format = 'png')
  plt.close()
plot_b(data)
```

(c) Code:

```
def part3_c(data):
    data_fft = np.fft.fft(data)
    print("shape of transformed data: ", data_fft.shape)
    print(data_fft[0])
    x = [i for i in range(data_fft.shape[0])]
    plt.plot(x, np.absolute(data_fft))
    plt.title("FFT")
    plt.xlabel("Time")
    plt.ylabel("Fourier Transform Magnitude")
    plt.savefig("3c.png", fomrat = 'png')
    plt.close()
part3_c(data)
```

分析：通过查看 Laurel Yanny 剪辑的上述傅里叶变换，在图的每一侧都有一组高峰和一组低峰，这与是否听到 Laurel 或 Yanny 有关。其中一组峰值代表 Yanny，对这些频率更敏感的人可能会听到。

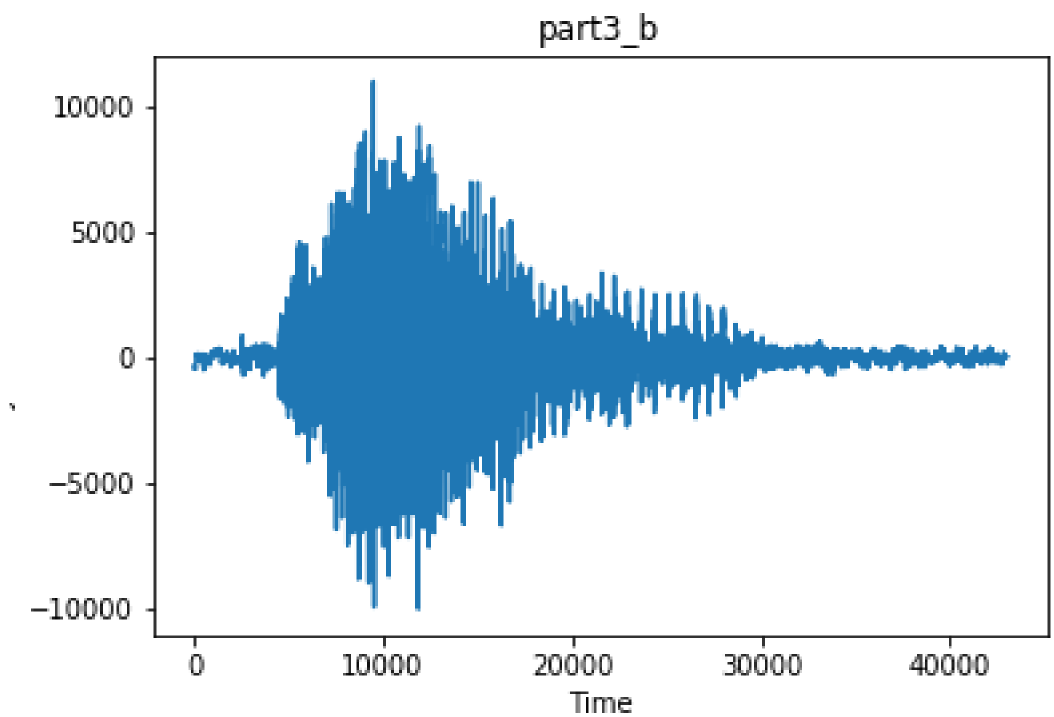

图 2: wav to array

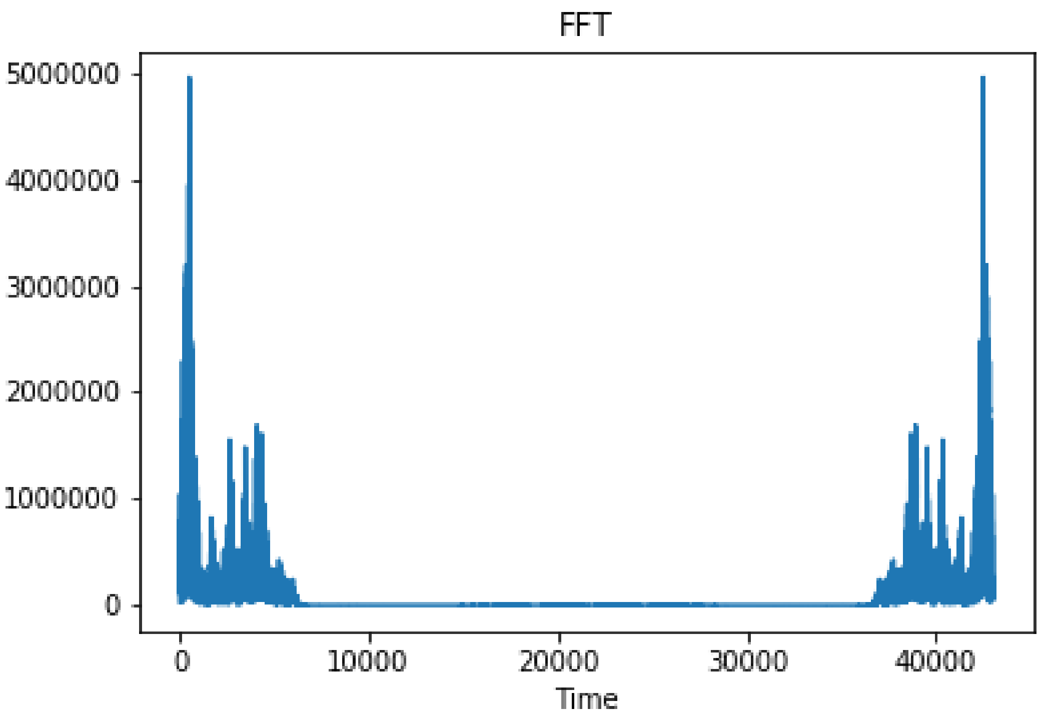
FFT
5000000
4000000
3000000
2000000
1000000
0
0
10000
20000
30000
40000
Time

图 3: FFT

(d) Code:

```
def part3_d(data):
  blocks = 500
  max_feq = 80
  total_chunks = data.shape[0] // blocks
  fourier_matrix = np.zeros((total_chunks, max_feq))
  for i in range(total_chunks):
    current_chunk = data[i * blocks : (i + 1) * blocks]
    curr_fourier = np.fft.fft(current_chunk)
    curr_fourier = np.absolute(curr_fourier)
    fourier_matrix[i, ] = curr_fourier[:80]
  fourier_matrix = np.sqrt(fourier_matrix)
  plt.imshow(fourier_matrix, cmap = 'hot')
  plt.xlabel("Chunk Index")
  plt.ylabel("Fourier Coefficients")
  plt.title("paert3_d")
  plt.savefig('p3_d.png', format = 'png')
  plt.close()
part3_d(data)
```

生成结果如下页图

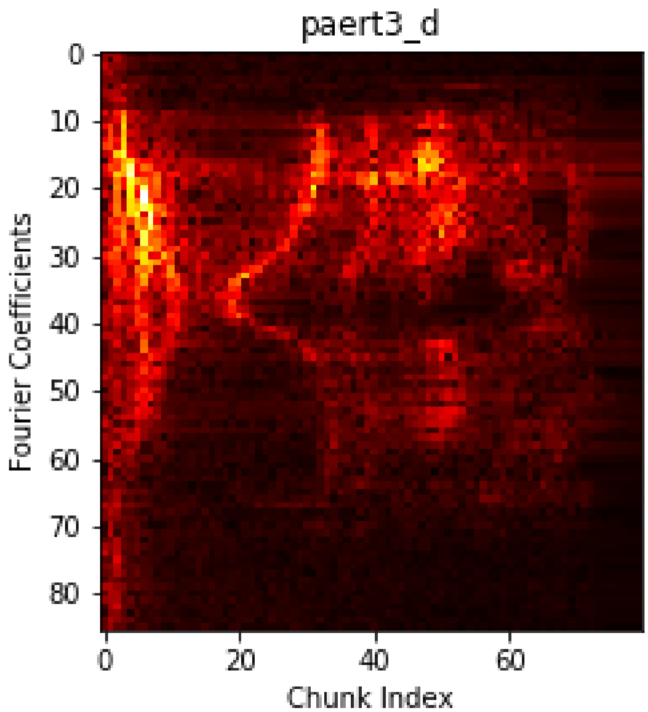
paert3_d
Fourier Coefficients
0
10
20
30
40
50
60
70
80
0
20
40
60
Chunk Index

图 4: part3d

(e) Code:

```
#e
def part3_e1(data, threshold, low = False):
  data = (np.absolute(data) * 1.0 / np.max(np.absolute(data)) * 42000).
                                                       astype(np.int16)
  with open(str(threshold) + ".wav", "wb") as f:
    sample_rate = sampleRate
    if low:
      sample_rate *= 1.3
    sample_rate = int(sample_rate)
    wavfile.write(f, sample_rate, data)
def part3_e2(data):
    thresholds = [40000]
    transformed_data = np.fft.fft(data)
    for threshold in thresholds:
      high = transformed_data.copy()
      high[:threshold] = 0
      high[:43008 - threshold] = 0
      low = transformed_data.copy()
      low[threshold:] = 0
      low[:43008 - threshold] = 0
      high_fft = np.fft.ifft(high)
      low_fft = np.fft.ifft(low)
      part3_e1(high_fft, "bigger" + str(threshold))
      part3_e1(low_fft, "smaller" + str(threshold), low = True)
part3_e2(data)
```

结论：找到一个清洗明确的分割阈值为 40000，分割后的音频放置文件夹中。