CS168 Spring Assignment 8

SUNet ID(s): 05794739
Name(s): Luis A. Perez
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Part 1

(a)
(a) Figure 1 on the LHS maps to Figure 5 on the RHS. Multiplying the fourier matrix M by 1 just means we sum elements in each row. For all rows except the first, the summation cancels out.

(b) Figure 2 on the LHS maps to Figure 4 on the RHS. Multiplying the fourier matrix M by the unit vector simply extract the corresponding column.

(c) Figure 3 on the LHS maps to Figure 1 on the RHS. This is becausee we're summing multiple columns of our fourier matrix, and the first and last column nearly cancel out.

(d) Fugure 4 on the LHS maps to Figure 7 on the RHS. This is because the figure is composed of just one frequent wave that goes from -1, 1, -1, 1, -1, ... and so on.

(e) Figure 5 on the LHS maps to Figure 3 on the RHS. This is because it's simply the sum of all of our frequencies, which lead to a stair-case pattern.

(f) Figure 6 on the LHS maps to Figure 6 on the RHS. This is because the fourier transform of a gaussian is a gausian.

(g) Figure 7 on the LHS maps to Figure 2 on the RHS. The distribution of elements is fairly complex, so it must consists of a combination of many different frequencies.

Part 2

(a) This corresponds to the probability of the event that we after rolling a six-sided die 100 times, the sum of the results is equal to precisely 250.

We can derive this directly. Consider simply the case $p * p$. In this situation, after convolving, we end with a vector of size 11. The first entry simply corresponds to the probability of rollowing two 1s. The second entry to the probability of rolling either a $1, 2$ or a $2, 1$. The third entry to the probability of rollowing a $1, 3$, $3, 1$, or $2, 2$ (sum to 4). More generally, the i-th entry corresponds to the coefficient of x^i, which corresponds to the probability of rolling 2 numbers such that they sum to $i + 2$.

Repeating this proceess n times means that the i-th entry of the resulting vector corresponds to the coefficient of the x^i term which corresponds to the probability of rolling n numbers such that they sum to $i + n$.

(b) Consider the k-th index. We show that they are equivalent for the LHS and RHS. Let us being with the LHS. We have:

$$\begin{aligned}
\mathcal{F}(\mathbf{f} * \mathbf{g})[k] &= \sum_{j=0}^{2N-1} (\mathbf{f} * \mathbf{g})[j] e^{-\frac{\pi i}{N} kj} && \text{(Definition of discrete fourier transform)} \\
&= \sum_{j=0}^{2N-1} \sum_{m=0}^{N-1} \mathbf{f}[m]\mathbf{g}[j-m] e^{-\frac{\pi i}{N} kj}
\end{aligned}$$

For the RHS, we have:

$$\begin{aligned}
(\mathcal{F}\mathbf{f}^+ \cdot \mathcal{F}\mathbf{g}^+)[k] &= \left(\sum_{j=0}^{2N-1} \mathbf{f}^+[j] e^{-\frac{\pi i}{N} kj} \right) \cdot \left(\sum_{j=0}^{2N-1} \mathbf{g}^+[j] e^{-\frac{\pi i}{N} kj} \right) \\
&= \left(\sum_{j=0}^{N-1} \mathbf{f}[j] e^{-\frac{\pi i}{N} kj} \right) \cdot \left(\sum_{j=0}^{N-1} \mathbf{g}[j] e^{-\frac{\pi i}{N} kj} \right) && \text{(Definition of } ^+\text{)} \\
&= \sum_{m=0}^{2N-1} \sum_{j=0}^{N-1} \mathbf{f}[j]\mathbf{g}[m-j] e^{-\frac{\pi i}{N} kj} && \text{(Definition of multiplication)}
\end{aligned}$$

The above verifes that the LHS is equal to the RHS. With this confirmation, we can implement a convolution by taking the inverse fourier transform of both sides. We'll have:

$$\mathbf{f} * \mathbf{g} = \mathcal{F}^{-1}(F\mathbf{f}^+ \cdot \mathcal{F}\mathbf{g}^+)$$

If the two tuples have different lengths, since the convolution operation is translation invariant, we can simply shift the smaller vector (WLOG, assume this to be $\mathbf{f}$) such that it is the same length as the larger vector. Then we can apply the same approach as before.

(c) We use the following code:

```
from scipy import fft

def convolve(x: List[int], y:List[int]):
    """Compute x*y.

    Only accepts real-valued x,y.
    """
    m, l = len(x), len(y)
    n = max(m, l)
    x = x + [0] * (n + max(n - m, 0))
    y = y + [0] * (n + max(n - l, 0))
    assert len(x) == 2*n
    assert len(y) == 2*n
    return np.rint(np.real(fft.ifft(fft.fft(x) * fft.fft(y))[:m + l - 1])).astype(i


def multiply(x: List[int], y: List[int]):
    """Multiplies x and y.

    Args:
        x: A list of digits. Lower indeces represent lower digits.
        y: Same as ax.

    Returns:
        A list of the same format representing the product x*y.
    """
    # Convolve the two using FFT.
    product = convolve(x, y)
    # Limit values to just be single digit.
    carry = 0
    fixed_product = []
    for val in product:
        digit = (val + carry) % 10
        carry = (val + carry) // 10
        fixed_product.append(int(np.rint(digit)))

    while carry > 0:
        digit = carry % 10
        carry = carry // 10
        fixed_product.append(int(np.rint(digit)))
    return fixed_product
```

```python
def to_list(x: str) -> List[int]:
    return [int(char) for char in reversed(x)]

def from_list(x: List[int]) -> str:
    return "".join([str(y) for y in reversed(x)])

def problem2c():
    x = "12345678901234567890"
    y = "987654321098765432109876543210"
    print(f"{x} x {y} = {from_list(multiply(to_list(x), to_list(y)))}")
```

We have the following results:

12345678901234567890 x 987654321098765432109876543210 = 1219326311370217952249657064223744

(d) The naive grade-school multiplication algorithm takes $O(n^2)$ time while the algorithm used above takes only $O(n \log n)$ time since the most expensive operation is the fast fourier transform (and its inverse). The element-wise product and processing are all linear time.

(e) TODO.

Part 2

(a) I hear "Laurel".

(b) The requested plot is available in Figure 1.

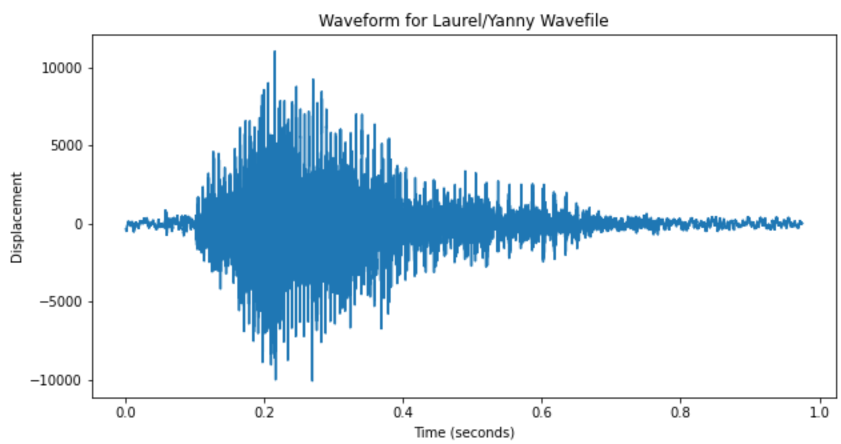

Figure 1: Laurel/Yanny Audio Waveform over Time

(c) The requested plot is available in Figure 2. We see that this plot has two mode – one around the low frequency range and one along the high frequency range. This is expected because these frequencies are just opposite of each other (inverse relation), so we they both correspond to the human audiable range.

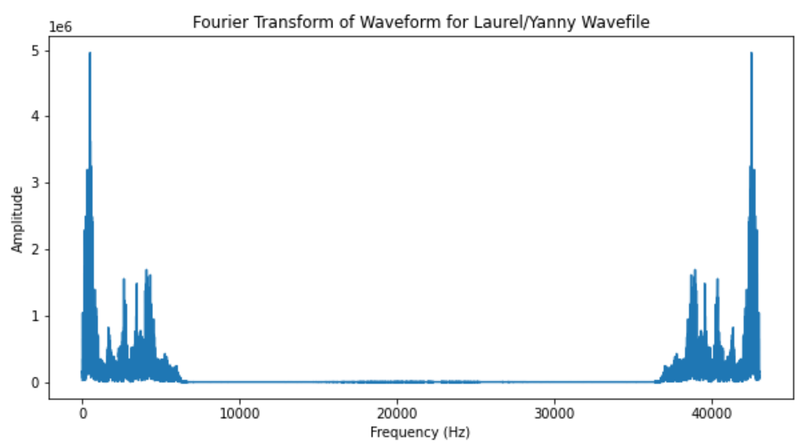

Figure 2: Fourier Transform of Laurel/Yanny Audio Waveform

(d) The requested spectogram is available in Figure 3.

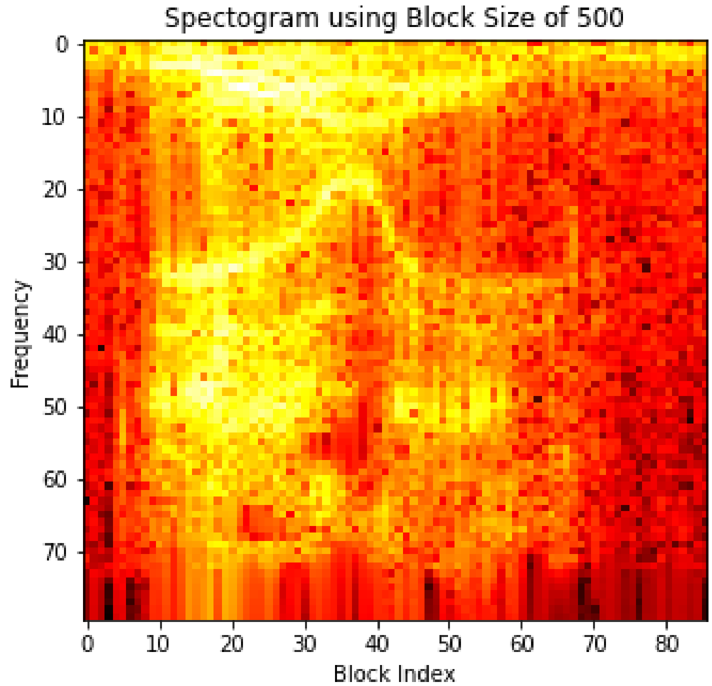

Figure 3: Laurel/Yanny Audio Spectogram

(e) We took the approach of zeroing out the low and high frequencies (on both ends of the polynomial, after performing a FFT). However, no matter the thresholds we explored, we always heard “Laurel” in both versions.

(f) It was difficult to hear a difference, but it appears that for the more slowed down versions (factors in 0.25 and 0.5), a clear initial y sounds was introduced into the audio.

HW8

May 30, 2020

1 CS 168 Spring Assignment 8

SUNet ID(s): 05794739

Name(s): Luis A. Perez

Collaborators: None

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

2 Imports

[1]:
```
import collections
import matplotlib.pyplot as plt
import scipy

import numpy as np
from PIL import Image
from sklearn import decomposition
import pandas as pd
import seaborn as sns
import os
import warnings

import IPython

from typing import Dict, List, Text, Tuple

# Make figure larger
plt.rcParams['figure.figsize'] = [10, 5]

# Set numpy seed for consistent results.
np.random.seed(1)
```

[2]:
```
class Globals:
    DATA_PATH = 'data/'
```

3 Part 2

3.1 Part 2c

[144]:
```
from scipy import fft

def convolve(x: List[int], y:List[int]):
    """Compute x*y.

    Only accepts real-valued x,y.
    """
    m, l = len(x), len(y)
    n = max(m, l)
    x = x + [0] * (n + max(n - m, 0))
    y = y + [0] * (n + max(n - l, 0))
    assert len(x) == 2*n
    assert len(y) == 2*n
    return np.rint(np.real(fft.ifft(fft.fft(x) * fft.fft(y))[:m + l - 1])).
 ↪astype(int)


def multiply(x: List[int], y: List[int]):
    """Multiplies x and y.

    Args:
        x: A list of digits. Lower indeces represent lower digits.
        y: Same as ax.

    Returns:
        A list of the same format representing the product x*y.
    """
    # Convolve the two using FFT.
    product = convolve(x, y)
    # Limit values to just be single digit.
    carry = 0
    fixed_product = []
    for val in product:
        digit = (val + carry) % 10
        carry = (val + carry) // 10
        fixed_product.append(int(np.rint(digit)))

    while carry > 0:
        digit = carry % 10
        carry = carry // 10
        fixed_product.append(int(np.rint(digit)))
    return fixed_product

def to_list(x: str) -> List[int]:
```

```
    return [int(char) for char in reversed(x)]

def from_list(x: List[int]) -> str:
    return "".join([str(y) for y in reversed(x)])
```

[145]:
```
def problem2c():
    x = "12345678901234567890"
    y = "987654321098765432109876543210"
    print(f"{x} x {y} = {from_list(multiply(to_list(x), to_list(y)))}")
```

[146]:
```
problem2c()
```

```
12345678901234567890 x 987654321098765432109876543210 =
121932631137021795224965706422374638011112635269 00
```

4 Problem 3

[163]:
```
from scipy.io import wavfile
```

[164]:
```
def load_wav():
    with open(os.path.join(Globals.DATA_PATH, 'laurel_yanny.wav'), 'rb') as f:
        sampleRate, data = wavfile.read(f)
    return sampleRate, data
```

[165]:
```
def save_wav(sampleRate, data, outfile='output.wav'):
    data = (data * 1.0 / np.max(np.abs(data))*32767).astype(np.int16)
    with open(os.path.join(Globals.DATA_PATH, outfile), 'wb') as f:
        wavfile.write(f, sampleRate, data)
```

[167]:
```
# Test.
save_wav(*load_wav())
```

4.1 Problem 3b

[168]:
```
def problem3b():
    sampleRate, data = load_wav()
    plt.title('Waveform for Laurel/Yanny Wavefile')
    plt.xlabel('Time (seconds)')
    plt.ylabel('Displacement')
    plt.plot(np.arange(len(data)) / sampleRate, data)
    plt.savefig('figures/laurel_yanny_waveform.png', format='png')
    plt.close()
```

[169]:
```
problem3b()
```

4.2 Problem 3c

[171]:
```
def problem3c():
    sampleRate, data = load_wav()
    data = fft.fft(data)

    plt.title('Fourier Transform of Waveform for Laurel/Yanny Wavefile')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Amplitude')
    # Only plot magnitudes of values.
    plt.plot(np.arange(len(data)), np.abs(data))
    plt.savefig('figures/laurel_yanny_waveform_fft.png', format='png')
    plt.close()
```

[172]:
```
problem3c()
```

4.3 Problem 3d

[207]:
```
def problem3d():
    block_size = 500
    max_freq = 80
    sampleRate, data = load_wav()
    heatmap = []
    # Ignore the last chunk.
    for i in range(0, len(data) - block_size, block_size):
        block = data[i:min(i+block_size, len(data))]
        transform = np.abs(fft.fft(block))
        heatmap.append(transform[:max_freq])
    print(f'We have {len(heatmap)} chunks.')
    heatmap = np.stack(tuple(heatmap)).T
    plt.imshow(np.log(heatmap), cmap='hot')
    plt.title(f'Spectogram using Block Size of {block_size}')
    plt.ylabel('Frequency')
    plt.xlabel('Block Index')
    plt.savefig('figures/spectogram.png', format='png')
    plt.close()
```

[208]:
```
x = problem3d()
```

```
We have 86 chunks.
```

4.4 Problem 3d

[222]:
```
def problem3e():
    sampleRate, data = load_wav()

    for t in range(100, 5000, 100):
        transform = fft.fft(data)
```

```
        low_only, high_only = transform.copy(), transform.copy()
        low_mask = (np.arange(len(low_only)) < t) | (np.arange(len(low_only)) >␣
↪(len(low_only) - t))
        low_only[~low_mask] = 0
        high_only[low_mask] = 0
        low_only = fft.ifft(low_only)
        high_only = fft.ifft(high_only)

        save_wav(sampleRate, low_only, outfile=f'[T={t}]low_only.wav')
        save_wav(sampleRate, high_only, outfile=f'[T={t}]high_only.wav')
```

[223]:
```
problem3e()
```

```
/Users/nautilik/.virtualenvs/cs168/lib/python3.7/site-
packages/ipykernel_launcher.py:2: ComplexWarning: Casting complex values to real
discards the imaginary part
```

4.5 Problem 3g

[252]:
```
def problem3g():
    sampleRate, data = load_wav()
    for scale in np.arange(0.25, 2.25, 0.05):
        scaled_sample = int(scale * sampleRate)
        save_wav(scaled_sample, data, outfile=f'[Rate={scaled_sample}]_output.
↪wav')
```

[253]:
```
problem3g()
```

[ ]:
```

```