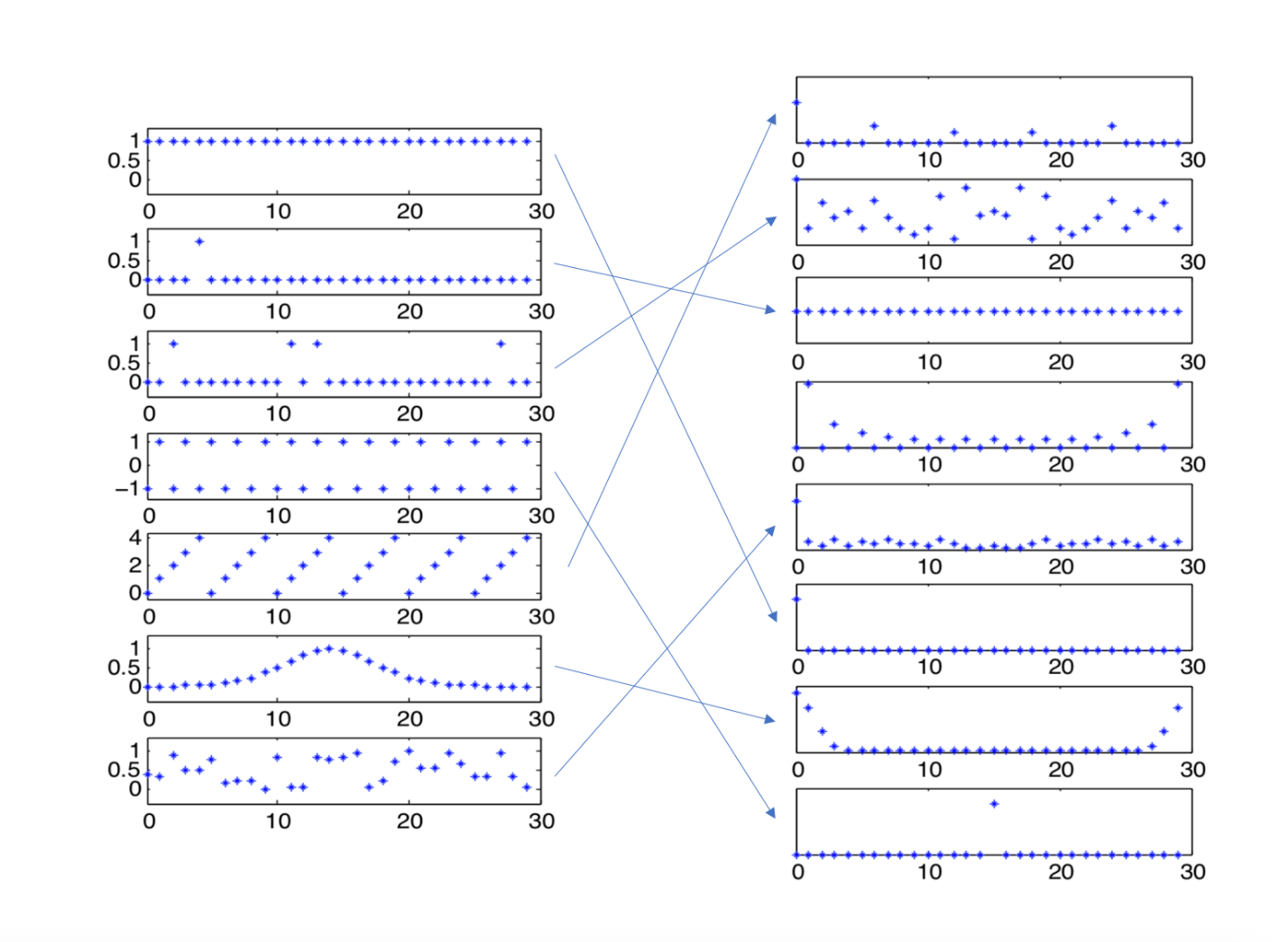
CS168 Miniproject 8

Kaylee Bement, Henry Lin

Starting from the top left discrete vector and moving downward:

Vector 1: The Fourier Transform of a constant term is a Dirac Delta function. Here the left is a constant, horizontal set of points while the right is a Dirac Delta function.

Vector 2: The Fourier Transform of a Dirac Delta function is a constant term. It is the reverse of the previous transformation.

Vector 3: We can see that the left vector represents several different frequencies that are put together. When we take the Fourier Transform of this, we should see a complicated wave consisting of 4 different frequencies, which is like the vector we chose (2).

Vector 4: We can see that the left vector has a consistent frequency of 2, and we can match it to a Fourier Transform that represents that.

Vector 5: The left vector closely represents a sawtooth wave except in discrete form. A sawtooth wave can be broken down into many different waves (4-5 in this case), which is what the matched vector represents.

Vector 6: The Fourier Transform of a Gaussian distribution is also a Gaussian distribution where you must put the sides together. This is shown as the left vector is a Gaussian and the right is also a Gaussian.

Vector 7: The left vector appears to be noise around a constant value between 0 and 1. The Fourier Transform of this would be a high first coefficient and a stream of small coefficients afterwards.

1. A. With probability q[150], the values of 100 die rolls will sum to 250.

To see this, recall that p[i] denotes the probability that i + 1 appears, or that 1 roll sums to i + 1. Thus, p \* p[i] is the probability that, given 2 rolls, the sum is i + 2, which makes sense since there will be (6 + 6 – 1) = 11 terms representing the possibility of rolling the values between 2 and 12, inclusive. Following this pattern q[i] is the probability that, given 100 rolls, the sum is i + 100. Therefore, q[150] = probability of 100 rolls summing to 150 + 100 = 250.

B.

To show that F(f \* g) = Ff+ · Fg+, we will focus on showing that F(f \* g)[m] = (Ff+ · Fg+)[m] for any index 0 ≤ m ≤ 2N – 1.

First, we will rewrite the right side of the equation using the definition of a Fourier Transform for a vector v of length n; c[m] = , where c is a vector of the Fourier Transform coefficients.

Next, because f+[a] and g+[a] are equal to 0 for N ≤ a ≤ 2N – 1, we can change the upper bound of both summations to N – 1.

Now we will multiply the summations together.

Next, let l = j + k. We will use l to make our equation look closer to the Fourier Transform of a convolution.

When l – j < 0, g+[l – j] = 0, meaning that the entire term will equal 0. Thus, we can shift l = j in the inner summation to l = 0 without affecting the summation. Further, because g+[a] is equal to 0 for N ≤ a, we can shift the upper bound of the same summation from j + N – 1 to 2N – 1.

Now we can move the summations and move any terms that don’t rely on j out of the j summation.

Since our inner summation only ranges from 0 to N-1, f+ can be changed to f and g+ can be changed to g.

For the left side of the equation, we are computing F(f \* g)[m]. Recall our definition of a Fourier Transform above for a vector v of length n; c[m] = , where c is a vector of the Fourier Transform coefficients. For F(f\*g), we are dealing with 2N elements, therefore the equation will be

From our definition of (f\*g)[l], we can modify this equation to

Thus, we have

Since this is true for any element m, F(f\*g) = Ff+ · Fg+.

-x-

An implementation of convolution of f and g using the Fast Fourier Transform: F-1(Ff+ · Fg+), where f+ and g+ are 2N-tuples obtained by padding f and g with zeros, and · denotes element-wise multiplication.

If the two tuples have different lengths, the shorter one can be padded with extra zeros.

C.

12193263113702179522496570642237463801111263526900

code:

def pad\_arrays(x, y):

x\_len = len(x)

y\_len = len(y)

x\_padded = x

y\_padded = y

if x\_len < y\_len:

x\_padded += [0 for i in range(y\_len - x\_len)]

elif y\_len < x\_len:

y\_padded += [0 for i in range(x\_len - y\_len)]

x\_padded += [0 for i in range(len(x\_padded))]

y\_padded += [0 for i in range(len(y\_padded))]

return x\_padded, y\_padded

def multiply(x, y):

x\_padded, y\_padded = pad\_arrays(x, y)

x\_fft = np.fft.fft(x\_padded)

y\_fft = np.fft.fft(y\_padded)

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))

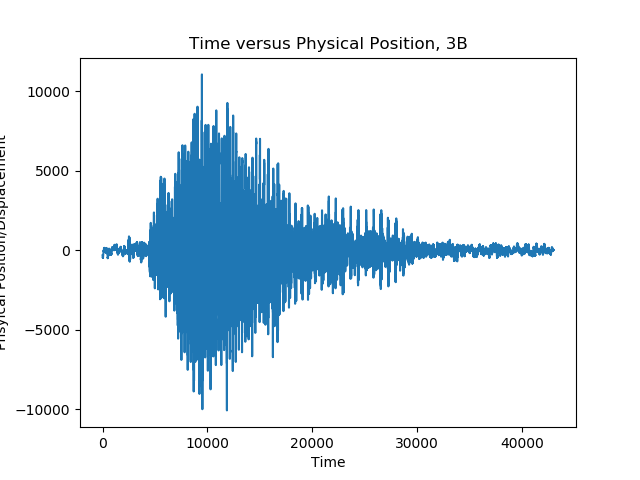
D.

As stated in the notes, the calculation of the convolution using this method takes O(nlogn) time, where n is the number of digits in the longest number. However, using the grade-school integer multiplication algorithm, this would take O(nm) time, where n and m are the number of digits in the each number. Therefore, the FFT method is much faster.

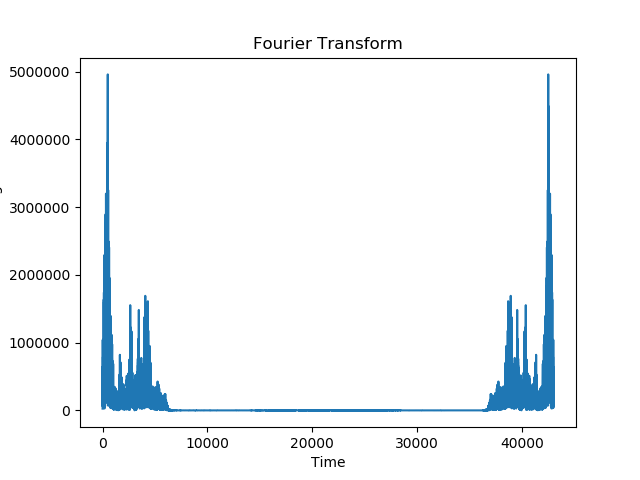
E.

1. A.

B.

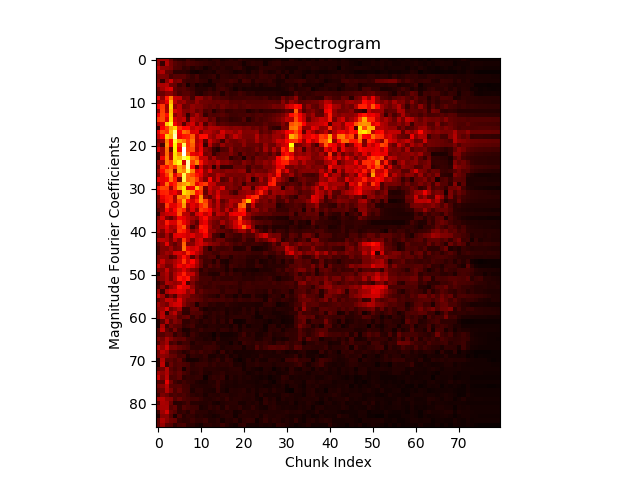


C.



From looking at the above Fourier Transform of the Laurel/Yanny clip, it seems like there is a grouping of high peaks and a grouping of low peaks on each side of the graph that most likely correspond to whether or not you hear Laurel or Yanny. The high peaks represent Yanny, so people more sensitive to those frequencies will likely hear that while the higher frequencies represent Laurel. This type of graph is expected since there is a large debate about which sound people hear.

D.



E. After trying several frequencies, we found that T = 42000 was the best choice for separating Yarry and Laurel. The Yarry audio was very clear and high pitched with this thresholding transform. On the other hand, the Laurel audio was low and much more like a murmur.

F.