# Numerical Methods

**Math 3338 – Spring 2022**

## Worksheet 26

## Fast Fourier Transforms

## 1 Reading

| | |
|---|---|
| CP | 7.2, 7.3, 7.4 |
| NMEP | - |

Table 1: Sections Covered

## 2 Motivation

The discrete cosine transform is an awesome tool to find the frequencies of incoming signals. The major disadvantage is the number of computations required to execute the transform. This causes the computation to be extremely slow, and sometimes you gotta go fast[1].

This leads to the Fast Fourier Transform, or FFT for short. The book gives an explanation for how this works, I highly recommend reading through and understanding it. The FFT is, essentially, the core of modern compression algorithms. MP3 finds the frequencies of music and discards "irrelevant" frequencies, the irrelevant part is sophisticated. JPEG does the same with images, break the image into rectangles, and apply FFT to each rectangle dropping the "irrelevant" data. MP4 is similar to JPEG expect on videos.

The FFT will be responsible for breaking modern cryptography once quantum computers become a thing. In a very simplified way, modern cryptography requires factoring large numbers. There exists an algorithm, Shor's algorithm, that can do this in polynomial time using a quantum computer. The key part is a Fourier transform that is only fast on a quantum computer.

## 3 Numpy FFT

This is built into numpy. The funcitons are contained in `numpy.fft`. Here are the important ones,

1. `rfft` The real Fourier transform
2. `irfft` The inverse real Fourier transform
3. `rfft2` The real Fourier transform on 2 dimensions
4. `irfft2` The inverse real Fourier transform on 2 dimensions

There are further FFTs for $N$ dimensions.

## 4 Looking Ahead on these Transforms

We'll be moving on from here, but the big question is "Why cosines?" The answer is it doesn't need to be. A current area of active research is Wavelets. These offer more accurate approximations using fewer "frequencies". I highly recommend going down the Wikipedia rabbit hole, it is fascinating and I *believe* JPEG uses stuff like this.

[1] `https://www.youtube.com/watch?v=ZZNwTMBNULY`

# Numerical Methods

### Math 3338 – Spring 2022

## Homework 26 (Due: Thursday, April 21)

Include all graphs in your write up of the problems.

**Problem 1 (1 pt)** Music! Let's play with music. For this problem you'll need a WAV file. I've provided one on Canvas, it's "I'll follow you into the dark" by Death Cab for Cutie. Here are my imports and how I load the file,

```
import matplotlib.pyplot as plt
import numpy as np
import wave
from numpy.fft import rfft,irfft,fftfreq
from scipy.io.wavfile import write

spf = wave.open("dark.wav")

signal = spf.readframes(-1)
signal = np.frombuffer(signal, "Int16")
fs = spf.getframerate()
```

With this we can plot the waveform,

```
fig,ax = plt.subplots(nrows=2,figsize = (10,4))

original_time = np.linspace(0, len(signal) / fs, num=len(signal))

ax[0].plot(original_time, signal)
```

To save the WAV file,

```
write("music/test.wav",int(fs*(len(song)/len(signal))),song)
```

where `song` is the result of irfft.

Here is what you should do,

1. Slice the sample data to only include a few seconds of the song. I used 30 to 32ish seconds. (this is optional, but it gives better results)
2. Take the rfft.
3. Do some stuff.
4. Plot the irfft on top of the original data, you may want an alpha value (.5 is good).
5. Plot the frequencies on the lower graph. You'll have to think about what the $x$-values should be.
6. Resave the WAV file and listen to it.

The do some stuff is the fun part. Try experimenting with the following,

1. Set all the frequencies with an amplitude smaller (take the absolute value) than some threshold equal to 0.
2. Set all the higher frequencies equal to zero.
3. Get creative

Document what stuff you do and how it affects the final audio of the song (you'll need to listen to the output).

**Problem 2 (1 pt)** Images! This problem will focus on a greyscale image, but it *should* work for any type of image, you many need to use a higher dimension Fourier Transform. I put an image of pretty bird on Canvas. Here is how I load the image, convert it to greyscale and make it a numpy array.

```
from PIL import Image
import numpy as np
from numpy.fft import rfft2,irfft2

# load the image
image = Image.open('images/bird.png').convert('L')

# convert image to numpy array
data = np.asarray(image)

fourier = rfft2(data)

#Magic code :)

im_data = np.asarray(irfft2(fourier),dtype="uint8")


# create Pillow image
image2 = Image.fromarray(im_data)
print(type(image2))

image2.save("images/new_bird.png")
```

In the "Magic Code" spot you can do a bunch of stuff. Try dropping certain frequencies, dropping small contributors, or something interesting.

Document what you do and comment on the results.

**Problem 3 (1 pt)** JPEG is a similar process to the above, except the first step is break the image into a grid, perform the above process (FFT and dropping frequencies) on each grid, and reconstructing each piece. Try breaking the bird image into a $3 \times 3$ grid and do stuff to it. You should be able to see the boundaries when you reconstruct. This problem may be challenging. Good luck.