Assignment 2: Fast Fourier Transform and Applications

1 Introduction

In this assignment you will implement two versions of the Discrete Fourier Transform (DFT). One of them will be a brute force approach that follows directly from the formula. The second one will be an implementation of the Fast Fourier Transform (FFT) which follows a divide-and-conquer approach to algorithm design. In particular, the FFT we will focus on is the Cooley-Tukey FFT.

At the end of this lab you should know how to:

1. Implement two types of DFT (naïve and FFT);
2. Understand why one version of the algorithm is better than the other;
3. Extend the FFT algorithm to multidimensional case (2-dimensions in this lab);
4. Use FFT to denoise an image.
5. Use FFT to compress an image file.

2 Background

2.1 The Discrete Fourier Transform

The Fourier transform of a signal decomposes it into a sum of its various frequency components. However, in the real world a lot of the signals we deal with are either discrete or, even if they are continuous, can only be approximated via sampling which gives a discretized approximation of the original signal. To address this, in class we saw how the continuous Fourier transform can be modified to obtain a discrete version. Recall the DFT and its inverse shown below respectively.

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi}{N}kn}, \qquad \text{for } k = 0, 1, 2, \ldots, N-1. \tag{1}$$

$$x_n = \frac{1}{N} \sum_{n=0}^{N-1} X_k e^{\frac{i2\pi}{N}kn}, \qquad \text{for } k = 0, 1, 2, \ldots, N-1. \tag{2}$$

For example, if we have a vector of size 10 and we apply equation (1) we will get another vector of size 10 at the output representing the DFT where the element of the output at index k will be the result of them sum X_k.

2.2 The Cooley-Tukey FFT

In 1965 Cooley and Tukey came up with their own algorithm for X_k that takes advantage of the symmetry of the FT to simplify the algorithmic complexity. The algorithm fits into the divide-and-conquer design paradigm that you are familiar with from e.g. mergesort.

Below we give a brief derivation for your understanding.

$$
\begin{aligned}
X_k &= \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi}{N}kn} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{\frac{-i2\pi}{N}k(2m)} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{\frac{-i2\pi}{N}k(2m+1)} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{\frac{-i2\pi}{N}k(2m)} + e^{\frac{-i2\pi}{N}k} \sum_{m=0}^{N/2-1} x_{2m+1} e^{\frac{-i2\pi}{N}k(2m)} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{\frac{-i2\pi}{N/2}k(m)} + e^{\frac{-i2\pi}{N}k} \sum_{m=0}^{N/2-1} x_{2m+1} e^{\frac{-i2\pi}{N/2}k(m)} \\
&= X_{even} + e^{\frac{-i2\pi}{N}k} X_{odd}
\end{aligned}
$$

The proof works as follows:

- Split the sum in the even and odd indices which we sum separately and then put together.
- Factor out a constant by reducing the exponent by one in the second term. This makes the coefficients the same for both sums so we can have them saved and precomputed in an array.
- Observe that multiplying by 2 is the same as dividing by a half.
- Finally, notice that the two terms are just a DFT of the even and odd indices which can be computed as separate problems and then finally summed.

The algorithm works by successively splitting the problem in halves a shown above. Convince yourselves that this can be done as many times as we want by just applying the same technique to the even and odd halves to split the problem to 4-subproblems, then 8, then 16…

In practice we would divide a big *x* array of say size 2048 in two equal parts recursively until the smaller parts became small enough to compute with the naïve method without suffering large efficiency penalties. Although we could divide the problem all the way down to sub-problems of size 2 there is usually a threshold beyond which splitting the problem creates more overhead than just computing it using the naïve method. The exact size where you choose to stop splitting the problems does not matter for this assignment as long as the runtime of your FFT is in the same order of magnitude as what is theoretically expected (more on this later).

2.3 The Two-Dimensional Fourier Transform (2DDFT)

The 2D-DFT can be thought of as a composition of separate 1-dimensional DFTs. Looking at the equation we define the 2DDFT for a 2D sample of signal f with N rows and M columns as:

$$F[k,l] = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} f[m,n] e^{\frac{-i2\pi}{M}km} e^{\frac{-i2\pi}{N}ln}$$
$$= \sum_{n=0}^{N-1}\Big(\sum_{m=0}^{M-1} f[m,n] e^{\frac{-i2\pi}{M}km}\Big) e^{\frac{-i2\pi}{N}ln}$$
$$\text{for } k = 0, 1, \ldots, M-1 \text{ and } l = 0, 1, \ldots, N-1.$$

Note that the term inside the brackets is a 1D DFT of the rows of the 2D matrix of values f and that the outer sum is another 1D DFT over the transformed rows performed along each column.

Similarly, we can obtain the inverse 2D transform, by applying multiple 1D inverse FFTs.

$$f[m,n] = \frac{1}{NM}\sum_{l=0}^{N-1}\sum_{k=0}^{M-1} F[k,l] e^{\frac{i2\pi}{M}km} e^{\frac{i2\pi}{N}ln}$$
$$= \frac{1}{NM}\sum_{l=0}^{N-1}\Big(\sum_{k=0}^{M-1} F[k,l] e^{\frac{i2\pi}{M}km}\Big) e^{\frac{i2\pi}{N}ln}$$
$$\text{for } m = 0, 1, \ldots, M-1 \text{ and } n = 0, 1, \ldots, N-1.$$

2.4 Summary of background and pre-requisites

Before proceeding with the rest of this lab document make sure you have accomplished these first steps:

1. You are familiar with basic programming tasks in Python. For this assignment you will need to use nothing more than loops, if statements and a few libraries which include: *numpy* for array manipulation, *matplotlib* for plotting, and optionally *open-cv* to manipulate the assignment's images. All of these libraries will be used at an elementary level which is easy to achieve if you don't know Python as long as you have experience with another programming language and look up the documentation.
2. You have read the course material on the Fourier transform.
3. You are familiar with the basics of the divide-and-conquer architecture from course prerequisites such as COMP-250 and COMP-251. If you don't remember much just review how mergesort works and its big-O complexity.

3 Lab requirements

Write a program using Python (version 3.5+, no Python 2) that:

- Performs DFT both with the naïve algorithm and the FFT algorithm discussed above.
- Performs the inverse operation. For the inverse just worry about the FFT implementation.
- Does the same thing for 2D Fourier Transforms (2d-FFT) and its inverse.
- Plots the resulting 2D DFT on a log scale plot.
- Saves the DFT on a csv file.

For this lab you must use Python. You are not allowed to use any prebuilt FT algorithm, but you may use external libraries that help you load and save data, manipulate images (cv2), plot (matplotlib or plotly) and carry out vectorized operations (numpy).

3.1 Calling syntax

Your application should be named fft.py, and it should be invoked at the command line using the following syntax:

```
python fft.py [-m mode] [-i image]
```

where the argument is defined as follows:

- mode (optional):
  - [1] (Default) for fast mode where the image is converted into its FFT form and displayed
  - [2] for denoising where the image is denoised by applying an FFT, truncating high frequencies and then displayed
  - [3] for compressing and saving the image
  - [4] for plotting the runtime graphs for the report

- image (optional): filename of the image we wish to take the DFT of. (Default: the file name of the image given to you for the assignment)

3.2 Output behavior

Your program's output depends on the mode you are running.

For the first mode your program should simply perform the FFT and output a one by two subplot of the original image and next to it its Fourier transform. The Fourier transform should be log scaled. An easy way to do this with matplotlib is to import *LogNorm* from *matplotlib.colors* to produce a logarithmic colormap.

For the second mode you should also output a one by two subplot. In this subplot you should include the original image next to its denoised version. To denoise all you need to do is to take the FFT of the image and set all the high frequencies to zero before inverting to get back the filtered original. Where you choose to draw the distinction between a "high" and a "low" frequency is up to you to design and tune for to get the best result.
Note: The FFT plot you produce goes from 0 to 2π so any frequency close to zero can be considered low (even frequencies near 2π) since 2π is just zero shifted by a cycle. Your program should print in the command line the number of non-zeros you are using and the fraction they represent of the original Fourier coefficients.

For mode three you need to do two things. Firstly, you must take the FFT of the image to compress it. The compression comes from setting some Fourier coefficients to zero. There are a few ways of doing this, you can threshold the coefficients' magnitude and take only the largest percentile of them. Alternatively, you can keep all the coefficients of very low frequencies as well as a fraction of the largest coefficients from higher frequencies to also filter the image at the same time. You should experiment with various schemes, decide what works best and justify it in your report.
Then your program should do the following: display a 2 by 3 subplot of the image at 6 different compression levels starting from the original image (0% compression) all the way to setting 95% of the coefficients to zero. To obtain the images just inverse transform the modified Fourier coefficients.
Secondly you should save the Fourier transform matrix of coefficients as a sparse matrix (e.g. csr

format). This will be useful for your report as it gives you an idea of how much memory you are saving. Additionally, your program should print in the command line the number of non zeros that are in each of the 6 images.

Finally, for the plotting mode your code should produce plots that summarize the runtime complexity of your algorithms. Your code should print in the command line the means and variances of the runtime of your algorithms versus the problem size. For more qualitative details about what you should plot see the report section.

4 Important dates, deliverables, and evaluation

4.1 Report

The code and the report are due at 23:59 on Mar. 20th, and it will count for 7.5% of your final grade. The split for the grade is 60% code - 40% report. Your report should include the following items:

- The names and McGill ID numbers of both group members
- Design: Describe the design of your algorithms
- Testing: Describe how you tested your algorithms to make sure they behave as required
- Analysis: Briefly explain the runtime of the naïve DFT for 1D. Derive and solve the recurrence for the FFT for 1D to get the runtime complexity. Argue about the 2D complexity of the two algorithms, (you don't need to solve the recurrence again an informal but logically sound argument is enough).
- Experiment:
    - Load the image provided to you and get its FFT transform. Plot it next to the original. Compare your result with the built-in np.fft.fft2 function. Remember to make your FFT plots log-scaled to improve visibility.
    - As explained in the previous section you can use FFT to denoise an image. Try to denoise the image as much as you can. Produce a few denoised versions of the image and explain what procedure gives you the best results (i.e. removing high frequencies or low frequencies, thresholding everything or keeping all the coefficients below/above a certain cutoff and thresholding the rest?)
    - Compression display a 2 by 3 subplot of the image at 6 different compression levels starting from the original image (0% compression) all the way to setting 95% of the coefficients to zero. You should also note the size of the saved sparse matrix files for each image and comment on the quality of the reconstruction.
    - Plotting for runtime: Create 2D arrays of random elements of various sizes (sizes must be square and powers of 2). Start from 2^5 and move up to 2^{10} or more if your computer can handle it. Gather data for the plot by re-running the experiment at least 10 times to obtain an average runtime for each problem size and a standard deviation.
      On your plot you must have problem size on the x axis and runtime in seconds on the y axis. You can plot two lines; one for the naïve method and one for the FFT. Plot your mean runtimes for each method and include error bars proportional to the standard deviation that represent a confidence interval defined by you. (For example, you can make it a 97% confidence interval by making the error bar length be twice the standard deviation).

4.2 Code

When submitting your report on myCourses, you should also upload a zip file containing your code. Make sure to include all source files required to compile and execute your program. Also include a readme file giving any special instructions required to compile your code and mentioning what version of Python you used when writing/testing your program.

5 Additional advice - Important

Before coding make sure you understand the basics of the DFT as well as the basics of python, numpy and matplotlib. If the image you are given does not have a length or width that is a power of 2 you can resize it with cv2 or you can pad it with zeros to the nearest power of 2 side length otherwise the FFT algorithm won't work!

Please remember to start early!