The Fast Fourier Transform
APPM 4600 Project
University of Colorado Boulder
Department of Applied Mathematics

1 Project Summary

In class, we talked about trigonometric interpolation. It turns out that when this interpolation involves uniformly distributed points, the cost of doing the interpolation can be reduced from $O(n^2)$ to $O(nlog(n))$. The technique for doing this is called the fast Fourier transform (FFT) and it exploits the fact that you can write the evaluation as beautifully nested series that all look the same but are scaled by a constant. In this project you will actually derive it and investigate its asymptotic behavior. Next you will learn the non-uniform FFT. What are the challenges of this method? How does it perform relative to the uniform FFT? What are recent developments on these methods?

2 Project Background

Two of the most foundational tools in modern math, physics, and engineering are Fourier series and the resulting Fourier Transformation, which allow for the decomposition of a time domain signal into its individual frequency components. This project will explore how to implement these techniques numerically by first discretizing the Fourier Transformation, achieving the Discrete Fourier Transformation (DFT). Once the method has been derived, we will explore how it's symmetries can be exploited to build a more efficient form of the algorithm, the Fast Fourier Transform (FFT).

2.1 Discrete Fourier Transform (DFT)

2.2 Mathematical background

In 1807, Joseph Fourier proposed that any piece-wise continuous function $h(t)$ could be expanded as an infinite sum of sinusoids. This discovery proved revolutionary, as the theory of the fourier series, and along side it the frequency domain of a continuous signal, allowed for scientists to achieve new feats in solving partial differential equations, signal processing, and many other fields. To state Fourier's discovery more mathematically, for any piece-wise continuous function $f(t)$ on the interval $[-L, L]$, it's periodic extension is given as

$$\tilde{f}(t) = A_0 + \sum_{n=1}^{\infty} A_n cos(nt) + \sum_{n=1}^{\infty} B_n sin(nt). \tag{1}$$

Where the fourier coefficients, the A's and B's, are given by what are referred to as the synthesis equations, given as

$$\begin{aligned} A_0 &= \frac{2}{\pi}\int_{-\pi}^{\pi} f(t)dt \\ A_n &= \frac{1}{\pi}\int_{\pi}^{\pi} f(t)cos(nt)dt \\ B_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)sin(nt)dt \end{aligned} \tag{2}$$

While this form of Fourier's invention is beautiful, it is often not practical, let alone possible, to use all of the infinite terms of the fourier series. Because of this, a signal's fourier series is often truncated after a given number of terms. The form of this truncated series is given below by equation (3)

$$f(t) \approx A_0 + \sum_{n=1}^{N} A_n cos(nt) + \sum_{n=1}^{N} B_n sin(nt). \tag{3}$$

With the advent of the modern computer, the next logical step for mathematicians was to determine a method by which to calculate the fourier coefficients numerically for a discrete signal. The result of this was the discrete fourier transformation (DFT). It is important that the use of the word "transformation" is a misnomer with this method, as the fourier transformation is related to, but not the same as, the fourier series. The DFT computes the truncated fourier series of a discrete signal. The DFT is given to be

$$\hat{f}_k = \sum_{j=0}^{N-1} f_j e^{-2\pi i j/N}. \tag{4}$$

In equation (4), $\hat{f}_k$ is the k^{th} fourier coefficient, N is the total number of samples in the discrete signal being analyzed, f_j is the j^{th} sample of the signal, and i is the imaginary unit, or $\sqrt{-1}$. A relationship between the DFT and the synthesis equations (equation (2)) can be found by applying Euler's identity to the complex exponential term. This will be explored later in this project.

2.3 Questions to Consider

1. Using Euler's identity, $e^{i\theta} = cos(\theta) + isin(\theta)$, show that another form of the fourier series can be written as

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{i\omega}.$$

   Also, derive a relationship between A_n, B_n, and c_n.

2. Using the relationship derived above, explain how $\hat{f}_k$ can be interpreted to give A_n and B_n.

3. Using equation (4) and letting $\omega_n = e^{-2\pi i/N}$construct a general form of a matrix system to solve for each of the discrete fourier coefficients $\hat{f}_i$ given N samples of your signal. This system should be of the form $\hat{f} = Ff$. What are some of the properties of F?

3 The Uniform Fast Fourier Transform (FFT)

3.1 Mathematical Background

Section 2 developed a method for approximating the truncated complex Fourier series for a given signal $f(t)$. However there are several drawbacks to the method explored, primarily that in order to find the fourier coefficients and transform a signal into the frequency domain, we needed to compute a matrix multiplication involving a dense NxN matrix, which has a computational complexity of $O(N^2)$. This can become an issue in many applications where large data sets need to be transformed, so a more efficient method is desirable. One such method is the fast fourier transform (FFT). The FFT is a method that takes advantage of the self symmetries within the DFT matrix system to cut down computational time and reduce the complexity of the DFT to $O(NlogN)$.

To build this optimized method, our signal needs to meet one requirement, the number of times that the signal is sampled must be a power of 2. This requirement can be relaxed with some simple tricks, but that is for later discussion. The demand for this requirement will reveal itself as we come to understand the FFT.

Once an adequate signal has been obtained the next logical step is to apply the fourier transformation to the signal, or in our case, the DFT. This results in the following matrix system

$$\hat{f} = F_{2^n \times 2^n} f.$$

Applying direct matrix multiplication to this sytem results in the DFT, which we have decided is inefficient for solving this problem. To get around this issue, some analysis can be performed on F to determine a matrix factorization that may allow for a less complex matrix multiplication. To do this, the signal vector f can be rearranged as follows

$$f = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{2^n-1} \end{bmatrix} \implies \hat{f} = \begin{bmatrix} f_0 \\ f_2 \\ \vdots \\ f_{2n-2} \\ f_1 \\ f_3 \\ \vdots \\ f_{2n-1} \end{bmatrix} = \begin{bmatrix} f_{even} \\ f_{odd} \end{bmatrix}$$

Once the signal has been rearranged, the new matrix system can be factored as

$$\hat{f} = R\hat{F} = \begin{bmatrix} I_{2^{n-1}} & D_{2^{n-1}} \\ I_{2^{n-1}} & -D_{2^{n-1}} \end{bmatrix} \begin{bmatrix} F_{2^{n-1}} & 0 \\ 0 & F_{2^{n-1}} \end{bmatrix} \begin{bmatrix} f_{even} \\ f_{odd} \end{bmatrix} \tag{5}$$

Where I_N denotes the $N \times N$ identity matrix and D is a diagonal matrix. At this point, the sparsity of both R and $\hat{F}$ can be abused to reduce the time needed to perform this matrix multiplication. But the savings do not stop there! Notice that the two matrices on the diagonal of the block matrix $\hat{F}$ are themselves DFT matrices which are of a size that is a multiple of 2. This means that again, the fast fourier transformation can be applied to the f_{even} and the f_{odd} matrix systems individually to further cut down on computational time, until only a series of 2x2 matrix solves need to be performed to compute the FFT.

3.2 Questions to consider

1. Using equation (4), break the sum into it's respective f_{odd} and f_{even} components. What do you notice about this new form? What does it look like when you use the ω_n substitution from 2.3.3?

2. Using your observations in the previous questions can you write a matrix that will perfrom the FFT? **Hint:** You will need to write a product of two matricies since the FFT uses a matrix factorization.

3. Generate some sample signals of varying lengths, at what point does your signal become so large that the FFT out perfroms the DFT? Does the DFT ever perform faster?

4. Not all signals that we want to process have a number of samples that is equivalent to 2^n. To apply the FFT to these signals it is common practice to pad the end of a signal with zeros until it has a length that is a power of 2. How does this impact the accuracy of the algorithm to correctly approximate these signals?

4 Software Expectations

You may use software for this project. However, you must demonstrate that you understand how the software works, what is does, and any limitations it may face. If you choose to explore the NUFFT we recommend using software.

5 Independent Directions

Possible next steps for this project include, but are not limited to:

1. Non-uniformly sampled FFT algorithms

2. The Niquist Sampling Criterion and how to beat it

3. Applications of the FFT in image compression

4. The discrete short-time fourier transform/gabor transform and the spectrogram

5. The multidimensional FFT and solving the heat equation by applyig it.

6 Helpful Sources

1. Van Loan, Computational frameworks for the Fast Fourier Transform, Frontiers in Applied Mathematics, 1992.

2. Kong, Python Programming and Numerical Methods, Chapter 24 (Fourier Transform)

3. Alok, Rokhlin "Fast Fourier Transforms for Nonequispaced Data". SIAM Journal on Scientific Computing, 1993.

4. Lee, Greengard, "The type 3 nonuniform FFT and its applications". Journal of Computational

Physics, 2005.