Fast Fourier Transformer Visualization

Fatima AlSaadeh
Rutgers University
Piscataway, NJ, USA
Email: fatima.alsaadeh@rutgers.edu

Ashley Dunn
Rutgers University
Piscataway, NJ, USA
Email: ashley.dunn@rutgers.edu

***Abstract*— Fast Fourier transformations are running millions of times a day; the algorithm behind them is essential in helping different fields of science and technology such as data analytics, medical imagery, and compression of images, audio, and video. The divide and conquer design of the fast Fourier transform algorithm made a breakthrough which dropped the complexity of polynomial multiplication from $O(n^2)$ to $O(n \log n)$ [1].**

I. Project Description

This project will include the fast Fourier transformer algorithm visualization and implementation. The visualization will cover how this important arithmetic algorithm works using a divide and conquer method to speed up polynomial multiplication. The main goal behind choosing this algorithm is to develop a deeper understanding of it as well as to highlight its applications. We will start the video with a section covering a mathematical introduction into the discrete Fourier transform, complex numbers, and polynomial arithmetic, all of which will provide the basis for us to introduce the fast Fourier transform. The video will then explain how we can multiply polynomials fast using the divide and conquer algorithm design, and cover the time complexity and the output of this algorithm. After understanding the basic idea of this algorithm we will try to implement it and test it over different data samples to show it in action. Finally we will cover the main important applications of this algorithm. In order to cover all these steps during the given timeline we will be building all parts simultaneously and focus on finishing the main version of all sections before trying to improve and fine tune them. This will ensure that we have a presentable project by the due date.

The project has four stages: Gathering, Design, Infrastructure Implementation, and User Interface.

A. Stage1 - The Requirement Gathering Stage.

The general system description: This system will analyze a given signal by computing the Discrete Fourier Transform to convert it from its given domain to the frequency domain. This process is usually long and slow on large sample sizes. Using the Fast Fourier transform, this application will make this process faster by breaking down the signal sample size into smaller sized samples, recursing on them separately, and then merging the results to produce the final representation of the signal in the frequency domain.

- The three types of users (grouped by their data access/update rights):
- The user's interaction modes: Biomedical Engineers
- The real world scenarios:
  - Scenario1 description: Biomedical engineers can use this system to analyze the electrocardiogram signals' compression to infer the frequency component of the generated periodic signals.
    System Data Input for Scenario1: Electrocardiogram signals.
    Input Data Types for Scenario1: Discrete sample numbers.
    System Data Output for Scenario1: Frequencies of heart rate.
    Output Data Types for Scenario1: Plot frequencies and complex numbers.
  - Scenario2 description: Biomedical engineers can use this system as part of the X-ray analysis to produce distortion free images from parts of the body.
    System Data Input for Scenario2: X-ray Images.
    Input Data Types for Scenario2: Discrete sample numbers.
    System Data Output for Scenario2: Representation of the image pixels in the frequency domain.
    Output Data Types for Scenario2: Plot frequencies and complex numbers.
- The user's interaction modes: Electrical Engineers
- The real world scenarios:
  - Scenario1 description: Circuits signal distortion detection.
    System Data Input for Scenario1: Circuits signals.
    Input Data Types for Scenario1: Discrete sample numbers.
    System Data Output for Scenario1: Frequencies of signals in order to show distortion.
    Output Data Types for Scenario1: Plot frequencies and complex numbers.
  - Scenario2 description: Sending signals through the air.
    System Data Input for Scenario2: signals.
    Input Data Types for Scenario2: Discrete sample numbers.
    System Data Output for Scenario2: Signal frequency domain.
    Output Data Types for Scenario2: Plot frequencies and complex numbers.

- The user's interaction modes: Audio/Music Engineers
- The real world scenarios:
  - Scenario1 description: This system can be used in audio and music detection. The application can be improved to read music signals and identify which notes or songs are being played. **System Data Input for Scenario1:** Audio signals.
    Input Data Types for Scenario1: Discrete sample numbers.
    System Data Output for Scenario1: Frequencies of the audio that can be compared with the stored frequencies of existing songs and notes and return the relation between them.
    Output Data Types for Scenario1: Plot frequencies and complex numbers and note or song.
  - Scenario2 description: This system can be used as part of audio compression to reduce the memory storage without losing the audio quality.
    System Data Input for Scenario2: audio signals.
    Input Data Types for Scenario2: Discrete sample numbers.
    System Data Output for Scenario2: Frequencies of the audio signals.
    Output Data Types for Scenario2: Plot frequencies and complex numbers.
- Project Time line and Division of Labor:
- Week 1 (Nov, 4 - 8):
  - Submit first project proposal. (Both)
  - Decide the tools and programming language. (Both)
  - Define the meeting times and divide the work.(Both)
  - Start the flow diagram of the system.(Fatima)
  - Start the Pseudo Code.(Ashley)
- Week 2 (Nov, 11 - 15):
  - Submit first project report.(Both)
  - Present the first project report.(Both)
  - Complete the flow diagram and Pseudo Code.(Both)
  - Collect data that will be used for testing.(Both)
- Week 3 (Nov, 18 - 22):
  - Start the algorithm visualization video recording.(Ashley)
  - Start the code implementation.(Fatima)
- Week 4 (Nov, 25 - 29):
  - Submit second project report.(Both)
  - Present the second report.(Both)
  - Second iteration over the algorithm visualization.(Fatima)
  - Second iteration over the code implementation.(Ashley)
  - Review and Testing. (Both)
- Week 5 (Dec, 2 - 6):
  - Prepare the final presentation. (Both)
  - Practice the presentation.(Both)
  - Third iteration over the work done. (Both)

B. Stage2 - The Design Stage.

Flow Diagram Description.

The flow diagram in Fig.1 describes the system which uses the recursive divide and conquers strategy for implementing the fast Fourier algorithm. Start with an input signal, which is recorded as Hertz and amplitude. That signal is then converted into a sampler of data points and ω, the sampling frequency, and an nth root of unity. From here, the diagram follows the pseudo-code. The algorithm has time complexity of $O(n \log n)$ and a time complexity of $O(n \log n)$. This is due to the recurrence relation:

$$T(n) = 2T(n/2) + O(n)$$

as the algorithm performs n iterations and has an array of size n each time, and at each recursive step divides the input into two halves. Thus, we can use the Master Theorem.

Flow Diagram.

See Fig. 1 on page 3

High Level Pseudo Code System Description.

The signal-processing function takes in an input signal in terms of amplitude verses time, digitizes it via sampling, and then performs the fast Fourier transform. The FFT function uses a divide and conquer strategy, separating the input array a into the even and odd powers of ω, thus expressing it as:

$$a(x) = a_e(x^2) + x * a_o(x^2)$$

FFT is called on each a_e and a_o portion, and then the results are put together.

Pseudo Code.

Algorithm 1 The fast Fourier transform [1]

Function FFT(a, ω)
Input: array a = [0, 1, . . ., n-1]
Output: ω a primitive nth root of unity

if ω = 0 **then**
  return a
end if

$s_{\text{even}}[0, 1, ..., \frac{n}{2} - 1] = \text{FFT}(a[0, 2, ..., n-2], \omega^2)$
$s_{\text{odd}}[0, 1, ..., \frac{n}{2} - 1] = \text{FFT}(a[1, 3, ..., n-1], \omega^2)$

for $j = 0$ to $\frac{n}{2} - 1$ **do**
  $r_j = s_{\text{even}}[j] + \omega^j s_{\text{odd}}[j]$
  $r_{j+\frac{n}{2}} = s_{\text{even}}[j] - \omega^j s_{\text{odd}}[j]$
end for
return $r[0, 1, ..., n-1]$

Function signal-processing(s_{in})
Input: s_{in} signal in time domain
Output: s_{out} signal in frequency domain

plot(s_{in})
a, ω = sampler(s_{in})
s_{out} = FFT(a, ω)
plot(s_{out})

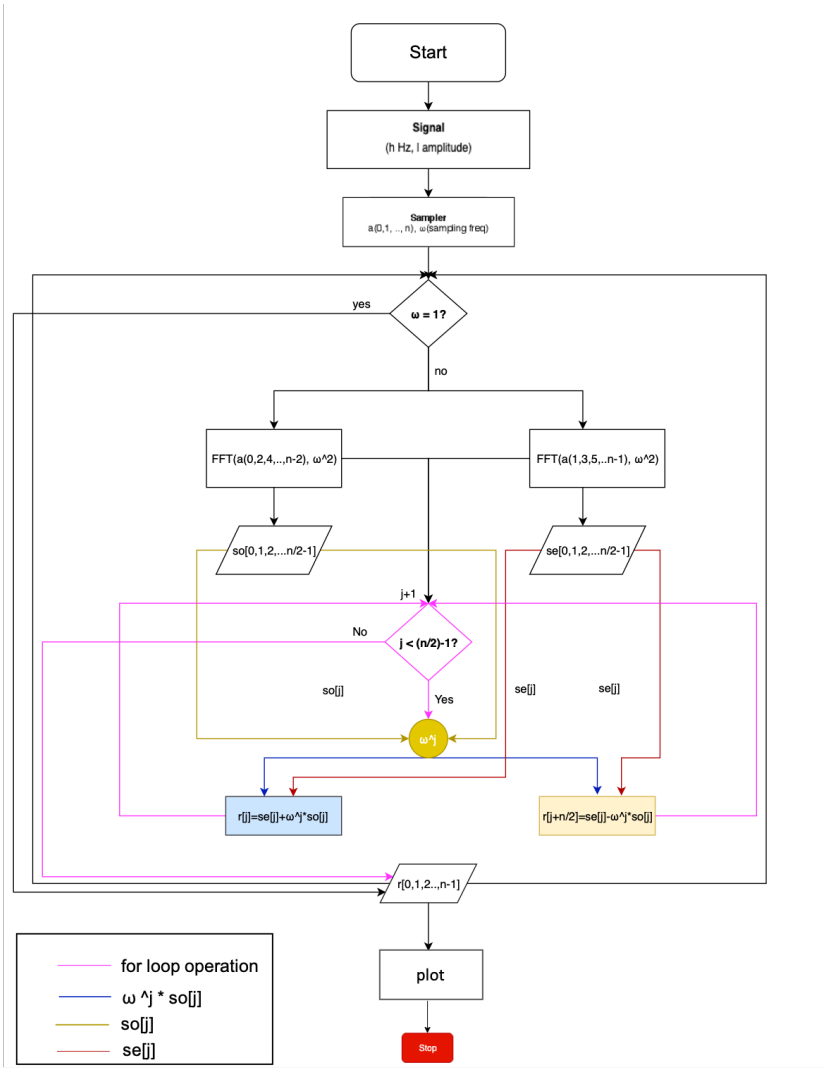

Fig. 1. Flow Diagram

Algorithms and Data Structures.

- Function FFT performs the fast Fourier transform on an input array using a nth root of unity, ω
- Function signal-processing takes the input signal in the form of a function of time, and uses FFT to output a function of frequency.
- Array a contains the sampled data points from the input function to signal-processing

Flow Diagram Major Constraints.

- Input a to the FFT function should be a vector of real numbers
- Input ω to the FFT function should be a complex number whose powers 1, ω , ω^2, ., ω^n-1 are the complex nth roots of unity.

C. Stage3 - The Implementation Stage.

Code

The programming language Python[2] and Pycharm[3] IDE were used to build this project. In order to implement the algorithm, a graphical user interface, voice recording, and converting audio to a wave, we used the following Python libraries: SciPy[4], NumPy[5], Matplotlib[6], wave, pyaudio[7], and tkinter.

The implementation of this FFT algorithm visualization application can be found in the following repository: git@github.com:fatiemahsaadeh/fft_visualization_project.git

Input Data

The input of this application starts as a wave function. This wave comes either from a combination of two sinusoids with frequency and amplitude specified by the user, or an audio wave recorded by the user. The FFT function will then take a user-specified number of samples across the resulting wave. These samples are in the form of amplitude/time coordinates. Additionally, the function takes in a primitive nth root of unity.

Sample Input Snippet

Getting 8 samples of 1 * sin(4 * 2 * π * t) signal

Amplitude	Time
-9.74927912e-01	0
-7.81831482e-01	0.001002
-4.89858720e-16	0.00200401
-4.33883739e-01	0.00300601
0.00000000e+00	0.00400802
4.33883739e-01	0.00501002
7.81831482e-01	0.00601202
9.74927912e-01	0.00701403

root = (0.707106+0.7071067j)

Output Data

The output of this application is the DFT vector calculated by the following equations using FFT algorithm and as the snippet below shows, the frequencies are symmetric around the zero axis.

$$X_{ke} = \sum_{n=0}^{N/2-1} [x_{2n}.e^{-(\frac{j*2*\pi}{N/2})*k*2n}] \quad (1)$$

$$X_{ko} = \sum_{n=0}^{N/2-1} [x_{2n+1}.e^{-(\frac{j*2*\pi}{N/2})*k*2n+1}] \quad (2)$$

Sample Output Snippet

Frequency
-1.253960+0.000000j
-1.717246+0.711307j
2.190643-2.190643j
0.153583-0.370782j
0.000000+0.000000j
0.153583+0.370782j
2.190643+2.190643j
-1.717246-0.711307j

Demo and sample findings

- Sample Input Data Size: 4096 bytes
- Output Data Size: 4072 bytes
- The FFT function memory usage:

Line	Mem usage	Increment
193	79.3 MB	0.1 MiB

D. Stage4 - User Interface.

The application's user interface contains two main components: the input windows and display window. The input windows give the user the ability to choose the original signal to perform our fast Fourier Transform on. They have two options to do this: inputting frequencies (fr1, fr2) and amplitudes (am1, am2) to create an input wave represented by

$$\text{am1} * \sin(\text{fr1} * 2 * \pi * t) + \text{am2} * \sin(\text{fr2} * 2 * \pi * t)$$

or record a one second audio clip. Next, one of two different display windows will appear. The user can choose to walk through the the algorithm, or to show the final results. They also must specify the sampling rate to be used by the algorithm.

Due to the nature of a divide and conquer algorithm, displaying every single step would be both overly time consuming and overly repetitive. For this reason, when the user decides to walk through the steps, we only show two algorithm calls at each recursive layer. Figure 2 shows a recursion tree which illustrates which calls to our fast Fourier Transform are dsiplayed to the user and which are not.

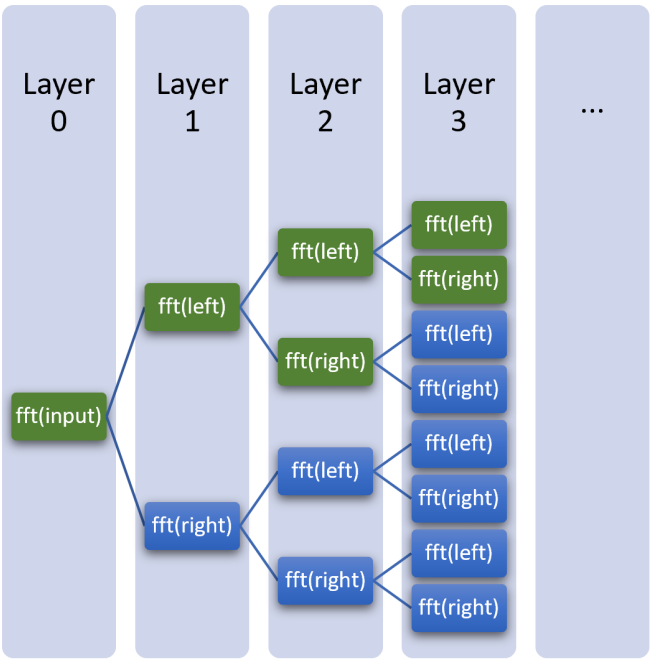

Fig. 2. Sample application recursion tree, with calls that are displayed to the user in green, and calls that are not displayed in blue.

The possible user interactions with the system are defining the frequencies and amplitudes of the sine waves to be added, recording an audio clip, choosing whether to track the FFT step by step or plot the final resulting signal, and clicking on the window to proceed to the next step of the process.

A window indicating an invalid input error will appear when the user inserts any value that cannot be converted to a float.

The results will be four plots showing the original signal (amplitude/time), and the resulted phase spectrum, amplitude spectrum and imaginary/real values of the current left and right points in each layer. (If the user chooses to only plot the final results, the last graph will be excluded.)

As described the first view is define the signal frequency and amplitude, the second view is to plot the original signal in time domain, and show the FFT algorithm step by step plotting the resulted signal in frequency domain, phase spectrum and each layer of the left and right recursion complex numbers. The third view will plot the final results of the time to frequency domain converting.

Deliverables

Python application:

- Initial Activation Statement: Python fft.py
- Initial UI: See Fig. 3. Application Input

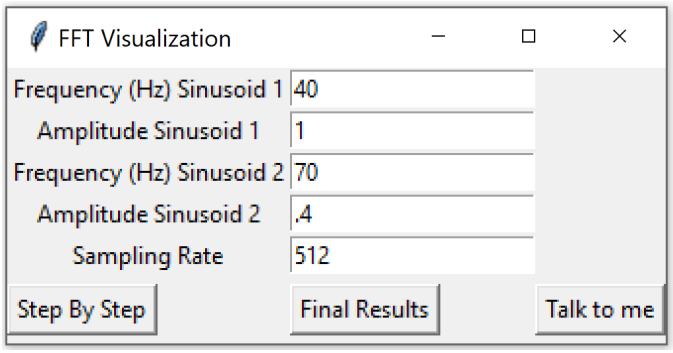

Fig. 3. Application Input

Sample User Navigation Paths:

Navigation Path 1

- Input values for Frequency and Amplitude of 2 sinusoids to be combined for an input signal
- Click "Step By Step"
- A graph of the original signal, current left and right values, the data in the phase spectrum, and the data in the amplitude spectrum will appear
- Click on screen to proceed to the next step of the algorithm

Navigation Path 2

- Input values for Frequency and Amplitude of 2 sinusoids to be combined for an input signal
- Click "Final Results"
- A graph of the original signal, final left and right values, the data in the phase spectrum, and the data in the amplitude spectrum will appear

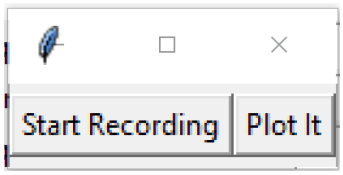

Fig. 4. Recording Window

Navigation Path 3

- Click "Talk to me"
- Click "Start Recording"
- Make some short noise
- Click "Plot it"
- A graph of the original signal, current left and right values, the data in the phase spectrum, and the data in the amplitude spectrum will appear
- Click on screen to proceed to the next step of the algorithm

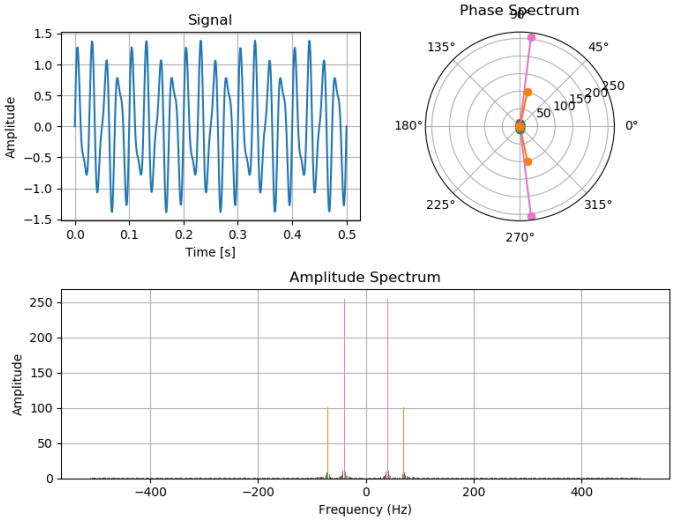

Fig. 5. Application Output

REFERENCES

[1] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani, *Algorithms*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 2008.

[2] Python3.7.5. [Online]. Available: https://docs.python.org/3.7/

[3] Pycharm. [Online]. Available: https://www.jetbrains.com/pycharm/

[4] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python," *arXiv e-prints*, p. arXiv:1907.10121, Jul 2019.

[5] T. Oliphant, "NumPy: A guide to NumPy," USA: Trelgol Publishing, 2006–, [Online; accessed ¡today¿]. [Online]. Available: http://www.numpy.org/

[6] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[7] J. de Langen. Playing and recording sound in python. [Online]. Available: https://realpython.com/playing-and-recording-sound-python