Fast Fourier Transform

April 2021

Instructor: Dr. Shafiei

Written By: Mohammad Amin Shafiee

Overview

In today's task, we want to calculate the Discrete Fourier Transform of an image by using the divide-and-conquer algorithm.

So let's see what is Discrete Fourier Transform and then calculate it.

Discrete Fourier Transform

Fourier Transform (also known as Fourier Analysis) converts a signal from its original domain (often time or space) to a representation in the frequency domain. In Discrete Fourier Transform (DFT) is obtained by decomposing a sequence of values into components of different frequencies.

The formula of DFT is:

$$X_k = \sum_{n=0}^{N-1} X_n * e^{-i\, 2\pi\, k\, n\, /\, N},\ k = 0, 1, \ldots, N-1$$

X_n : is our discrete function

(The function Has N samples for simplicity assume its period [-N/2, N/2]).

N: is period

As you can see it requires $O(N^2)$ Computation.

Fast Fourier Transform

So as you all see above the naive method takes much time to compute the DFT so it raises the question can we do better? (Tim Roughgarden). It turns out yes let's see how they did it.

Because the Period of function is N:

$$X_{N+k} = \sum_{n=0}^{N-1} X_n * e^{-i\, 2\pi k\, n\, /\, N}$$

Same as Function Above Because function will repeat over the period.

So with the above derivation let's continue and calculate X_k:

$$X_k = \sum_{n=0}^{N-1} X_n * e^{-i\,2\pi k\, n / N}$$

$$= \sum_{m=0}^{N/2-1} X_{2m} * e^{-i\,2\pi k\,(2m) / N} + \sum_{m=0}^{N/2-1} X_{2m+1} * e^{-i\,2\pi k\,(2m+1) / N}$$

$$= \sum_{m=0}^{N/2-1} X_{2m} * e^{-i\,2\pi k\, m / (N/2)} + e^{-i\,2\pi\, k\, / N} \sum_{m=0}^{N/2-1} X_{2m+1} * e^{-i\,2\pi k\, m / (N/2)}$$

$$= X_{even} + e^{-i\,2\pi k / N} X_{odd}$$

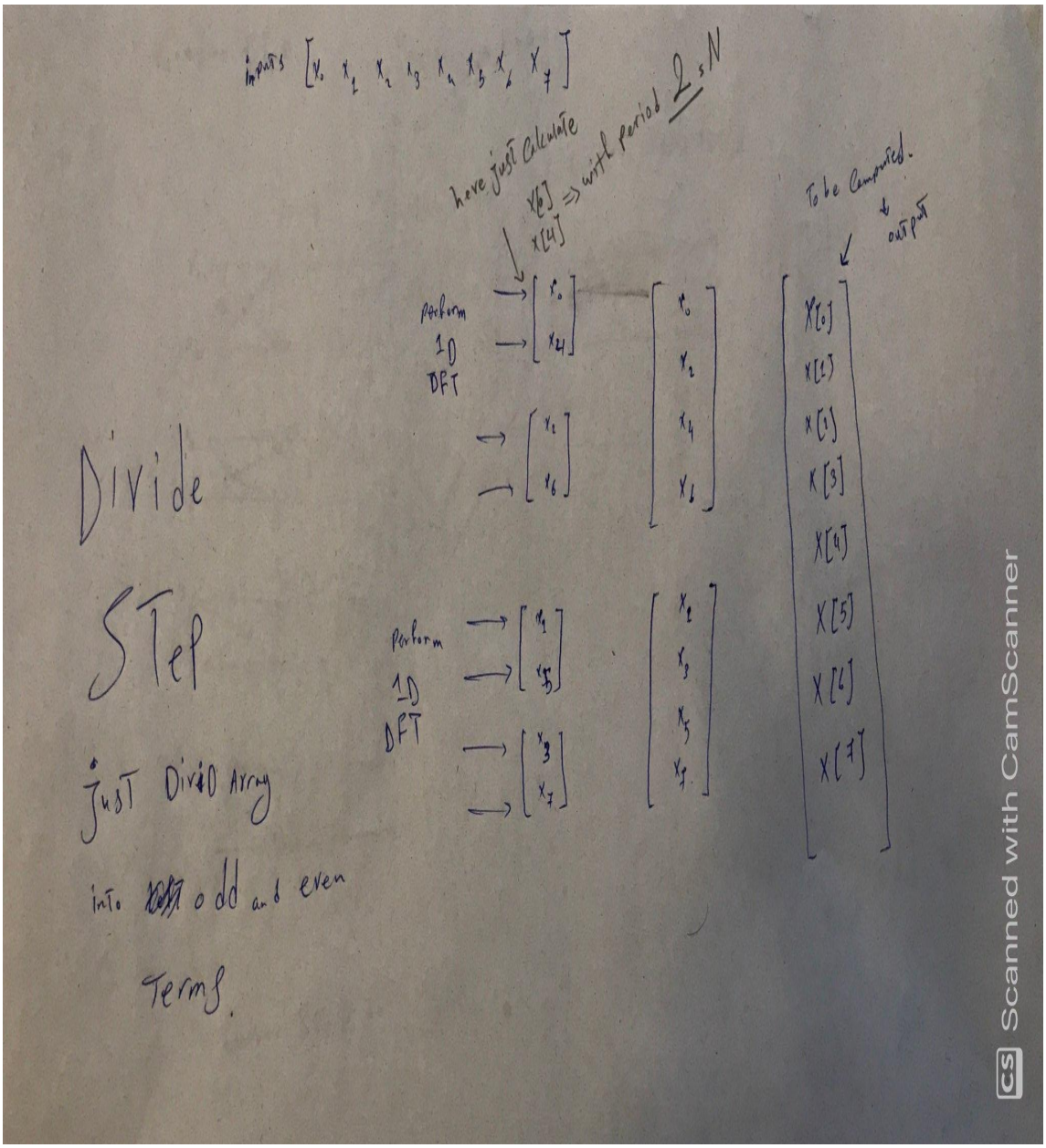
We've split the single Discrete Fourier transform into two terms which themselves look very similar to smaller Discrete Fourier Transforms, one on the odd-numbered values, and one on the even-numbered values.

$\left(\sum_{m=0}^{N/2-1} X_{2m} * e^{-i\,2\pi k\, m / (N/2)} \right.$ is even one and the other term is the odd one).

So far we haven't saved any computation still the same as the naive approach $O(N^2)$. So what is the trick? as all of you may suggest why we don't make each subproblem smaller M is N/2 So like the earlier approach, continue our divide step until there like 2, 4, or even 1 term is left. So with this approach, our computation becomes much smaller because we have 2 divide step (divide the array into odd and even terms) and linear time for computing 1, 2, 4, …, terms($O(N)$).

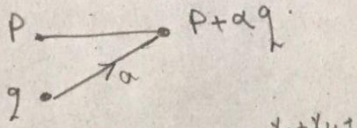
In the picture below, I have drawn the divide and conquer steps for an array with a length of 8 terms.

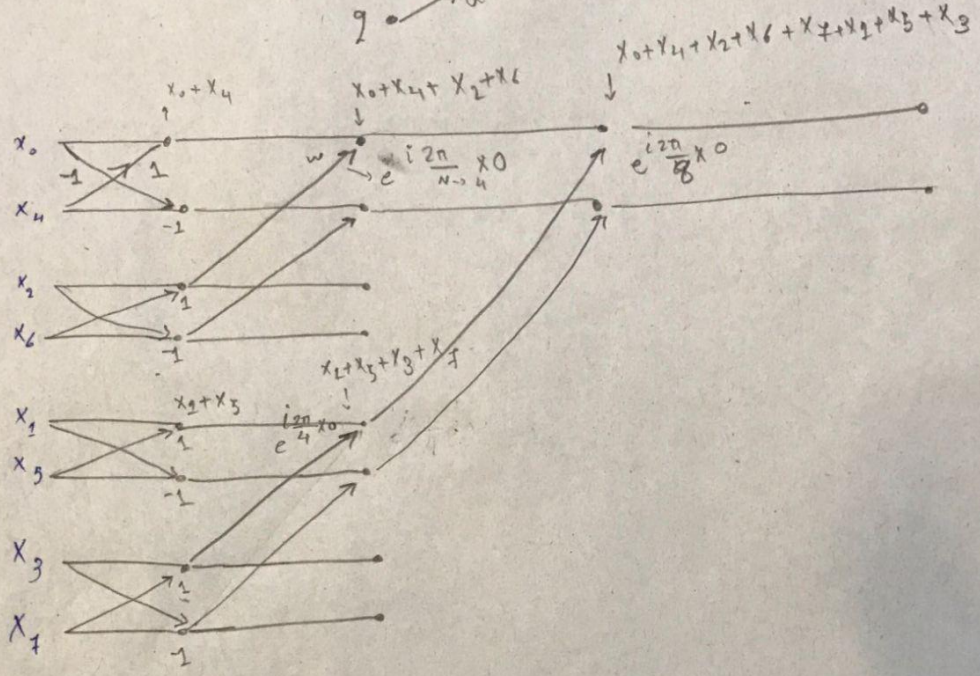
Divide-Step:


inputs
have just calculate
x[0]
x[4]
⇒ with period 2 ≤ N
To be computed.
& output
perform
1D
DFT
perform
1D
DFT
Divide
STep
Just Divid Array
into odd and even
Terms.
Scanned with CamScanner

Combine-step:

Combine Step





$X[0]$

$X[1]$

Continue.

In Combine Step → Just do this $\text{evenTerms} + e^{\frac{-j2\alpha k}{N}} \text{odd Terms}$

$$X[k] = x_{even}[k] + e^{-j\frac{2\alpha k}{N}} x_{odd}[k]$$

Implementation:

I Have Handel reading the image and padding for the algorithm and implemented 1D Fourier Transform. Your task is just to implement the _FFT function which is located in FFT.py (you just implement the algorithm).

Attention:

The resulting image is not the same as the NumPy result image because the NumPy does 2D DFT in the most efficient way, so don't worry if your result is not the same as the NumPy result but 1D DFT is the same as NumPy 1D DTF.

For implementing this task you must use python3.

Scoring:

Your implementation will be scored against the NumPy.ftt package. Don't be worried about scoring very much I just want you to learn some application about the divide and conquer algorithm. The error will be tolerated about 10 percent of real value so there is no panic situation (😂).

Be care full outside and stay safe.

Best regard Shafiee.