Signal Processing Algorithms & Architecture
Dr. Anirban Dasgupta
Department of Electronics & Electrical Engineering
Indian Institute of Technology Guwahati
Lec 9: Fast Fourier Transforms

Hello everyone, welcome to a fresh, new lecture on efficient algorithms for Fourier analysis, which are nothing but our Fast Fourier Transforms. This is part of the course on signal processing algorithms and architectures. I am Dr. Anirban Dasgupta and let us get started. So, now, like in all algorithms, we try to find the complexities in DFT as well. We will try to find what the complexities are or I would say the computational cost. So, we see that this is the equation-

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{j2\pi kn}{N}}$$

If you perform the DFT in matrix form, it will be clear. So, if this is our x[n] and this is the DFT matrix, which is N×N; this is N×1. So, we see there are N^2 multiplications.

I am not considering additions here because multiplications are the costliest operations compared to additions. So, what is the big deal in N^2 computation? Many algorithms have N^2 computation. But say, in a real scenario, we will not compute DFTs of 8-point or 16-point. Typically, we will be interested in computing DFTs which are of very high order and let us see how, with increasing N, This N^2 term increases or the number of multiplications increases.

N	2	8	16	64	128	512	1024
N^2	4	64	256	4096	16384	262144	1048576

So with 2, we get 4, which is reasonable, for 8, we get 64. But as you see, we go for a higher order DFTs the number of multiplications they are extremely large and if you are trying to solve a problem like processing my words, I say hello. So, if I say hello, say it is taking around 3 seconds, and if I say my speech it is sampled at 48 KHz; you can imagine just how many samples this hello will have? And if you do the DFT of the total hello, then you can understand how many multiplications there are will be required, which is difficult for real-time processing. So, can we do better? So here comes the FFT to our rescue. So, the Fast Fourier Transforms are a set of. efficient algorithms to compute the

DFT and its inverse. So, what it does is reduce the complexity of $O(N^2)$ to the $O(NlogN)$. Now, if you have studied algorithms course you know that in divide and conquer methods, we typically reduce an N^2 complexity to $NlogN$ complexity and to give you a better understanding, let Let us see if we are doing $NlogN$ and N^2, see how

N	2	8	16	64	128	512	1024
N^2	4	64	256	4096	16384	262144	1048576
$N\ log_2(N)$	2	24	64	384	896	4608	10240

much this is reducing. So as your n increases, you see. Your N^2 is too much here.

But your $NlogN$ is still reasonable, and it is still quite less compared to N^2 for a value of 1024. Now, you must be thinking why I have taken these are typical values of N, because when I am going to do this $\log_2 n$ operation, so it will give me an integer if I use these values. So, what makes FFT bring down the complexity from $O(N^2)$ to the $O(NlogN)$? And the secret behind it is the twiddle factor. What are the twiddle factors? So, this W_N

$$W_N = e^{-j\frac{2\pi}{N}}$$

is called the twiddle factor. So why are these twiddle factors so important? So, if you see the symmetry property and the periodicity property, of the twiddle factors, they reduce the the number of computations is immense and these two factors help FFT to reduce the complexity of the normal DFT algorithm. So, what is the symmetry factor? Symmetry factor means that I just need to compute-

$$W_N^{k+\frac{N}{2}} = -W_N^k \text{ for all } k$$

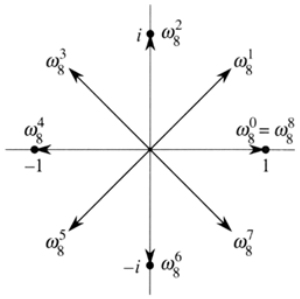
$$W_N^{k+N} = W_N^k \text{ for all } k$$

Because from n to $n/2 - 1$, these can be obtained using the DFTs of the first half and this is by the property you can easily verify this. So, just try this as a take-home assignment. So, just plug in these values and you will see that this is easily verified. Similarly, the periodicity. Property is also easily verified. Now the periodicity property stems from the fact, as I said in the previous lecture, that These discrete Fourier transforms are considered to be periodic sequences, that is the range.

From $-\infty$ to $+\infty$ we are mainly considering the main period, which is 0 to n-1. So, this is just a pictorial representation of the twiddle factor, which is nothing but, so if you see this symmetry property, so this is 7, so of course this is the DC.

The component and the rest are like these two are symmetric and these two are symmetric, and this is the middle point, and this is for the real part. If you see the imaginary part, you also do.

You can easily verify this property and periodicity means it is basically a repetition. So, I have just repeated it for one more period to show you. So, this is n equal to 8. So, this typically repeats, and to understand this, this is the diagram in the complex plane.



So, the twiddle factor is nothing but the n^{th} root of unity. So, how do you say "n^{th} root of unity"? So, 1 can be written as- $e^{j2\pi}$. Why? Because cos(2π) is 1 and sin(2π) is 0. So, I can just write this as by N and to the power N , so this becomes the n^{th} root of unity.

So, if I plug in n equal to 2, I get my plus and Minus 1. If I plug in equals 4, I get these 4 points. So, if you draw the unit circle and plot n points on it. The circle at equal distances, or I would say equally. angles, then those are the roots of your unity.

So, if you see this figure, note the symmetry property. It is very visual from the plot as well as the periodicity. Periodicity means after completion one complete cycle, it will again repeat. Being of a circular nature, it will go on repeating. After every cycle and symmetry, you can see the symmetry very easily. So now we will use divide and conquer approach to efficiently compute our DFT. So, as I said, this direct computation is not. using this symmetry and periodicity. Properties of the twiddle factor. So, this is what we are interested in computing. the DFT and in FFT what we will do is We will use a divide-and-conquer strategy. And the first step is that we will divide this DFT into two DFTs, and one way of dividing is one DFT will be of 2 points and the other.

We will be down by 2 points, which we will overcome eventually. But how did all this start? So, it started by factorizing this N into $N = LM$. Now you might be wondering. What if N is a prime number? What if N cannot be factored easily? So, you can always do zero-padding.

So, you pad with sufficient zeros. to ensure a factorization. The second thing is that I know that x[n] is a One-dimensional vector or a sequence. Now, what we will do in this approach is we will map this one-dimensional vector x n into factors L cross m as a two-dimensional array or matrix and why do we want to do this? this will now create smaller signals where we can compute the DFTs of the smaller ones signals and we know that if the signal If the size is small, the computations will be less. So how can we map this 1D space into 2D? Is this a unique way, or are there multiple ways? For example, if I have 1, 2, 3, 4, I can Write 1, 2, 3, and 4 as 2 by 2 matrices, like 1, 2, 3, 4 or 1, 2, 3, 4. So, there can be two possible mappings accordingly. One can be thought of

$$n = Ml + m$$

$$n = Lm + l$$

Similarly, when we get the frequency, it is also, instead of a 1D array, we will get a 2D one. Array like, say, alpha, beta, gamma, and delta. So, we can read this out row-wise. Alpha, beta, gamma, delta, or we can read this column-wise: alpha, gamma, beta, delta.

So, similarly in frequency, we can also have two possible mappings:

$$k = qL + p$$

$$k = pM + q$$

If it is confusing. To you, let us take a simple illustration. So here we have this sequence. So, what is M? M is 4. So small m varies along with this. So, if I say 0, 1, 2, and 3, and L varies downwards to 0 and 1. So, what is the value of capital M? It is 4, which means this equation is N equals 4L plus M.

$$x[n] = [2,3,5,6,9,1,4,7], M = 4, L = 2$$

$n = Ml + m$

2	3	5	6
9	1	4	7

$n = Lm + l$

2	5	9	4
3	6	1	7

$k = qL + p$

$X[0]$	$X[2]$	$X[4]$	$X[3]$
$X[1]$	$X[3]$	$X[5]$	$X[7]$

$k = pM + q$

$X[0]$	$X[1]$	$X[2]$	$X[3]$
$X[4]$	$X[5]$	$X[6]$	$X[7]$

So, put the M that varies from 0 to 3. Small m. So, put L varies from 0 to 1. So, if L is 0, then M is 0; if this value, M is 1, is 0. This value, which is n equals 1 and m equals 2, Is this value, n equals 2 and m equals 3, equal to n equals 3? And then the next value will be. l equals 1 and m is 0. So that makes n equal to 4. If L is 1, then M is 1, and N is 5. Then M is 2, L is 1, and N is 6. M is 3, L is 1, and it is index 7. So, you see, what am I doing is? I am technically arranging this X row-wise. The first row is 2, 3, 5, 6, and the second row is 9, 1, 4, 7. Another way is that I just do it in columns wise like 23,569,147. Similarly, the frequencies that I get, Now, if you have this idea clear, let us start with the algorithm. And as I said, depending on the arrangement I can have two such algorithms.

So, the first algorithm is that we will store the signal column-wise. Column-wise means saying this, like this. So, what will we do then? So, we will compute them M point DFT of each row.

So this is the signal-stored column wise, and this is the formula to find out the M-point DFTs. So, for an N-point DFT, we know the computational complexity is N^2 . So, for M-point DFT, it will be M^2. And how many such DFTs are we computing? There are L rows. So, L such DFTs, we will compute. So, the cost will be $(LM)^2$. Now this signal, which I get, I say this matrix is f L q, and we will multiply with the factored twiddle factor w, n, L, q to this output f of L, q. And this I say, this is g of L q. This is just multiplying two matrices. So, it will be a cost of L multiplied by m. What is next? Next is now this GLQ. We will do the L-point DFT. of each column. So again, as I said, L point DFT. will take L squared computations for multiplications and We are doing M such DFTs.

So, the cost will be $(ML)^2$ and finally, we have got this matrix X, which is the DFTs. We just need to know. Read out the values. Now, I say that considering these multiplications and the storing of the signal column-wise and reading the values row wise. So, we will not consider these. These are kind of constant operations. So, if you just read the values out row-wise. So, we will get the final DFT, and if you verify it will be the exact X[K] that we will get by without doing so many things. We will just do direct computation of the endpoint DFT. Now you must be thinking that. Here we are doing a lot of computations. We are first doing the M-point DFT of the rows, then multiplying by a rectangular array and then L point DFT again of the columns. So, won't it be more

computations? Well, math will give the answer and if I add these three, like I'm. square, lm, and ml square. So, if I take ml common, it is l plus m plus one. So, 1 does not count towards complexity. So, it will be n. So, m times l is n. because that is the factor we have done. So, N times L plus M, which is less than. N squared, and you check with.

Different values of N. Say if you take N equal to 30, check 30 squared and if you factor it as 6 plus 5, then into 6 plus 5. So, 30 squared is 900, and this is 11 times 30, which is 330, much less. So, these are the two algorithms. So, if you find it complex because when you are doing what you will if the sequence is big because of a smaller sequence you can directly find the DFT; it will hardly matter.

That is what we can see in the n versus n squared plot. So, if n is very small, it is not a big issue to do direct computation. But if n is big, then you will probably do not do it by hand, you will be doing it on a computer. So, you just remember these steps because the rest are formulas which I have already discussed it. So, algorithm one is to store the signal column. wise, then find M-point DFTs of the rows, compute G, and compute L point DFTs. of the columns, read the DFT values row-wise and the second algorithm is you store the signal row-wise, compute L-point DFTs of columns.

Compute this matrix G, then compute M points. DFTs of rows and then read the values column-wise. So, another very easy technique to remember is column-wise storing the signal DFTs of rows, then DFTs of columns, and reading DFTs row-wise. So, there is an alternative fashion. Similarly, if you store the signal row-wise, first do the DFTs of the columns, then do DFTs of the rows and then read the values column-wise. So, if you have understood this, divide and conquer the approach, then it is very easy to understand the most common form of the FFT which is the Radix-2 FFT, or it is also called the radix-2 Cooley-Tukey FFT algorithm based on the two computer scientists, Cooley and Tukey. So, what is this? So, suppose I say N is highly composite and it is made up of a lot of factors of r such that it is r^v. Then such algorithms will be called your Radix R algorithm and if this r is 2, it is. called the Radix 2 algorithm. Now, why is this efficient? We will know very shortly. So, first, what will it do? So, here also L is into m. So, one will be N divided by 2, and the other will be 2. And what is the specialty about 2? Is that if you see the 2-point DFT? It is just adding and subtracting. So, you do not need any difficult computation in that. You just need to add the values to get X(0) and the difference of the values, X(1) will give you the capital. And then again, this N/2 can be further divided into N/4, into 2, and into 2. And in this way, you keep on doing. Unless you are computing all two-point DFTs. So, to start with, as I said, m is N/2 and one is 2, and here n will be n and will vary from 0 to $\frac{n}{2}$ -1.

So, essentially, what are we doing? We are taking even samples and the odd samples. So, if I have signals like 3, 2, 1, 6, 0, and 2, so this will form my first sub signal $F_1[n]$, and this will form the second sub signal $F_2[n]$. So what are we doing? Essentially, we are skipping some values of the signal and forming the sub signal and this process is called decimation or downsampling.

So there is a difference in decimation and down sampling. which we will cover in multirate signal processing. But as of now, understand that since we are skipping some signals. In the time domain, this is called the decimation in time algorithm and just to clearly explain the mathematics, so this is my original formula for DFT, where I say this is for the even values of n. This is for the odd values of n, so even values I replace my N as twice M-

$$N = 2M$$

and this is for the even values. This is for the odd values: the two small DFTs we are calculating and now we will use some properties like we know,

$$(W_n)^2 = W_{n/2}$$

So here we will write down the DFT matrix as n/2 so which is the meaning we are halving the size of the DFT matrix and this is very evident from the calculations. So this X[k], which is an endpoint DFT is broken down into two DFTs one is of length n/2, and the other is also of length n/2. And this is the multiplication with that twiddle factor which is giving that g thing in the divide and conquer approach. Great. So now we have two N/2-point DFTs. Again, in this case, since this is N by 2 point DFT, now my k will be varying from 0 to $\frac{n}{2}$ - 1. But we need values from. $n/2$ to $n\ -\ 1$ also. Now here comes my symmetry property so the rest, since it is symmetric, so we will use the symmetry property and with this, we can extend this to the other half of the DFT, and we keep on doing this, as I said. So, from n we get two N/2- point DFTs, then from N/2, we get 4 N/4 -4 point DFTs, and we keep on doing. Until finally, we end up in 2-point DFTs and 2 point DFTs, as I said, are just adding and subtracting. So, what is essentially the computation? So, we have to do these n values that are the nth thing and then the height of the tree if you see, so every time. It is halving itself, so this height will be-

$$\log_2 N$$

and hence the complexity is reducing to $N\ log_2\ N$. So the two algorithms, this is called a butterfly diagram is somewhat looks like a butterfly's wings. So, this is the decimation-in-

time algorithm. How would you understand it? See the input side. So, on the input side, the first sample is x[0].

Then three samples are. skipped x[1], x[2], and x[3], and then I get x[4]. Similarly, I get x[2], and then. I skip three samples; I take x[6]. Then I take x[1], skip 3 samples, and take x[5] and similarly take x[3], and then skip 3 samples, take x[7]. Now, why three samples? because there are two stages, so each stage is divided into half of the previous stage like a recursion kind of thing, or you can consider this as storing the DFT values column-wise. So, this is just like x if you see x[0] coming here then x[1], x[2], x[3], x[4], x[5], x[6] and x[7]. So, this is going to be the first DFT we are doing the DFT column-wise, right? And in the frequency, we will read it sequentially. The other version is the decimation in frequency or DIF algorithm. This is also the butterfly diagram. Here we will take a look at the reading store.

Signal row-wise like it is, but when we will read the frequencies we will read them column-wise. So here we see that the frequencies are decimated.

So, we have to see this and then we will get this. So, that is the decimation in frequency algorithm. The remaining thing is self-explanatory. This is just multiplying with the twiddle factors as per the formula and this is just a simple signal flow graph. So just these two are the pictorial representations. This is for an 8-point FFT. This is for a 16-point FFT which is in three stages.

So, thank you very much.

We will meet again in the next lecture.