Finite Element Method and Computational Structural Dynamics

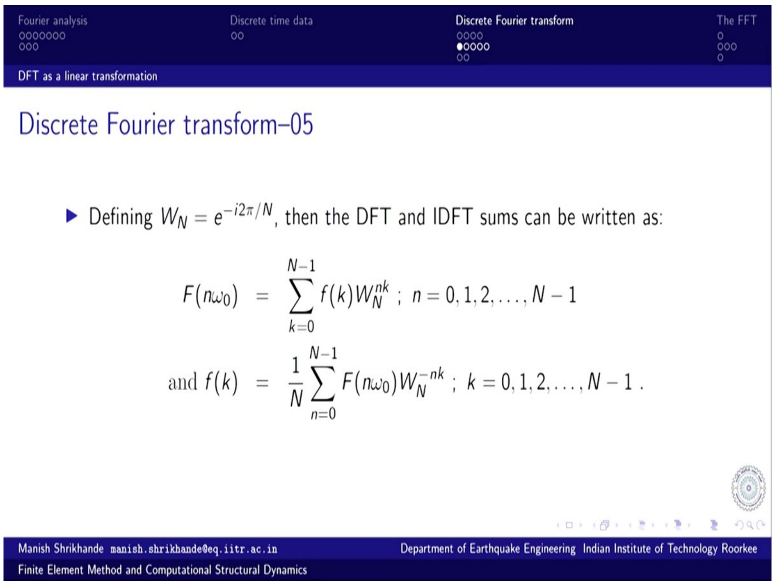
Prof. Manish Shrikhande
Department of Earthquake Engineering
Indian Institute of Technology, Roorkee

Lecture - 58
Discrete Fourier Transform-V

Hello friends. So, we have seen the nitty-gritties of the Computation of Discrete Fourier Transform, and how one period of time domain data and one period of frequency domain data corresponds or serves as a good approximation of continuous time and continuous frequency domain Fourier transform pairs, which are derived, which are representative of these periodic functions that we do with discrete time samples and discrete frequency samples.

So, we studied we looked at the problem for discrete Fourier transform as a linear mapping problem linear transformation.

(Refer Slide Time: 01:14)



Discrete Fourier transform–05

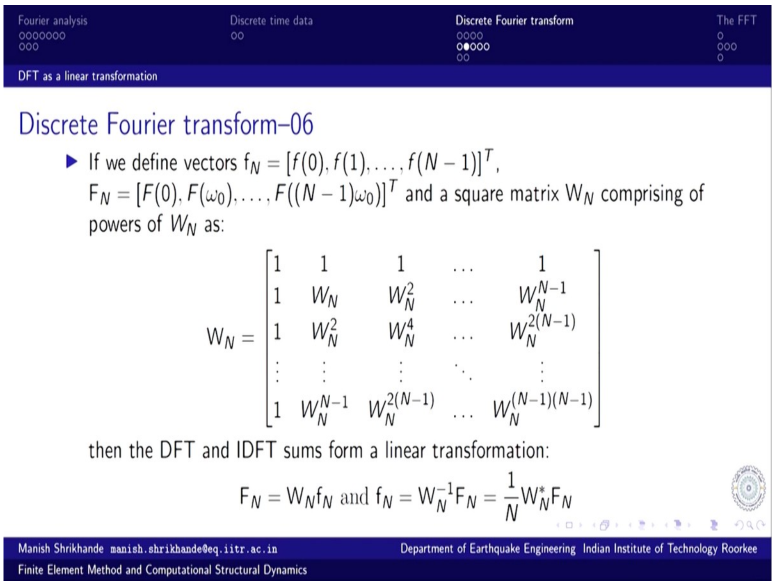
- Defining $W_N = e^{-i2\pi/N}$, then the DFT and IDFT sums can be written as:

$$F(n\omega_0) = \sum_{k=0}^{N-1} f(k) W_N^{nk} \; ; \; n = 0, 1, 2, \ldots, N-1$$

$$\text{and } f(k) = \frac{1}{N} \sum_{n=0}^{N-1} F(n\omega_0) W_N^{-nk} \; ; \; k = 0, 1, 2, \ldots, N-1 \, .$$

We saw that the Fourier transforms, the discrete Fourier transform and inverse discrete Fourier transform. They can be represented as a linear transformation between two sets of numbers. So, N number of data points can be mapped onto another N number of data points by using the transformation matrix, a square transformation matrix. And those the square transformation matrix comprises of complex exponentials.

(Refer Slide Time: 01:49)



Discrete Fourier transform–06

- If we define vectors $\mathrm{f}_N = [f(0), f(1), \dots, f(N-1)]^T$, $\mathrm{F}_N = [F(0), F(\omega_0), \dots, F((N-1)\omega_0)]^T$ and a square matrix W_N comprising of powers of W_N as:

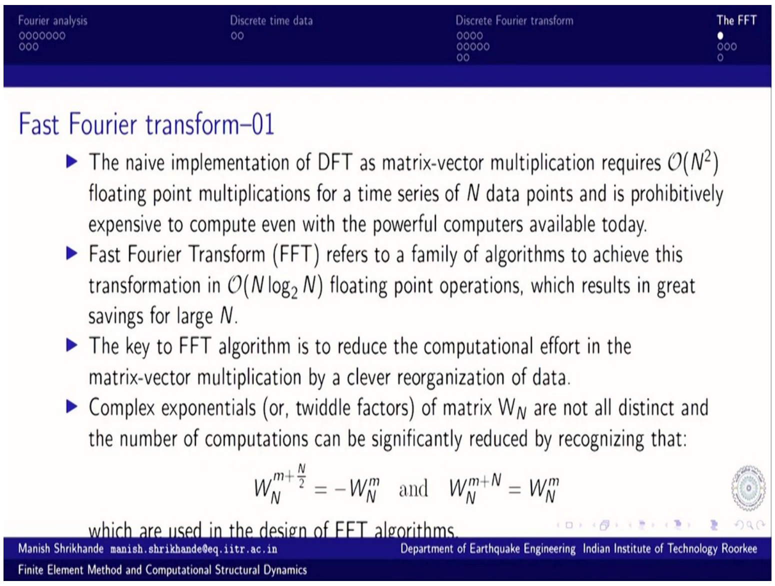
$$\mathrm{W}_N = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N & W_N^2 & \dots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \dots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & W_N^{(N-1)(N-1)} \end{bmatrix}$$

then the DFT and IDFT sums form a linear transformation:

$$\mathrm{F}_N = \mathrm{W}_N \mathrm{f}_N \text{ and } \mathrm{f}_N = \mathrm{W}_N^{-1}\mathrm{F}_N = \frac{1}{N}\mathrm{W}_N^{*}\mathrm{F}_N$$

So, now the key to computation of discrete Fourier transform lies in efficient computation of these linear transformation so, how do we do this?

(Refer Slide Time: 02:00)



Fast Fourier transform–01

- The naive implementation of DFT as matrix-vector multiplication requires $\mathcal{O}(N^2)$ floating point multiplications for a time series of N data points and is prohibitively expensive to compute even with the powerful computers available today.
- Fast Fourier Transform (FFT) refers to a family of algorithms to achieve this transformation in $\mathcal{O}(N \log_2 N)$ floating point operations, which results in great savings for large N.
- The key to FFT algorithm is to reduce the computational effort in the matrix-vector multiplication by a clever reorganization of data.
- Complex exponentials (or, twiddle factors) of matrix W_N are not all distinct and the number of computations can be significantly reduced by recognizing that:

$$W_N^{m+\frac{N}{2}} = -W_N^m \quad \text{and} \quad W_N^{m+N} = W_N^m$$

which are used in the design of FFT algorithms.

So, naive implementation of discrete Fourier transform as a matrix-vector multiplication as we define it here as $F_N = W_N f_N$. So, this is W_N is this square matrix of complex exponentials multiplied by N number of time domain data discreetly sample time domain data. So, this entire thing complex multiplication it matrix-vector multiplication is an N square operation.

So, the computational effort the number of floating point multiplications that we deal with in this particular implementation is of the order of N^2, N is the number of data points. So, obviously, the number of it the computational effort increases exponentially as N increases. And we discuss that normally we would keep N as integer power of 2, that is that follows in a short while the reason why it has to be integer power of 2.

So, but just matrix-vector multiplication as we see it requires the order computational effort is of the order of N square. So, that is actually proportional to the number of floating point multiplications that happen there, and it is prohibitively expensive to compute even with the powerful computers that are available today. So, it takes very very long and the effort actually grows exponentially. So, very very quickly so, N^2.

So, you can see that if N = 2, the effort is 4 of proportional to 4. If N = 8, the effort is proportional to 64, and if N = 122, 128 then effort is approximately may be proportional to way above 1000. So, it grows very quickly. And very soon, it may hit I mean the performance limit and there might be more efficient techniques available to solve to handle the problem. And this transform may not be feasible.

And that is that was the state of affairs until late 20th Century, until 1965 to be precise. So, while these Fourier series and transform analysis was known for a long-long time and everybody knew the importance and the ease with which the problems could be transformed and the elegant structure, that it afforded in the transform domain which could be theoretically, it require simple to solve if only we could transform in the first place.

So, this transformation was a problem for reasonably and for any decent size of data and computation of transform was a major challenge. So, the while the importance of the transform analysis was recognized very early, but there was no way to make use of it, because there was no efficient way to compute the transform.

So, Fast Fourier Transform FFT as we call, it refers to a family of algorithms to achieve this transformation in N log N floating point operations multiplications. So, N log N so, logarithm is to the base 2. So, if you look at it so, logarithm to base 2 that is just a constant and it is then proportional to linear power.

So, the computational effort actually rises very very slowly it is a linear progression. So, compare that with the simple naive implementation of DFT as simple matrix vector multiplication N square and compare it to with N log of N base to the base 2.

So, log of N to the base 2 is a constant so, that will be for any given value of N. So, it will be linearly proportional to N first power of N. And that results in great saving for large N.

So, for example, if N = 10000. So, for fast Fourier transform the computational effort would be proportional to 10000, but for matrix-vector multiplication that effort would be proportional to 10^8. So, almost factor of 10000 difference.

So, that is a effect that the kind of scaling that happens the in the computational effort that we say we tend to save by making use of fast Fourier transform algorithm. And that is why it is set to be one of the major technological advances in 20th Century. The discovery of fast Fourier transform algorithms.

It changed the way data was analyzed, it changed the way these signals were analyzed, it changed the complete field I mean the radar applications and I mean applications are immense. The moment it becomes feasible to implement transform very quickly, then the application areas are immense and several new vistas opened up.

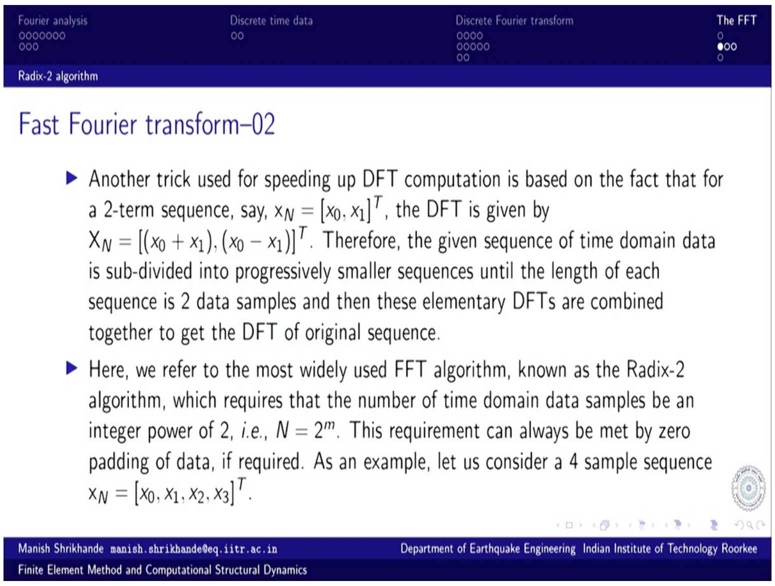
And it completely revolutionized the way data was analyzed and looked at. So, the key what is the key to FFT algorithm how can such dramatic savings be achieved. So, key to FFT algorithm is to reduce the computational effort in matrix-vector multiplication by a clever reorganization of data that is all we do. So, the power of observation look at the pattern the how the computations are arranged and identification of pattern and trying to find a simpler way of arriving at same results in a more efficient manner.

So, complex exponentials or the twiddle factors that we discussed earlier of the matrix W_N, we need to appreciate that they are not all distinct. And number of computations can be significantly reduced again because those are complex exponentials, those are of course, periodic with period 2 pi.

So, the number of computations can be significantly reduced by recognizing that $W_N^{m+\frac{N}{2}}$, that is the power that is raised is actually equal to negative of W_N^m. And then W_N^{m+N} is simply = W_N^m , because this is going to be periodic with period N. And beyond after N by 2, this is going to be negative complex conjugate. So, it is simply we can just make use of conjugation rule and know that it is just going to be the negative of that.

And these identities are essentially used in the design of FFT algorithm. So, how does it work? So, let us just for simplicity and for ease of application, ease of explanation I take example of just 4 set 4 samples time domain data, which is represented by only 4 samples. And that 4 sample will when we process it go through the entire process that will explain how fast Fourier transform algorithm works. And that will explain how this, how the savings are achieved. And of course, the concepts that we use are general enough and they can be extended to any length of data.

(Refer Slide Time: 11:18)



Fourier analysis | Discrete time data | Discrete Fourier transform | The FFT

Radix-2 algorithm

Fast Fourier transform–02

- Another trick used for speeding up DFT computation is based on the fact that for a 2-term sequence, say, $x_N = [x_0, x_1]^T$, the DFT is given by $X_N = [(x_0 + x_1), (x_0 - x_1)]^T$. Therefore, the given sequence of time domain data is sub-divided into progressively smaller sequences until the length of each sequence is 2 data samples and then these elementary DFTs are combined together to get the DFT of original sequence.
- Here, we refer to the most widely used FFT algorithm, known as the Radix-2 algorithm, which requires that the number of time domain data samples be an integer power of 2, *i.e.*, $N = 2^m$. This requirement can always be met by zero padding of data, if required. As an example, let us consider a 4 sample sequence $x_N = [x_0, x_1, x_2, x_3]^T$.

Manish Shrikhande manish.shrikhande@eq.iitr.ac.in | Department of Earthquake Engineering Indian Institute of Technology Roorkee

Finite Element Method and Computational Structural Dynamics

So, another trick that is used for speeding up the DFT computations is based on the fact that the 2-term sequence say x_N = x_0 and x_1. If there are only 2 sequences the only 2-terms, then the discrete Fourier transform is given by, just the sum of these two numbers and the difference of these two numbers, because that will be corresponding to just 0 and %pi right.

And the they are going to be real, so no imaginary part there. And those numbers are given by addition of 2 numbers and subtraction of 2 numbers we can work out the complex exponentials using complex exponentials, and we arrive at the same results here as given here.

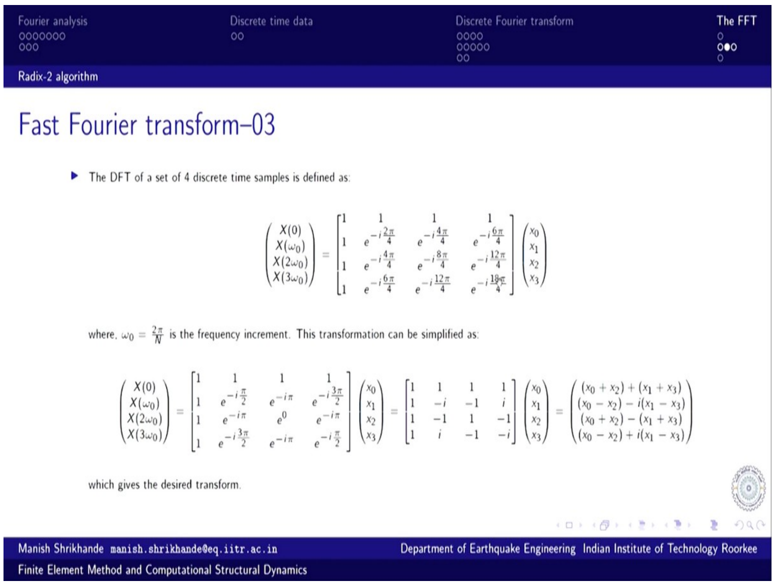
So, if the given sequence of time domain data, if we can find some way to subdivide it into progressively smaller sequences until the length of each sequence is 2 data samples right. So, we kind of segregate or divide data repeatedly into smaller and smaller, smaller and smaller sets until we end up with two samples in each set and then we know that the Fourier transformers those two those two samples is just the sum of those two samples and difference of those two samples. So, and then using this result we build up the remaining computation subsequently.

So, these elementary discrete Fourier transforms of two sample set are combined together to get the discrete Fourier transform of original larger sequence. So, how this is actually done? So, in this particular example that we take we use the most widely used FFT algorithm known as Radix-2 algorithm and this algorithm, this was also the one of the very first FFT algorithms that was proposed by Cooley and Tukey known as Cooley-Tukey algorithm both of them working at Bell Labs.

So, Radix-2 algorithm and this algorithm requires that number of time domain data samples be an integer power of 2. So, N now that comes the requirement that total number of data points that has to be 2^m, m being a positive integer. And this can requirement can always be met by zero padding of the data, if required. So, it will be rarely by happy coincidence that the number of data points that we have the waveform would be exactly equal to integer power of 2.

So, before we apply Fourier transfer algorithm, before we use this data for Fourier transform it is necessary that the length of the data should be made equal to integer power of 2, and that is done by zero padding. So, as an example as I said we consider four sample sequence. So, that is x_N is given by x_0, x_1, x_2, x_3. So, 4 data points in discrete time samples.

(Refer Slide Time: 15:08)



Fast Fourier transform–03

- The DFT of a set of 4 discrete time samples is defined as:

$$\begin{pmatrix} X(0) \\ X(\omega_0) \\ X(2\omega_0) \\ X(3\omega_0) \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & e^{-i\frac{2\pi}{4}} & e^{-i\frac{4\pi}{4}} & e^{-i\frac{6\pi}{4}} \\ 1 & e^{-i\frac{4\pi}{4}} & e^{-i\frac{8\pi}{4}} & e^{-i\frac{12\pi}{4}} \\ 1 & e^{-i\frac{6\pi}{4}} & e^{-i\frac{12\pi}{4}} & e^{-i\frac{18\pi}{4}} \end{bmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

where, $\omega_0 = \frac{2\pi}{N}$ is the frequency increment. This transformation can be simplified as:

$$\begin{pmatrix} X(0) \\ X(\omega_0) \\ X(2\omega_0) \\ X(3\omega_0) \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & e^{-i\frac{\pi}{2}} & e^{-i\pi} & e^{-i\frac{3\pi}{2}} \\ 1 & e^{-i\pi} & e^{0} & e^{-i\pi} \\ 1 & e^{-i\frac{3\pi}{2}} & e^{-i\pi} & e^{-i\frac{\pi}{2}} \end{bmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} (x_0 + x_2) + (x_1 + x_3) \\ (x_0 - x_2) - i(x_1 - x_3) \\ (x_0 + x_2) - (x_1 + x_3) \\ (x_0 - x_2) + i(x_1 - x_3) \end{pmatrix}$$

which gives the desired transform.

And for these 4 discrete time samples the Fourier transform is given by X(0), $X(\omega_0)$, $X(2\omega_0)$, $X(3\omega_0)$. And that is given by this twiddle factors the complex exponential matrix and numbers. Now, if we look at these numbers I am sorry for the small font because I had to compress the whole thing here.

So, first thing that we see here that these are of course, complex exponentials and they appear as a product of k and N. So, in this particular case it becomes a product of m N and k both 3 3. So, that becomes 9 multiplied by 2π. So, it becomes $18\frac{\pi}{4}$. So, 4 is the number of data points N. So, this becomes $18\frac{\pi}{4}$.

Now, this $18\frac{\pi}{4}$ I will just take this last sample here. So, this exponent is $e^{-i18\frac{\pi}{4}}$. Now, $18\frac{\pi}{4}$ is of course, greater than 2π and these complex exponentials they are periodic with π. So, I can all trap these all these complex exponentials I can just consider modulo 2π. And that expression now leads to this expression here.

So, here this $18\frac{\pi}{4}$ is exactly I mean numerically equivalent to $e^{-i\frac{\pi}{2}}$ simply. And similarly all these expressions they would be wrapped back into the, range I mean fundamental range of frequencies discrete frequencies; outside the discrete fundamental frequency, they would all be wrapped back, because of the periodicity making use of periodicity.

And then we make use of evaluate this complex exponentials and then we see that this particular thing is $e^{-i\frac{\pi}{2}}$. So, at $\frac{\pi}{2}$ cosine term is 0. So, all that we will be left with is $-i\sin\frac{\pi}{2}=1$.

So, that is what we end up here as - i and next term here is $e^{-i\pi}$ so, that would be equal to $\cos\pi$ and $\cos\pi=-1$. So, we have this matrix that is available. And when we combine these two we end up with very interesting result.

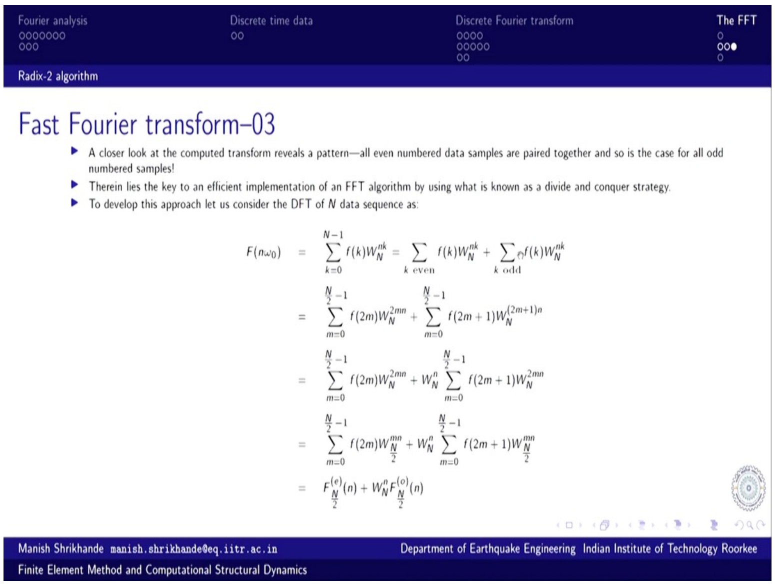
First term is $x_0+x_2+x_1+x_3$. Second term here is $x_0-x_2-i\times(x_1-x_3)$. Third term here is $x_0+x_2-x_1+x_3$. So, that repeats I mean real number again. And followed by $x_0-x_2+ix_1-x_3$ so, the conjugate of the earlier complex number. And this is the Fourier I mean desired discrete Fourier transform.

So, first simplification that happens is we need to look at what are the complex exponentials. And we only need to find out what are the unique values of the complex exponential, the periodic values they can always be wrapped back into the fundamental range. And the values that we have already computed that can be reused. So, that is one way first step of saving computation and there will be this will be a massive saving in its own right.

Second interpretation second observation is the combination how the terms get combined. If you look at these only the even number terms are occurring together and only odd number terms are occurring together, in the transform while computing transform. So, even number terms for the time domain data and odd number terms in the time domain data.

So, even numbered terms in time domain data they combine with even number term. And odd number term they combine with odd number term, in making up the transform. So, that gives us very interesting observation and that gives us a hint, that maybe we can just segregate even data and from odd data. So, we can find out what is the even number your data then we know how to combine them some of the even number and odd number and that is all it requires.

(Refer Slide Time: 20:42)



Fast Fourier transform–03

- A closer look at the computed transform reveals a pattern—all even numbered data samples are paired together and so is the case for all odd numbered samples!
- Therein lies the key to an efficient implementation of an FFT algorithm by using what is known as a divide and conquer strategy.
- To develop this approach let us consider the DFT of N data sequence as:

$$\begin{aligned}
F(n\omega_0) &= \sum_{k=0}^{N-1} f(k)W_N^{nk} = \sum_{k \text{ even}} f(k)W_N^{nk} + \sum_{k \text{ odd}} f(k)W_N^{nk} \\
&= \sum_{m=0}^{\frac{N}{2}-1} f(2m)W_N^{2mn} + \sum_{m=0}^{\frac{N}{2}-1} f(2m+1)W_N^{(2m+1)n} \\
&= \sum_{m=0}^{\frac{N}{2}-1} f(2m)W_N^{2mn} + W_N^{n} \sum_{m=0}^{\frac{N}{2}-1} f(2m+1)W_N^{2mn} \\
&= \sum_{m=0}^{\frac{N}{2}-1} f(2m)W_{\frac{N}{2}}^{mn} + W_N^{n} \sum_{m=0}^{\frac{N}{2}-1} f(2m+1)W_{\frac{N}{2}}^{mn} \\
&= F_{\frac{N}{2}}^{(e)}(n) + W_N^{n} F_{\frac{N}{2}}^{(o)}(n)
\end{aligned}$$

So, a closer look at the computed transform it reveals a pattern all even number data samples are paired together. And so, is the case for all odd numbered data points, odd numbered samples. And therein lies the key for efficient implementation of FFT algorithm by using what is known as a divide and conquer strategy.

So, we know now that there is something to do with the arrangement and the placement of numbers in that array. So, even the numbers that are placed in even position they somehow combine only with the numbers placed on even position other numbers placed on even position even indices. And the numbers that are placed on odd indices they somehow combined with only the numbers that are placed on odd indices. So, we can separate them out.

So, we look at the this approach divide the sequence so, if we again go back to Fourier summation and will this entire summation of N number of data points we split it into 2 parts. Where k is even and another summation, where k the sample number they are odd numbered samples.

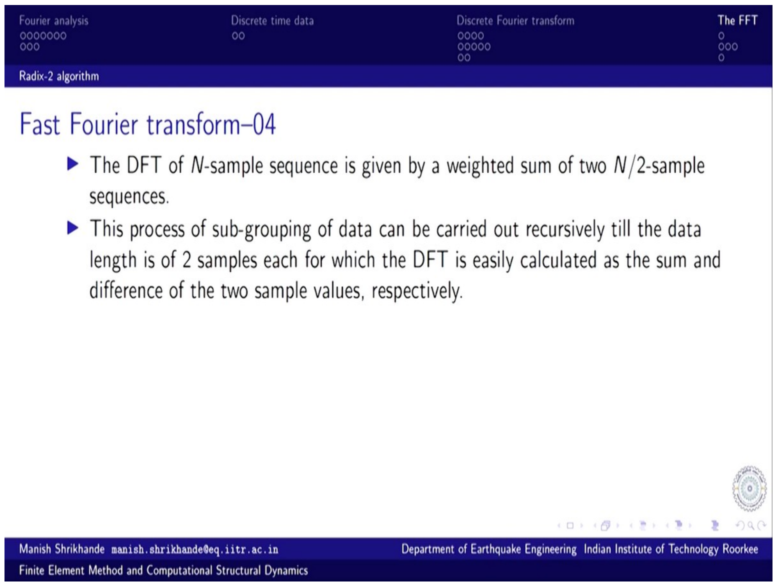
And then when we look at this arrange these numbers suitably and then it can be found that we can work out this representation the Fourier transform that we have of N number of data points, it can be defined as a linear combination or the of the Fourier transform of even number of data point, so half of the length Fourier transform of half of the length of data.

So, while we are computing Fourier transform of N length data that can be represented as some of Fourier transform of half-length data that is even numbered data. And the twiddle factor again W_N^N times the odd numbered data Fourier transform. So, essentially we reduce the problem from N number of data points we reduce the problem to N by 2 number of data points to such cases.

So, eventually this is what happens in each cycle we reduce the problem size by a factor of 2. And eventually we will continue this cycle repeatedly until we end up with only 2 data points. And once we have these 2 data points, then we know the Fourier transform of 2 data points that is just sum of the numbers and difference of the two numbers. And that is the Fourier transform.

So, once we do that then we can again substitute it back into this relation that even part Fourier transform $+W_N^N$ times odd part Fourier transform and then we can work our way backwards. And by addition we can find out the original waveform Fourier transform. So, that in a sense is the basic algorithm of fast Fourier transform.

(Refer Slide Time: 24:30)



Fourier analysis | Discrete time data | Discrete Fourier transform | The FFT

Radix-2 algorithm

Fast Fourier transform–04

- The DFT of N-sample sequence is given by a weighted sum of two $N/2$-sample sequences.
- This process of sub-grouping of data can be carried out recursively till the data length is of 2 samples each for which the DFT is easily calculated as the sum and difference of the two sample values, respectively.

Manish Shrikhande manish.shrikhande@eq.iitr.ac.in Department of Earthquake Engineering Indian Institute of Technology Roorkee

Finite Element Method and Computational Structural Dynamics

So, in this what we have done here is DFT of N-sample sequence is given by a weighted sum of two N by 2 sample sequences. This process of sub-grouping of data can be carried out recursively till the data length is of 2 samples each for which the DFT is easily calculated as the sum and difference of the two sample values respectively.

So subsequently, this computed DFT how smaller sub-groups they are added together weighted some as we said complex exponential integer power of complex exponential. And they are added together to obtain the DFT of full and sample sequence.

And this whole sequence of operation can be completed in of the order of the computational effort is of the order of N log N base 2. And floating point operations as against total computational effort of N^2 in implementation of DFT as a simple linear transformation operation as matrix-vector multiplication.

And while I discussed this with very simple example of four sample points and worked out how it works out in practice and how it will end up in physical I mean actual implementation by dividing it; dividing a given sample set into two-halves and then again looking at each of those into two-halves and again those 4 into again having them again and again until we end up with 2 samples at a time and again work backward.

So, this algorithm although makes it very easy visually it is very difficult to visualize; I mean the how the algorithm proceeds and but there are some very good excellent implementations of FFT algorithm. And there are I mean the happy coincidence does not end here in discovering the combination of segregation of data into even and odd, which allows us to divide it into half at every scale, every cycle.

If we go into the implementation detail we will find that the correct position they are obtained of for the combination of the terms they are obtained by reversing the binary representation of indices. So, whatever is the index level index position value of the index and if we represent that index in binary format, and then reverse the bits just find out what is the if it is 1 0 we will call it 0 1, if it is 0 1 we use 1 0.

And that bit reversed arrangement that corresponds to the target arrangement that we are looking for in this Radix-2 algorithm. And this happy coincidence is a very very efficient way to implement this FFT algorithm. And of course, we can have theoretical analysis of this FFT algorithm why this works and why this has to be bit reversal is actually the correct way of positioning of data.

We can, there are several analysis that you can look at the signal flow analysis and graph theory, which explain the implementation of fast Fourier transform algorithms. We will not get into those details because as a practitioner of structural dynamics it is of more relevance for us to understand the concept of Fourier transform or discrete Fourier transform, and under what conditions it is analogous or it is representative of the continuous wave form and continuous frequency transform.

And what are the warning zones and what are the problem potential problem areas that we need to take care of. Because implementation of Fourier transform algorithm requires no doubt it requires considerable skill, but there are several excellent implementations of Fourier transform fast Fourier transform algorithm, that are available in public domain and users can readily use it.

So, it is worthwhile to spend more time in actually using the fast implementation available implementation of fast Fourier transform to calculate or the results or to for the interpretation of the results, that are available from using those algorithms.

A very powerful tool that is available for public domain is FFT W. So, that is available for free download. And of course, JNU scientific library includes excellent implementation of fast Fourier transform and then you can also download several I mean efficient codes for fast Fourier transfer, Fortran codes from Netlify if required.

So, with that we end our lecture here Fast Fourier Transform. Next lecture we will discuss about some of the applications how these Fourier transform is actually used in structural dynamics, how what we what do we do with this Fourier transform and how it helps in analysis that is not so obvious in other time marching schemes so to say.

Thank you.