Fast Fourier Transform (FFT)

Elena Punskaya
www-sigproc.eng.cam.ac.uk/~op205

Some material adapted from courses by
Prof. Simon Godsill, Dr. Arnaud Doucet,
Dr. Malcolm Macleod and Prof. Peter Rayner

Computations for Evaluating the DFT

- Discrete Fourier Transform and its Inverse can be computed on a digital computer

$$X_p = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi}{N}np}, \quad p \in \{0, 1, ..., N-1\} \tag{1}$$

$$x_n = \frac{1}{N} \sum_{p=0}^{N-1} X_p e^{j\frac{2\pi}{N}np}, \quad n \in \{0, 1, 2, ..., N-1\} \tag{2}$$

- What are computational requirements?

Operation Count

How do we evaluate computational complexity?

- count the number of real multiplications and additions
- by ¨real¨ we mean either fixed-point or floating point operations depending on the specific hardware
- subtraction is considered equivalent to additions
- divisions are counted separately

Other operations such as loading from memory, storing in memory, loop counting, indexing, etc are not counted (depends on implementation/architecture) – present *overhead*

Operation Count – Modern DSP

Modern DSP:

a real multiplication and addition is a single machine cycle $y=ax+b$ called MAC (multiply/accumulate)

maximum number of multiplications and additions must be counted, not their sum

traditional claim ¨multiplication is much more time-consuming than addition¨ becomes obsolete

Discrete Fourier Transform as a Vector Operation

Again, we have a sequence of sampled values x_n and transformed values X_p, given by

$$X_p = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi}{N}np}, \quad p \in \{0, 1, ..., N-1\} \tag{1}$$

Let us denote:

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

Then:

$$X_p = \sum_{n=0}^{N-1} x_n \times W^{np}, \quad p \in \{0, 1, ..., N-1\}$$

Matrix Interpretation of the DFT

This Equation can be rewritten in matrix form:

$$X_p = \sum_{n=0}^{N-1} x_n \times W^{np}, \quad p \in \{0, 1, ..., N-1\}$$

as follows:

$$\begin{bmatrix} X_0 \\ X_1 \\ \dots \\ X_{N-1} \end{bmatrix} = \underbrace{\begin{bmatrix} W^0 & W^0 & W^0 \dots & W^0 \\ W^0 & W^1 & W^2 \dots & W^{N-1} \\ \vdots & \vdots & \vdots \dots & \vdots \\ W^0 & W^{N-1} & W^{2(N-1)} \dots & W^{(N-1)^2} \end{bmatrix}}_{N \times N \text{ matrix}} \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_{N-1} \end{bmatrix}$$

complex numbers

can be computed off-line and stored

DFT Evaluation – Operation Count

Multiplication of a complex $N \times N$ matrix by a complex N-dimensional vector.

No attempt to save operations ⇒

N^2 complex multiplications ("x"s)

$N(N-1)$ complex additions ("+"s)

First row and first column are 1 however ⇒

save $2N-1$ "x"s, $(N-1)^2$ complex "x"s & $N(N-1)$ "+"s

Complex "x": 4 real "x"s and 2 real "+"s;

complex "+" : 2 real "+"s

Total: $4(N-1)^2$ real "x"s & $4(N-0.5)(N-1)$ real "+"s

DFT Computational Complexity

Thus, for the input sequence of length N the number of arithmetic operations in direct computation of DFT is proportional to N^2.

For $N=1000$, about a *million* operations are needed!

In 1960s such a number was considered prohibitive in most applications.

Discovery of the Fast Fourier Transform (FFT)

When in 1965 Cooley and Tukey ¨first¨ announced discovery of Fast Fourier Transform (FFT) in 1965 it revolutionised Digital Signal Processing.

They were actually 150 years late – the principle of the FFT was later discovered in obscure section of one of Gauss' (as in Gaussian) own notebooks in 1806.

Their paper is the most cited mathematical paper ever written.

The Fast Fourier Transform (FFT)

The FFT is a highly elegant and efficient algorithm, which is still one of the most used algorithms in speech processing, communications, frequency estimation, etc – one of the most highly developed area of DSP.

There are many different types and variations.

They are just computational schemes for computing the DFT – not new transforms!

Here we consider the most basic radix-2 algorithm which requires N to be a power of 2.

FFT Derivation 1

Lets take the basic DFT equation: $$X_p = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi}{N}np}$$

Now, split the summation into two parts: one for even n and one for odd n:

$$\begin{aligned} X_p &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \, e^{-j\frac{2\pi}{N}(2n)p} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{N}(2n+1)p} \quad (*) \\ &= \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}np} + e^{-j\frac{2\pi}{N}p} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}np} \\ &= A_p + W^p B_p \end{aligned}$$

Take outside

where

$$A_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}np}$$

$$B_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}np}$$

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

FFT Derivation 2

Thus, we have:

$$A_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}np}$$

$$B_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}np}$$

- Notice now that A_p and B_p are themselves DFTs each of length $N/2$:
  - A_p is the DFT of a sequence $\{x_{2n}\} = \{x_0, x_2, ...x_{N-4}, x_{N-2}\}$
  - B_p is the DFT of a sequence $\{x_{2n+1}\} = \{x_1, x_3, ...x_{N-3}, x_{N-1}\}$

- We know, however that the DFT is periodic in the frequency domain (in this case with period N/2). This leads to further simplifications, as follows.

FFT Derivation 3

Now, take the same equation again where we split the summation into two parts: one for even n and one for odd n:

$$X_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n}\, e^{-j\frac{2\pi}{N}(2n)p} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{N}(2n+1)p} \quad (*)$$

And evaluate at frequencies $p+N/2$

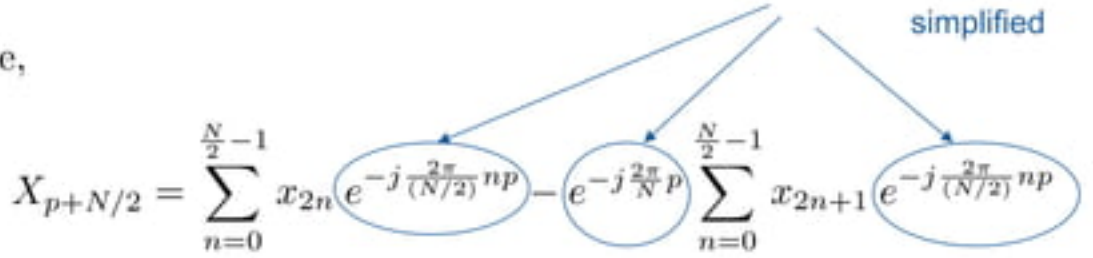
$$X_{p+N/2} = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}n(p+N/2)} + e^{-j\frac{2\pi}{N}(p+N/2)} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}n(p+\frac{N}{2})}$$

Now, simplify terms as follows:

$$e^{-j\frac{2\pi}{(N/2)}n(p+N/2)} = e^{-j\frac{2\pi}{(N/2)}np}, \quad e^{-j\frac{2\pi}{N}(p+N/2)} = -e^{-j\frac{2\pi}{N}p}$$

FFT Derivation 4

Hence,



$$X_{p+N/2} = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}np} - e^{-j\frac{2\pi}{N}p} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}np}$$

$$= A_p - W^p B_p$$

with A_p W^p and B_n defined as before

$$A_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}np}$$

$$B_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}np}$$

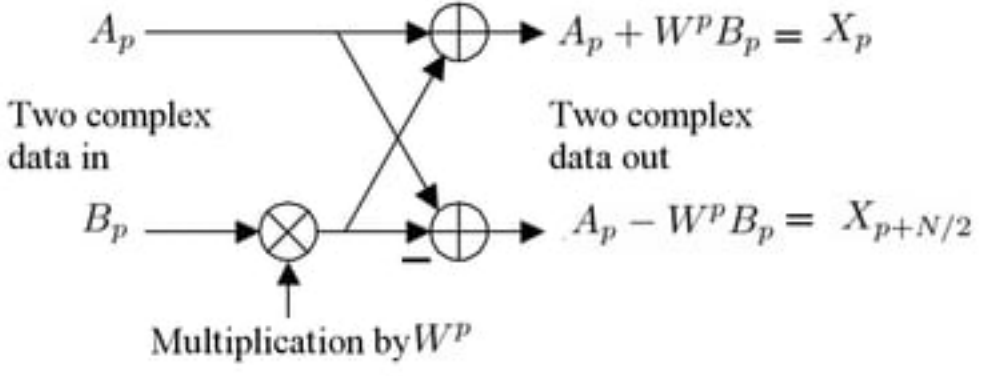
$$W = e^{-j\frac{2\pi}{N}}$$

FFT Derivation 5

Now, compare the equations for $X_{p+N/2}$ with that for X_p:

$$X_p = A_p + W^p B_p, \qquad X_{p+N/2} = A_p - W^p B_p$$

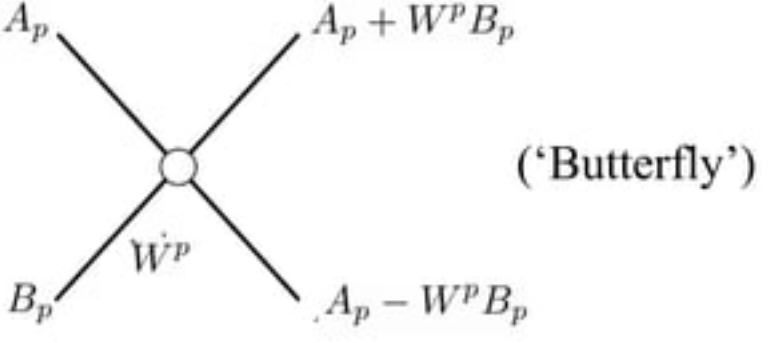
This defines the FFT butterfly structure:



FFT Derivation - Butterfly

Or, in more compact form:

$$X_p = A_p + W^p B_p, \quad X_{p+N/2} = A_p - W^p B_p$$



('Butterfly')

FFT Derivation - Redundancy

Look at the two required terms (W^p assumed precomputed and stored):

$$A_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}np}, \quad B_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}np},$$

- The terms A_p and B_p need only be computed for $p = 0, 2, ..., N/2 - 1$, since $X_{p+N/2}$ has been expressed in terms of A_p and B_p - hence we have uncovered redundancy in the DFT computation.
- Thus calculate the A_p and B_p for $p = 0, 1, ..., N/2 - 1$ and use them for calculation of both X_p and $X_{p+N/2}$

FFT Derivation - Computational Load

$$A_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\left(\frac{2\pi}{N/2}\right)np}$$

$$B_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\left(\frac{2\pi}{N/2}\right)}$$

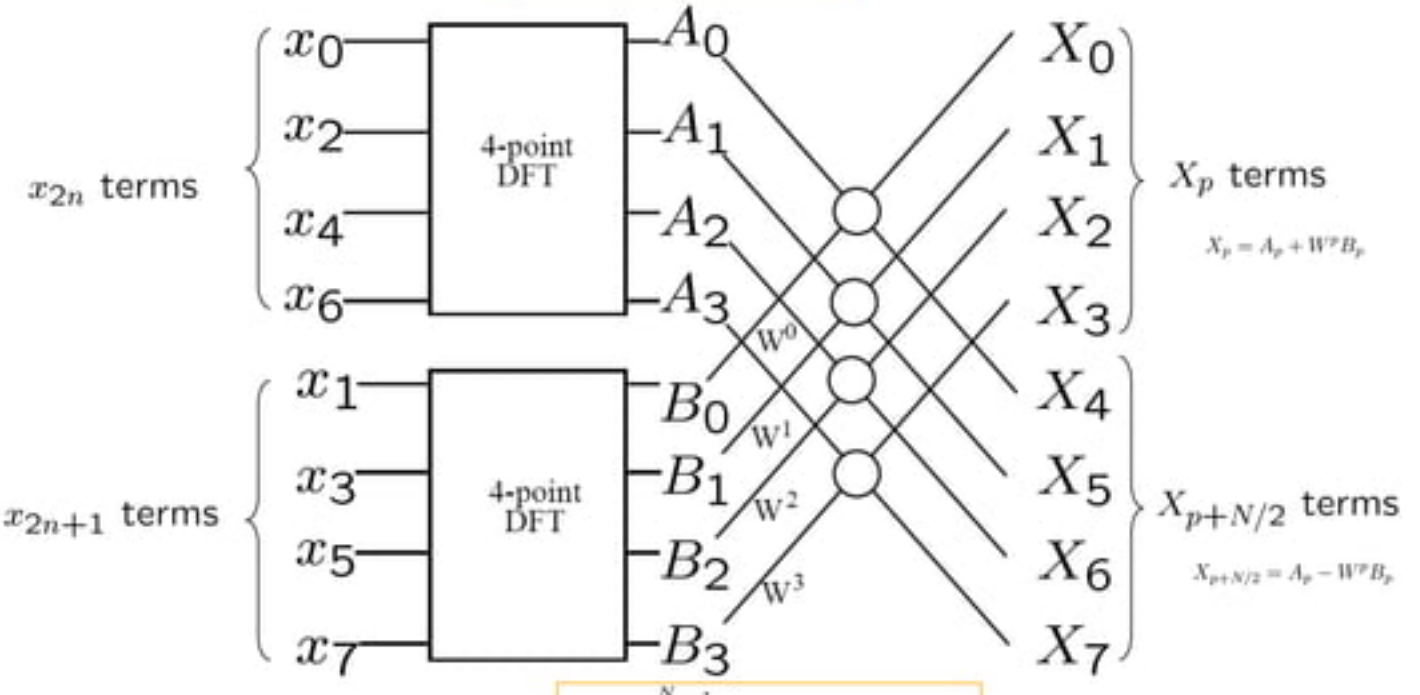
- The number of complex multiplies and additions is:
  - A_p requires $N/2$ complex multiplies and additions; so does B_p. The total for all $p = 0, 1, ..., N/2-1$ is then $2(N/2)^2$ multiplies and additions for the calculation of all the A_p and B_p terms.
  - $N/2$ multiplies for the calculation of $W^p B_p$ for all $p = 0, 1, 2, ..., N/2-1$
  - $N = N/2 + N/2$ additions for calculation of $A_p + W^p B_p$ and $A_p - W^p B_p$

for particular p

- Thus total number of complex multiplies and additions is approximately $N^2/2$ for large N
- The computation is approximately halved compared to the direct DFT evaluation

Flow Diagram for a N=8 DFT

$$A_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{(N/2)}np}$$



$$B_p = \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{(N/2)}np}$$

Further decomposition

We obtained $X_p = A_p + W^p B_p, \quad X_{p+N/2} = A_p - W^p B_p$

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

Now A_0, A_1, A_2, A_3 - $N/2$-point DFT

$$A_p = \alpha_p + W_{N/2}^p \beta_p \qquad A_{p+N/2} = \alpha_p - W_{N/2}^p \beta_p$$

- using redundancy just discovered

B_0, B_1, B_2, B_3 - $N/2$-point DFT

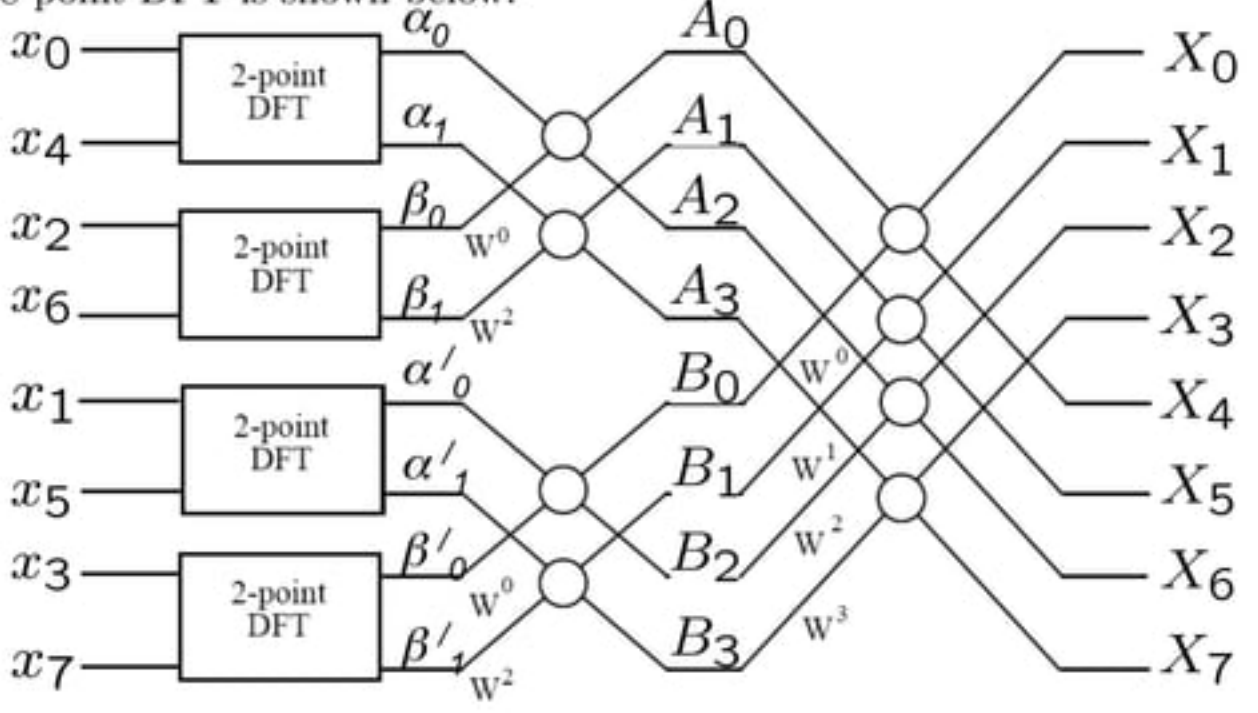
$$B_p = \alpha'_p + W_{N/2}^p \beta'_p \qquad B_{p+N/2} = \alpha'_p - W_{N/2}^p \beta'_p$$

- using redundancy just discovered

Finally $W_{N/2}^p = e^{-j2\pi p/N/2} = e^{-j2\pi 2p/N} = W_N^{2p}$

Flow Diagram for a N=8 DFT - Decomposition

Assuming that $(\frac{N}{2})$ is even, the same process can be carried out on each of the $(\frac{N}{2})$ point DFTs to further reduce the computation. The flow diagram for incorporating this extra stage of decomposition into the computation of the $N = 8$ point DFT is shown below.



Further decomposition

We obtained $X_p = A_p + W^p B_p, \quad X_{p+N/2} = A_p - W^p B_p$

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

Now A_0, A_1, A_2, A_3 - $N/2$-point DFT

$$A_p = \alpha_p + W_{N/2}^p \beta_p \qquad A_{p+N/2} = \alpha_p - W_{N/2}^p \beta_p$$

- using redundancy just discovered

B_0, B_1, B_2, B_3 - $N/2$-point DFT

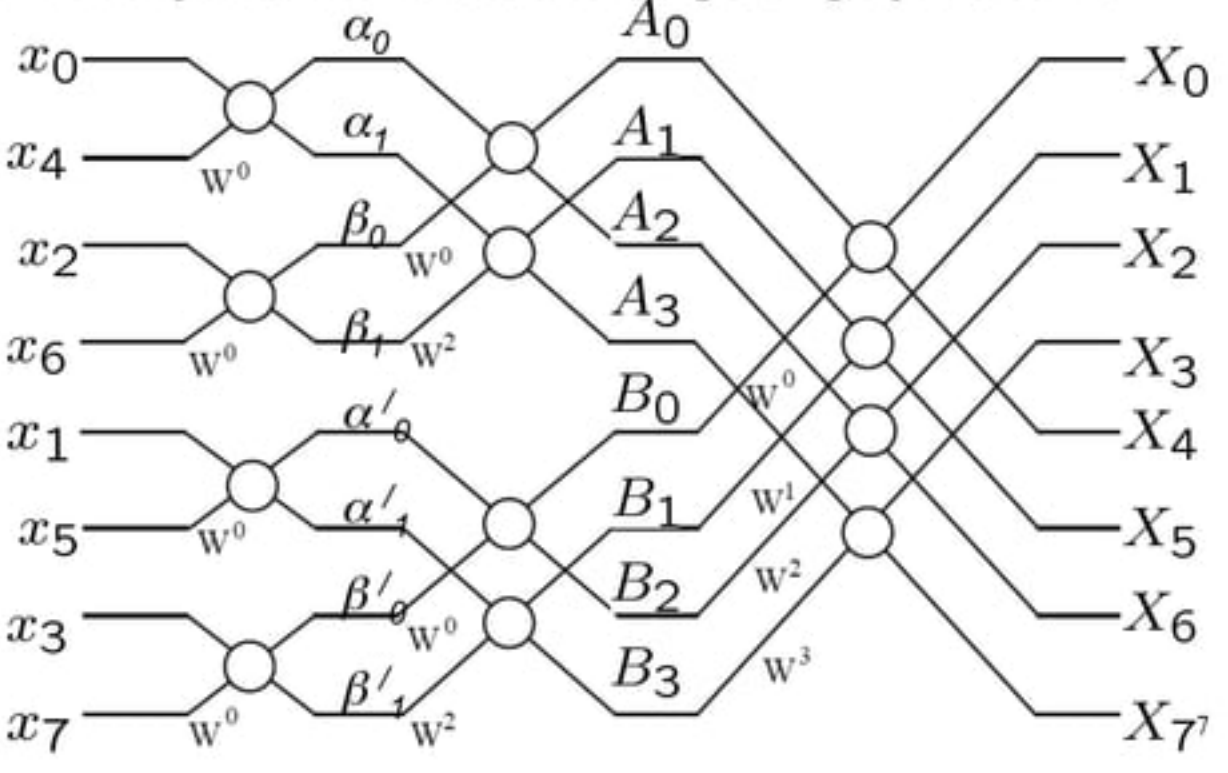
$$B_p = \alpha'_p + W_{N/2}^p \beta'_p \qquad B_{p+N/2} = \alpha'_p - W_{N/2}^p \beta'_p$$

- using redundancy just discovered

Finally $W_{N/2}^p = e^{-j2\pi p/N/2} = e^{-j2\pi 2p/N} = W_N^{2p}$

Flow Diagram for a N=8 DFT – Decomposition 2

It can be seen that if $N = 2^M$ then the process can be repeated M times to reduce the computation to that of evaluating N single point DFTs.



Bit-reversal

Examination of the final chart shows that it is necessary to shuffle the order of the input data. This data shuffle is usually termed *bit-reversal* for reasons that are clear if the indices of the shuffled data are written in binary.

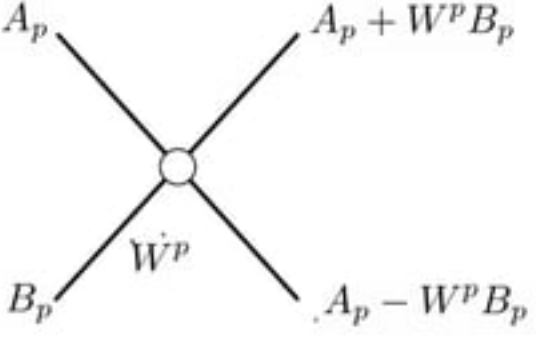
Binary	Bit Reverse	Decimal
000	000	0
001	100	4
010	010	2
011	110	6
100	001	1
101	101	5
110	011	3
111	111	7

FFT Derivation Summary

- The FFT derivation relies on redundancy in the calculation of the basic DFT
- A recursive algorithm is derived that repeatedly rearranges the problem into two simpler problems of half the size
- Hence the basic algorithm operates on signals of length a power of 2, i.e.

$$N = 2^M$$ (for some integer M)

- At the bottom of the tree we have the classic FFT `butterfly' structure



Computational Load of full FFT algorithm

The type of FFT we have considered, where $N = 2^M$, is called a radix-2 FFT.

It has $M = \log_2 N$ stages, each using $N / 2$ butterflies

Complex multiplication requires 4 real multiplications and 2 real additions

Complex addition/subtraction requires 2 real additions

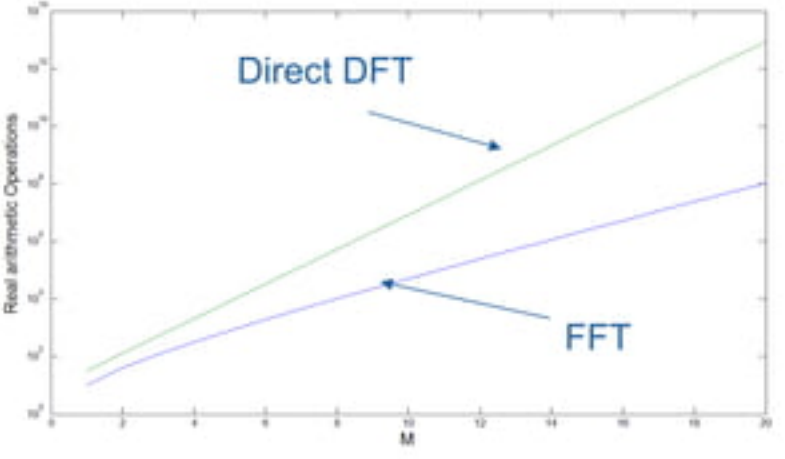
Thus, butterfly requires 10 real operations.

Hence the radix-2 N-point FFT requires $10(N / 2)\log_2 N$ real operations compared to about $8N2$ real operations for the DFT.

Computational Load of full FFT algorithm

The radix-2 N-point FFT requires $10(N/2)\log_2 N$ real operations compared to about $8N^2$ real operations for the DFT.

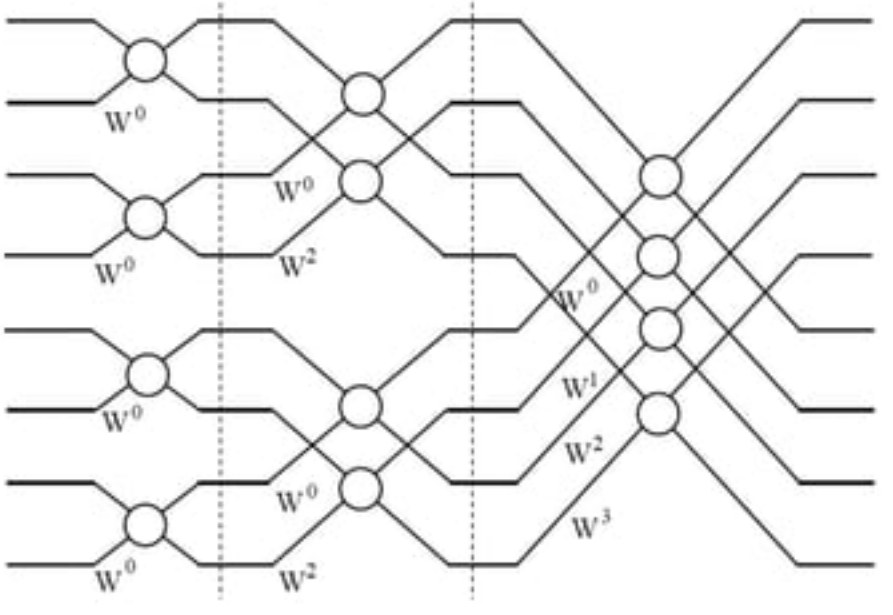
This is a huge speed-up in typical applications, where N is 128 – 4096:



Further advantage of the FFT algorithm

The FFT algorithm has a further significant advantage over direct evaluation of the DFT expression in that computation can be performed *in-place*. This is best illustrated in the final flow chart where it can be seen that after two data values have been processed by the *butterfly* structure, those data are not required again in the computation and they may be replaced, in the computer or in the chip memory, with the values at the output of the *butterfly* structure.

Input



Output

The Inverse FFT (IFFT)

The IDFT is different from the DFT:

$$X_p = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi}{N}np}, \quad p \in \{0, 1, ..., N-1\} \tag{1}$$

$$x_n = \frac{1}{N} \sum_{p=0}^{N-1} X_p e^{j\frac{2\pi}{N}np}, \quad n \in \{0, 1, 2, ..., N-1\} \tag{2}$$

- it uses positive power of W^p instead of negative ones
- There is an additional division of each output value by N

Any FFT algorithm can be modified to perform the IDFT by

- using positive powers instead of negatives
- Multiplying each component of the output by $1/N$

Hence the algorithm is the same but computational load increases due to N extra multiplications

Many types of FFT

The form of FFT we have described is called ***"decimation in time";*** there is a form called ***"decimation in frequency"*** (but it has no advantages).

The **"radix 2"** FFT must have length *N* a power of 2. Slightly more efficient is the **"radix 4"** FFT, in which 2-input 2-output butterflies are replaced by 4-input 4-output units. The transform length must then be a power of 4 (**more restrictive**).

A completely different type of algorithm, the Winograd Fourier Transform Algorithm (WFTA), can be used for FFT lengths equal to the product of a number of mutually prime factors (e.g. 9*7*5 = 315 or 5*16 = 80). The WFTA uses fewer multipliers, but more adders, than a similar-length FFT.

Efficient algorithms exist for FFTing **real (not complex) data at about 60% the effort of the same-sized complex-data FFT.**

The Discrete Cosine and Sine Transforms (**DCT and DST) are similar real-signal algorithms used in image coding.**

Applications of the FFT

There FFT is surely the most widely used signal processing algorithm of all. It is the basic building block for a large percentage of algorithms in current usage

Specific examples include:

- Spectrum analysis – used for analysing and detecting signals
- Coding – audio and speech signals are often coded in the frequency domain using FFT variants (MP3, …)
- Another recent application is in a modulation scheme called OFDM, which is used for digital TV broadcasting (DVB) and digital radio (audio) broadcasting (DAB).
- Background noise reduction for mobile telephony, speech and audio signals is often implemented in the frequency domain using FFTs

Practical Spectral Analysis

- Say, we have a symphony recording over 40 minutes long – about 2500 seconds
- Compact-disk recordings are sampled at 44.1 kHz and are in stereo, but say we combine both channels into a single one by summing them at each time point
- Are we to compute the DFT of a sequence one hundred million samples long?

a) unrealistic – speed and storage requirements are too high

b) useless – we will get a very much noiselike wide-band spectrum covering the range from 20 Hz to 20 kHz at high resolution including all notes of all instruments with their harmonics

Short-Time Spectral Analysis

- We actually want a sequence of short DFTs, each showing the spectrum of a signal of relatively short interval
- If violin plays note E during this interval – spectrum would exhibit energies at ~~not~~ the frequency of note E (329 Hz) and characteristic harmonics of the violin
- If the interval is short enough – we may be able to track note-to-note changes
- If the frequency resolution is high enough – we will be able to identify musical instruments playing

Remember: our – ear – excellent spectrum analyser?

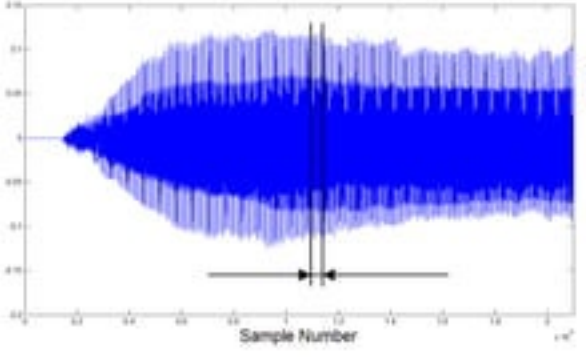
Short-Time Spectral Analysis – Computational Load

- If we take a time interval of about 93 milliseconds (signal of length 4096) the frequency resolution will be 11 Hz (adequate)
- The number of distinct intervals in the symphony is 40 x 60/0.093 - about 26,000
- We may choose to be conservative and overlap the intervals (say 50%)
- In total, need to compute 52,000 FFTs of length 4096

the entire computation will be done in considerably less time than the music itself

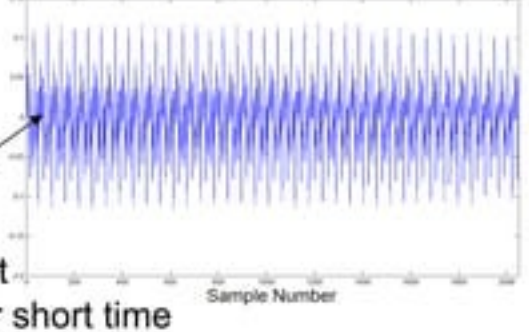
Short-time spectral analysis also use windowing (more on this later)

Case Study: Spectral analysis of a Musical Signal



Sample rate is
10.025 kHz
(T=1/10,025 s)

Extract a short segment



Looks almost periodic over short time interval

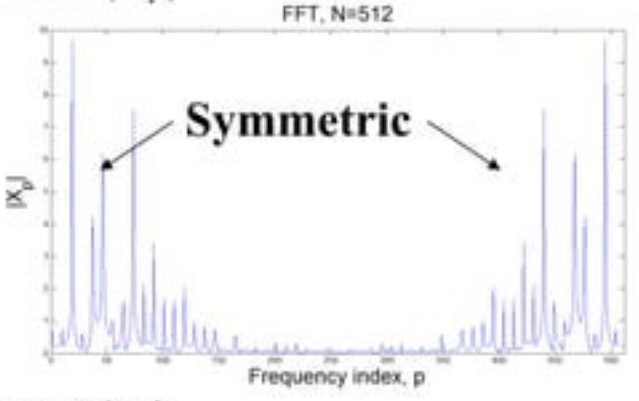
Load this into Matlab as a vector x

Take an FFT, N=512

X=fft(x(1:512));

Case Study: Spectral analysis of a Musical Signal, FFT

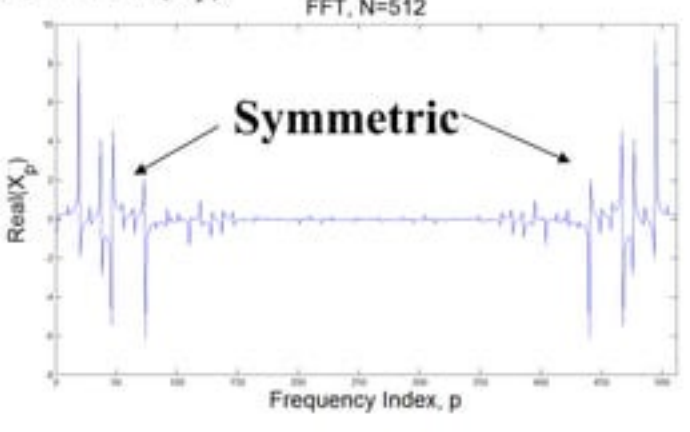
Modulus $|X_p|$



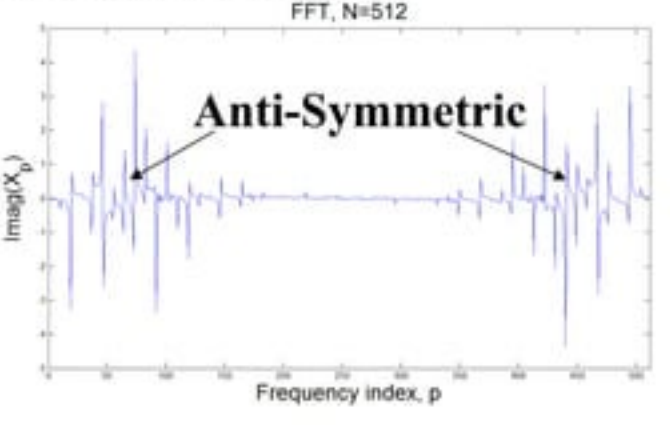
Note Conjugate symmetry as data are real:

$$X_p = X^*_{N-p}$$

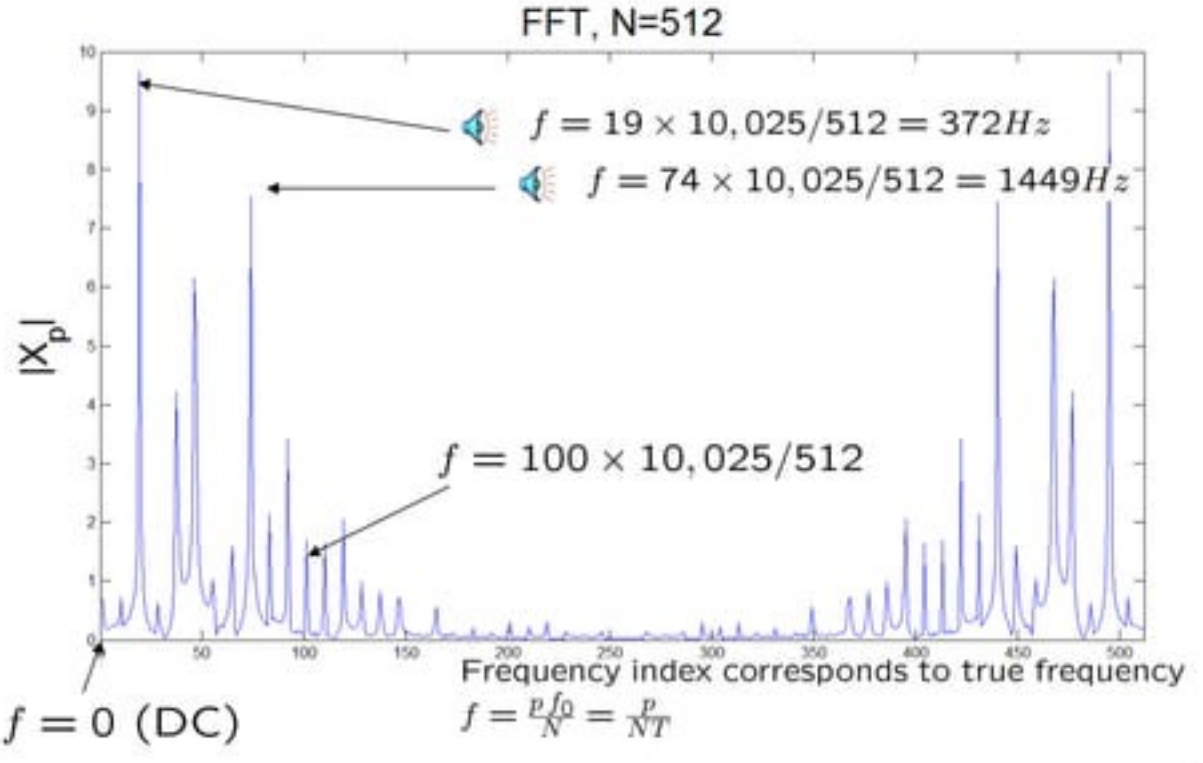
Real Part $\Re(X_p)$



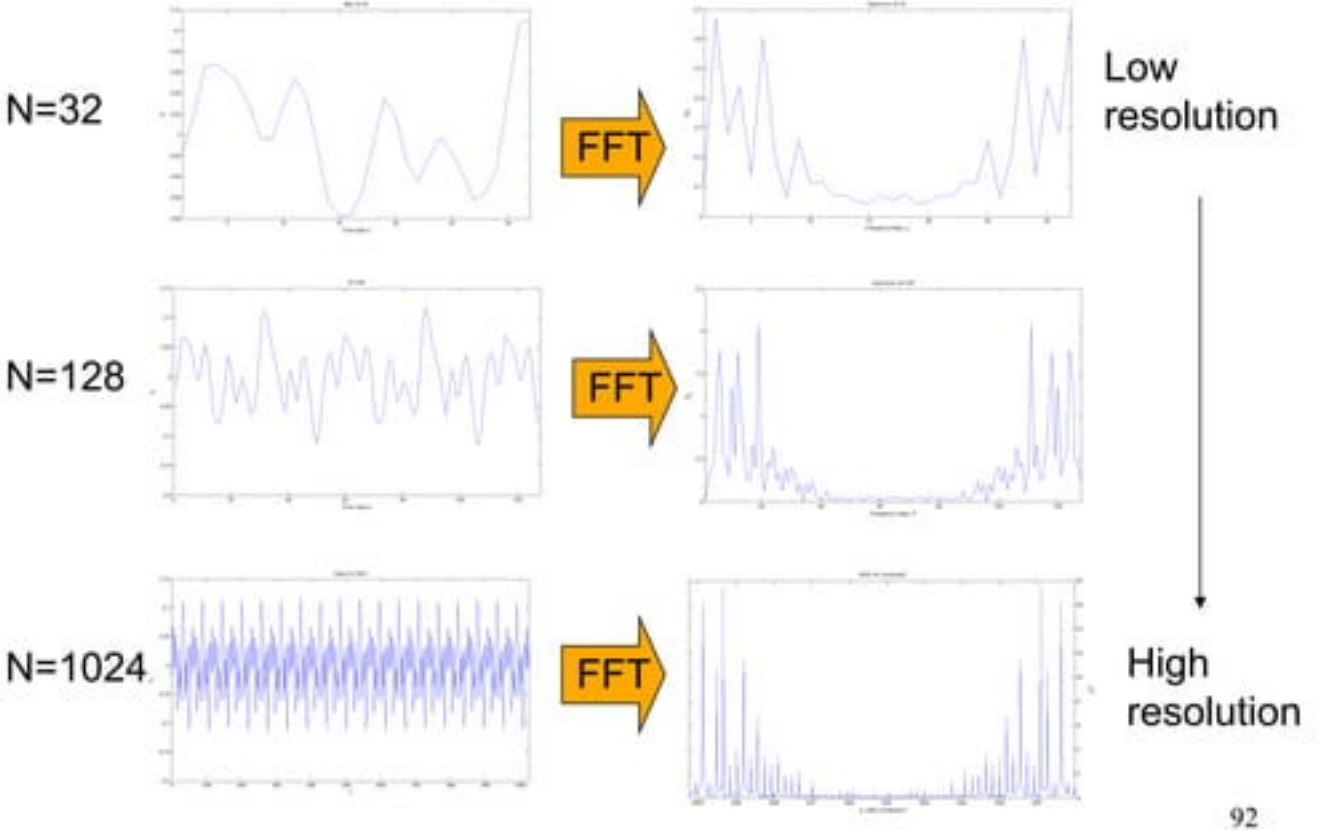
Imaginary Part $\Im(X_p)$



Case Study: Spectral analysis of a Musical Signal - Frequencies



The Effect of data length N



The DFT approximation to the DTFT

DTFT at frequency $\omega = \dfrac{2\pi p}{NT}$

$$X(e^{j\frac{2\pi}{N}p}) = \sum_{n=-\infty}^{\infty} x_n e^{-j\frac{2\pi}{N}np}$$

DFT:

$$X_p = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}np}$$

- Ideally the DFT should be a `good' approximation to the DTFT
- Intuitively the approximation gets better as the number of data points *N* increases
- This is illustrated in the previous slide – resolution gets better as *N* increases (more, narrower, peaks in spectrum).
- How to evaluate this analytically?

  View the truncation in the summation as a multiplication by a rectangle window function

Analysis

Consider DTFT:

$$X(e^{j\omega T}) = \sum_{n=-\infty}^{\infty} x_n \, e^{-jn\omega T}$$

Truncate the summation, as for the DFT:

$$X_w(e^{j\omega T}) = \sum_{n=0}^{N-1} x_n \, e^{-jn\omega T}$$

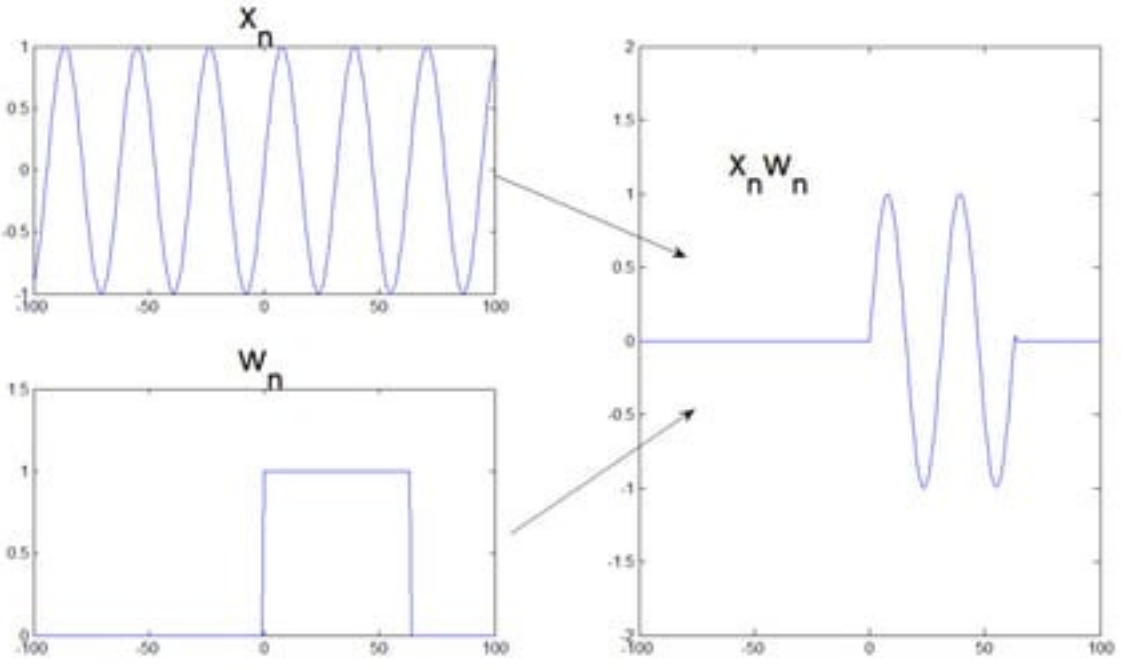
Now, note that this is equivalent to an infinite summation, but with x_n pre-multiplied by a rectangle window function:

$$X_w(e^{j\omega T}) = \sum_{n=-\infty}^{\infty} w_n x_n \, e^{-jn\omega T}$$

with

$$w_n = \begin{cases} 1, & n = 0, 1, 2, ..., N-1 \\ 0, & \text{otherwise} \end{cases}$$

Pre-multiplying by rectangular window



Spectrum of the windowed signal

Now, take the DTFT of the windowed signal $x_w = w_n x_n$ directly:

$$X_w(e^{j\omega T}) = \sum_{n=-\infty}^{\infty} \{x_n\, w_n\}\, e^{-jn\omega T}$$

DTFT of w_n is $W(e^{j\theta})$

$$= \sum_{n=-\infty}^{\infty} x_n \left\{ \frac{1}{2\pi} \int_0^{2\pi} W(e^{j\theta}) e^{jn\theta} d\theta \right\} e^{-jn\omega T}$$

Inverse DTFT of $W(e^{j\theta})$

$$= \frac{1}{2\pi} \int_0^{2\pi} W(e^{j\theta}) \sum_{n=-\infty}^{\infty} x_n\, e^{-jn(\omega T - \theta)}\, d\theta$$

$$X_w(e^{j\omega T}) = \frac{1}{2\pi} \int_0^{2\pi} W(e^{j\theta})\, X(e^{j(\omega T - \theta)})\, d\theta$$

We see that the spectrum of the windowed signal is the convolution of the infinite duration signal spectrum and the window spectrum.

Fourier Transform of the Rectangular Window

What is the DTFT of the window w_n?

[Subst. $\theta = \omega T$ - makes no difference to form of results]:

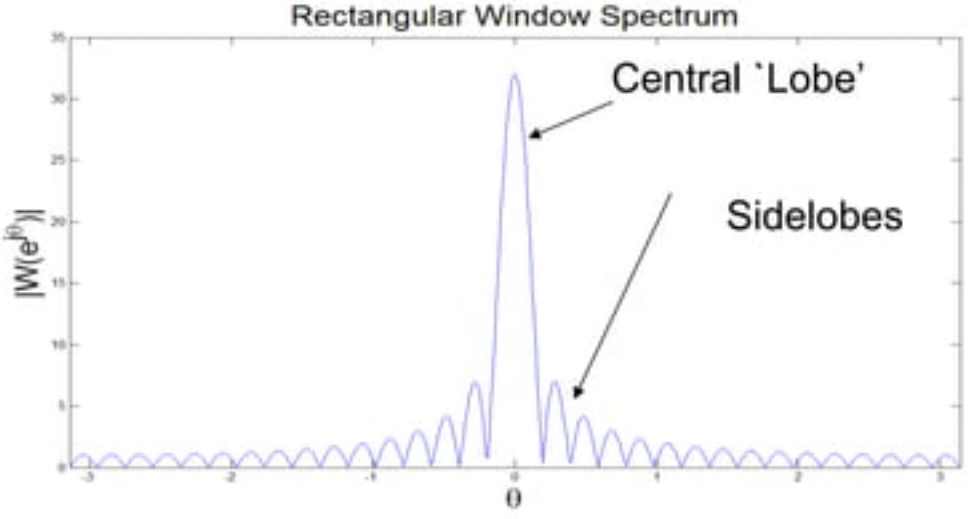
$$W(e^{j\theta}) = \sum_{n=0}^{N-1} 1.e^{-jn\theta}$$

$$= e^{-j(N-1)\theta/2} \frac{\sin(N\theta/2)}{\sin(\theta/2)}$$

[Check you can get this result yourself as an extra examples question]

Rectangular Window Spectrum

N=32



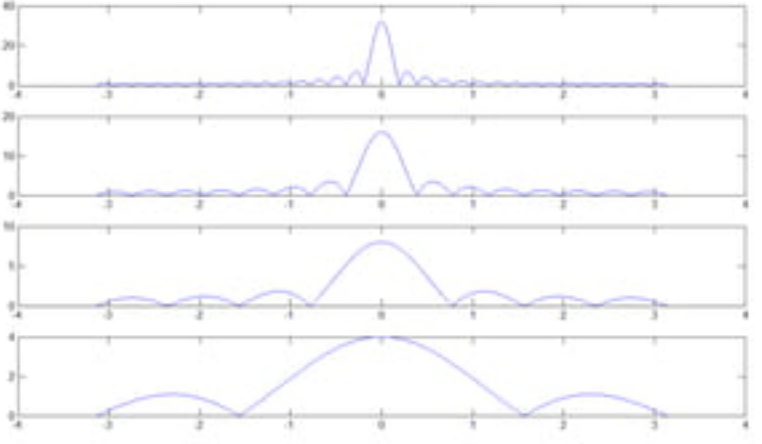
Rectangular Window Spectrum – Lobe Width

N=32

N=16

N=8

N=4



Lobe width inversely
Proportional to N

Now, imagine what happens when the sum of two frequency components is DFT-ed:

$$x_n = \exp(j\omega_1 nT) + \exp(j\omega_2 nT)$$

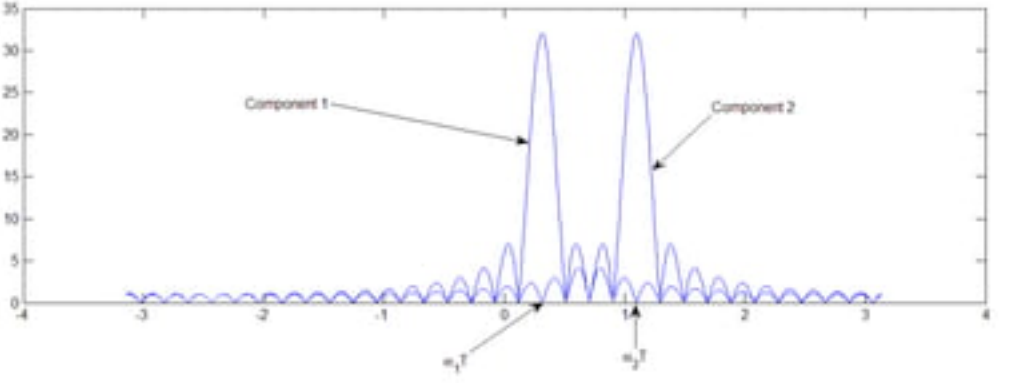
The DTFT is given by a train of delta functions:

$$X(e^{j\omega T}) = 2\pi \sum_{n=-\infty}^{+\infty} \delta(\omega T + 2n\pi - \omega_1 T) + \delta(\omega T + 2n\pi - \omega_2 T)$$

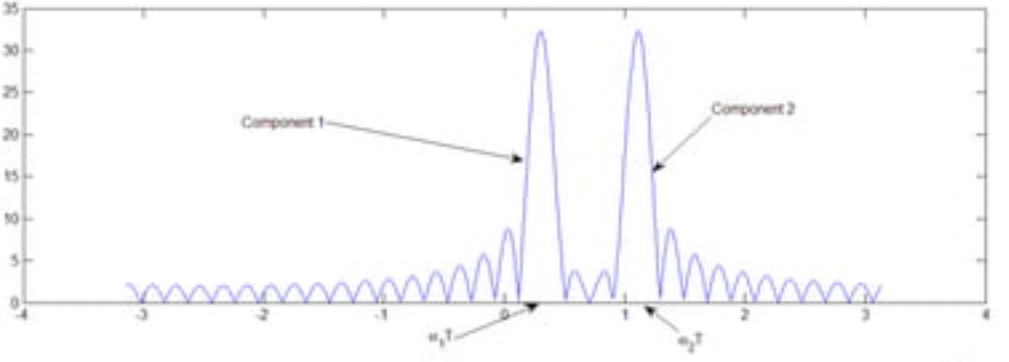
Hence the windowed spectrum is just the convolution of the window spectrum with the delta functions:

DFT for the data

Both components separately



Both components Together



Brief summary

- The rectangular window introduces broadening of any frequency components (`smearing') and sidelobes that may overlap with other frequency components (`leakage').
- The effect improves as N increases
- However, the rectangle window has poor properties and better choices of w_n can lead to better spectral properties (less leakage, in particular) – i.e. instead of just truncating the summation, we can pre-multiply by a suitable window function w_n that has better frequency domain properties.
- More on window design in the filter design section of the course – see later

Summary so far …

- The Fourier Transform (FT) is a way of transforming a continuous signal into the frequency domain
- The Discrete Time Fourier Transform (DTFT) is a Fourier Transform of a sampled signal
- The Discrete Fourier Transform (DFT) is a discrete numerical equivalent using sums instead of integrals that can be computed on a digital computer
- The FFT is a highly elegant and efficient algorithm, which is still one of the most used algorithms in speech processing, communications, frequency estimation, etc – one of the most highly developed area of DSP

Thank you!