Introduction to Fourier Transforms

Iain Bethune, Gavin Pringle, Joachim Hein
EPCC
The University of Edinburgh

Lectures over next week

- Iain Bethune – FFT

  – Wed 30th Sep – Lecture: Intro to FFTs
  – Mon 5th Oct – Lecture: FFT Libraries
  – Tue 6th Oct – Practical: FFT image processing
  – Wed 7th Oct – Lecture: Parallel FFTs

Overview

- The Fourier Transform
  - Who, what, why?
  - Fourier Series
  - Mathematical properties of the Fourier Transform
- Discrete Fourier Transform
  - Introduction to first exercise
- Fast Fourier Transform
  - A brief overview
  - Worked example of 4-point DFT

Fourier Transfoms

- Jean Baptiste Joseph Fourier (1768-1830) first employed what we now call Fourier Transforms whilst working on the theory of heat
  - The Fourier transform first appeared in “On the Propagation of Heat in Solid Bodies”, memoir to Paris Institute, 21 Dec., 1807.
- Linear Transform which takes temporal or spatial information and converts into information which lies in the frequency domain
  - And visa versa
  - Frequency domain also known as Fourier space, Reciprocal space, or G-space -> “Spectral Methods”
- Mathematical tool which alters the problem to one which is more easily solved

Pictures of Joseph Fourier

Who would use Fourier Transforms?

- Physical Sciences
  - Cosmology (P^3M N-body solvers)
  - Fluid mechanics
  - Computational Chemistry (See L09)
  - Quantum physics
  - Signal and image processing
    - Antenna studies
    - Optics
- Numerical analysis
  - Linear systems analysis
  - Boundary value problems
  - Large integer multiplication (Prime finding)
- Statistics
  - Random process modelling
  - Probability theory

Caveat: different disciplines use different notation, normalisation, and sign conventions

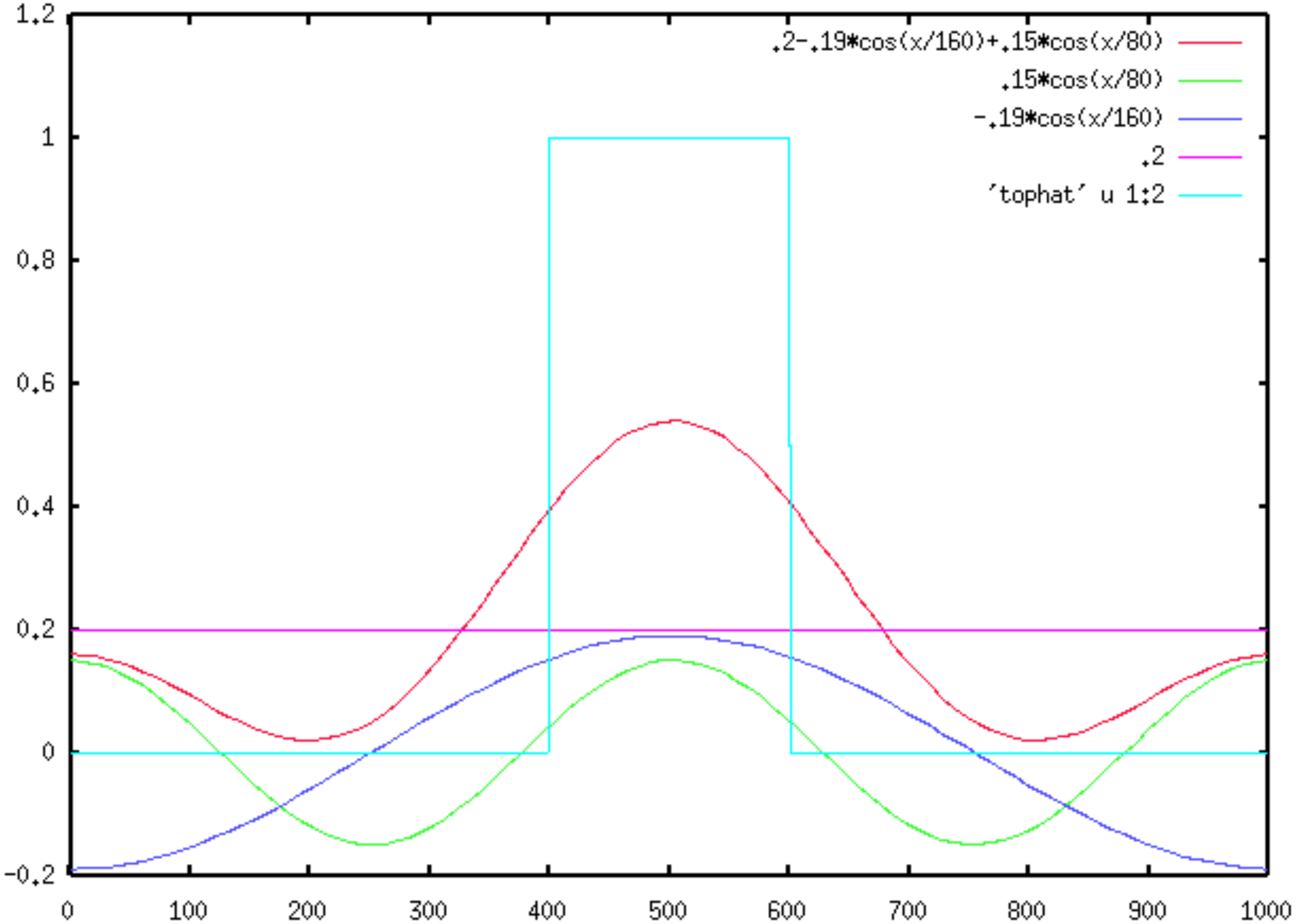
Fourier's Theorem

- Fourier's Theorem:
  - All periodic signals may be represented by an infinite sum of sines and cosines of different periods and amplitudes.
- The cosines and sines are associated with the symmetrical and asymmetric information, respectively
- Fourier Transforms encode this information via

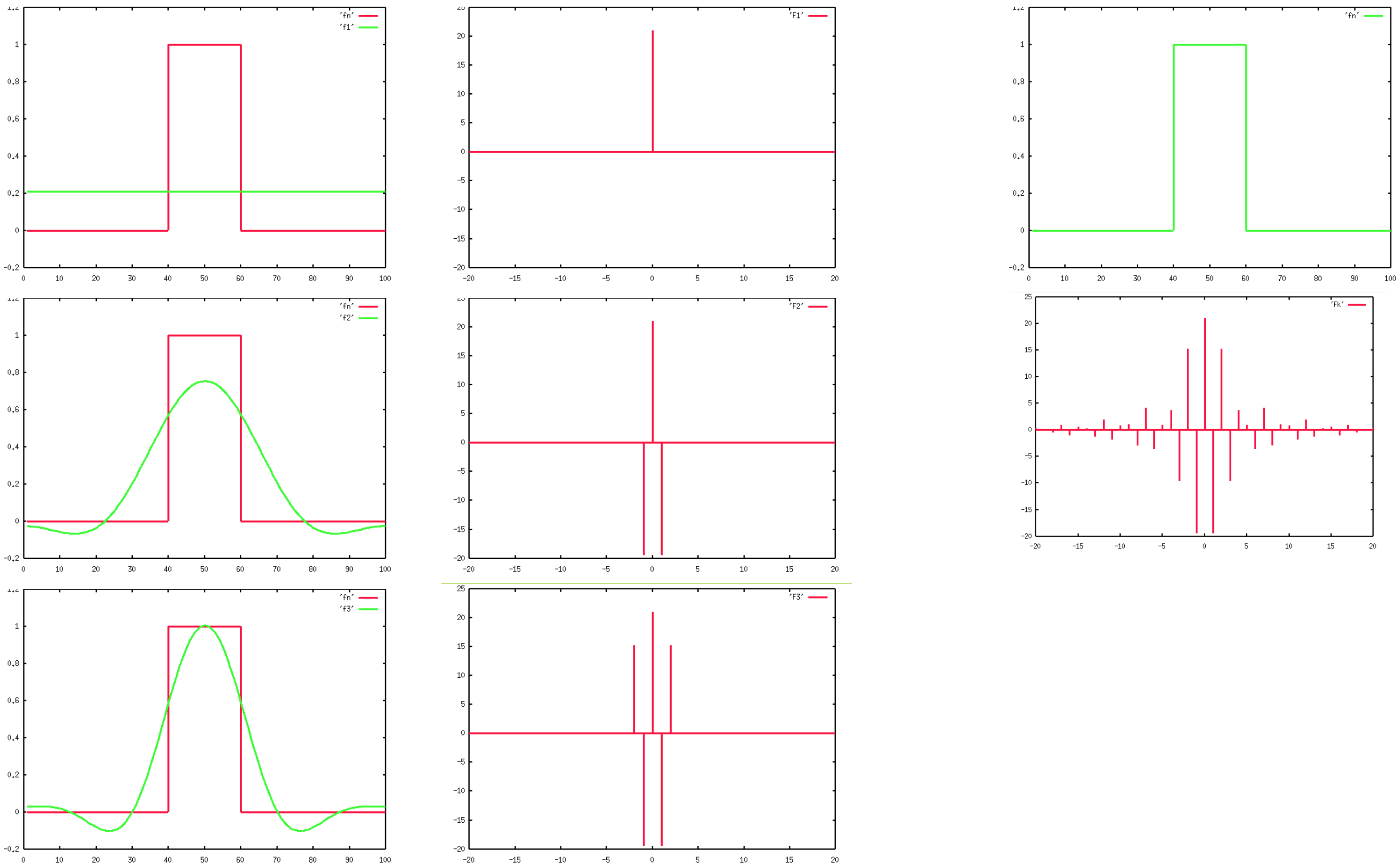
$$e^{i\theta} = \cos\theta + i\sin\theta$$

- NB: Any signal may be considered periodic, by replicating the non-zero part to infinity.

- The top hat function, along with the individual 1st, 2nd and 3rd Fourier components and their sum.



Example: The Top Hat Function



Mathematics of the Fourier Transform

- The Fourier Transform of a complex function *f (x)* is given as

$$F(s) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi xs} dx$$

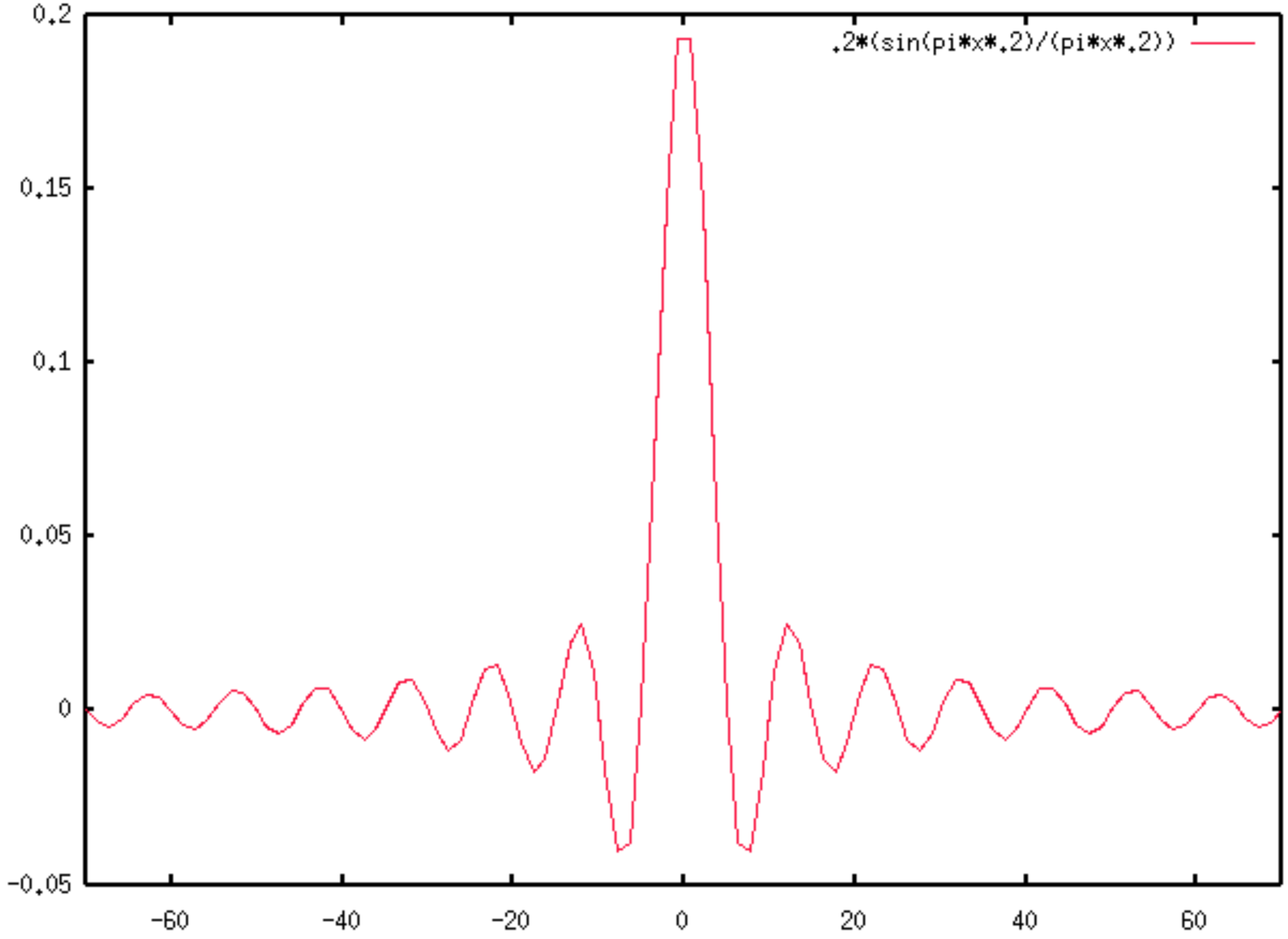
- The inverse Fourier Transform is given as

$$f(x) = \int_{-\infty}^{\infty} F(s) e^{i2\pi xs} ds$$

- The Fourier pair is defined as

$$f(x) <=> F(s)$$

Example: The Top Hat Function



Properties: Scaling

- Time scaling

$$f(at) \Leftrightarrow \frac{1}{|a|} F\left(\frac{s}{a}\right)$$

- Frequency scaling

$$\frac{1}{|b|} f\left(\frac{t}{b}\right) \Leftrightarrow F(bs)$$

Properties: Shifting

- Time shifting

$$f(t - t_0) \Leftrightarrow F(s)e^{2\pi i s t_0}$$

- Frequency shifting

$$f(t)e^{-2\pi i s_0 t} \Leftrightarrow F(s - s_0)$$

Properties: Convolution Theorem

- Say we have two functions, $g(t)$ and $h(t)$, then the convolution of the two functions is defined as

$$g \otimes h \equiv \int_{-\infty}^{\infty} g(\tau)h(t-\tau)d\tau$$

- The Fourier Transform of the convolution is simply the product of the individual Fourier Transforms

$$g \otimes h \Leftrightarrow G(s)H(s)$$

- The correlation of the two functions is defined by

$$Corr(g,h) \equiv \int_{-\infty}^{\infty} g(\tau + t)h(\tau)d\tau$$

- The Fourier Transform of the correlation is simply

$$Corr(g,h) \Leftrightarrow G(s)H(-s)$$

Discrete Fourier Transform

- The Discrete Fourier Transform of N complex points f_k is defined as

$$F_n \equiv \sum_{k=0}^{N-1} f_k e^{2\pi i k n / N}$$

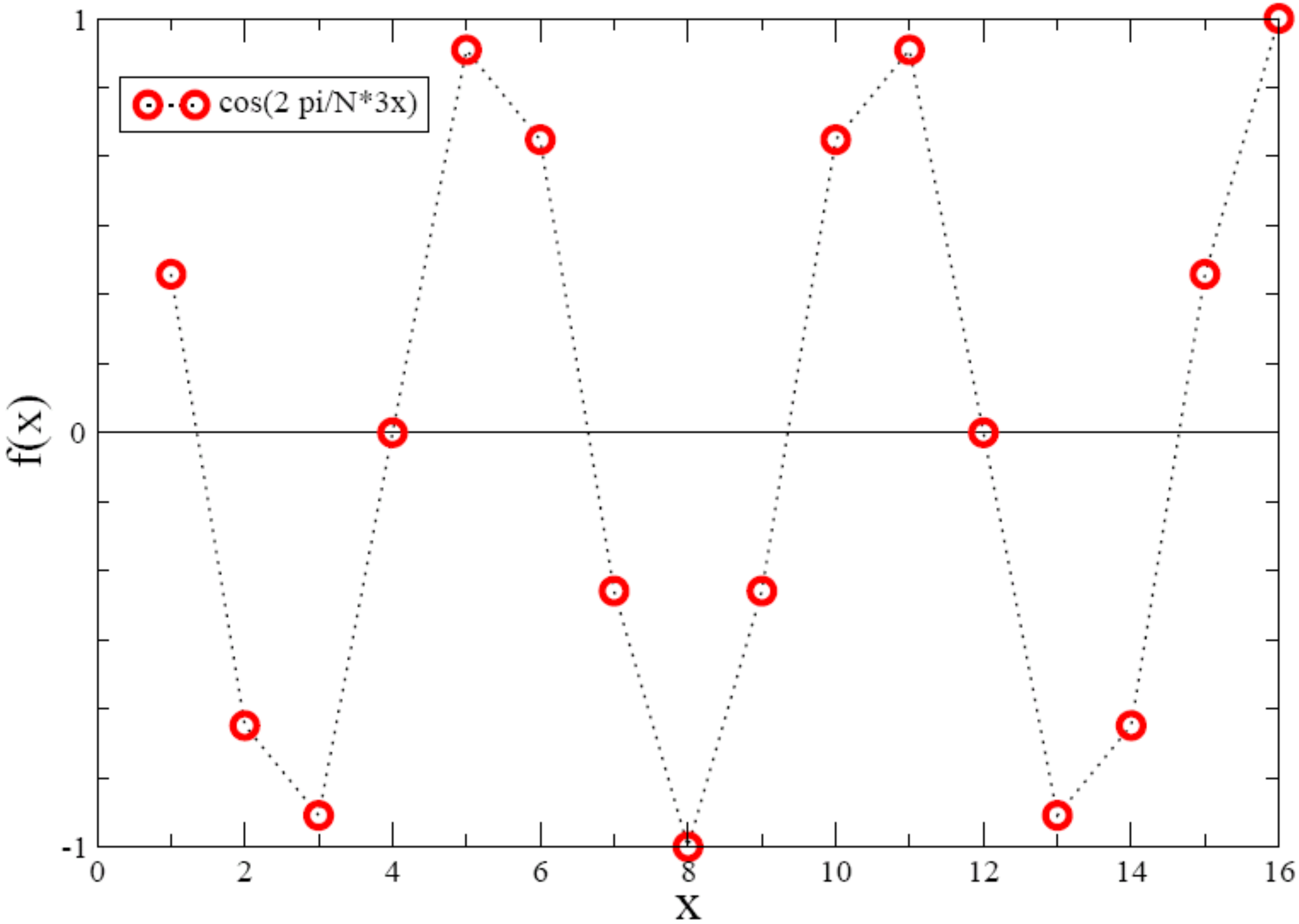
- The inverse Discrete Fourier Transform, which recovers the set of f_k s exactly from F_ns is

$$f_k = \frac{1}{N} \sum_{n=0}^{N-1} F_n e^{-2\pi i k n / N}$$

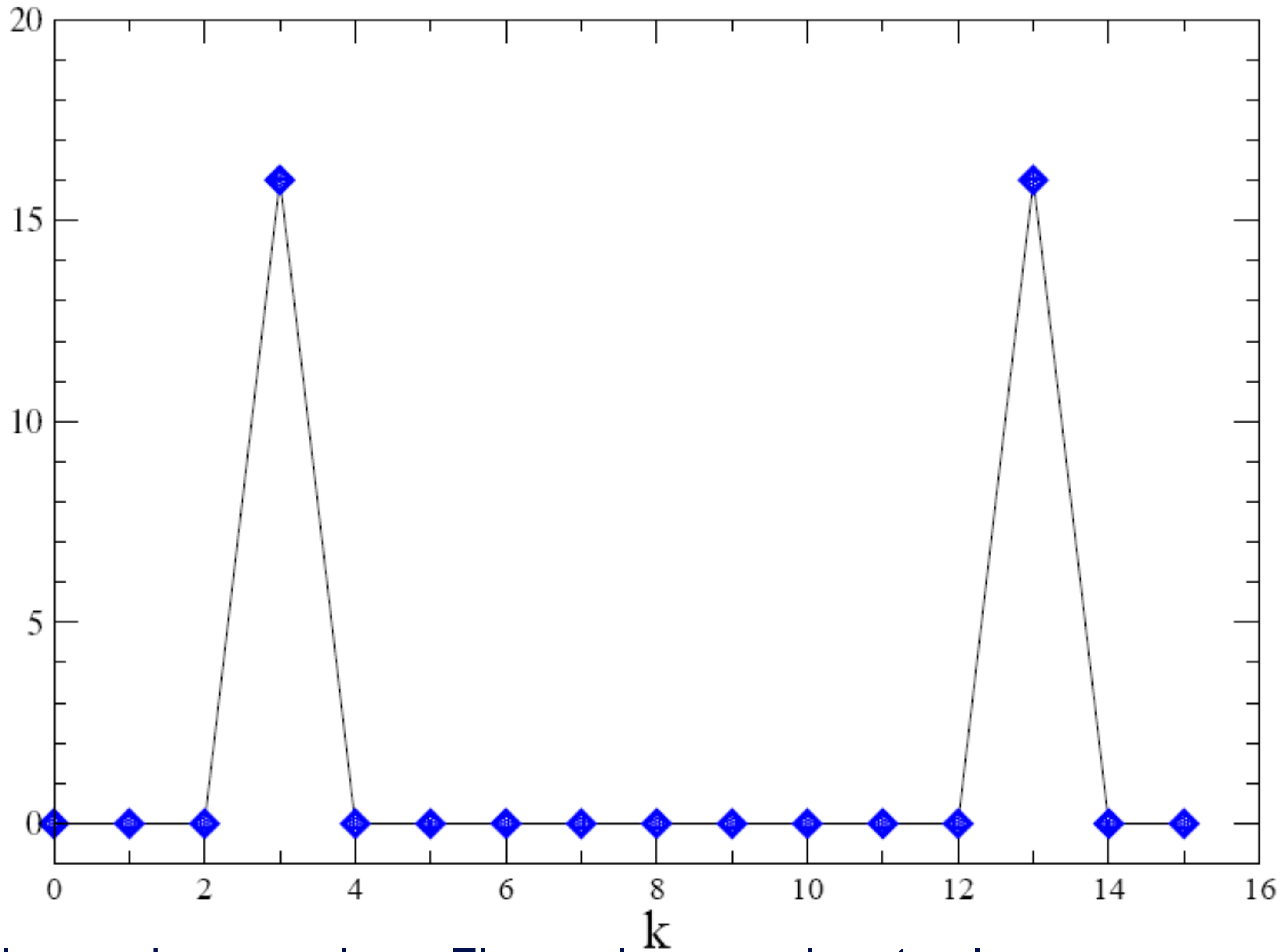
- Both the input function and its Fourier Transform are periodic

Example: Cosine function

3 periods for N=16:



Example: Cosine function



- FT is genuine complex – Figure shows real part only
- Peak of height N at k=3 and k=N-3

Discrete Fourier Transform

- The DFT can be rewritten as

$$F_n = a_0 + \sum_{k=1}^{N-1}\left(a_k \cos\left(2\pi k \frac{n}{N}\right) + b_k i \sin\left(2\pi k \frac{n}{N}\right)\right)$$

- Thus, the DFT essentially returns real number values for a_k and b_k , stored in a complex array
  - a_k and b_k are functions of f_k
  - remaining trigonometric constants (twiddle factors) may be pre-computed for a given N

- The scaling, shifting, convolution and correlation relationships, which hold for the continuous case, also hold for the discrete case.

DFT Laboratory

- Visit and play with the DFT Laboratory
  - Copied from Stanford University
  - Written by Dave Hale, Landmark Graphics
  - Local copy: http://www.epcc.ed.ac.uk/~ibethune/FFTlab/FftLab.html
  - Original: sepwww.stanford.edu/oldsep/hale/FftLab.html

- Suggested experiments in the handout

- Only takes 15 minutes, please attempt before next lecture!

Fast Fourier Transform

- What is the computational cost of the DFT?
  - Each of the *N* points of the DFT is calculated in terms of all the N points in the original function: $O(N^2)$

$$F_n \equiv \sum_{k=0}^{N-1} f_k e^{2\pi i k n / N}$$

  - Very expensive to compute, even for moderate N

Fast Fourier Transform

- In 1965, J.W. Cooley and J.W. Tukey published a DFT algorithm which is of $O(N \log N)$
  - N is a power of 2
  - FFTs in general are not limited to powers of 2, however, the order may resort to $O(N^2)$
  - Essentially a divide-and-conquer algorithm (details to follow)
  - In hindsight, faster than $O(N^2)$ algorithms were previously, independently discovered
    - Gauss was probably first to use such an algorithm in 1805

Fast Fourier Transform

- FFT is an efficient method for computing the DFT
  - Orders of magnitude faster, even for small values of N

N	N^2	$N \log_2(N)$
128	16384	896

- For further reading, implementation details consult:
  - *Numerical Recipes. The Art of Scientific Computing, 3rd Edition*, 2007, Cambridge University Press (www.nr.com)

FFT Implementation

- Algorithm based on Danielson & Lanczos (1942)

$$F_n = \sum_{k=0}^{N-1} f_k e^{2\pi ikn/N}$$

$$F_n = \sum_{k=0}^{N/2-1} f_{2k} e^{2\pi i(2k)n/N} + \sum_{k=0}^{N/2-1} f_{2k+1} e^{2\pi i(2k+1)n/N}$$

$$F_n = \sum_{k=0}^{N/2-1} f_{2k} e^{2\pi ikn/(N/2)} + e^{2\pi in/N} \sum_{k=0}^{N/2-1} f_{2k+1} e^{2\pi ikn/(N/2)}$$

$$F_n = F_n^e + W_N^n F_n^o \qquad W_N = e^{2\pi i/N}$$

FFT Implementation

- Can continue to break down into smaller and small FFTs

$$F_n = F_n^e + W_N^n F_n^o$$

$$F_n = F_n^{ee} + W_{N/2}^n F_n^{eo} + W_N^n F_n^{oe} + W_{N/2}^n W_N^n F_n^{oo}$$

$$F_n = F_n^{ee} + W_N^{2n} F_n^{eo} + W_N^n F_n^{oe} + W_N^{3n} F_n^{oo}$$

- For a 4 element DFT (N=4), each of the remaining 1-element DFTs must be one of the f_k we started with – but which ones?

FFT Implementation

- Bit reversal
- Set e=0, o=1, and reverse the order in binary

$$F_n^{ee} = f_{00} = f_0 \qquad F_n^{eo} = f_{10} = f_2 \qquad \text{etc…}$$

- Swapping elements by bit reversal is O(N)
- Now build up the F_n by combining the reordered f_ks

FFT Implementation

- Recall:

$$F_n = F_n^e + W_N^n F_n^o$$

- i.e. we can find all the components of an N-length DFT via 2 N/2-length DFTs – these are periodic with period N/2 so

$$F_n^e = F_{n-N/2}^e \qquad F_n^o = F_{n-N/2}^o \qquad W_N^n = -W_N^{n-N/2}$$

$$F_n = \begin{cases} F_n^e + W_N^n F_n^o & \text{if } n < N/2 \\ F_{n-N/2}^e - W_N^{n-N/2} F_{n-N/2}^o & \text{if } n \geq N/2 \end{cases}$$

FFT Implementation

- So first combine N=1 DFTs pairwise:

f_0	f_2	f_1	f_3

becomes

f_0+f_2	f_1+f_3	f_0-f_2	f_1-f_3

FFT Implementation

- Then combine the N=2 DFTs in even/odd pairs:

f_0+f_2	f_1+f_3	f_0-f_2	f_1-f_3

becomes

$f_0+f_2+f_1+f_3$	$f_0-f_2+W(f_1-f_3)$	$f_0+f_2-(f_1+f_3)$	$f_0-f_2-W(f_1-f_3)$

=

$f_0+f_2+f_1+f_3$	$f_0-f_2+ if_1-if_3$	$f_0+f_2-f_1-f_3$	$f_0-f_2-if_1+if_3$

FFT Implementation

- Try e.g. taking the transform of (1, 2, 3, 4)
- Gives (10, -2-2i, -2, -2+2i)

- Compare with e.g. FFT Calculator
  - http://www.random-science-tools.com/maths/FFT.htm
  - Or implement your own using FFTW (see later)

- Correct answer, (modulo choice of sign for imaginary part)

FFT Implementation

- Psuedocode example given in Num. Recipes Ch. 12

- Key Points

  - $Log_2(N)$ steps
  - Each step we update N elements
  - Overall runtime is O(NlogN)

  - This is a real pain to implement (either by hand or in code)

  - You don’t want to ever do this!