EE123
Digital Signal Processing

Lecture 8
FFT II
Lab1

based on slides by J.M. Kahn

M. Lustig, EECS UC Berkeley

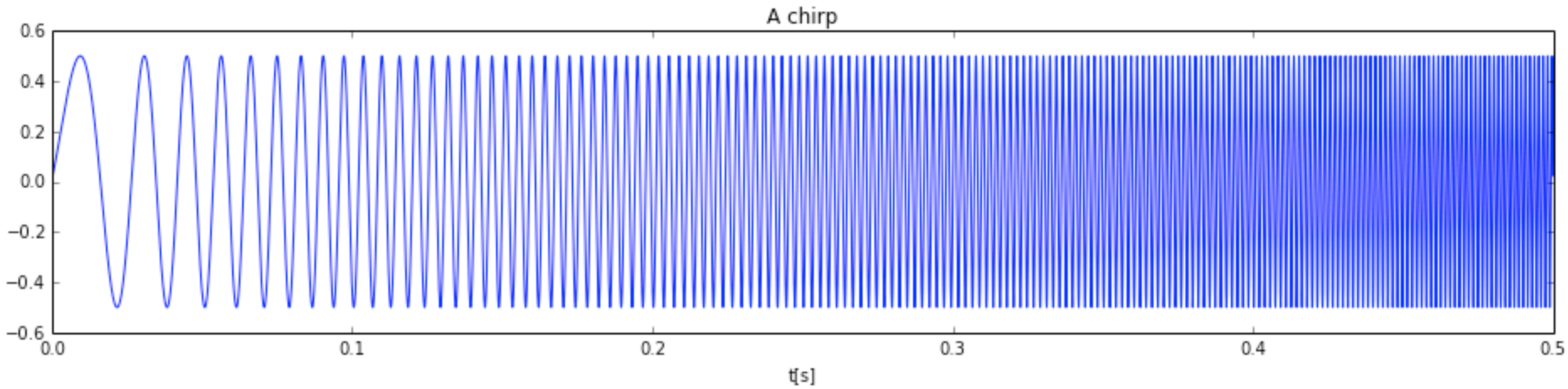
Announcements

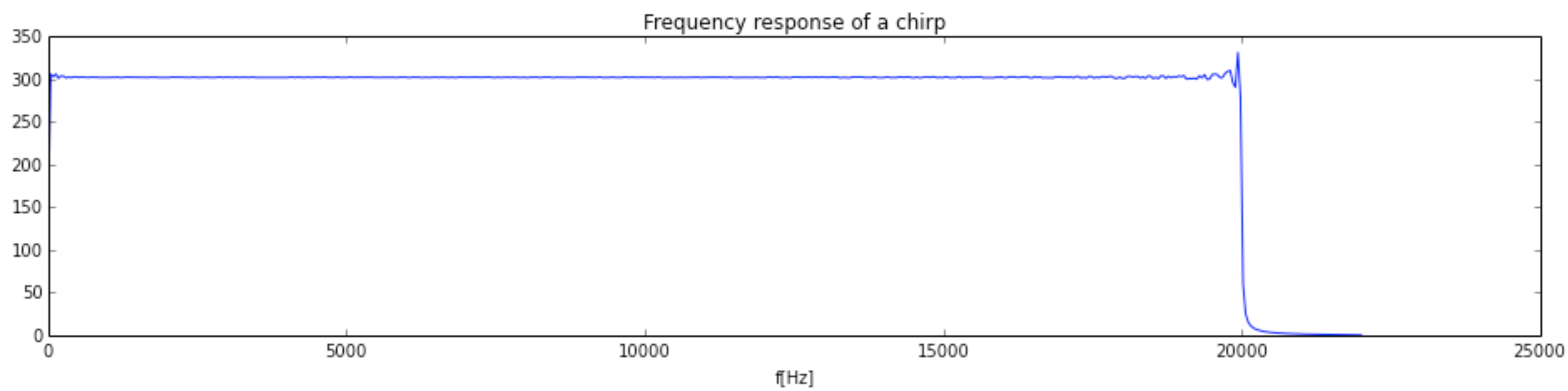
- Last time:
  - Started FFT
- Today
  - Lab 1
  - Finish FFT
- Read Ch. 10.1-10.2

- Midterm 1: Feb 22nd

Lab1

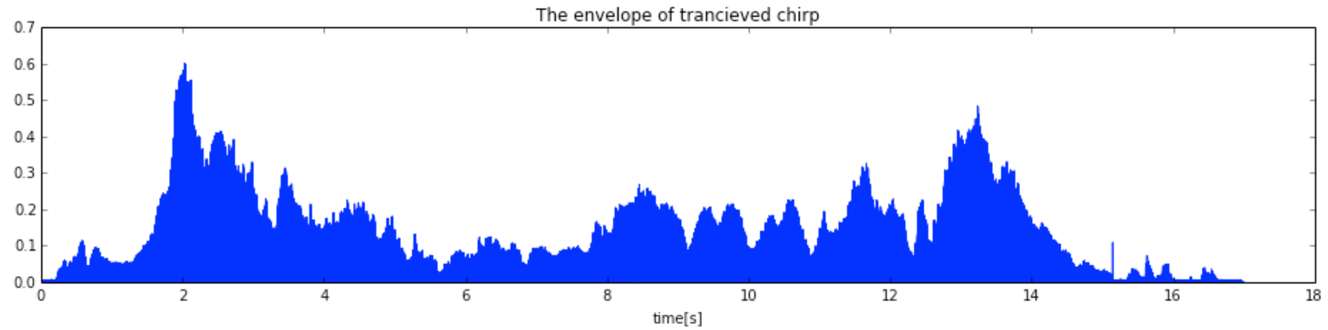
- Generate a chirp

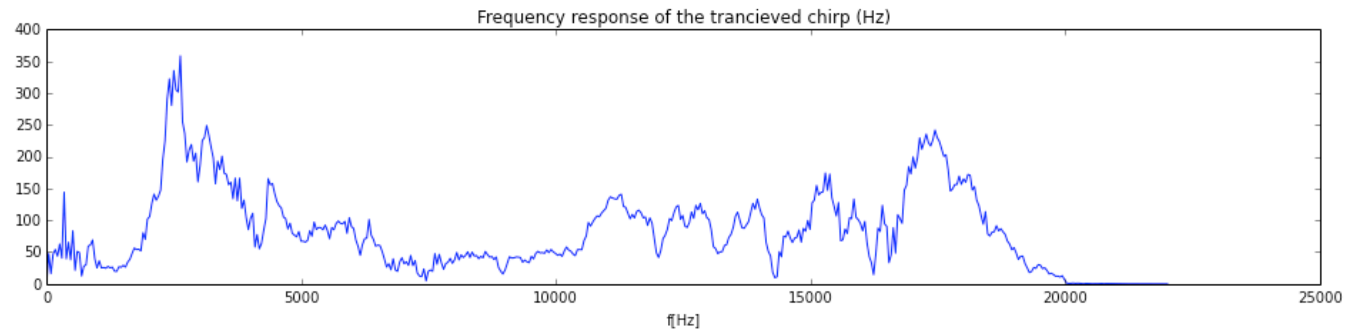


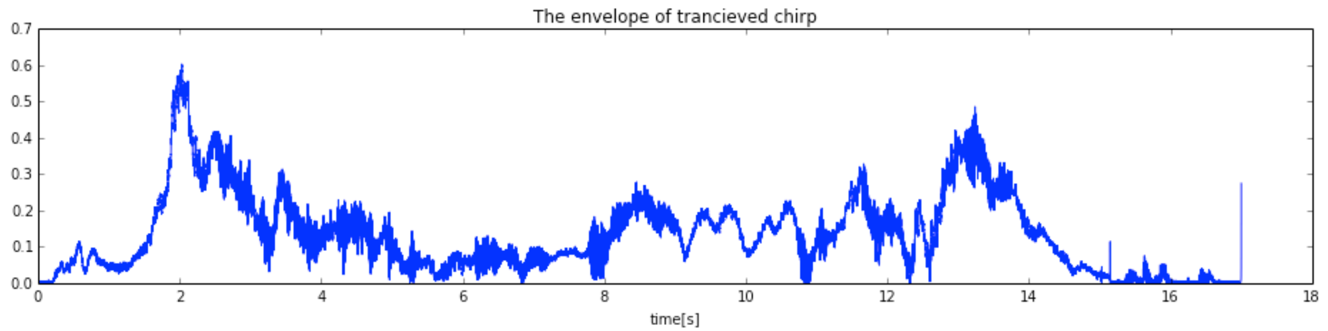


Lab1

- Play and record chirp

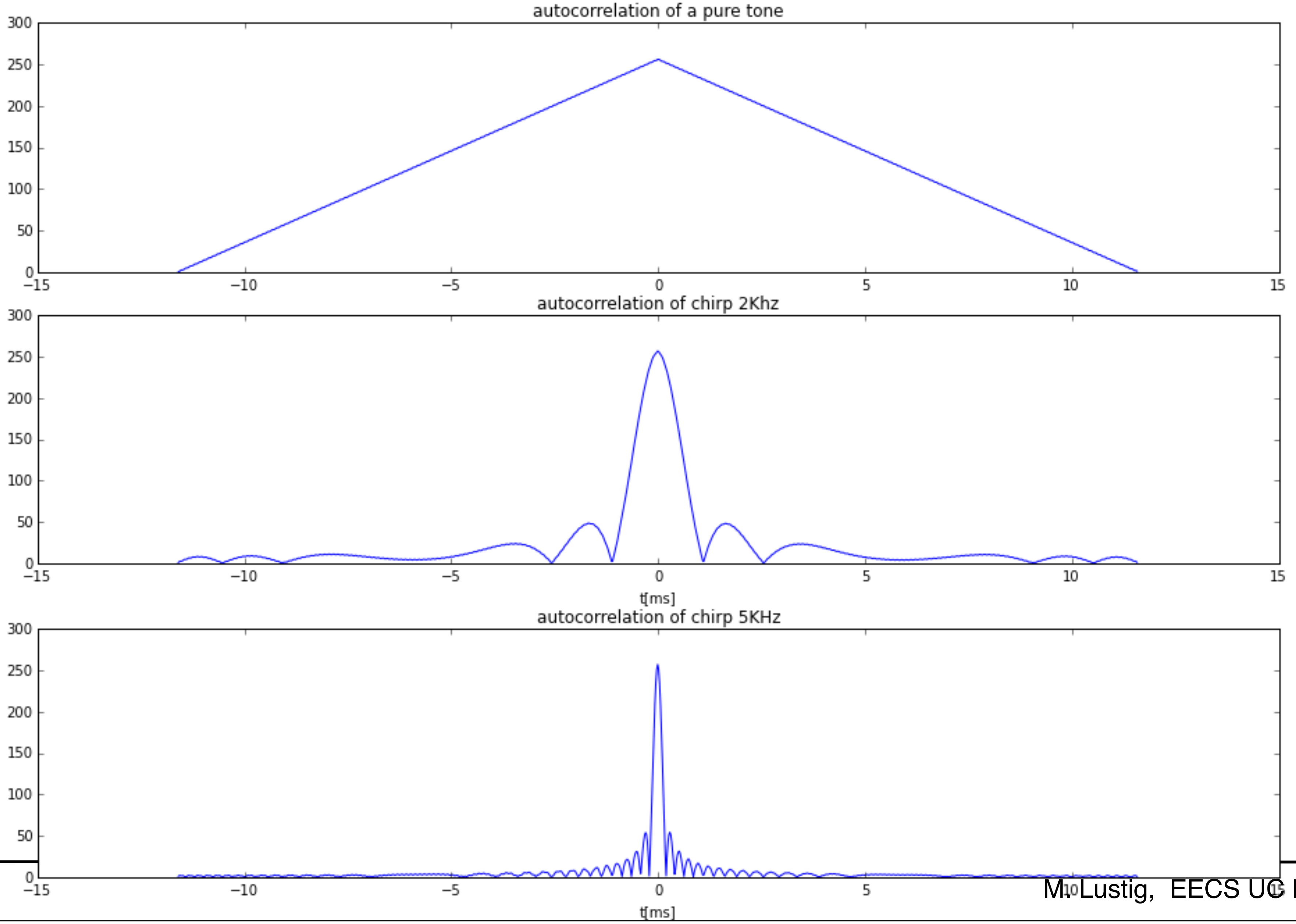






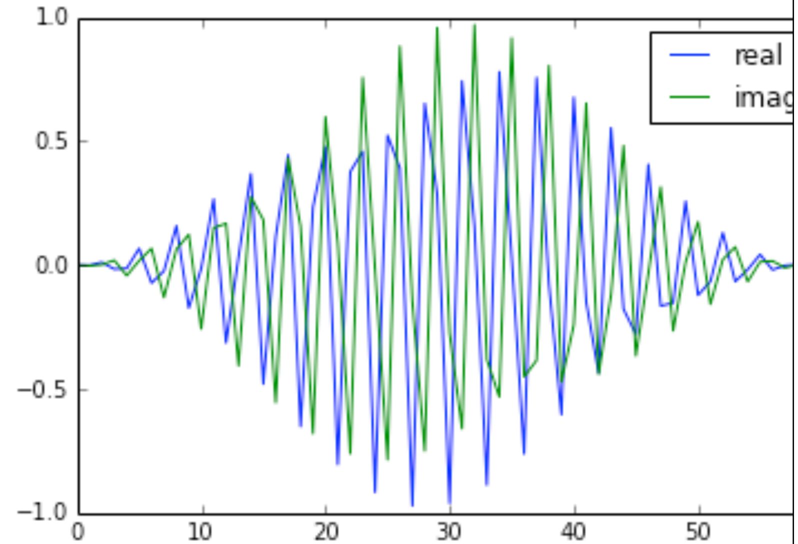
Lab 1

- Auto-correlation of a chirp - pulse compression

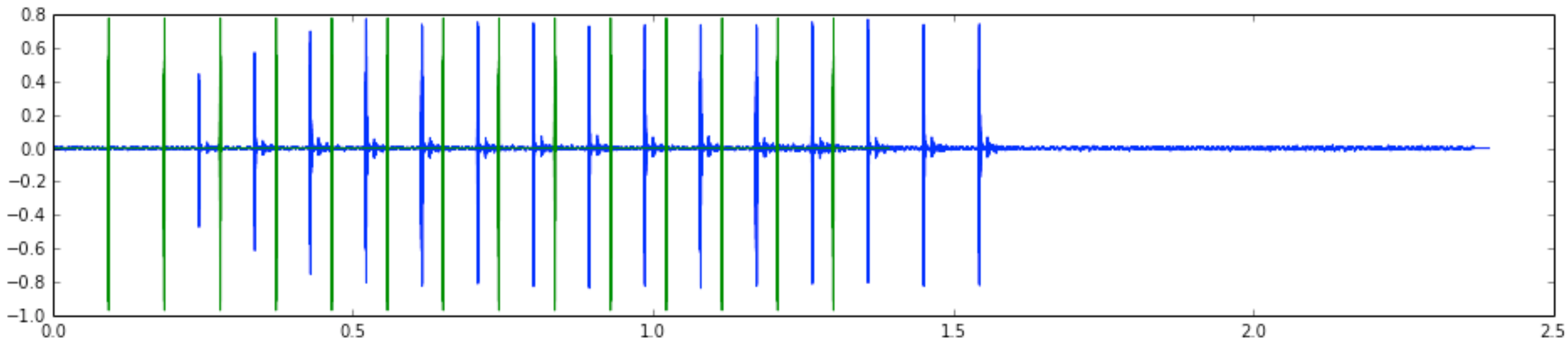


Lab I part II - Sonar

- Generate a pulse - analytic
- Use real part for pulse train
- Transmit and record

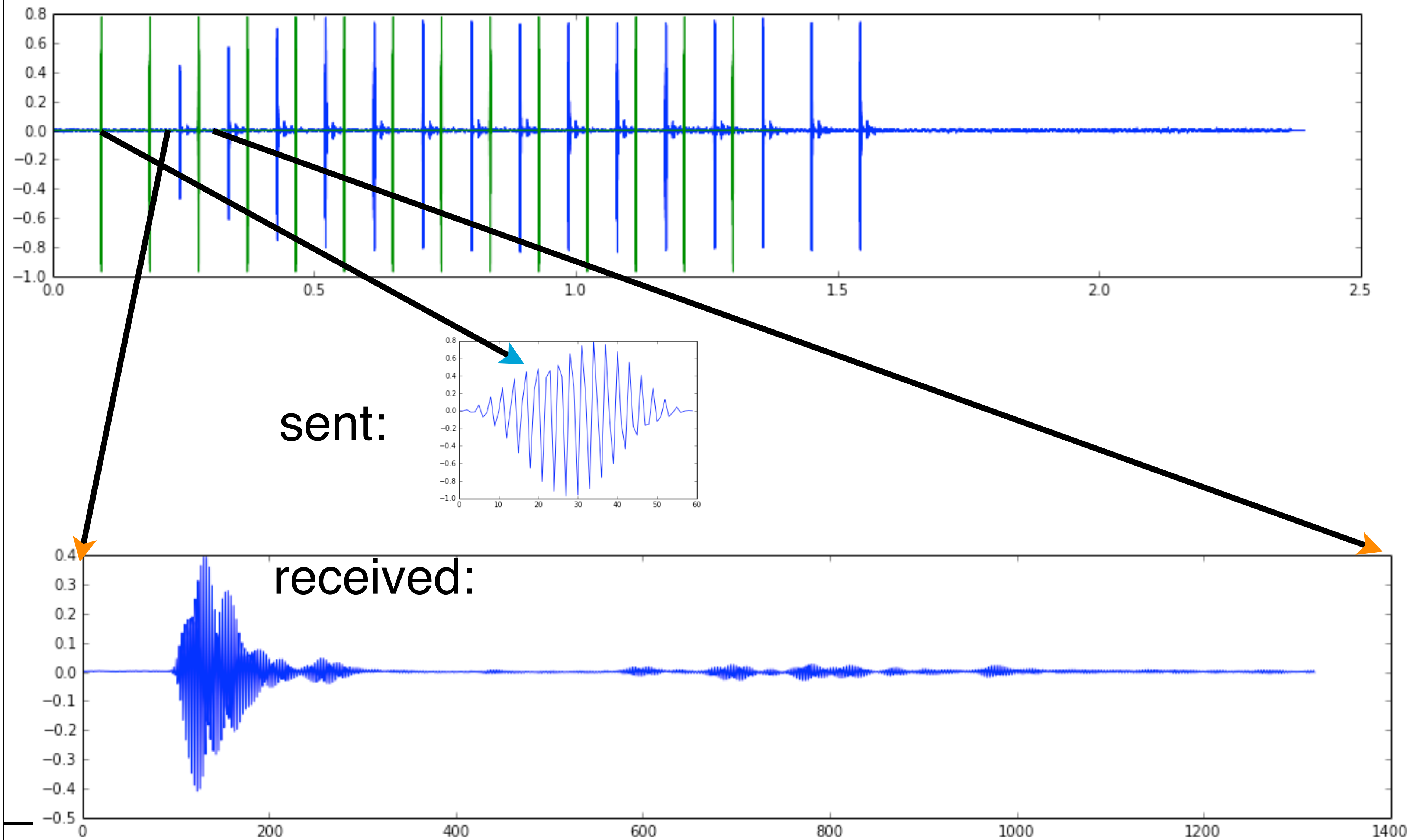


Sent and recorded:



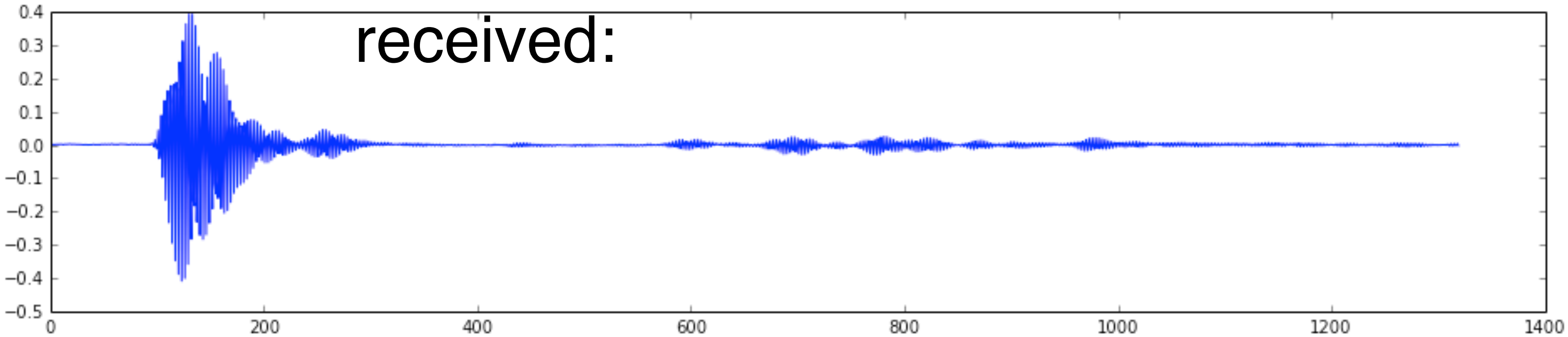
Lab I part II - Sonar

- Extract a pulse

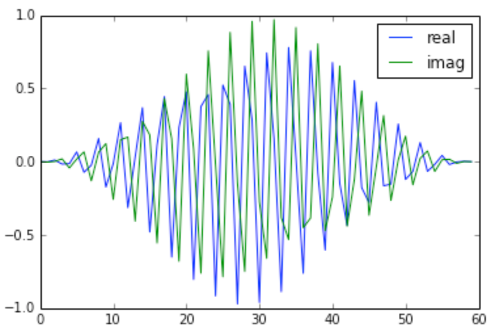


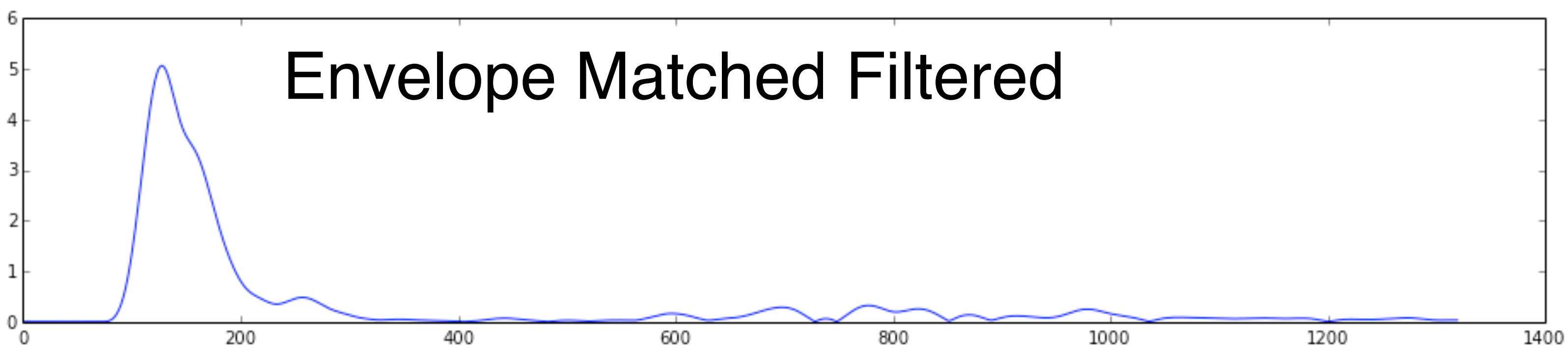
Lab I part II - Sonar

- Matched Filtering



Filter:

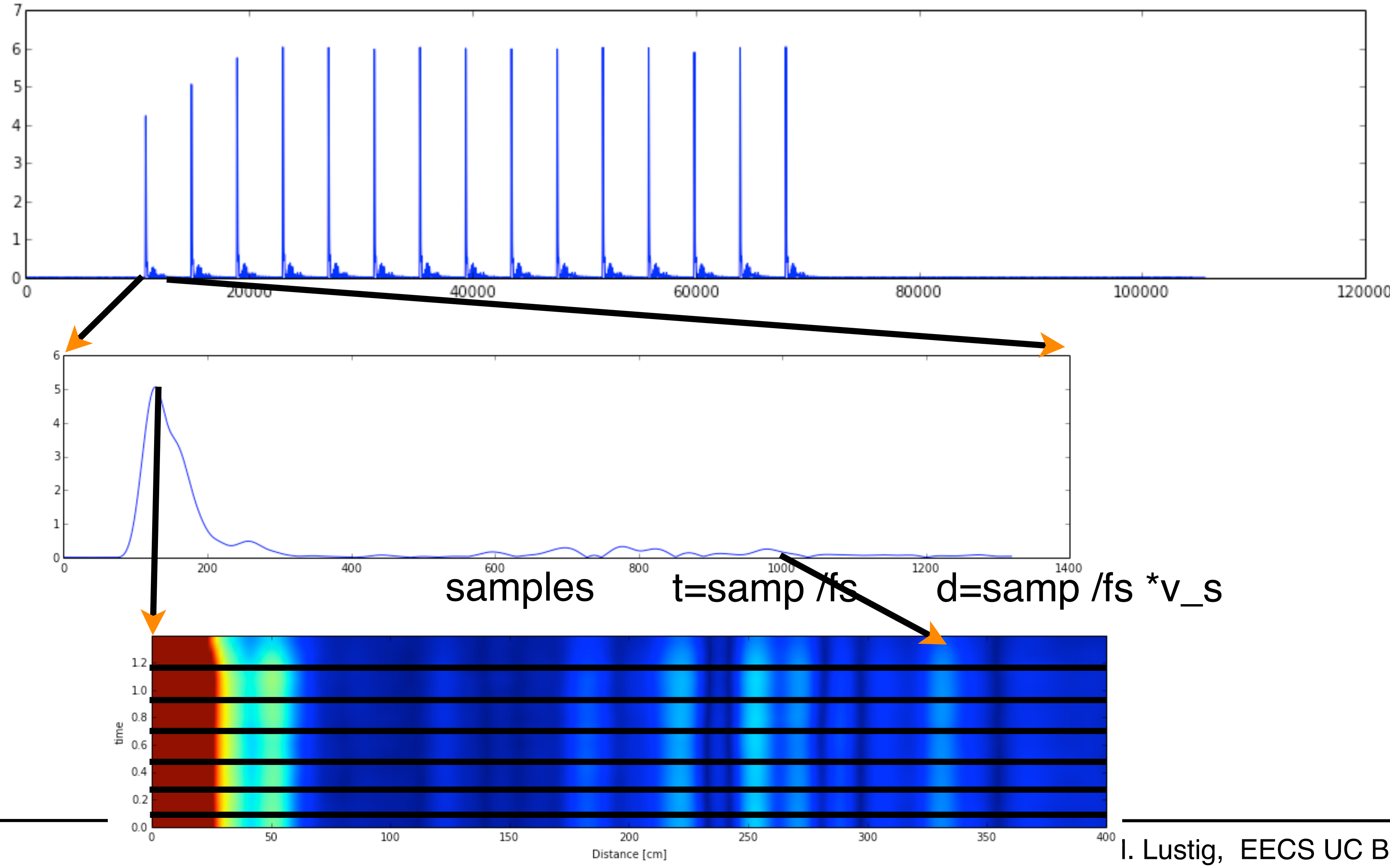




Lab I part II - Sonar

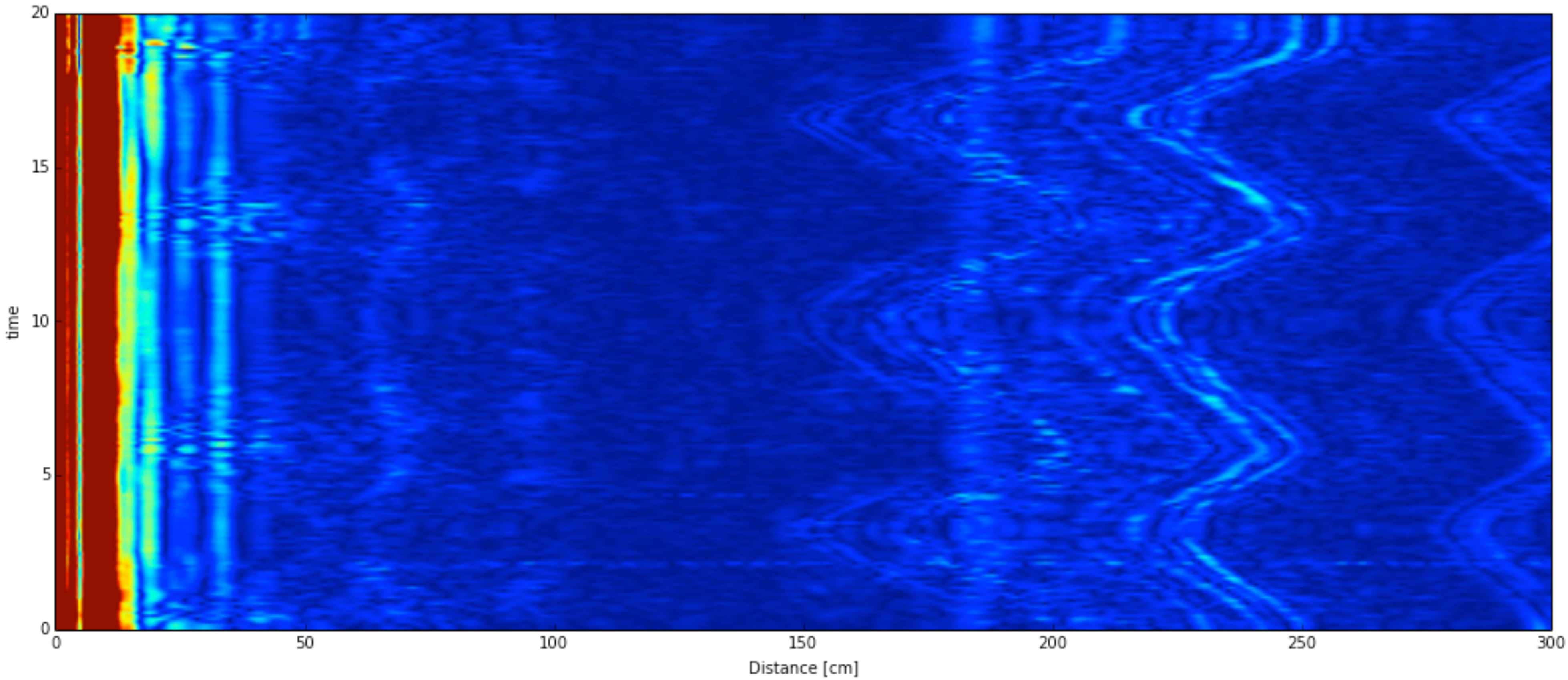
- Display echos vs distance

Matched Filter:



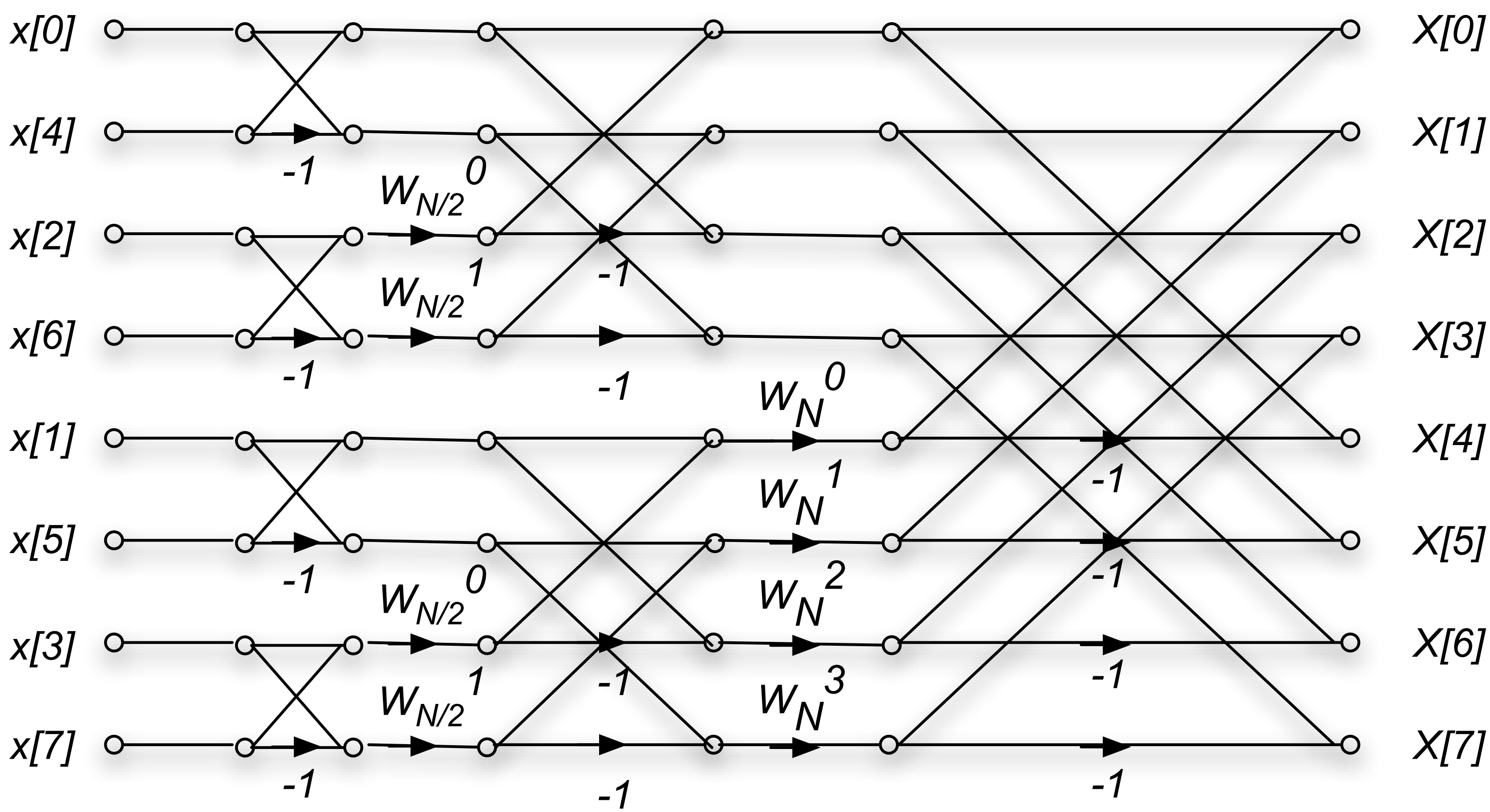
Lab I part II - Sonar

- Real time demo



Decimation-in-Time Fast Fourier Transform

Combining all these stages, the diagram for the 8 sample DFT is:



This the decimation-in-time FFT algorithm.

Decimation-in-Time Fast Fourier Transform

- In general, there are $\log_2 N$ stages of decimation-in-time.
- Each stage requires $N/2$ complex multiplications, some of which are trivial.
- The total number of complex multiplications is $(N/2)\log_2 N$.
- The order of the input to the decimation-in-time FFT algorithm must be permuted.
  - First stage: split into odd and even. Zero low-order bit first
  - Next stage repeats with next zero-lower bit first.
  - Net effect is reversing the bit order of indexes

Decimation-in-Time Fast Fourier Transform

This is illustrated in the following table for $N = 8$.

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

Decimation-in-Frequency Fast Fourier Transform

The DFT is

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}$$

If we only look at the even samples of $X[k]$, we can write $k = 2r$,

$$X[2r] = \sum_{n=0}^{N-1} x[n] W_N^{n(2r)}$$

We split this into two sums, one over the first $N/2$ samples, and the second of the last $N/2$ samples.

$$X[2r] = \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n + N/2] W_N^{2r(n+N/2)}$$

Decimation-in-Frequency Fast Fourier Transform

But $W_N^{2r(n+N/2)} = W_N^{2rn} W_N^N = W_N^{2rn} = W_{N/2}^{rn}$.
We can then write

$$
\begin{aligned}
X[2r] &= \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n+N/2] W_N^{2r(n+N/2)} \\
&= \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n+N/2] W_N^{2rn} \\
&= \sum_{n=0}^{(N/2)-1} \left(x[n] + x[n+N/2]\right) W_{N/2}^{rn}
\end{aligned}
$$

This is the $N/2$-length DFT of first and second half of $x[n]$ summed.

Decimation-in-Frequency Fast Fourier Transform

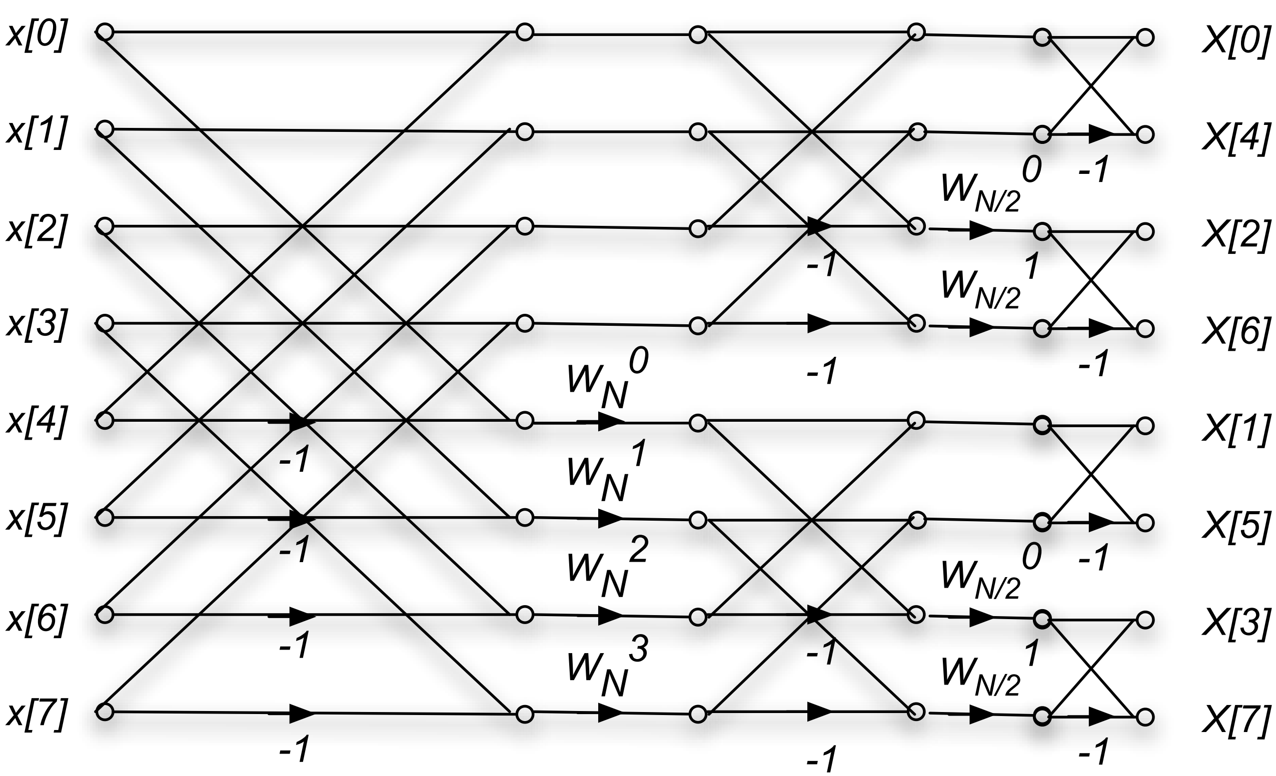
$$X[2r] = \mathrm{DFT}_{\frac{N}{2}}\left\{(x[n] + x[n+N/2])\right\}$$
$$X[2r+1] = \mathrm{DFT}_{\frac{N}{2}}\left\{(x[n] - x[n+N/2])\, W_N^n\right\}$$

(By a similar argument that gives the odd samples)

Continue the same approach is applied for the $N/2$ DFTs, and the $N/4$ DFT's until we reach simple butterflies.

Decimation-in-Frequency Fast Fourier Transform

The diagram for and 8-point decimation-in-frequency DFT is as follows

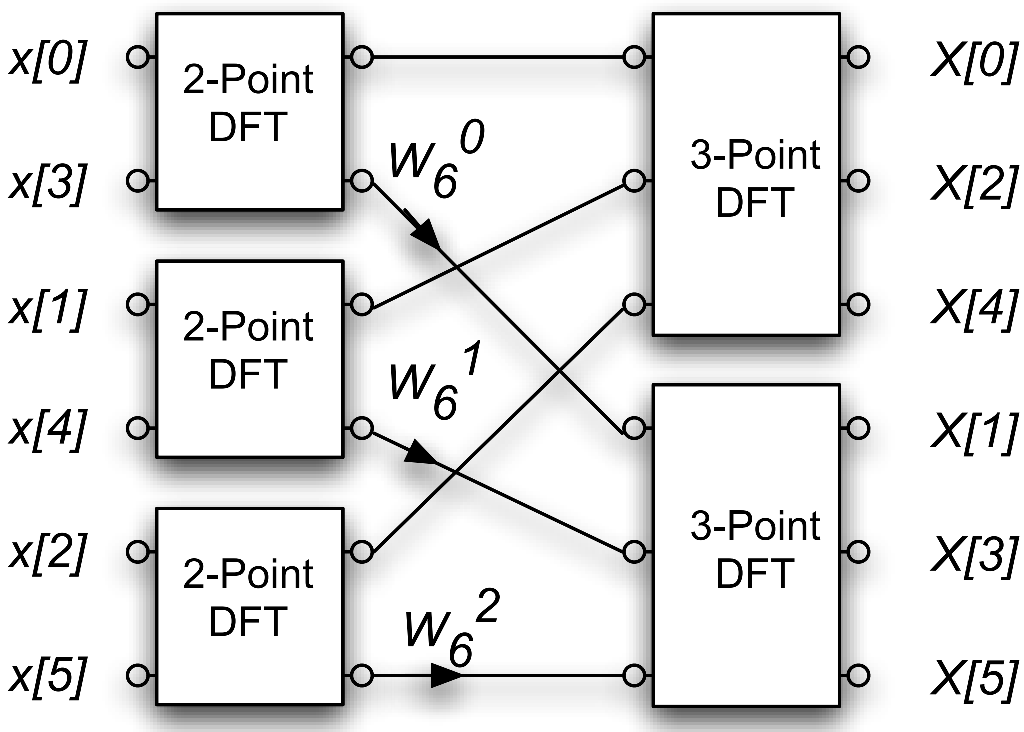


This is just the decimation-in-time algorithm reversed!
The inputs are in normal order, and the outputs are bit reversed.

Non-Power-of-2 FFT's

A similar argument applies for any length DFT, where the length N is a composite number.

For example, if $N = 6$, a decimation-in-time FFT could compute three 2-point DFT's followed by two 3-point DFT's



Non-Power-of-2 FFT's

Good component DFT's are available for lengths up to 20 or so. Many of these exploit the structure for that specific length. For example, a factor of

$$W_N^{N/4} = e^{-j\frac{2\pi}{N}(N/4)} = e^{-j\frac{\pi}{2}} = -j \qquad \textit{Why?}$$

just swaps the real and imaginary components of a complex number, and doesn't actually require any multiplies.
Hence a DFT of length 4 doesn't require any complex multiplies.
Half of the multiplies of an 8-point DFT also don't require multiplication.
Composite length FFT's can be very efficient for any length that factors into terms of this order.

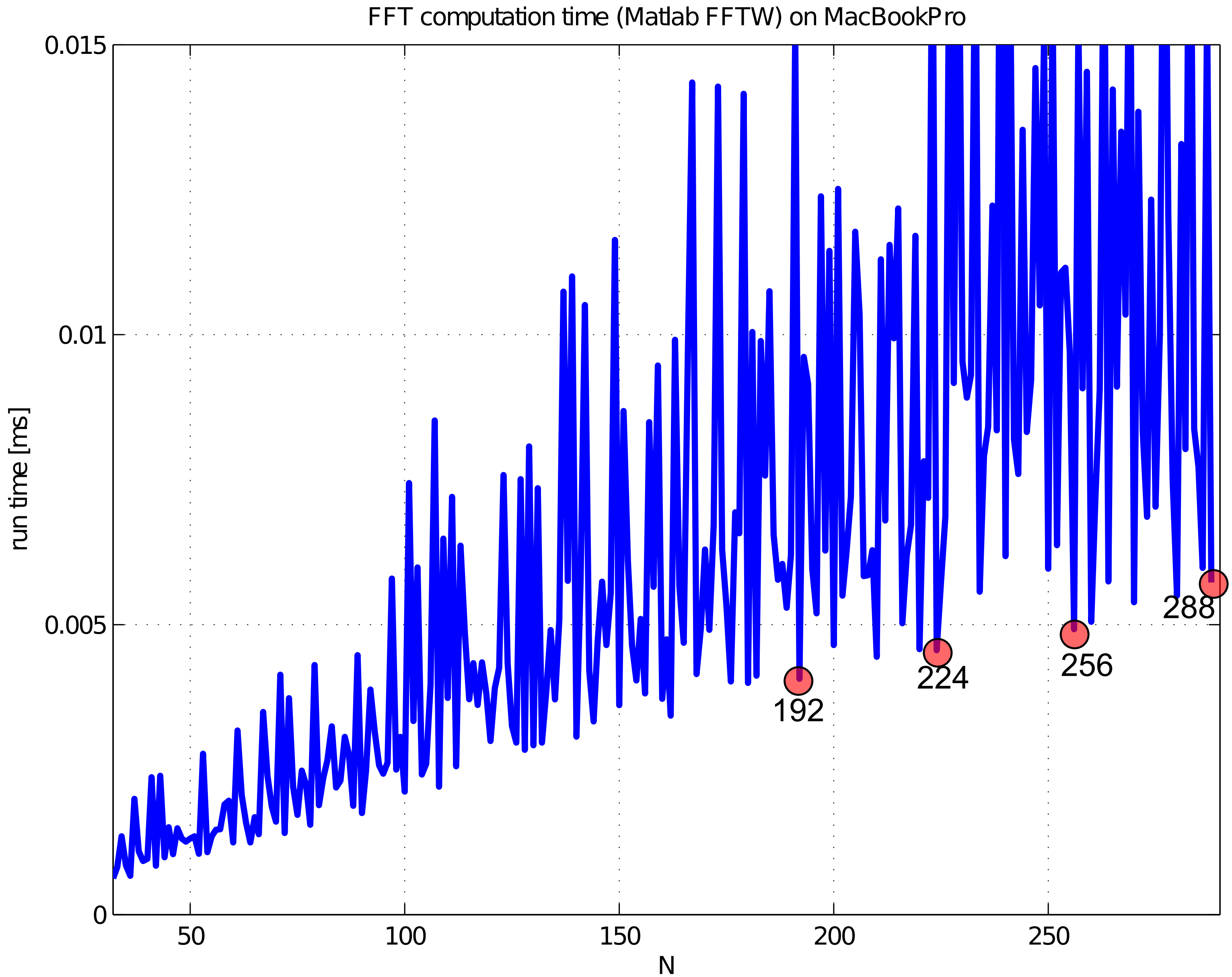
For example $N = 693$ factors into

$$N = (7)(9)(11)$$

each of which can be implemented efficiently. We would perform

- 9×11 DFT's of length 7
- 7×11 DFT's of length 9, and
- 7×9 DFT's of length 11

- Historically, the power-of-two FFTs were much faster (better written and implemented).
- For non-power-of-two length, it was faster to zero pad to power of two.
- Recently this has changed. The free FFTW package implements very efficient algorithms for almost any filter length. Matlab has used FFTW since version 6


FFT computation time (Matlab FFTW) on MacBookPro
run time [ms]
N
0.015
0.01
0.005
0
50
100
150
200
250
192
224
256
288

FFT as Matrix Operation

$$\begin{pmatrix} X[0] \\ \vdots \\ X[k] \\ \vdots \\ X[N-1] \end{pmatrix} = \begin{pmatrix} W_N^{00} & \cdots & W_N^{0n} & \cdots & W_N^{0(N-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ W_N^{k0} & \cdots & W_N^{kn} & \cdots & W_N^{k(N-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ W_N^{(N-1)0} & \cdots & W_N^{(N-1)n} & \cdots & W_N^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} x[0] \\ \vdots \\ x[n] \\ \vdots \\ x[N-1] \end{pmatrix}$$

- W_N is fully populated $\Rightarrow N^2$ entries.

FFT as Matrix Operation

$$\begin{pmatrix} X[0] \\ \vdots \\ X[k] \\ \vdots \\ X[N-1] \end{pmatrix} = \begin{pmatrix} W_N^{00} & \cdots & W_N^{0n} & \cdots & W_N^{0(N-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ W_N^{k0} & \cdots & W_N^{kn} & \cdots & W_N^{k(N-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ W_N^{(N-1)0} & \cdots & W_N^{(N-1)n} & \cdots & W_N^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} x[0] \\ \vdots \\ x[n] \\ \vdots \\ x[N-1] \end{pmatrix}$$

- W_N is fully populated $\Rightarrow N^2$ entries.
- FFT is a decomposition of W_N into a more sparse form:

$$F_N = \begin{bmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{bmatrix} \begin{bmatrix} W_{N/2} & 0 \\ 0 & W_{N/2} \end{bmatrix} \begin{bmatrix} \text{Even-Odd Perm.} \\ \text{Matrix} \end{bmatrix}$$

- $I_{N/2}$ is an identity matrix. $D_{N/2}$ is a diagonal with entries $1,\ W_N,\ \cdots,\ W_N^{N/2-1}$

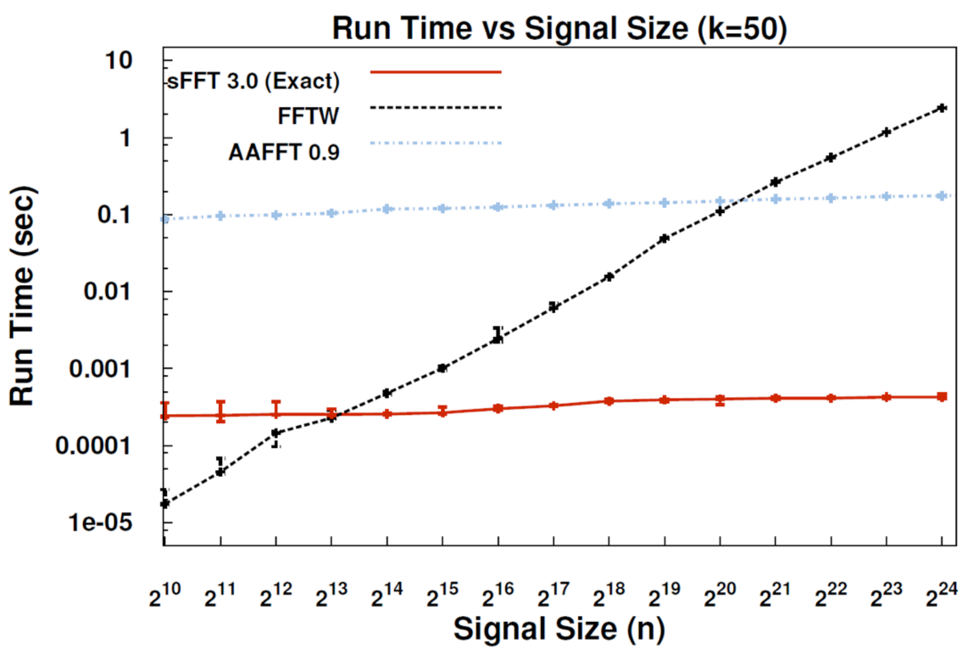
FFT as Matrix Operation

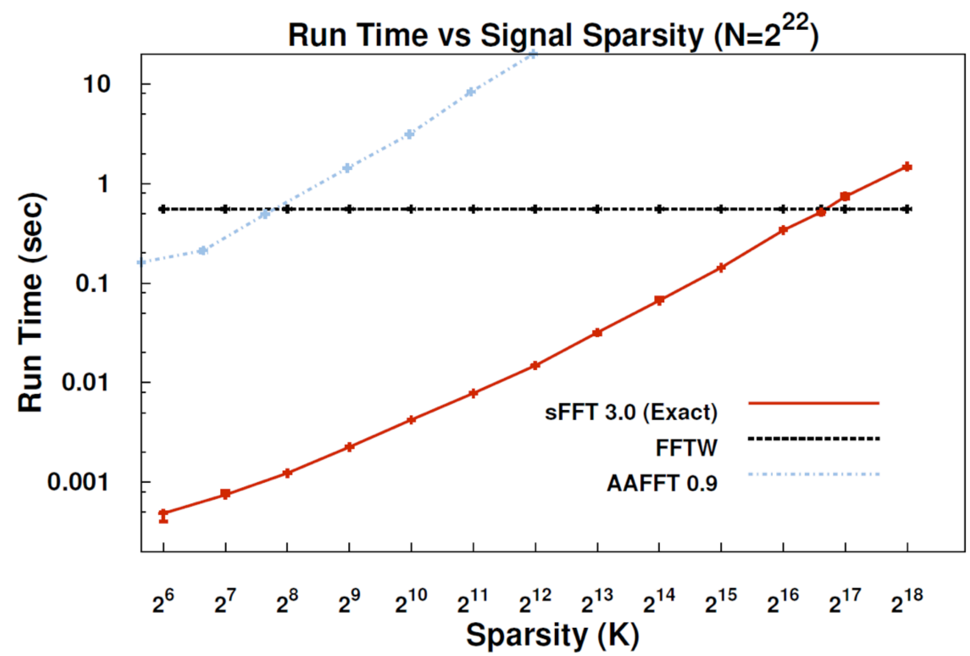
Example: $N = 4$

$$F_4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & W_4 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -W_4 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Beyond NlogN

- What if the signal x[n] has a k sparse frequency
  - A. Gilbert et. al, “Near-optimal sparse Fourier representations via sampling
  - H. Hassanieh et. al, “Nearly Optimal Sparse Fourier Transform”
  - Others……
- O(K Log N) instead of O(N Log N)





From: http://groups.csail.mit.edu/netmit/sFFT/paper.html