FFT

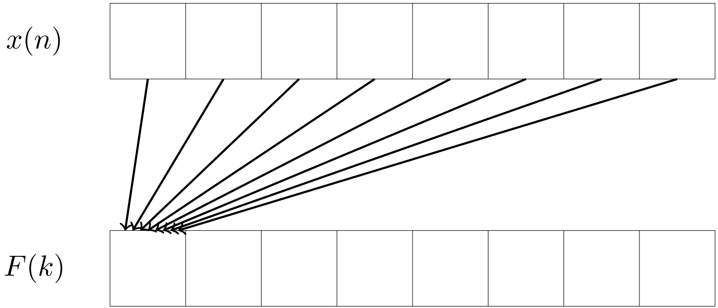
DFT

$$x(n) \qquad n = 0, \ldots, (N-1)$$

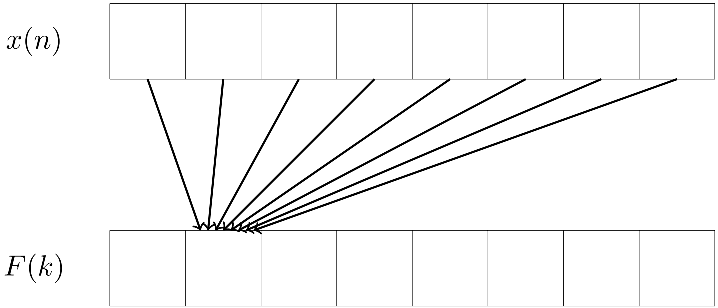
$$W_N = e^{\frac{-2\pi i}{N}}$$

$$F(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad k = 0, \ldots, (N-1)$$

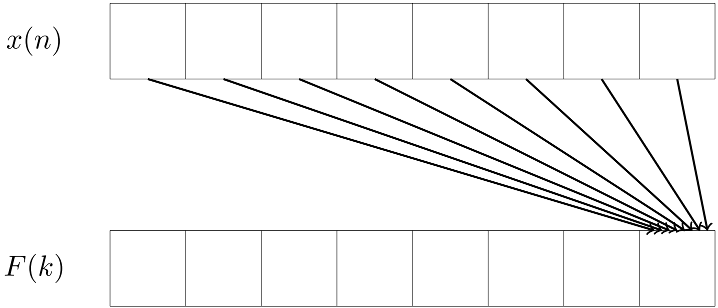
DFT: Computation scheme



DFT: Computation scheme



DFT: Computation scheme



DFT: Implementation

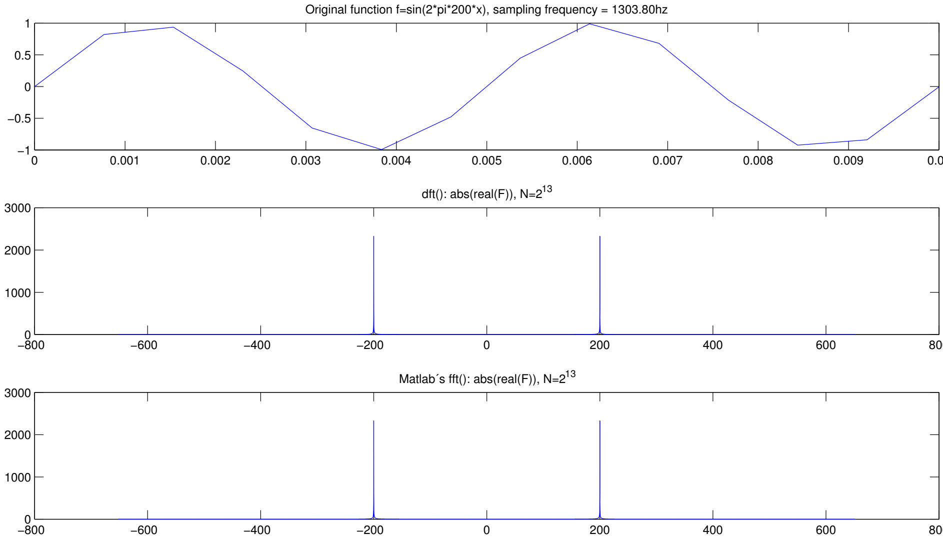
```
function f = dft(x)
  f=zeros(size(x));
  N=length(x);
  Wn=exp(-2*pi*1i/N);
  for k=1:N
    for n=1:N
      f(k)=f(k)+ x(n) * Wn^{(k-1)*(n-1)}
    end
  end
end
```

DFT: Time complexity

```
function f = dft(x)
  f=zeros(size(x)); % O(N)
  N=length(x); % O(N)
  Wn=exp(-2*pi*1i/N); % O(1)
  for k=1:N % O(N*N)
    for n=1:N
      f(k)=f(k)+ x(n) * Wn^{(k-1)*(n-1)}  % O(1)
    end
  end
end
```

$O(n^2)$ operations

DFT: Output



Original function f=sin(2*pi*200*x), sampling frequency = 1303.80hz
dft(): abs(real(F)), N=2^13
Matlab´s fft(): abs(real(F)), N=2^13

DFT: Properties

$$W_N^{j+N/2} = -W_N^j$$

$$W_N^{j+N/2} = e^{\frac{-2\pi i(j+N/2)}{N}} = e^{\frac{-2\pi ij}{N} + \frac{-2\pi i(N/2)}{N}}$$
$$= e^{\frac{-2\pi ij}{N}} e^{-\pi i} = -W_N^j$$

$$W_N^{j+N} = W_N^j$$

$$W_N^{j+N} = W_N^{j+N/2+N/2} = W_N^j$$

$$F(k+N) = F(k)$$

$$F(k+N) = \sum_{n=0}^{N-1} x(n) W_N^{(k+N)n} = \sum_{n=0}^{N-1} x(n) W_N^{kn} W_N^{Nn}$$
$$= \sum_{n=0}^{N-1} x(n) W_N^{kn} = F(k)$$

FFT: Derivation (1)

$$
\begin{aligned}
F_N(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} &= \sum_{n=0}^{N/2-1} x(2n) W_N^{k(2n)} + \sum_{n=0}^{N/2-1} x(2n+1) W_N^{k(2n+1)} \\
&= \sum_{n=0}^{N/2-1} x(2n) W_N^{k2n} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_N^{k(2n)} \\
&= \sum_{n=0}^{N/2-1} x(2n) W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_{N/2}^{kn} \\
&= F_{N/2}^{even}(k) + W_N^k F_{N/2}^{odd}(k) \qquad k = 0, \ldots, (N-1)
\end{aligned}
$$

W_N^k are called *twiddle factors*

FFT: Derivation (2)

¿Value of $F_{N/2}^{even}(k)$ and $F_{N/2}^{odd}(k)$ if $k \geq N/2$?

Define $N_1 = \{0, \ldots, N/2-1\}$ and $N_2 = \{N/2, \ldots, N-1\}$

$$F_N(k) = F_{N/2}^{even}(k) + W_N^k F_{N/2}^{odd}(k)$$

$$= \begin{cases} F_{N/2}^{even}(k) + W_N^k F_{N/2}^{odd}(k) & \text{if } k \in N_1 \\ F_{N/2}^{even}(k) + W_N^k F_{N/2}^{odd}(k) & \text{if } k \in N_2 \end{cases}$$

If $k \geq N/2 \rightarrow k = N/2 + k'$

$$= \begin{cases} F_{N/2}^{even}(k) + W_N^k F_{N/2}^{odd}(k) & \text{if } k \in N_1 \\ F_{N/2}^{even}(N/2+k') + W_N^{N/2+k'} F_{N/2}^{odd}(N/2+k') & \text{if } k' \in N_1 \end{cases}$$

$$= \begin{cases} F_{N/2}^{even}(k) + W_N^k F_{N/2}^{odd}(k) & \text{if } k \in N_1 \\ F_{N/2}^{even}(k') - W_N^{k'} F_{N/2}^{odd}(k') & \text{if } k' \in N_1 \end{cases}$$

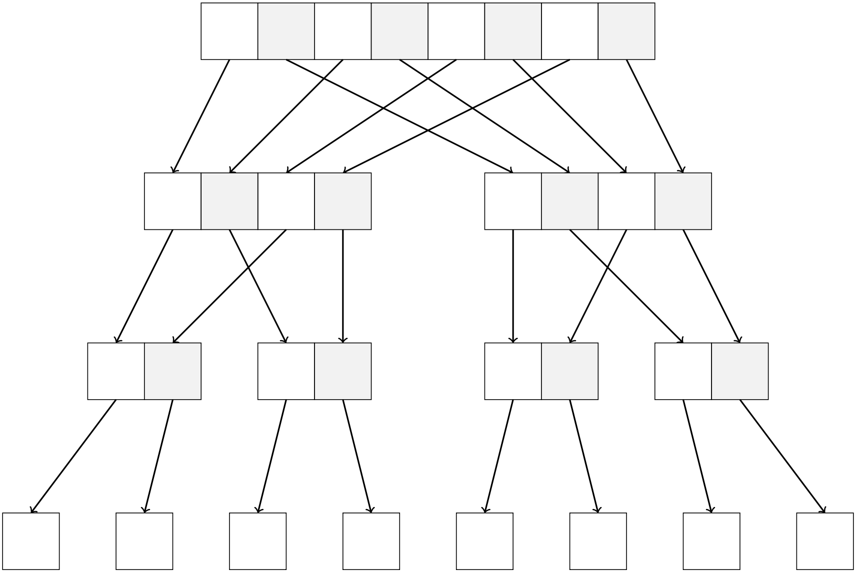
FFT: Divide and conquer

Calculating $F_N(k)$ for all k

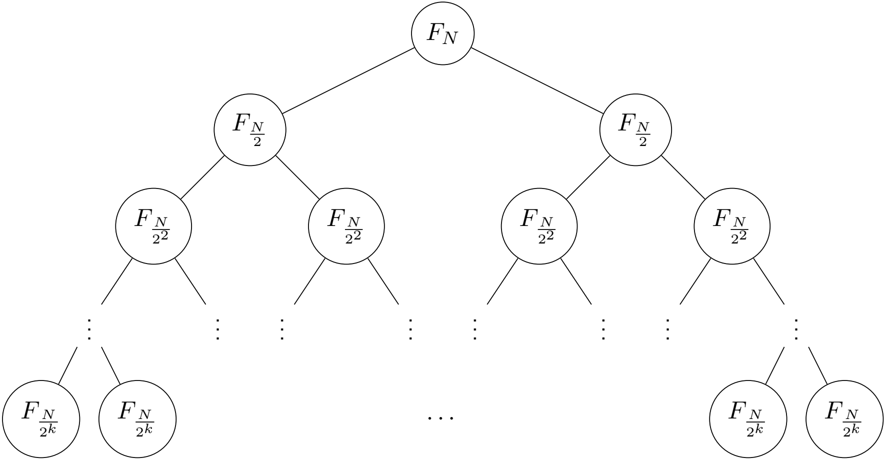
1. Split $x(n)$ into $x_{odd}(n)$ and $x_{even}(n)$.
2. Calculate $F^{even}_{N/2}(k)$ and $F^{odd}_{N/2}(k)$ for all k
3. Calculate $F_N(k)$ for all k as:

$$F_N(k) = \begin{cases} F^{even}_{N/2}(k) + W^k_N F^{odd}_{N/2}(k) & \text{if } k \in N_1 \\ F^{even}_{N/2}(k') - W^{k'}_N F^{odd}_{N/2}(k') & \text{if } k' \in N_1,\ k = N/2 + k' \end{cases} \tag{1}$$

FFT: Recursion splits



FFT: Recursion tree



FFT: Implementation

```
function f = facuft(x)
N=length(x);
Wn=exp(-2*pi*1i/N);

if N==1
  f= x;
else
  x_even=x(1:2:end);
  x_odd=x(2:2:end);
  f_even=facuft(x_even);
  f_odd=facuft(x_odd);

  f=zeros(size(x));
  first_half_indices=0:(N/2-1);
  twiddle_factors=Wn.^first_half_indices;
  f(1:(N/2))=f_even + twiddle_factors .* f_odd;
  f((N/2+1):end)=f_even - twiddle_factors .* f_odd;
end
```

FFT: Time complexity

```
function f = facuft(x)
N=length(x);%O(N)
Wn=exp(-2*pi*1i/N);%O(1)

if N==1
  f= x;  %O(1)
else
  x_even=x(1:2:end);%O(N)
  x_odd=x(2:2:end);  %O(N)
  f_even=facuft(x_even);% T(N/2)
  f_odd=facuft(x_odd);%T(N/2)

  f=zeros(size(x));%O(N)
  first_half_indices=0:(N/2-1);%O(N)
  twiddle_factors=Wn.^first_half_indices;%O(N)
  f(1:(N/2))=f_even + twiddle_factors .* f_odd;%O(N)
  f((N/2+1):end)=f_even-twiddle_factors.*f_odd;%O(N)
end
```

FFT: Solving recurrence

$$T(n) = \begin{cases} D & \text{if } n = 1 \\ 2T(\frac{n}{2}) + Cn & \text{if } n > 1 \end{cases}$$

FFT: Solving recurrence

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(\frac{n}{2}) + n & \text{if } n > 1 \end{cases}$$

FFT: Solving recurrence

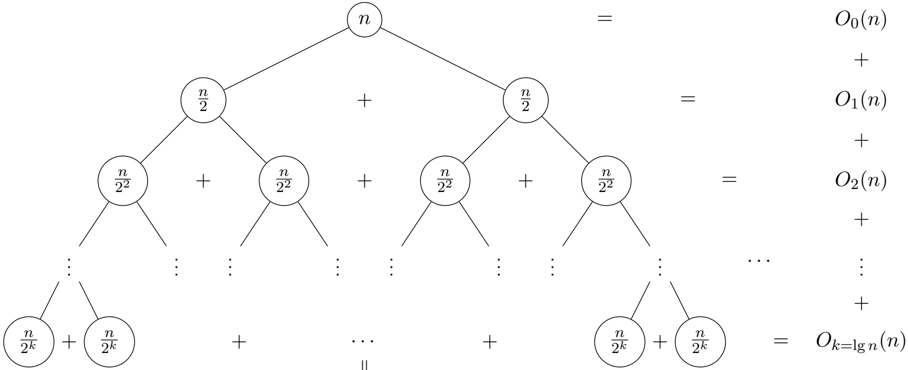
$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(\frac{n}{2}) + n & \text{if } n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + n = 2(2T(\frac{n}{2^2}) + \frac{n}{2}) + n = 2^2 T(\frac{n}{2^2}) + 2\frac{n}{2} + n \\ &= 2^2 T(\frac{n}{2^2}) + 2n = \cdots = 2^k T(\frac{n}{2^k}) + kn \end{aligned}$$

If $\dfrac{n}{2^k} = 1 \rightarrow k = \log_2(n)$:

$$T(n) = 2^{\log_2(n)} T(1) + \log_2(n) n = n + n \log_2(n) \in O(n \log(n))$$

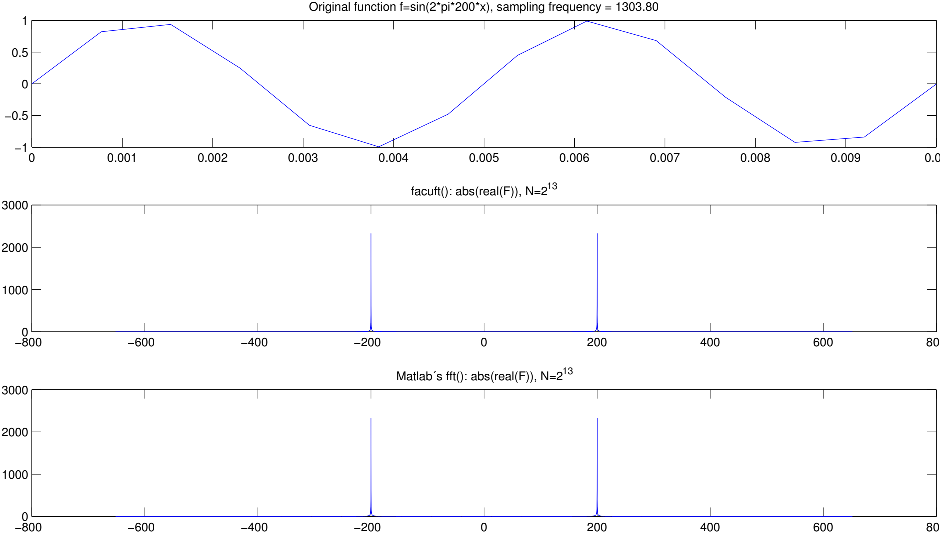
FFT: Time complexity with recursion tree



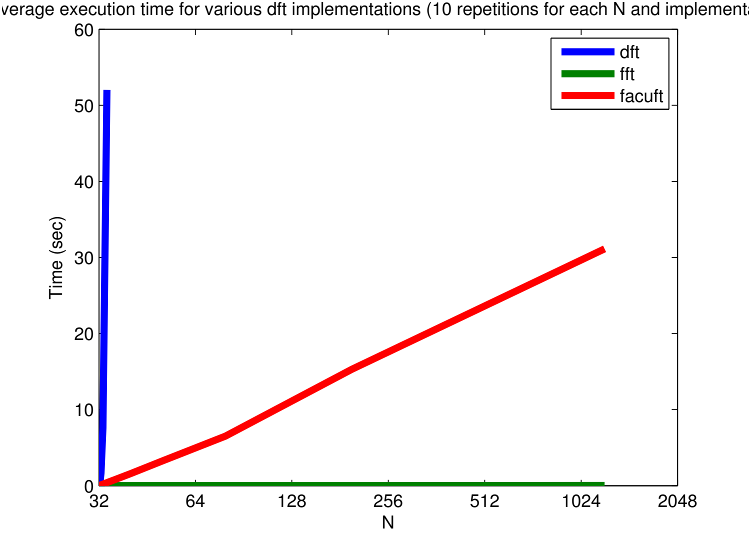
$$\Downarrow$$

$$O\left(\sum_{i=0}^{k} 2^i \cdot \frac{n}{2^i}\right) = O\left(\sum_{i=0}^{k} n\right) = O(k \cdot n) \Leftrightarrow O(n \cdot \lg n)$$

FFT: Output


Original function f=sin(2*pi*200*x), sampling frequency = 1303.80
1
0.5
0
–0.5
–1
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
facuft(): abs(real(F)), N=2^{13}
3000
2000
1000
0
–800
–600
–400
–200
0
200
400
600
Matlab´s fft(): abs(real(F)), N=2^{13}
3000
2000
1000
0
–800
–600
–400
–200
0
200
400
600

FFT: Empirical running time



FFT: Implementation improvements

- Recursion kills performance $\rightarrow$ bigger base cases
- In every recursion call, split the input x into K subarrays instead of 2 (tree depth $= \log_K(n)$, *fatter* trees) $\rightarrow$ Radix-m algorithms.
- Recursion kills performance $\rightarrow$ iterative implementation
- Calculate in-place (ie, no temporary arrays)
- Avoid pure matlab (matlab's `fft` is written in C)
- DFT's lower bound not known, but many results pointing to $DFT \in \Omega(n \log(n))$

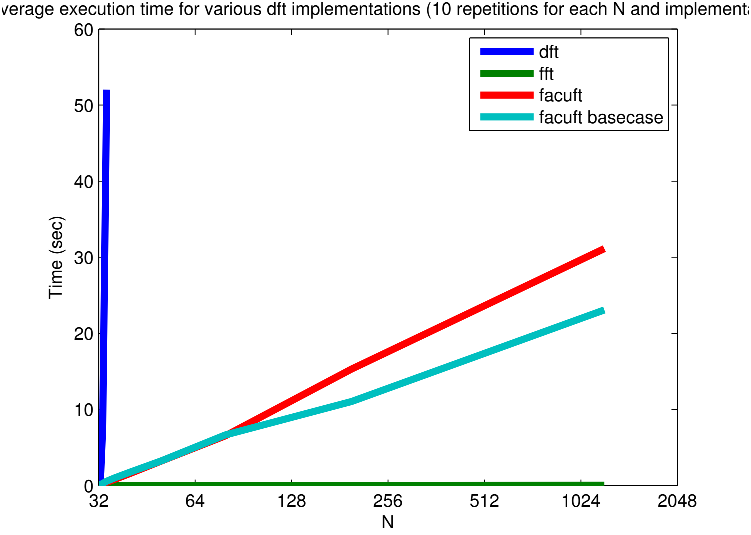
FFT: Bigger base case

```
function f = facuft_basecase(x)
N=length(x);
Wn=exp(-2*pi*1i/N);

if N<=2^3
  f= dft(x);
else
  x_even=x(1:2:end);
  x_odd=x(2:2:end);
  f_even=facuft(x_even);
  f_odd=facuft(x_odd);

  f=zeros(size(x));
  first_half_indices=0:(N/2-1);
  twiddle_factors=Wn.^first_half_indices;
  f(1:(N/2))=f_even + twiddle_factors .* f_odd;
  f((N/2+1):end)=f_even - twiddle_factors .* f_odd;
end
```

FFT: Empirical running time



Matrix DFT

$$F_N(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} = (W_N^{0k}, \ldots, W_N^{(N-1)k})\mathbf{x}^T$$

$$\mathbf{W_N} = \begin{bmatrix} W_N^{0\times 0} & W_N^{0\times 1} & \ldots & W_N^{0\times (N-1)} \\ W_N^{1\times 0} & W_N^{1\times 1} & \ldots & W_N^{1\times (N-1)} \\ & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & & \\ W_N^{(N-1)\times 0} & W_N^{(N-1)\times 1} & \ldots & W_N^{(N-1)\times (N-1)} \end{bmatrix}$$

$$(\mathbf{W_{N}}_{(k,n)} = W_N^{k\times n})$$

$$F_N(k) = (\mathbf{W_N}\mathbf{x}^T)_k$$

$$\mathbf{F_N}\mathbf{x} = \mathbf{W_N}\mathbf{x}^T$$

FFT Review

Calculating $F_N(k)$ for all k

1. Split $x(n)$ into $x_{odd}(n)$ and $x_{even}(n)$.
2. Calculate $F^{even}_{N/2}(k)$ and $F^{odd}_{N/2}(k)$ for all k
3. Calculate $F_N(k)$ for all k as:

$$F_N(k) = \begin{cases} F^{even}_{N/2}(k) + W^k_N F^{odd}_{N/2}(k) & \text{if } k = 0, \dots, N/2-1 \\ F^{even}_{N/2}(k') - W^{k'}_N F^{odd}_{N/2}(k') & \text{if } k = N/2, \dots, N-1 \end{cases} \quad (2)$$

$(k = k' + N/2)$

Factorization

Operations **1**, **2** and **3** can be expressed as a matrix too.

Matrix FFT: Factorization

$$\mathbf{F_N x} = \begin{bmatrix} \mathbf{I} & \mathbf{D_N} \\ \mathbf{I} & -\mathbf{D_N} \end{bmatrix} \begin{bmatrix} \mathbf{F_{N/2}} & 0 \\ 0 & \mathbf{F_{N/2}} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \\ 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix} \mathbf{x}$$

$$= \mathbf{C}_N \begin{bmatrix} \mathbf{F_{N/2}} & 0 \\ 0 & \mathbf{F_{N/2}} \end{bmatrix} \mathbf{P}_N \mathbf{x}$$

D diagonal, with twiddle factors

$$\mathbf{D_{N(i,i)}} = W_N^i, \qquad i = 0, \dots, N-1 \tag{3}$$

Matrix FFT: Factorization

$$
\begin{aligned}
\mathbf{F_N x} &= \mathbf{C}_N \mathbf{C}_{N/2} \begin{bmatrix} \mathbf{F_{N/2^2}} & 0 & 0 & 0 \\ 0 & \mathbf{F_{N/2^2}} & 0 & 0 \\ 0 & 0 & \mathbf{F_{N/2^2}} & 0 \\ 0 & 0 & 0 & \mathbf{F_{N/2^2}} \end{bmatrix} \mathbf{P}_{N/2} \mathbf{P}_N \mathbf{x} \\
&= \mathbf{C}_N \mathbf{C}_{N/2} \mathbf{C}_{N/2^2} \ldots \mathbf{C}_1 \mathbf{I} \mathbf{P}_1 \ldots \mathbf{P}_{N/2^2} \mathbf{P}_{N/2} \mathbf{P}_N \mathbf{x} \\
&= \mathbf{C}_N \mathbf{C}_{N/2} \mathbf{C}_{N/2^2} \ldots \mathbf{C}_1 \mathbf{P}_1 \ldots \mathbf{P}_{N/2^2} \mathbf{P}_{N/2} \mathbf{P}_N \mathbf{x}
\end{aligned}
$$

Matrix FFT: Iterative

Calculating $F_N(k)$

1. Apply $\log_2(n)$ permutations $\mathbf{P_i}$ to $\mathbf{x}$ to obtain $\mathbf{x}'$
2. Apply $\log_2(n)$ combination operations $\mathbf{C_i}$ to $\mathbf{x}'$ to obtain $F_N(\mathbf{x})$

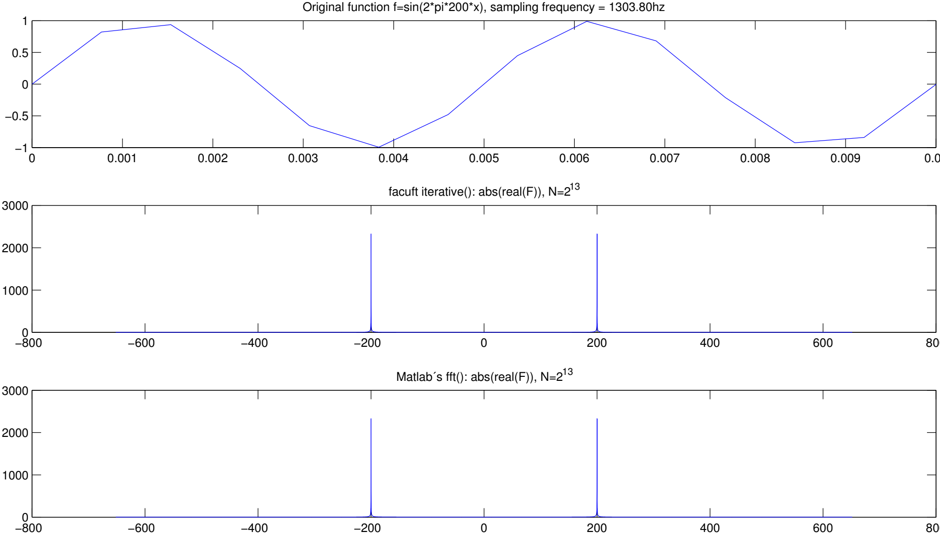
Running time

- Even though each permutation and combination operation is represented by an N-by-N matrix, it can be applied in $O(n)$ time
- Each item above is therefore $O(n \log(n))$
- Total running time is still $O(n \log(n))$

Iterative FFT: Implementation

```
function f=facuft_iterative(x)
    f=permutations(x); %O(n log(n))
    f=combinations(f); %O(n log(n))
end
function x=permutations(x)
    N=length(x); %O(n)
    levels=log2(N); %O(1)
    for l=0:(levels-2) %O(n log(n))
        x=permutation_for_level(l,x,N); %O(n)
    end
end
function f=combinations(f)
    N=length(f); %O(n)
    levels=log2(N); %O(1)
    for l=(levels-1):-1:0 %O(n log(n))
        f=combinations_for_level(l,f,N); %O(n)
    end
end
```

Iterative FFT: Output



Iterative FFT: Empirical running time