Fourier Transforms –
The *Fast* Fourier Transform

CS370 Lecture 22 – March 6, 2017

DFT By Definition

Our definition of the DFT is $F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}$.

Its inverse, the IDFT is $f_n = \sum_{k=0}^{N-1} F_k W^{nk}$.

Today:

- Write these in matrix form.
- Make them ***fast***.

A Matrix View of DFT

Our usual DFT is $F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}$.

If F and f be vectors of Fourier coefficients and input data, we can write the DFT as a matrix multiplication,

$$F = Mf$$

where M is a matrix where whose k^{th} column is: $\frac{1}{N}\begin{bmatrix} W^0 \\ W^{-k} \\ W^{-2k} \\ \dots \\ W^{-(N-1)k} \end{bmatrix}$.

A Matrix View of DFT

e.g., For $N = 4$, we have $F_k = \frac{1}{4}\sum_{n=0}^{3} f_n W^{-nk}$, which in matrix form is

$$\underbrace{\frac{1}{4}\begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^{-1} & W^{-2} & W^{-3} \\ W^0 & W^{-2} & W^{-4} & W^{-6} \\ W^0 & W^{-3} & W^{-6} & W^{-9} \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}}_{f} = \underbrace{\begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{bmatrix}}_{F}$$

$$M \quad f = F$$

A Matrix View of IDFT

Recall our orthogonality identity:
$$\sum_{j=0}^{N-1} W^{jk} W^{-jl} = \sum_{j=0}^{N-1} W^{j(k-l)} = N\delta_{k,l}$$

Our orthogonality identity leads to $\overline{M^T} M = \frac{1}{N} I$, where I is the identity matrix. For $N = 4$:

$$\underbrace{\frac{1}{4}\begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix}}_{\overline{M^T}} \underbrace{\frac{1}{4}\begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^{-1} & W^{-2} & W^{-3} \\ W^0 & W^{-2} & W^{-4} & W^{-6} \\ W^0 & W^{-3} & W^{-6} & W^{-9} \end{bmatrix}}_{M} = \underbrace{\frac{1}{4}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\frac{1}{N} I}$$

Therefore $M^{-1} = N\overline{M^T}$.

The IDFT becomes $f = M^{-1}F = N\overline{M^T}F$.

A Matrix View of IDFT

e.g., For $N = 4$, we have IDFT $f_n = \sum_{k=0}^{3} F_k W^{nk}$ in matrix form:

$$\underbrace{4}_{N} \cdot \underbrace{\frac{1}{4}\begin{bmatrix} 1 & W^0 & W^0 & W^0 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^4 & W^6 \\ 1 & W^3 & W^6 & W^9 \end{bmatrix}}_{\overline{M^T}} \underbrace{\begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{bmatrix}}_{F} = \underbrace{\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}}_{f}$$

$$N \quad \overline{M^T} \quad F = f$$

Summary: Matrix Form of DFT pair

DFT: $F = Mf$

Inverse DFT: $f = M^{-1}F = N\overline{M^T}F$

where the k^{th} column of M is $\frac{1}{N}\begin{bmatrix} W^0 \\ W^{-k} \\ W^{-2k} \\ \dots \\ W^{-(N-1)k} \end{bmatrix}$

Making Fourier Transforms Fast

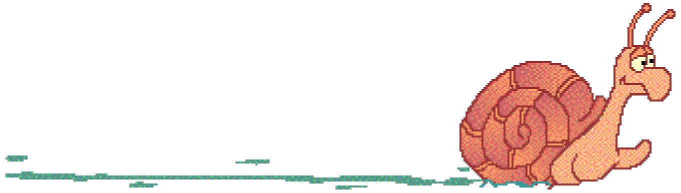
We’ve examined the discrete Fourier transform and its inverse.

Next, we want to determine a more *efficient* way to compute them.

Questions:

- What is the complexity of the naïve method?
- What properties of the DFT allow it to be sped up?
- What is the complexity of the new method?

Slow Fourier Transform



A direct implementation of $F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}$ takes $O(N^2)$ complex floating point operations. Essentially two nested ***for*** loops:

For $k = 0{:}N-1$ //iterate over all k unknown coeffs
    $F_k = 0$ //initialize coefficient to zero
    For $n = 0{:}N-1$ //iterate over all n data values
        $F_k \mathrel{+}= f_n W^{-nk}$ //increment by scaled data value
    End
    $F_k = F_k / N$ //normalize
End

A Faster Fourier Transform

Design a *divide and conquer* strategy.

We'll:

1. Split the full DFT into *two* DFT's of *half* the length.
2. Repeat recursively.
3. Finish at the base case: the DFT of individual pairs of numbers.

(If $N \neq 2^m$ for some m, we can pad our initial data with zeros.)

Key question: *How* can we split up the DFT?

Dividing it up

The usual DFT of the sequence f_n is:

$$F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk}$$

It can be split into two separate DFT's of two arrays of ***half the length*** (N/2):

$$g_n = \frac{1}{2}\left(f_n + f_{n+\frac{N}{2}}\right)$$

$$h_n = \frac{1}{2}\left(f_n - f_{n+\frac{N}{2}}\right) W^{-n}$$

The "W-factor" (or sometimes twiddle factor).

where $n \in \left[0, \frac{N}{2} - 1\right]$. Then $F_{even} = DFT(g)$ and $F_{odd} = DFT(h)$.

Dividing it up - Example

$$f = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 3 \\ 2 \\ 2 \\ 1 \\ 3 \end{bmatrix} \qquad \begin{bmatrix} 2 \\ 2 \\ 1 \\ 3 \end{bmatrix} = g$$

$$\begin{bmatrix} 0 \cdot W^0 \\ 0 \cdot W^{-1} \\ 0 \cdot W^{-2} \\ 0 \cdot W^{-3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = h$$

Apply

$$g_n = \frac{1}{2}\left(f_n + f_{n+\frac{N}{2}}\right)$$

$$h_n = \frac{1}{2}\left(f_n - f_{n+\frac{N}{2}}\right) W^{-n}$$

for $n = 0,1 \ldots \frac{N}{2} - 1.$

DFT of g is $G = \left[2, \frac{1+i}{4}, -\frac{1}{2}, \frac{1-i}{4}\right]$. (e.g. found by def'n.)

DFT of h is $H = [0,0,0,0]$.

Recover DFT of f by ***interleaving*** G & H (even and odd entries, respectively):

$$F = \left[2, 0, \frac{1+i}{4}, 0, -\frac{1}{2}, 0, \frac{1-i}{4}, 0\right].$$

G H

Dividing it up

How did we arrive at this specific splitting?

$$g_n = \frac{1}{2}\left(f_n + f_{n+\frac{N}{2}}\right)$$
$$h_n = \frac{1}{2}\left(f_n - f_{n+\frac{N}{2}}\right)W^{-n}$$

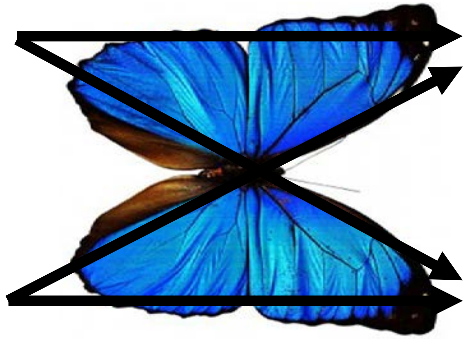
Let’s see how we can manipulate our DFT definition into this form.

Visualizing – A Butterfly operation

We can think of each step as taking a pair of numbers and producing two outputs:

f_n

$f_{n+\frac{N}{2}}$

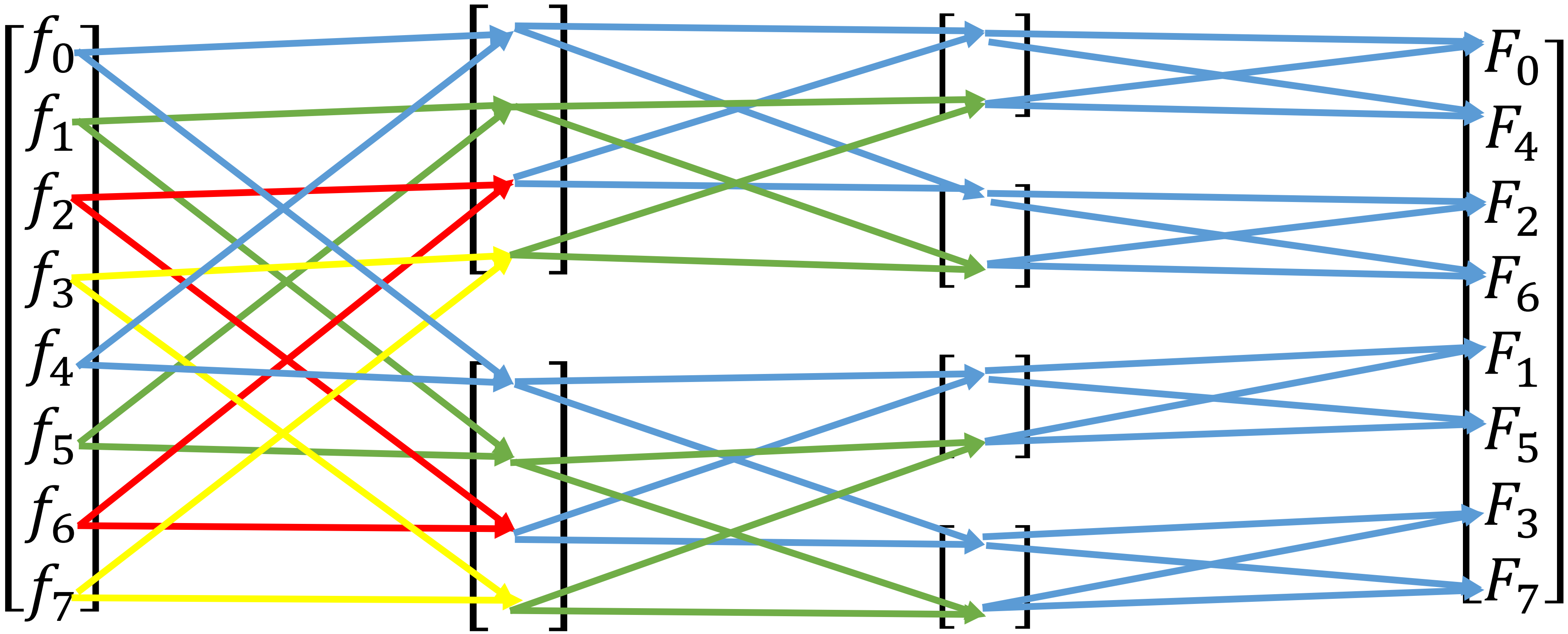


$$\overbrace{\frac{1}{2}\left(f_n + f_{n+\frac{N}{2}}\right)}^{g_n}$$

$$\underbrace{\frac{1}{2}\left(f_n - f_{n+\frac{N}{2}}\right) W^{-n}}_{h_n}$$

The (admittedly mild) resemblance to a butterfly gives its name.

Big Picture – Recursive Butterfly FFT alg'm

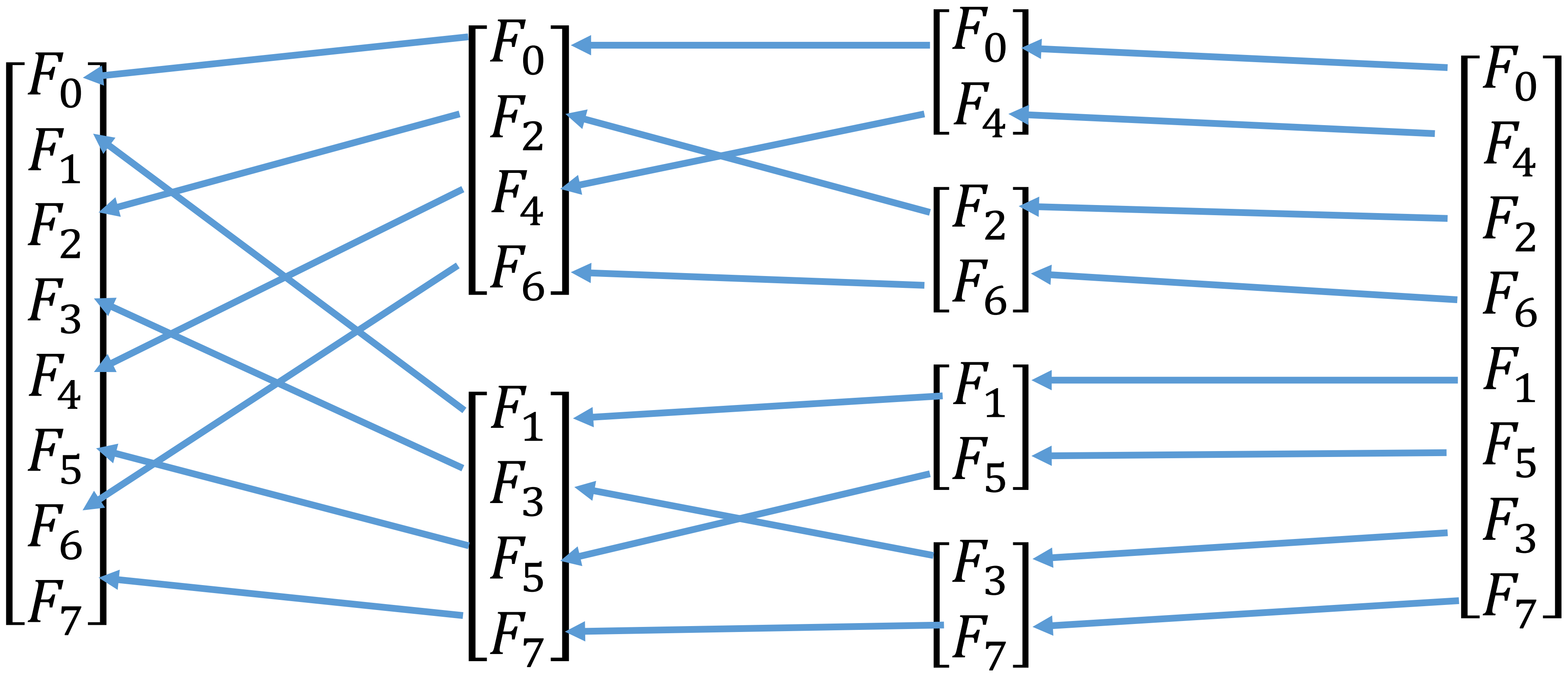


$N = 8 = 2^3$, so we have 3 recursive stages.

Coefficient output order is scrambled!

Reassembling The Result Vector

Each step applies an even-odd index splitting for the result location. So, the output ends up in “bit-reversed” positions, which we must undo.



Bit-Reversed Output

Consider the binary indices of the data: output indices have bits in reverse order!

So we must “unscramble” the coefficients as a final step.

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix} = \begin{bmatrix} f_{000} \\ f_{001} \\ f_{010} \\ f_{011} \\ f_{100} \\ f_{101} \\ f_{110} \\ f_{111} \end{bmatrix} \xrightarrow{\text{Perform All Butterfly Stages}} \begin{bmatrix} F_0 \\ F_4 \\ F_2 \\ F_6 \\ F_1 \\ F_5 \\ F_3 \\ F_7 \end{bmatrix} = \begin{bmatrix} F_{000} \\ F_{100} \\ F_{010} \\ F_{110} \\ F_{001} \\ F_{101} \\ F_{011} \\ F_{111} \end{bmatrix}$$