Computing the DFT

DFT: \( X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \)

\( W_N = e^{-j\frac{2\pi}{N}} \)

IDFT: \( x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \)

- DFT is \( N \) equally-spaced samples of DT Fourier transform
  \( \omega_k = \frac{2\pi k}{N}, \quad 0 \leq \omega_k < 2\pi \)

- Note similarity of DFT and IDFT
  - given DFT implementation \( \rightarrow \) easy to get IDFT implementation

\( \Rightarrow \) Computational complexity is the key implementation issue

Computational complexity

Measures of complexity:
- Number of multiplies and adds
- Chip area
- Power requirements

Direct implementation of the DFT:
\[ \sum_{n=0}^{N-1} x[n] W_N^{kn} \]

Each sample \( X[k] \) requires:
\( N \) complex multiplies \( \quad N - 1 \) complex adds

Thus an \( N \)-pt DFT requires: \( \mathcal{O}(N^2) \) computations

\( \Rightarrow \) For large \( N \), computational burden is huge
Computing the DFT (continued)

We want algorithms for DFT that reduce number of computations. These methods usually exploit symmetry/periodicity of $W_N^{kn}$.

Brief history:

- 1805: Gauss – origins of the fast Fourier transform
- 1905: Runge
- 1942: Danielson/Lanczos $\leftarrow O(N \log N)$
- 1965: Cooley and Tukey
  - developed fast algorithms for DFT when $N$ is a composite number (i.e., $N$ is the product of two or more integers)

$\Rightarrow$ Cooley and Tukey’s work sparked a flurry of research leading to the FFT

Fast Fourier Transform

FFT is an algorithm for computing the DFT

Basic idea (Cooley & Tukey):

Divide and Conquer! $\rightarrow$ split DFT into smaller and smaller transforms

If $N$ is a power of 2: $N = 2^\nu$

- Number of computations is $O(N \log_2 N)$
- Called radix-2

Two categories:

- Decimation-in-Frequency
  - compute even/odd frequency samples separately
- Decimation-in-Time
  - compute using even/odd time samples separately
Decimation in frequency algorithm

Suppose we have a 1024-point signal

- What length DFT is sufficient to represent this signal? **1024 points**

- Suppose only even DFT samples are needed. How can we get them?
  Undersampling in frequency \( \Rightarrow \) time aliasing
  Thus: time-alias and compute 512-point transform to get even samples

- Suppose only odd DFT samples are needed. How can we get them?
  - Multiply time sequence by complex exponential
    \( \Rightarrow \) this produces shift in frequency domain
  - Time alias the new sequence
  - Take 512-point transform

---

Decimation in frequency: even samples (Assume \( N \) is a power of 2)

To get even DFT samples:

\[
X[2r] = \sum_{n=0}^{N-1} x[n] W_N^{2r n} \quad r = 0, \ldots, \frac{N}{2} - 1
\]

\[
= \sum_{n=0}^{N/2-1} x[n] W_N^{2r n} + \sum_{n=-N/2}^{N-1} x[n] W_N^{2r n}
\]

\[
= \sum_{n=0}^{N/2-1} x[n] W_N^{2r n} + \sum_{n=0}^{N/2-1} x \left[ n + \frac{N}{2} \right] W_N^{2r n} W_N^{2r \left( \frac{N}{2} \right)}
\]

Note: \( W_N^{2r \left( \frac{N}{2} \right)} = W_N^{rN} = e^{-j2\pi rN} = e^{-j2\pi r} = 1 \)

Even samples: \( X[2r] = \sum_{n=0}^{N/2-1} \left( x[n] + x \left[ n + \frac{N}{2} \right] \right) W_N^{r n} \)

\( \frac{N}{2} \)-pt DFT of time-aliased sequence
Decimation in frequency: odd DFT samples

Odd DFT samples:

\[ X[2r+1] = \sum_{n=0}^{N/2-1} \left[ x[n]W_N^n + x \left[ n + \frac{N}{2} \right] W_N^n W_N^{nr} e^{-j\pi} \right] W_N^{nr} \]

\[ X[2r+1] = \sum_{n=0}^{N/2-1} \left[ x[n] - x \left[ n + \frac{N}{2} \right] \right] W_N^n W_N^{nr} \]

\( N/2 \)-pt DFT of time-aliased & modulated sequence

- \( x[n] - \left[ n + \frac{N}{2} \right] \) is another form of time-aliasing
- the \( W_N^n \) term is the modulation

8-point DFT example \( \Rightarrow \)

Decimation in frequency example: 8-point DFT

- \( W_N \) terms are called twiddle factors

Decompose \( \frac{N}{2} \) DFT's into smaller DFT's \( \Rightarrow \)
Decimation in frequency example: 8-point DFT

- \( \frac{N}{4} \)-point DFT's are 2-point DFT's if \( N = 8 \)

What does a 2-point DFT look like?

2-point DFT

\[
X[k] = \sum_{n=0}^{1} x[n]e^{-j\frac{2\pi}{2}nk} = x[0]e^{-j0} + x[1]e^{-j\pi k}
\]

Thus the 2-point DFT is trivial: just add and subtract!

\[
X[0] = x[0] + x[1] \\
X[1] = x[0] - x[1]
\]

Flowgraph for 2-point DFT:

- Known as a butterfly calculation

Insert butterfly for 2-pt DFT into structure
Decimation in frequency example: 8-point DFT

- Butterfly structure
- Input in normal order; output is bit-reversed

FFT Computations

First stage:

\[ \frac{N}{2} \] complex multiplies  \[ N \] complex adds

Can decompose into \( \log_2 N \) stages. Final stage has 2-pt DFT’s.

Comparison: decimation-in-frequency vs. direct

<table>
<thead>
<tr>
<th>Operation</th>
<th>Decimation-in-frequency</th>
<th>Direct</th>
</tr>
</thead>
<tbody>
<tr>
<td>complex multiplies</td>
<td>( \frac{N}{2} \log_2 N )</td>
<td>( N^2 )</td>
</tr>
<tr>
<td>complex adds</td>
<td>( N \log_2 N )</td>
<td>( N(N - 1) )</td>
</tr>
</tbody>
</table>

Example: \( N = 1024 \)  \( N^2 = 1,048,576 \)  \( N \log_2 N = 10,240 \)  
Computations reduced by 2 orders of magnitude!
Other FFT algorithms, e.g., decimation in time

Recall tranposition theorem for flowgraphs:
- interchange input and output
- reverse direction of all arrows
- Result is a flowgraph with the same input/output behavior

Transpose decimation-in-frequency flowgraph
⇒ decimation-in-time flowgraph

Which implementation/flowgraph to use?
- In-place calculation
- Indexing strategy
- Random or sequential access memory
- Store coefficients or compute recursively