FAST FOURIER TRANSFORM

EEEN 462 – ANALOGUE COMMUNICATION

Friday, December 19, 2025

FOURIER TRANSFORM (RECAP)

1. **Fourier Transform** decomposes a function into its constituent frequencies.

$$X(f) = \int_{-\infty}^{\infty} x(t)\, e^{-j2\pi ft}\, dt$$

1. **For continuous signals,** we use the Continuous Fourier Transform (CFT).
2. **For discrete signals,** we use the Discrete Fourier Transform (DFT).
3. **Key Insight:** Any periodic signal can be represented as a sum of sinusoids with different frequencies, amplitudes, and phases.

DISCRETE FOURIER TRANSFORM - RECAP

1. **Discrete-time signals**, we use the DFT:

$$X[k] = \Sigma_{n=0}^{N-1} x[n] \cdot e^{-j(2\pi/N)kn}, \text{ for } k = 0, 1, \ldots, N-1$$

where:

- x[n] is the discrete-time signal (N samples)
- X[k] is the DFT output (frequency bins)
- N is the number of samples
- k represents frequency index

2. **Inverse DFT (IDFT)** reconstructs the time-domain signal from frequency components:

$$x[n] = (1/N)\, \Sigma_{k=0}^{N-1} X[k] \cdot e^{j(2\pi/N)kn}, \text{ for } n = 0, 1, \ldots, N-1$$

THE NEED FOR FAST FOURIER TRANSFORM

Direct computation of DFT requires $O(N^2)$ operations:

- For each of N frequency bins (k values), we need N complex multiplications and N-1 additions
- Total: $N \times N = N^2$ complex multiplications

N (Samples)	DFT Operations	FFT Operations	Speed-up Factor
64	4,096	384	10.7x
256	65,536	2,048	32x
1024	1,048,576	10,240	102x
4096	16,777,216	49,152	341x

FFT USES DIVIDE AND CONQUER APPROACH

The key insight behind FFT is the divide-and-conquer strategy as follows:

1. Divide the N-point DFT into two N/2-point DFTs
2. Exploit symmetry and periodicity of the complex exponential (twiddle factors)
3. Recursively apply until we reach 2-point DFTs (butterflies)

RADIX-2 FFT ALGORITHM

- The most common FFT is the r**adix-2 Cooley-Tukey algorithm**, which requires N to be a power of 2.
- Algorithm steps:
  1. If N = 1, return x[0] (base case)
  2. Separate x[n] into even and odd indexed samples:
     - Even: $x_e[m] = x[2m]$ for m = 0, 1, ..., N/2-1
     - Odd: $x_o[m] = x[2m+1]$ for m = 0, 1, ..., N/2-1
  3. Compute N/2-point FFT of both sequences: $X_e[k]$ and $X_o[k]$
  4. Combine using the butterfly operation:

$$X[k] = X_e[k] + W_N^k \cdot X_o[k]$$
$$X[k+N/2] = X_e[k] - W_N^k \cdot X_o[k]$$

where $W_N^k = e^{-j(2\pi/N)k}$ are the twiddle factors

The algorithm recursively applies this decomposition until reaching 2-point DFTs (butterflies). Each stage has N/2 butterflies, and there are $\log_2 N$ stages.

FFT BUTTERFLY STRUCTURE

1. The butterfly is the fundamental computational unit of the FFT:
2. Each butterfly performs:
   1. One complex multiplication ($B \times W_N^k$)
   2. One complex addition ($A + WB$)
   3. One complex subtraction ($A - WB$)

3. **Key Property:** The butterfly structure exploits the symmetry of twiddle factors: $W_N^{k+N/2} = -W_N^k$, which reduces computations by half.

DECIMATION-IN-TIME (DIT) FFT

1. In Decimation-in-Time (DIT) FFT, we split the time-domain sequence into even and odd parts:

$$X[k] = \Sigma_{n\ even}\ x[n]\ W_N^{kn} + \Sigma_{n\ odd}\ x[n]\ W_N^{kn}$$
$$= \Sigma_{m=0}^{N/2-1}\ x[2m]\ W_N^{2mk} + \Sigma_{m=0}^{N/2-1}\ x[2m+1]\ W_N^{k(2m+1)}$$

2. Since $W_N^2 = W_{N/2}$, we get:

$$X[k] = \Sigma_{m=0}^{N/2-1}\ x[2m]\ W_{N/2}^{mk} + W_N^k\ \Sigma_{m=0}^{N/2-1}\ x[2m+1]\ W_{N/2}^{mk}$$
$$= X_e[k] + W_N^k\ X_o[k]$$

DECIMATION-IN-FREQUENCY (DIF) FFT

- In Decimation-in-Frequency (DIF) FFT, we split the frequency-domain sequence into even and odd parts:

- For k even: $X[2r] = \Sigma_{n=0}^{N/2-1} (x[n] + x[n+N/2]) W_{N/2}^{rn}$

- For k odd: $X[2r+1] = \Sigma_{n=0}^{N/2-1} (x[n] - x[n+N/2]) W_N^n W_{N/2}^{rn}$

KEY DIFFERENCES BETWEEN DIT AND DIF

Aspect	DIT FFT	DIF FFT
Decomposition	Time sequence split	Frequency sequence split
Butterfly Input	Natural order	Natural order
Butterfly Output	Bit-reversed order	Natural order
Twiddle Factor	Between stages	Before butterfly
Complexity	Same: O(N log N)	Same: O(N log N)

FFT COMPLEXITY ANALYSIS

1. The computational savings of FFT compared to DFT are dramatic:
   a) DFT: N^2 complex multiplications
   b) FFT: $(N/2) \log_2 N$ complex multiplications
2. For an N-point FFT:
   - Number of stages: $\log_2 N$
   - Butterflies per stage: $N/2$
   - Complex multiplications per butterfly: 1
   - Total complex multiplications: $(N/2) \log_2 N$
   - Complex additions: $N \log_2 N$

WRITING FFT FUNCTION IN MATLAB - *(Radix-2 Cooley-Tukey)*

```
function X = myFFT(x)
N = length(x);
    if bitand(N, N-1) ~= 0                      % Check if N is power of 2
      error('Input length must be a power of 2');
   end
  if N == 1  % Base case
      X = x;
      return;
   end
even = myFFT(x(1:2:end));                   % Split into even and odd indices
   odd = myFFT(x(2:2:end));
   k = 0:N/2-1;                              % Compute twiddle factors
   W = exp(-2*pi*1i*k/N);                    % Twiddle factors
X = [even + W.*odd, even - W.*odd];          % Combine results
end
```

APPLICATIONS OF FFT

1. **Spectral Analysis**
    - Analyzing frequency content of signals (audio, vibrations, RF)
2. **Communications**

    OFDM in 4G/5G, DSL, WiFi, software-defined radio
3. **Image Processing**

    JPEG compression, filtering, convolution via multiplication
4. **Medical**

    MRI, ECG analysis, ultrasound imaging
5. **Other applications:** audio compression (MP3), speech recognition, radar, sonar, seismic analysis, and solving partial differential equations.

EXAMPLE: USING INBUILT FFT FUNCTION IN MATLAB

```
% Generate a test signal
Fs = 1000; % Sampling frequency
t = 0:1/Fs:1-1/Fs; % Time vector
f1 = 50; f2 = 120; % Frequencies
x = 0.7*sin(2*pi*f1*t) + sin(2*pi*f2*t);

% Compute FFT
N = length(x);
X = fft(x);
X_mag = abs(X); % Magnitude spectrum

% Frequency vector
f = (0:N-1)*(Fs/N);
plot(f, X_mag)
xlabel('Frequency (Hz)')
ylabel('Magnitude')
```

PRACTICAL CONSIDERATIONS

1. Windowing

- Apply window functions (Hamming, Hanning) to reduce spectral leakage from finite observation intervals.

2. Aliasing

- Ensure sampling rate ≥ 2× maximum frequency (Nyquist theorem) to avoid aliasing.

3. Zero Padding

- Add zeros to increase FFT size for better frequency resolution (interpolation in frequency domain).

CONCLUSION

1. Key Takeaways:

- FFT reduces DFT complexity from $O(N^2)$ to $O(N \log N)$
- Radix-2 FFT uses divide-and-conquer with butterfly operations
- Both DIT and DIF approaches provide the same computational benefits
- FFT enables real-time spectral analysis in countless applications

2. Why FFT Matters for EE Students

- Understanding FFT is essential for:
- Digital signal processing (DSP) system design
- Communications engineering (OFDM, software-defined radio)
- Image and audio processing applications
- Embedded systems with real-time signal processing requirements