From DFT to Fast FFT: A Practical Derivation and Implementation Notes

LineInTuner Notes

September 12, 2025

1 The Discrete Fourier Transform (DFT)

Given a length-N sequence $x[n]$, the DFT is

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi kn/N}, \quad k = 0, 1, \ldots, N-1. \tag{1}$$

This direct computation is $\mathcal{O}(N^2)$: N outputs, each summing N complex multiplications.

In many applications we only need the magnitude spectrum $|X[k]| = \sqrt{\mathrm{Re}\{X[k]\}^2 + \mathrm{Im}\{X[k]\}^2}$.

2 Even/Odd Decomposition (Why FFT Works)

Assume N is even. Split the input into its even and odd-indexed subsequences:

$$x_e[r] = x[2r], \tag{2}$$

$$x_o[r] = x[2r+1], \qquad r = 0, \ldots, \frac{N}{2} - 1. \tag{3}$$

Then

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \tag{4}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x_e[r] e^{-j2\pi k(2r)/N} + \sum_{r=0}^{\frac{N}{2}-1} x_o[r] e^{-j2\pi k(2r+1)/N} \tag{5}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x_e[r] e^{-j2\pi kr/(N/2)} + e^{-j2\pi k/N} \sum_{r=0}^{\frac{N}{2}-1} x_o[r] e^{-j2\pi kr/(N/2)} \tag{6}$$

$$= E[k \bmod N/2] + W_N^k\, O[k \bmod N/2], \tag{7}$$

where $E[\cdot]$ and $O[\cdot]$ are the $N/2$-point DFTs of x_e and x_o, and $W_N^k = e^{-j2\pi k/N}$ are the *twiddle factors*.

This identity expresses an N-point DFT in terms of two $(N/2)$-point DFTs and $\mathcal{O}(N)$ additional work. Recursing yields $\mathcal{O}(N \log_2 N)$ complexity.

3 Radix-2 Decimation-in-Time (DIT) FFT

For $N = 2^m$, the DIT FFT proceeds in $m = \log_2 N$ stages. An efficient *iterative* in-place algorithm has two key components:

3.1 Bit-Reversal Permutation

The recursive even/odd splitting implies a particular data access order. The iterative algorithm first permutes the input to *bit-reversed* order: interpret the index i in binary with m bits and reverse those bits to obtain j; swap entries at i and j when $i < j$. This ensures subsequent stages operate on contiguous subproblems.

3.2 Butterfly Computation

Let m denote the butterfly size at a given stage, doubling each stage: $m = 2, 4, 8, \ldots, N$. Define the primitive twiddle for the stage as

$$w_m = e^{-j2\pi/m}. \tag{8}$$

Process the array in blocks of length m. Within each block, for $j = 0, 1, \ldots, m/2 - 1$, maintain a running twiddle $w = w_m^j$ and perform the butterfly:

$$\begin{aligned} t &= w \cdot a[i + j + m/2], &(9)\\ u &= a[i + j], &(10)\\ a[i + j] &\leftarrow u + t, &(11)\\ a[i + j + m/2] &\leftarrow u - t. &(12) \end{aligned}$$

Here $a[\cdot]$ is the in-place complex working array. Advancing w by a complex multiply $w \leftarrow w \cdot w_m$ avoids repeated trigonometric calls.

After the final stage, $a[k] = X[k]$. For magnitude output, return $|a[k]|$.

4 Why Bit-Reversal and Butterflies Yield the DFT

The even/odd decomposition proves that an N-point DFT equals a combination of two $(N/2)$-point DFTs with twiddles. Applying the same decomposition recursively yields a computation DAG (dependency graph). The bit-reversal permutation reorders the input so that the DAG can be evaluated *iteratively* with contiguous memory accesses: each stage merges pairs of sub-transforms already computed at the previous stage. The butterflies are exactly those merge operations. Thus, the iterative algorithm computes precisely the same $X[k]$ as the definition.

5 Complexity and Practical Notes

- **Complexity**: $\mathcal{O}(N \log_2 N)$ vs $\mathcal{O}(N^2)$ for direct DFT.
- **Twiddle reuse**: One $\sin / \cos$ per stage; use complex multiplies to step twiddles within the stage.
- **Sign convention**: The forward FFT uses the negative sign in the exponent, consistent with the DFT above.
- **Normalization**: Often omitted in the forward transform; apply $1/N$ or $2/N$ (single-sided) when converting to amplitude.
- **Real inputs**: A real FFT can halve computation/storage by exploiting conjugate symmetry, but a standard complex FFT is simpler to implement correctly first.
- **Windowing**: For non-coherent tones, apply a window (e.g., Hann) to reduce spectral leakage before the FFT.

6 Rust-Oriented Implementation Sketch

Assume input x of length $N = 2^m$ and separate real/imag arrays.

```
// 1) Initialize
for i in 0..N { re[i] = x[i]; im[i] = 0.0; }

// 2) Bit-reversal permutation (m = log2(N))
for i in 0..N {
  j = reverse_bits(i, m);
  if i < j { swap(re[i], re[j]); swap(im[i], im[j]); }
}

// 3) Butterfly stages
for msize in [2, 4, 8, ..., N] {
  theta = -2*pi / msize;
  (wm_sin, wm_cos) = sin_cos(theta);
  for block in (0..N step msize) {
    w_re = 1.0; w_im = 0.0; // w = 1
    for j in 0..msize/2 {
      i1 = block + j;
      i2 = i1 + msize/2;
      // t = w * a[i2]
      t_re = w_re * re[i2] - w_im * im[i2];
      t_im = w_re * im[i2] + w_im * re[i2];
      // u = a[i1]
      u_re = re[i1]; u_im = im[i1];
      // a[i1] = u + t
      re[i1] = u_re + t_re; im[i1] = u_im + t_im;
      // a[i2] = u - t
      re[i2] = u_re - t_re; im[i2] = u_im - t_im;
      // w *= wm
      tmp_re = w_re * wm_cos - w_im * wm_sin;
      tmp_im = w_re * wm_sin + w_im * wm_cos;
      w_re = tmp_re; w_im = tmp_im;
    }
  }
}

// 4) Magnitude
for k in 0..N { out[k] = sqrt(re[k]^2 + im[k]^2); }
```

7 Validation

- **Delta input**: $x[n] = \delta[n]$ yields $X[k] = 1$ for all k (unnormalized).
- **Coherent cosine**: $x[n] = \cos(2\pi k_0 n/N)$ produces peaks at k_0 and $N - k_0$ with magnitude $\approx N/2$.
- Compare the fast FFT output against a slow DFT on small N for parity.