Concepts and Algorithms of Scientific and Visual Computing

–Fast Fourier Transform–

CS448J, Autumn 2015, Stanford University

Dominik L. Michels

Fast Fourier Transform (FFT)

For practical application, given a signal $x \in \ell^2(\mathbb{Z})$, we only take a finite number of elements into account. W.l.o.g. we describe

$$\mathbf{x} = (x_0, \ldots, x_{n-1})$$

with a tuple of length n, which is a power of two.

We decompose $\mathbf{x}$ in elements with even respectively odd indexes:

$$\mathbf{x}_e = (x_0, x_2, \ldots, x_{n-2}), \quad \mathbf{x}_o = (x_1, x_3, \ldots, x_{n-1}).$$

The corresponding polynomials are defined by

$$p_{\mathbf{x}}(\chi) = \sum_{j=0}^{n-1} x_j \chi^j, \quad p_{\mathbf{x}_e}(\chi) = \sum_{j=0}^{\frac{n}{2}-1} x_{2j} \chi^j, \quad p_{\mathbf{x}_o}(\chi) = \sum_{j=0}^{\frac{n}{2}-1} x_{2j+1} \chi^j.$$

Fast Fourier Transform (FFT)

The identity

$$p_{\mathbf{x}}(x) = p_{\mathbf{x}_e}(x^2) + x\, p_{\mathbf{x}_o}(x^2)$$

follows by simple calculation.

Furthermore, for the primitive n-th root of unity $\Omega_n := \exp(2\pi i/n)$ holds $\Omega_n^{k-n} = \Omega_n^k$ for all $k \in \mathbb{Z}$, and therefore

$$p_{\mathbf{x}}(\Omega_n^k) = \begin{cases} p_{\mathbf{x}_e}(\Omega_n^{2k}) + \Omega_n^k \cdot p_{\mathbf{x}_o}(\Omega_n^{2k}) & \text{if } k \in \{0, \ldots, \frac{n}{2} - 1\} \\ p_{\mathbf{x}_g}(\Omega_n^{2k-n}) + \Omega_n^k \cdot p_{\mathbf{x}_o}(\Omega_n^{2k-n}), & \text{if } k \in \{\frac{n}{2}, \ldots, n-1\} \end{cases}.$$

Fast Fourier Transform (FFT)

According to this scheme, the Fourier transform of

$$\hat{\mathbf{x}} = \left(p_{\mathbf{x}}(1), p_{\mathbf{x}}(\Omega_n), \ldots, p_{\mathbf{x}}(\Omega_n^{n-1})\right)$$

can be computed from

$$\hat{\mathbf{x}}_e = \left(p_{\mathbf{x}_e}(1), p_{\mathbf{x}_e}(\Omega_n^2), \ldots, p_{\mathbf{x}_e}(\Omega_n^{n-2})\right), \quad \hat{\mathbf{x}}_o = \left(p_{\mathbf{x}_o}(1), p_{\mathbf{x}_o}(\Omega_n^2), \ldots, p_{\mathbf{x}_o}(\Omega_n^{n-2})\right)$$

leading to a recursive algorithm, the so-called Cooley-Tukey fast Fourier transform (FFT).

This works for signals with lengths given by powers of two; see [Cooley, Tukey 1965]. It was extended later in [Bluestein 1970] to arbitrary $n \in \mathbb{N}$.

Fast Fourier Transform (FFT)

Cooley-Tukey fast Fourier transform for an input signal vector $\mathbf{x} = (x_0, \ldots, x_{n-1})$.

```
function FFT(x, Ω_n)
begin
    if n = 1 then
        ζ_0 ← x_0
    end
    else
        x_e ← (x_0, x_2, ..., x_{n-2})        (φ_0, ..., φ_{n/2-1}) ← FFT(x_e, Ω_n^2)
        x_o ← (x_1, x_3, ..., x_{n-1})        (ψ_0, ..., ψ_{n/2-1}) ← FFT(x_o, Ω_n^2)
        for k ← 0 to n/2 − 1 do
            ζ_k ← φ_k + Ω_n^k · ψ_k
            ζ_{n/2+k} ← φ_k + Ω_n^{n/2+k} · ψ_k
        end
    end
    return (ζ_0, ..., ζ_{n-1})
end
```

Fast Fourier Transform (FFT)

The runtime of the Cooley-Tukey fast Fourier transform can be described in dependence of the signal length n by

$$T(n) = 2 \cdot T(n/2) + \mathcal{O}(n)$$

with $T(1) \in \mathcal{O}(1)$.

Iterative substitution leads to the time complexity

$$T(n) \in \mathcal{O}(n \log(n)).$$

Using the identity

$$\mathrm{FFT}^{-1}(\zeta, \Omega_n) = \frac{1}{n} \mathrm{FFT}(\zeta, \Omega_n^{-1})$$

we also obtain the time complexity $\mathcal{O}(n \log(n))$ for the inverse FFT.

Two-dimensional Discrete Fourier Transform

For a given finite two-dimensional signal

$$x : \{0, \ldots, n_1 - 1\} \times \{0, \ldots, n_2 - 1\} \to \mathbb{C},$$

its Fourier transform $\hat{x} : \{0, \ldots, n_1 - 1\} \times \{0, \ldots, n_2 - 1\} \to \mathbb{C}$ is given by

$$\hat{x}(k_1, k_2) = \frac{1}{\sqrt{n_1 n_2}} \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} x(j_1, j_2) \exp\left(-2\pi i \left(j_1 \frac{k_1}{n_1} + j_2 \frac{k_2}{n_2}\right)\right)$$

as a natural extension of the one-dimensional case.

Two-dimensional Discrete Fourier Transform

Moreover, the Fourier transform can be expressed using the equivalent representation

$$\begin{aligned}\hat{x}(k_1,k_2) &= \frac{1}{\sqrt{n_1 n_2}}\sum_{j_1=0}^{n_1-1}\sum_{j_2=0}^{n_2-1} x(j_1,j_2)\exp\left(-2\pi i\left(j_1\frac{k_1}{n_1}+j_2\frac{k_2}{n_2}\right)\right)\\ &= \frac{1}{\sqrt{n_2}}\sum_{j_2=0}^{n_2-1}\left(\frac{1}{\sqrt{n_1}}\sum_{j_1=0}^{n_1-1} x(j_1,j_2)\exp(-2\pi i\, j_1\frac{k_1}{n_1})\right)\exp\left(-2\pi i\, j_2\frac{k_2}{n_2}\right),\end{aligned}$$

in which the inner term (in brackets) represents a column-wise Fourier transform and the outer term represents a row-wise Fourier transform.

Each Fourier transform can be carried out using a FFT leading to the time complexity

$$n_2\cdot\mathcal{O}(n_1\log(n_1)) + n_1\cdot\mathcal{O}(n_2\log(n_2)) = \mathcal{O}(n\log(n))$$

with $n := n_1 n_2$. A parallelization with $c := \max\{n_1, n_2\}$ processors leads to $\mathcal{O}(c\log(c))$.

Multidimensional Discrete Fourier Transform

More general, the Fourier transform of for a signal

$$x : \{0, \dots, n_1 - 1\} \times \cdots \times \{0, \dots, n_d - 1\} \to \mathbb{C}$$

of dimension d is defined by

$$\hat{x}(k_1, \dots, k_d) = \frac{1}{\sqrt{\prod_{l=1}^{d} n_l}} \sum_{j_1=0}^{n_1-1} \dots \sum_{j_d=0}^{n_d-1} x(j_1, \dots, j_d) \exp\left(-2\pi i \sum_{l=1}^{d} j_l \frac{k_l}{n_l}\right).$$

As in the one-dimensional case, the time complexity is given by $\mathcal{O}(n \log(n))$ with $n := \prod_{l=1}^{n} n_l$ using the FFT.