Fast Fourier Transform

Recall we found that the exponential polynomial interpolating a function f at $x_j = \frac{2\pi j}{N}$ is given by:

$$P = \sum_{k=0}^{N-1} c_k E_k, \quad c_k = (f, E_k)_N$$

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} f(x_j)(\lambda^k)^j, \quad \lambda = e^{-2i\pi/N}$$

Thus computing P requires $\mathcal{O}(N^2)$ computations
(N operations to compute each of the N c_k).

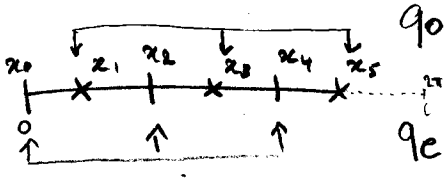
The FFT is based on a tremendous simplification that allows to compute P in $\mathcal{O}(N \log N)$ operations.

For example: $N = 2^{20} \approx 1M$

$$N^2 = 2^{40} \simeq 1G, \quad N \log_2 N \approx 20M$$

Basically factor of a million improvement! (in this case)

Here is the basic fact that is used by FFT:



$q_o \equiv$ trig interpolant at odd nodes
$q_o(x_{2j}) = f(x_{2j+1}) \quad 0 \le j \le n-1$

$q_e =$ trig interpolant at even nodes
$q_e(x_{2j}) = f(x_{2j}) \quad 0 \le j \le n-1$

$P \equiv$ trig interpolant at all nodes

$P(x_j) = f(x_j) \quad 0 \le j \le 2n-1$

$$P(x) = \frac{1}{2}(1 + e^{inx}) q_e(x) + \frac{1}{2}(1 - e^{inx}) q_o\left(x - \frac{\pi}{n}\right)$$

Proof:

q_o, q_e have degree $\le n-1$
$(e^{ix})^n$ has degree n
$\Rightarrow p$ has degree $\le 2n-1$

We need to show p interpolates f at the $2n$ nodes

$$x_0, x_1, \dots, x_{2n-1}; \quad x_j = \frac{\pi j}{n}, \quad 0 \le j < 2n-1$$

$$p(x_j) = \frac{1}{2}\left[1+E_n(x_j)\right] q_e(x_j) + \frac{1}{2}\left[1-E_n(x_j)\right] q_o\left(x_j - \frac{\pi}{n}\right)$$

$0 \le j \le 2n-1$

$$E_n(x_j) = \exp\left[i n \frac{\pi j}{n}\right] = e^{ij\pi} = \begin{cases} +1 & j \text{ even} \\ -1 & j \text{ odd} \end{cases}.$$

thus for j even:

$$p(x_j) = q_e(x_j) = f(x_j)$$

— for j odd:

$$p(x_j) = q_o\left(x_j - \frac{\pi}{n}\right) = q_o(x_{j-1}) = f(x_j) \quad \text{QED}$$

Now this relation is also useful in practice since we can get coeff of p from those of q_e and q_o easily.

Theorem

Let

$$q_e = \sum_{j=0}^{n-1} \alpha_j E_j$$

$$q_o = \sum_{j=0}^{n-1} \beta_j E_j$$

$$p = \sum_{j=0}^{2n-1} \gamma_j E_j$$

Then for $0 \le j \le n-1$:

$$\gamma_j = \frac{1}{2}\alpha_j + \frac{1}{2} e^{-ij\pi/n} \beta_j$$

$$\gamma_{j+n} = \frac{1}{2}\alpha_j - \frac{1}{2} e^{-ij\pi/n} \beta_j$$

proof:

$$q_0\left(x-\frac{\pi}{n}\right)=\sum_{j=0}^{n-1}\beta_j E_j\left(x-\frac{\pi}{n}\right)$$

$$=\sum_{j=0}^{n-1}\beta_j e^{ij(x-\pi/n)}=\sum_{j=0}^{n-1}\beta_j e^{-i\pi j/n}E_j(x)$$

$$P(x)=\frac{1}{2}\left(1+E_n(x)\right)p(x)+\frac{1}{2}\left(1-E_n(x)\right)q\left(x-\frac{\pi}{n}\right)$$

$$P=\frac{1}{2}\sum_{j=0}^{n-1}\left[(1+E_n)\alpha_j E_j+(1-E_n)\beta_j e^{-i\pi j/n}E_j\right]$$

$$=\frac{1}{2}\sum_{j=0}^{n-1}\underbrace{\left[\alpha_j+\beta_j e^{-i\pi j/n}\right]}E_j+\underbrace{\left[\alpha_j-\beta_j e^{-i\pi j/n}\right]}E_{j+n}$$

(since $E_j E_n=E_{j+n}$) QED.

The relation between odd and even interpolants can be made more general by using the following operators: (linear)

$L_n f$ = trig poly interpol nodes $x_j=\frac{2\pi j}{n}$, $0\le j\le n-1$

$(T_h f)(x)=f(x+h)$ (translation)

Of course we have:

$$L_n f=\sum_{k=0}^{n-1}(f,E_k)_n E_k$$

And in our formula:

$$P=L_{2n}f$$

$$q^e=L_n f$$

$$q^o=L_n T_{\pi/n} f$$

Thus:

$$L_{2n} f = \frac{1}{2}(1+E_n) L_n f + \frac{1}{2}(1-E_n) T_{-\pi/n} L_n T_{\pi/n} f \quad (*)$$

We now want to design an algorithm to compute $L_N f$ with $N = 2^m$.

Notation: ($\downarrow$ interp on 2^n nodes)

$$P_k^{(n)} = L_{2^n} T_{\frac{2k\pi}{N}} f \qquad 0 \le n \le m, \quad 0 \le k \le 2^{m-n} - 1$$

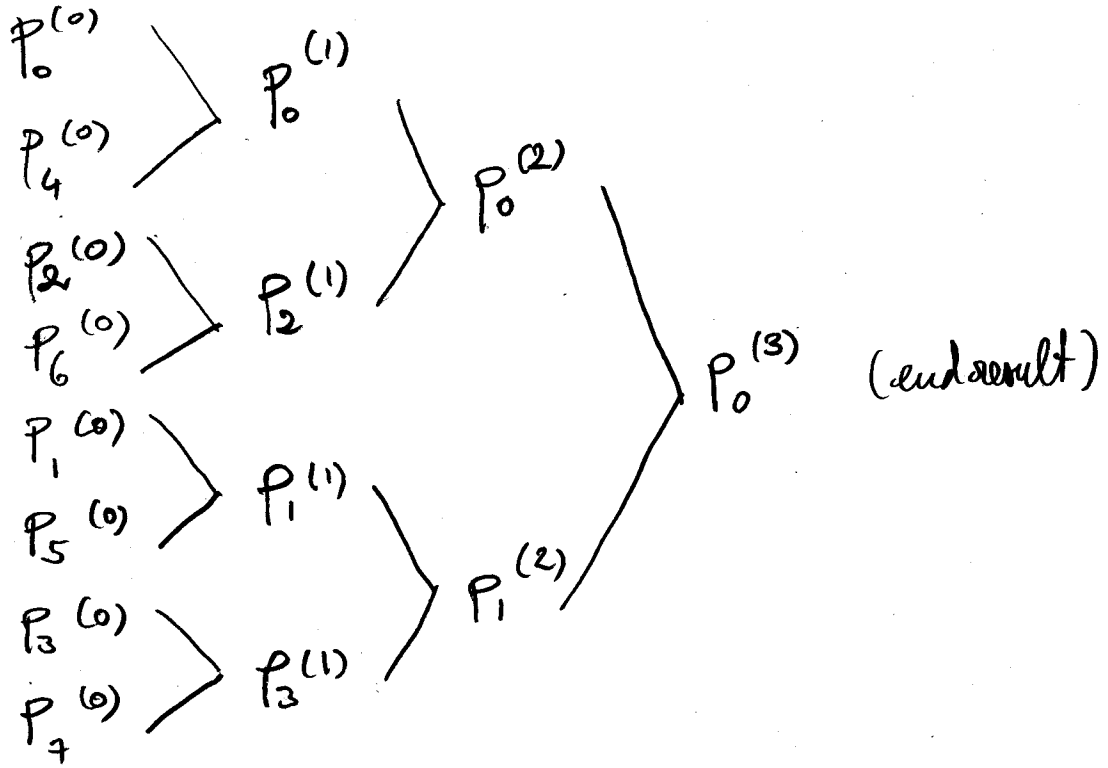
($P_k^{(n)}$: starting at node k)

$= $ trig interp of degree $2^n - 1$ s.t.

$$P_k^{(n)}\left(\frac{2\pi j}{2^n}\right) = f\left(\frac{2\pi k}{N} + \frac{2\pi j}{2^n}\right), \qquad 0 \le j \le 2^n - 1$$

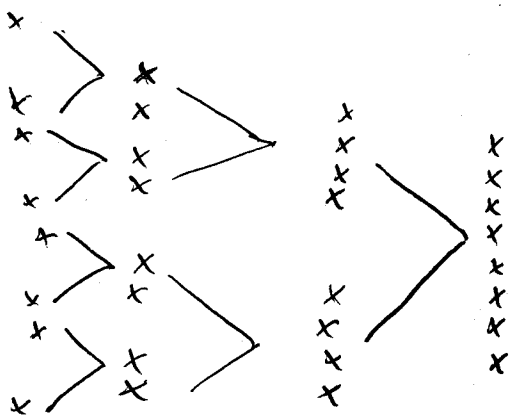
Applying $(*)$:

$$P_k^{(n+1)}(x) = \frac{1}{2}\left(1 + e^{i 2^n x}\right) P_k^{(n)}(x) + \frac{1}{2}\left(1 - e^{i 2^n x}\right) P_{k+2^{m-n-1}}^{(n)}\left(x - \frac{\pi}{2^n}\right)$$



2^0 pts $\quad$ 2^1 pts $\quad$ 2^2 pts $\quad$ 2^3 pts.

2^2 sep $\quad$ 2^1 sep. $\quad$ 2^0 sep $\quad$ 0 sep



How do we get an $N \log N$ operation count?

Let $R(n)$ be the #of multiplication needed to compute coeff in interp polynomial for points $\frac{2\pi i}{n}$, $0 \le j \le n-1$.

Then clearly:

$$\underbrace{R(2n)}_{\substack{\text{interp poly} \\ \text{coef for } 2n \text{ pts.}}} \le \underbrace{2R(n)}_{\substack{\text{compute interp} \\ \text{poly on even \& odd}}} + \underbrace{2n}_{\substack{\text{mult to convert} \\ \text{even \& odd into true one.}}}$$

We shall show that $R(2^m) \leq m 2^m$ by induction.

$m=0$: no multiplications are involved because constant interpolant

$\Rightarrow R(2^0) = 0$

Assume $R(2^m) \leq m 2^m$ holds:

$$R(2^{m+1}) = R(2\,2^m) \leq 2R(2^m) + 2\,2^m$$

$$\leq 2m2^m + 2^{m+1} = (m+1)2^{m+1}$$

So total number of operations is $\mathcal{O}(N \log_2 N) = \mathcal{O}(m2^m)$

In Matlab: fft and ifft

fftw3 (fastest Fourier transform in the west)

not restricted to powers of 2, but N should have many prime factors

⚠ careful with normalization; e.g. matlab uses:

$$X_k = \sum_{j=0}^{N-1} x_j\, e^{-2i\pi kj/N} = N(x, E_k)_N$$

$$x_j = \frac{1}{N} \sum_{k=0}^{N-1} X_k\, e^{2i\pi kj/N}$$ (inverse fast Fourier transform)

$$= (X, \overline{E_j})_N$$

Convolution using FFT

Convolution has many uses in e.g. signal processing:

$$h = g * f = f * g$$

$$h(x) = \int_{-\infty}^{\infty} g(y)\, f(x-y)\, dy = \int_{-\infty}^{\infty} f(y)\, g(x-y)\, dy$$

h = output signal
g = input signal
f = filter "impulse response"

In the discrete case we can think of f, h, g as sets of N samples:

$$h_n = \sum_{m=0}^{N-1} g_m f_{n-m} \equiv \text{discrete convolution}$$

Or in matrix form:

$$\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \\ h_{N-1} \end{bmatrix} = \begin{bmatrix} f_0 & f_{-1} & f_{-2} & \cdots & f_{1-N} \\ f_1 & f_0 & f_{-1} & \cdots & f_{2-N} \\ f_2 & f_1 & f_0 & \cdots & f_{3-N} \\ \vdots & & & & \\ f_{N-1} & \cdots & & & f_0 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{N-1} \end{bmatrix} \qquad (\star)$$

$$\underline{h} = T_N \underline{g} \quad \text{where } T_N = \text{Toeplitz matrix}$$

naive implementation would cost $O(N^2)$

Convolution can be done efficiently using FFT.

Here is the key result we need:

$$(\text{ifft}(F.*G))_n = \frac{1}{N}\sum_{k=0}^{N-1} F_k G_k e^{\frac{2i\pi kn}{N}}$$

$$= \frac{1}{N}\sum_{k=0}^{N-1}\left(\sum_{j=0}^{N-1} f_j e^{-2i\pi kj/N}\right)\left(\sum_{j'=0}^{N-1} g_{j'} e^{-2i\pi kj'/N}\right) e^{2i\pi kn/N}$$

$$= \sum_{j=0}^{N-1} f_j \sum_{j'=0}^{N-1} g_{j'} \underbrace{\frac{1}{N}\sum_{k=0}^{N-1} e^{\frac{2i\pi k}{N}(n-j-j')}}_{(E_{n-j}, E_{j'})_N}$$

Now note we have:

$$(E_{n-j}, E_{j'})_N = \begin{cases} 1 & \text{if } n-j-j' \text{ is divible by } N \\ 0 & \text{otherwise} \end{cases}$$

ie. $n-j-j' = 0 \bmod N$

$j' = n-j \bmod N$

Thus:

$$(\text{ifft}(F.*G))_n = \sum_{j=0}^{N-1} f_j\, g_{(n-j) \bmod N}$$

So we can evaluate a slightly different convolution efficiently using fft:

1. `F=fft(f); G=fft(g);` $\quad O(N\log N)$
2. `H= F.*G` $\quad O(N)$
3. `h=ifft(H)` $\quad O(N\log N)$

total $O(N\log N)$ operations

In matrix vector product form this becomes:

$$\begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{N-1} \end{bmatrix} = \underbrace{\begin{bmatrix} f_0 & f_{-1} & f_{-2} & \cdots & f_{1-N} \\ f_1 & f_0 & f_{-1} & \cdots & f_{2-N} \\ \vdots & & & & \\ f_{1-N} & \cdots & & & f_0 \end{bmatrix}}_{C_N} \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{N-1} \end{bmatrix}$$

C_N = circulant matrix.

So the FFT gives us a tool to compute matrix vector products of circulant matrices in $O(N \log N)$ operations!

How do we use it to compute the matvec product (☆)?

The trick is to formulate (☆) as a M-V prod with larger matrix that is circulant:

$$\text{Let } B_N = \begin{bmatrix} 0 & f_{N-1} & f_{N-2} & \cdots & f_2 & f_1 \\ f_{1-N} & 0 & f_{N-1} & \cdots & & f_2 \\ f_{2-N} & f_{1-N} & 0 & & & f_3 \\ \vdots & & & \ddots & & \vdots \\ f_{-1} & f_{-2} & \cdots & & f_{1-n} & 0 \end{bmatrix} \in \mathbb{R}^{N \times N}$$

It is easy to check that $C = \begin{bmatrix} T_N & B_N \\ B_N & T_N \end{bmatrix}$ is a circulant matrix

So we can efficiently evaluate:

$$\begin{bmatrix} T_N g \\ B_N g \end{bmatrix} = \underbrace{\begin{bmatrix} T_N & B_N \\ B_N & T_N \end{bmatrix}}_{C} \begin{bmatrix} g \\ 0 \end{bmatrix}$$

throw away

← zero padding

in $O(2N \log(2N))$
$= O(N \log N)$ operations.