

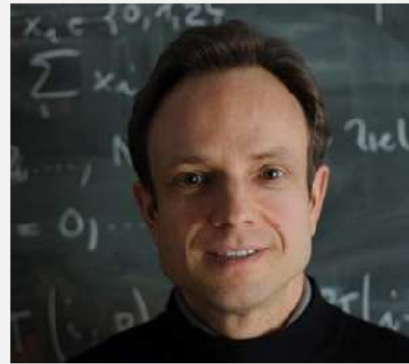
Course Video

Section 4.3: Fast Fourier Transform (FFT)

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 18, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Def. 4.2.1.18

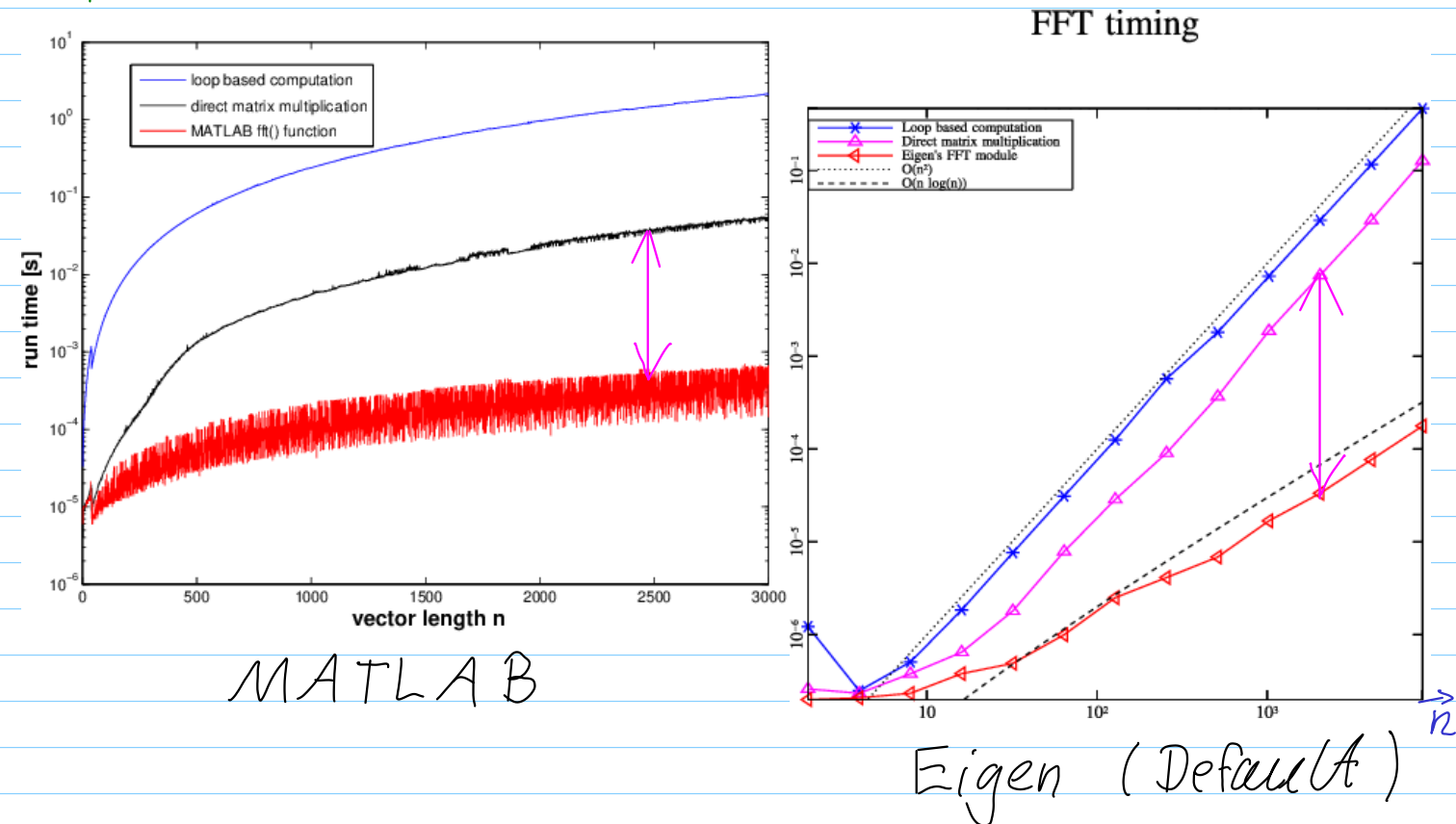
Duration: 30 minutes

IV. Filtering Algorithms

4.3. Fast Fourier Transform (FFT)

DFT : $\underline{c} = F_n \underline{y}$: cost = $O(n^2)$ for $n \rightarrow \infty$

Exp 4.3.0.2 : Runtimes of DFT implementations



2

The secret of MATLAB's/EIGEN's/PYTHON's `fft()`:

the **Fast Fourier Transform (FFT)** algorithm

(discovered by C.F. Gauss in 1805, rediscovered by Cooley & Tuckey in 1965, one of the "top ten algorithms of the century").

§ 4.3.0.3 : FFT : Derivation & Complexity

Consider $n = 2m$:

even/odd splitting

$$\begin{aligned}
 c_k &= \sum_{j=0}^{n-1} y_j e^{-\frac{2\pi i}{n} jk} = \sum_{j=0}^{m-1} y_{2j} e^{-\frac{2\pi i}{n} 2jk} + \sum_{j=0}^{m-1} y_{2j+1} e^{-\frac{2\pi i}{n} (2j+1)k} \\
 &= \sum_{j=0}^{m-1} y_{2j} \underbrace{e^{-\frac{2\pi i}{m} jk}}_{=\omega_m^{jk}} + e^{-\frac{2\pi i}{n} k} \sum_{j=0}^{m-1} y_{2j+1} \underbrace{e^{-\frac{2\pi i}{m} jk}}_{=\omega_m^{jk}} \quad k \in \mathbb{Z}. \quad (4.3.0.4)
 \end{aligned}$$

$\underbrace{\quad}_{=\hat{c}_k^{\text{even}}}$ $\underbrace{\quad}_{=\hat{c}_k^{\text{odd}}}$

$$\begin{aligned}
 \hat{c}^{\text{even}} &= \text{DFT}_m [y_{2j}]_{j=0}^{m-1} \\
 \hat{c}^{\text{odd}} &= \text{DFT}_m [y_{2j+1}]_{j=0}^{m-1}
 \end{aligned}$$

Idea for $n = 2^L$: "Divide & conquer" recursion

C++ code 4.3.0.5: Recursive FFT → GITLAB

```

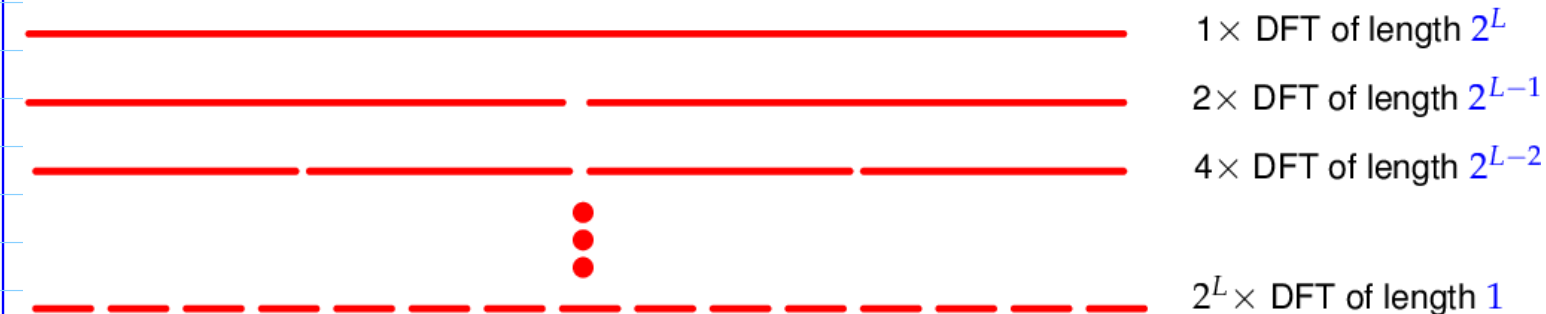
2 // Recursive DFT for vectors of length  $n = 2^L$ 
3 VectorXcd fftrec(const VectorXcd& y) {
4     using idx_t = VectorXcd::Index;
5     using comp = std::complex<double>;
6     // Nothing to do for DFT of length 1
7     const idx_t n = y.size();
8     if (n == 1) return y;
9     if (n % 2 != 0) throw std::runtime_error("size(y) must be even!");
10    const Eigen::Map<const
        Eigen::Matrix<comp, Eigen::Dynamic, Eigen::Dynamic, Eigen::RowMajor>>
11        Y(y.data(), n/2, 2); // for selecting even and odd components
12    const VectorXcd c1 = fftrec(Y.col(0)), c2 = fftrec(Y.col(1));
13    const comp omega = std::exp(-2*M_PI/n*comp(0,1)); // root of unity  $\omega_n$ 
14    comp s(1.0, 0.0);
15    VectorXcd c(n);
16    // Scaling of DFT of odd components plus periodic copying
17    for (long k = 0; k < n; ++k) {
18        c(k) = c1(k%(n/2)) + c2(k%(n/2))*s;
19        s *= omega;
20    }
21    return c;
22 }

```

$c1 \leftrightarrow \hat{c}^{\text{even}}$
 $c2 \leftrightarrow \hat{c}^{\text{odd}}$
 $\text{cost} = O(n)$

Computational cost ?

Visualization of computational cost of `fftrec()` from Code 4.3.0.5:



- ③ $\triangleright O(2^L)$ ops. on every level
 $\triangleright \text{cost} = O(2^L \cdot L) = O(n \cdot \log n)$ for $n \rightarrow \infty$
 $"\ll O(n^2)"$

Beyond $n = 2^L$:

Rem. 4.3.0.8 : FFT for $n = pq$

Re-indexing: generalizes even/odd splitting

$$c_k = \sum_{j=0}^{n-1} y_j \omega_n^{jk} \stackrel{[j:=lp+m]}{=} \sum_{m=0}^{p-1} \sum_{l=0}^{q-1} y_{lp+m} e^{-\frac{2\pi i}{pq}(lp+m)k} = \sum_{m=0}^{p-1} \omega_n^{mk} \underbrace{\sum_{l=0}^{q-1} y_{lp+m} \omega_q^{l(k \bmod q)}}_{\leftrightarrow \text{DFT}_q}. \quad (4.3.0.9)$$

I : $\mathbb{Z}_m := \text{DFT}_q [y_{lp+m}]_{l=0}^{q-1} : p\text{-times}$

$$(z_m)_k = z_{m,k} := \sum_{l=0}^{q-1} y_{lp+m} \omega_q^{lk}, \quad 0 \leq m < p, \quad 0 \leq k < q.$$

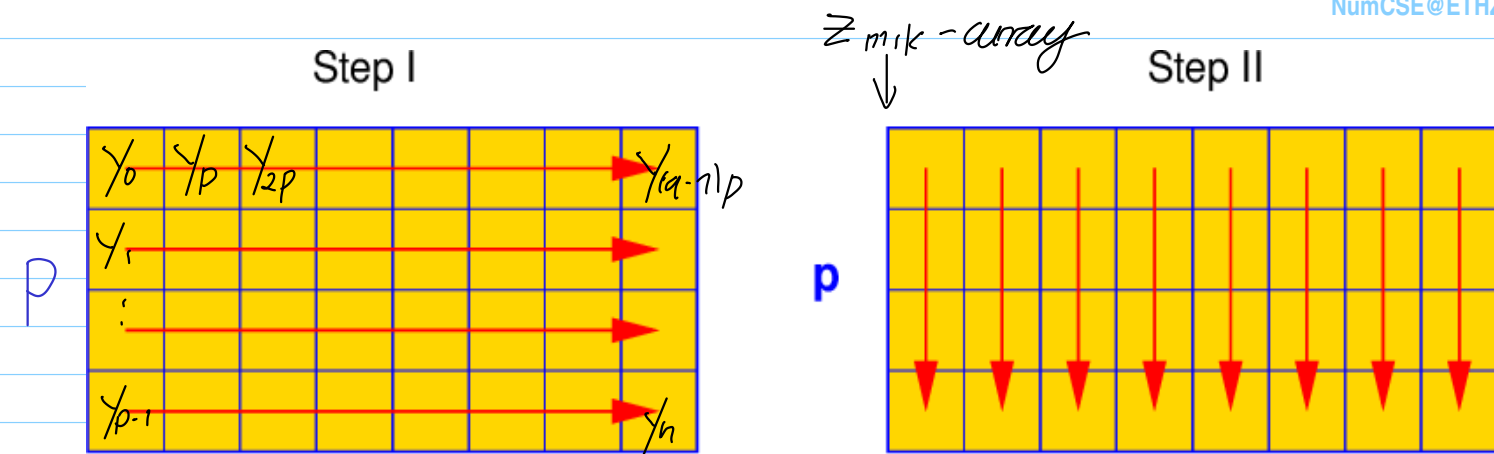
II

$$c_{rq+s} = \sum_{m=0}^{p-1} e^{-\frac{2\pi i}{pq}(rq+s)m} z_{m,s} = \sum_{m=0}^{p-1} (\omega_n^{ms} z_{m,s}) \omega_p^{mr}$$

$\downarrow$ n multiplications

$k = rq + s : 0 \leq r < p, \quad 0 \leq s < q$

$\leftrightarrow \text{DFT}_p$



Related to reduction of 2D DFT of $C^{m,n}$ to $m \times \text{DFT}_n + n \times \text{DFT}_m$!

Special treatment of prime n :

The execution time for fft depends on the length of the transform. It is fastest for powers of two. It is almost as fast for lengths that have only small prime factors. It is typically several times slower for lengths that are prime or which have large prime factors $\rightarrow$ Ex. 4.3.0.12.



Never implement DFT/FFT by yourself!
 Under all circumstances use high-quality numerical libraries!

④

Review questions 4.3.0.13

5

⑦