

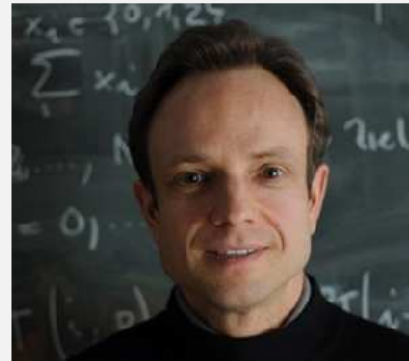
Course Video

Section 4.2.2: Discrete Convolution via DFT

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 4, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Prerequisites.

- Orthogonal diagonalization (linear algebra)

Dependencies. Section 4.2.1

Duration: 14 minutes

IV. Filtering Algorithms

4.2. Discrete Fourier Transform (DFT)

4.2.2. Discrete Convolution via DFT

Recall : discrete periodic convolution :

$$(y_k) = (p_k) \star_n (x_k) \Leftrightarrow y_k = \sum_{j=0}^{n-1} p_{k-j} x_j, \quad k=0, \dots, n-1$$

C++11 code 4.2.2.1: Discrete periodic convolution: straightforward implementation
→ [GITLAB](#)

```

2 Eigen::VectorXcd pconv(const Eigen::VectorXcd &u, const Eigen::VectorXcd &x) {
3     const int n = x.size();
4     Eigen::VectorXcd z = VectorXcd::Zero(n);
5     // "naive" two-loop implementation of discrete periodic convolution
6     for (int k = 0; k < n; ++k) {
7         for (int j = 0, l = k; j <= k; ++j, —l) z[k] += u[l] * x[j];
8         for (int j = k + 1, l = n - 1; j < n; ++j, —l) z[k] += u[l] * x[j];
9     }
10    return z;
11 }

```

$$\text{Cost} = O(n^2)$$

Assume : library DFT "ultra fast"

2

$$\underline{z}_x = \sum_{j=0}^{n-1} u_{k-j} x_j \longleftrightarrow \underline{z} = \underline{C} \underline{x}, \quad \underline{C} = [c_{ij}]_{i,j=0}^{n-1}$$



Idea:

(4.2.1.17) >

$$\underline{z} = \underbrace{\mathbf{F}_n^{-1}}_{\text{DFT}} \text{diag}(\underbrace{\mathbf{F}_n \underline{u}}_{\text{DFT}}) \underbrace{\mathbf{F}_n \underline{x}}_{\text{DFT}}$$

componentwise •

Lemma 4.2.1.16. Diagonalization of circulant matrices (→ Def. 4.1.4.12)

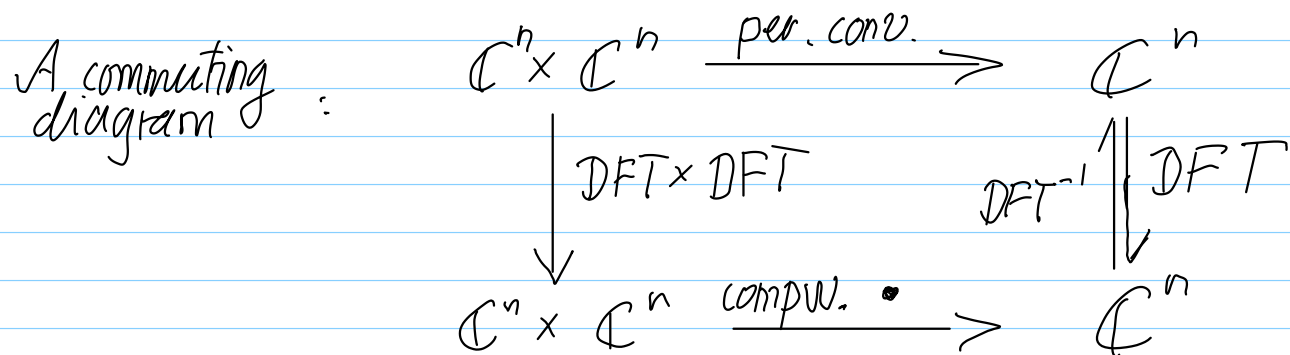
For any circulant matrix $\mathbf{C} \in \mathbb{K}^{n,n}$, $c_{ij} = u_{i-j}$, $(u_k)_{k \in \mathbb{Z}}$ n -periodic sequence, holds true

$$\mathbf{C} \bar{\mathbf{F}}_n = \bar{\mathbf{F}}_n \text{diag}(d_1, \dots, d_n) \quad , \quad [d_0, \dots, d_{n-1}]^\top = \mathbf{F}_n [u_0, \dots, u_{n-1}]^\top .$$

Theorem 4.2.2.2. Convolution theorem

The discrete periodic convolution $*_n$ between n -dimensional vectors $\underline{u}$ and $\underline{x}$ is equal to the inverse DFT of the component-wise product between the DFTs of $\underline{u}$ and $\underline{x}$; i.e.:

$$(\underline{u}) *_n (\underline{x}) := \left[\sum_{j=0}^{n-1} u_{k-j} x_j \right]_{k=0}^{n-1} = \mathbf{F}_n^{-1} [(\mathbf{F}_n \underline{u})_j (\mathbf{F}_n \underline{x})_j]_{j=1}^n . \quad (4.2.2.3)$$

**C++11 code 4.2.2.4: Discrete periodic convolution: DFT implementation** → GITLAB

```

2 Eigen::VectorXcd pconvfft(const Eigen::VectorXcd &u,
3                           const Eigen::VectorXcd &x) {
4     Eigen::FFT<double> fft;
5     return fft.inv(((fft.fwd(u)).cwiseProduct(fft.fwd(x))).eval());
6 }

```

Discrete conv. by periodic convolution & zero padding

$$\underline{a}, \underline{b} \in \mathbb{C}^n : \underline{a} *_n \underline{b} = \begin{bmatrix} \underline{a} \\ 0 \end{bmatrix} *_{2n-1} \begin{bmatrix} \underline{b} \\ 0 \end{bmatrix} \in \mathbb{R}^{2n-1}$$

C++11 code 4.2.2.5: Implementation of discrete convolution (→ Def. 4.1.3.3) based on periodic discrete convolution → GITLAB

```

2 Eigen::VectorXcd fastconv(const Eigen::VectorXcd &h,
3                           const Eigen::VectorXcd &x) {
4     assert(x.size() == h.size());
5     const Eigen::Index n = h.size();
6     // Zero padding, cf. (4.1.4.16), and periodic discrete convolution
7     // of length 2n-1, Code 4.2.2.4
8     return pconvfft(
9         (Eigen::VectorXcd(2 * n - 1) << h, Eigen::VectorXcd::Zero(n - 1)) <<
10         .finished(),
11         (Eigen::VectorXcd(2 * n - 1) << x, Eigen::VectorXcd::Zero(n - 1)) <<
12         .finished());
13 }

```

③

→ Review questions 4.2.2.6

(Try to answer without consulting tablet or lecture notes)

4

5

7