

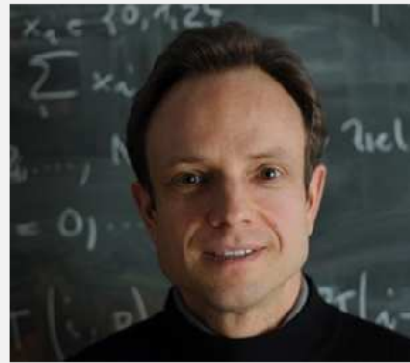
Course Video

Section 4.2.5: Two-Dimensional DFT

Prof. R. Hiptmair, SAM, ETH Zurich

Date: October 17, 2020

(C) Seminar für Angewandte Mathematik, ETH Zürich



Dependencies. Section 4.2.2

Duration: minutes

IV. Filtering Algorithms

4.2. Discrete Fourier Transform (DFT)

4.2.5. Two-Dimensional DFT

2D signals $\longleftrightarrow$ images $\longleftrightarrow$ matrices

§ 4.2.5.1 Matrix Fourier modes

Recall: DFT $\longleftrightarrow$ Inf. into Fourier of modesbasis vectors: $(F_n)_{:,k} = [\omega_n^{k\#}]_{\#=0}^{n-1}$

Idea: 2D Fourier basis matrices by tensor product

$$\{ (F_m)_{:,j}, (F_n)_{:,l}^T \}_{\substack{j=0,\dots,m-1 \\ l=0,\dots,n-1}} \subset \mathbb{C}^{m,n}$$

2D DFT of $Y \in \mathbb{C}^{m,n}$: $\text{DFT}_{m,n}(Y) = \underbrace{\sum_{j=0}^{m-1} \sum_{l=0}^{n-1} (Y)_{j,l} (F_m)_{:,j} (F_n)_{:,l}^T}_{=: C}$

[C++-indexing]

$$C = \sum_{j=0}^{m-1} \sum_{\ell=0}^{n-1} (Y)_{j,\ell} (F_m)_{:,j} (F_n)_{:, \ell}^T$$

$$(C)_{k_1, k_2} = \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} y_{j_1, j_2} \omega_m^{j_1 k_1} \omega_n^{j_2 k_2} = \sum_{j_1=0}^{m-1} \omega_m^{j_1 k_1} \left(\sum_{j_2=0}^{n-1} \omega_n^{j_2 k_2} y_{j_1, j_2} \right), \quad 0 \leq k_1 < m, 0 \leq k_2 < n.$$

m 1D DFTs of rows of Y
 n 1D DFT of cols of $DFT_{m,n}(Y)$

$$(C)_{k_1, k_2} = \sum_{j_1=0}^{m-1} (F_n(Y)_{j_1,:})_{k_2} \omega_m^{j_1 k_1} \Rightarrow C = F_m (F_n Y^T)^T = F_m Y F_n. \quad (4.2.5.4)$$

Lemma 4.2.1.14. Properties of Fourier matrix

The scaled Fourier-matrix $\frac{1}{\sqrt{n}} F_n$ is unitary ($\rightarrow$ Def. 6.2.1.2): $F_n^{-1} = \frac{1}{n} F_n^H = \frac{1}{n} \bar{F}_n$.

$$C = \sum_{j_1=0}^{m-1} \sum_{j_2=0}^{n-1} y_{j_1, j_2} (F_m)_{:,j_1} (F_n)_{:,j_2}^T \Rightarrow Y = F_m^{-1} C F_n^{-1} = \frac{1}{mn} \bar{F}_m C \bar{F}_n. \quad (4.2.5.5)$$

C++ code 4.2.5.6: Two-dimensional discrete Fourier transform [→ GITLAB](#)

```

2 template <typename Scalar>
3 void fft2(Eigen::MatrixXcd &C, const Eigen::MatrixBase<Scalar> &Y) {
4     using idx_t = Eigen::MatrixXcd::Index;
5     const idx_t m = Y.rows(), n=Y.cols();
6     C.resize(m,n);
7     Eigen::MatrixXcd tmp(m,n);
8
9     Eigen::FFT<double> fft; // Helper class for DFT
10    // Transform rows of matrix Y
11    for (idx_t k=0; k<m; k++) {
12        Eigen::VectorXcd tv(Y.row(k));
13        tmp.row(k) = fft.fwd(tv).transpose();
14    }
15
16    // Transform columns of temporary matrix
17    for (idx_t k=0; k<n; k++) {
18        Eigen::VectorXcd tv(tmp.col(k));
19        C.col(k) = fft.fwd(tv);
20    }
21 }

```

C++ code 4.2.5.7: Inverse two-dimensional discrete Fourier transform [→ GITLAB](#)

```

2 template <typename Scalar>
3 void ifft2(Eigen::MatrixXcd &C, const Eigen::MatrixBase<Scalar> &Y) {
4     using idx_t = Eigen::MatrixXcd::Index;
5     const idx_t m = Y.rows(), n=Y.cols();
6     fft2(C, Y.conjugate()); C = C.conjugate() / (m*n);
7 }

```

(4)

§ 4.2.5.9. Periodic convolution of matrices

1D:

$$\mathbf{y} = \mathbf{p} *_n \mathbf{x} := \left[\sum_{j=0}^{n-1} x_j p_{(k-j) \bmod n} \right]_{k=0}^{n-1}, \quad \mathbf{p}, \mathbf{x} \in \mathbb{C}^n, \quad (4.2.5.10)$$

2D:

$$\mathbf{X} *_m, n \mathbf{Y} := \mathbf{B}(\mathbf{X}, \mathbf{Y}) = \left[\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} (\mathbf{Y})_{(k-i) \bmod m, (\ell-j) \bmod n} \right]_{\substack{k=0, \dots, m-1 \\ \ell=0, \dots, n-1}}, \quad \mathbf{X}, \mathbf{Y} \in \mathbb{C}^{m,n}.$$

→ Diagonal in basis $\{(\mathbf{F}_m)_{:,j} (\mathbf{F}_n)_{:, \ell}^T\}_{\substack{j=0, \dots, m-1 \\ \ell=0, \dots, n-1}}$

$$\begin{aligned} (\mathbf{B}(\mathbf{X}, \mathbf{Y}))_{k,\ell} &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} (\mathbf{Y})_{(k-i) \bmod m, (\ell-j) \bmod n} \\ &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} \omega_m^{r(k-i)} \omega_n^{s(\ell-j)} \\ &= \left(\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} \overline{\omega_m^{ri}} \omega_n^{sj} \right) \cdot \omega_m^{rk} \omega_n^{s\ell}. \end{aligned}$$

$$\mathbf{B}(\mathbf{X}, \underbrace{(\mathbf{F}_m)_{:,r} (\mathbf{F}_n)_{s,:}}_{\text{"eigenvector"}}) = \underbrace{\left(\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} \overline{\omega_m^{ri}} \omega_n^{sj} \right)}_{\text{"eigenvalue", see Eq. (4.2.5.3)}} (\mathbf{F}_m)_{:,r} (\mathbf{F}_n)_{s,:}. \quad (4.2.5.13)$$

≡ an entry of $\text{DFT}_{m,n}(\mathbf{X})!$

1D result:

Theorem 4.2.2.2. Convolution theorem

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

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

Theorem 4.2.5.14. 2D convolution theorem

For any $\mathbf{X}, \mathbf{Y} \in \mathbb{C}^{m,n}$, we have

$$\mathbf{X} *_m, n \mathbf{Y} = \text{DFT}_{m,n}^{-1} (\text{DFT}_{m,n}(\mathbf{X}) \odot \text{DFT}_{m,n}(\mathbf{Y})),$$

where $\odot$ stands for the entrywise multiplication of matrices of equal size.

C++ code 4.2.5.15: DFT-based 2D discrete periodic convolution → GITLAB

```
// DFT based implementation of 2D periodic convolution
template <typename Scalar1, typename Scalar2, class EigenMatrix>
void pmconv(const Eigen::MatrixBase<Scalar1> &X,
            const Eigen::MatrixBase<Scalar2> &Y, EigenMatrix &Z) {
    using Comp = std::complex<double>;
    using idx_t = typename EigenMatrix::Index;
    using val_t = typename EigenMatrix::Scalar;
    const idx_t n = X.cols(), m = X.rows();
    if ((m != Y.rows()) || (n != Y.cols()))
        throw std::runtime_error("pmconv: size mismatch");
    Z.resize(m, n);
    Eigen::MatrixXcd Xh(m, n), Yh(m, n);
    // Step 1: 2D DFT of Y
    fft2(Yh, (Y.template cast<Comp>()));
    // Step 2: 2D DFT of X
    fft2(Xh, (X.template cast<Comp>()));
    // Steps 3, 4: inverse DFT of component-wise product
    ifft2(Z, Xh.cwiseProduct(Yh));
}
```

5

Ex. 4.2.5.16 (Deblurring by DFT)

image $\longleftrightarrow P \in \mathbb{R}^{m,n}$ (pixel matrix)

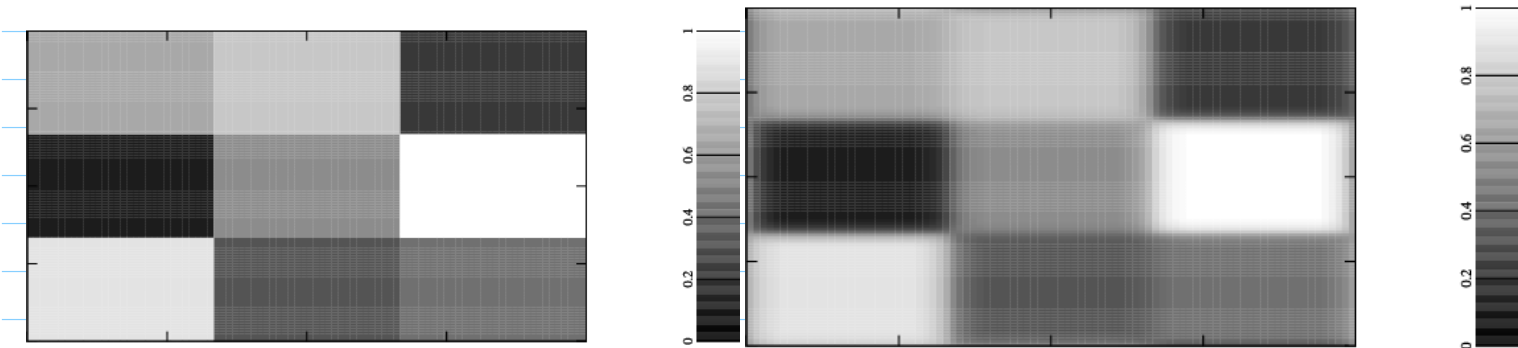
Blurring operator ($L \ll m, n$)

$$(C := \mathcal{B}(P))_{l,j} := \sum_{k,q=-L}^L \underset{\substack{\uparrow \\ \text{PSF}}}{S_{k,q}}(P)_{(l+k) \bmod m, (j+q) \bmod n}$$

Here: $S_{k,q} = \frac{1}{1+k^2+q^2}$ & normalized, $L=5$

Original

Blurred image



$\mathcal{B} \triangleq$ a 2D discrete periodic convolution

$$\begin{aligned} (C)_{l,j} &= c_{l,j} = \sum_{k=-L}^L \sum_{q=-L}^L S_{k,q}(\mathbf{P})_{(l+k) \bmod m, (j+q) \bmod n} \\ &= \sum_{k=-L}^L \sum_{q=-L}^L S_{-k,-q}(\mathbf{P})_{(l-k) \bmod m, (j-q) \bmod n} \\ &= \sum_{k=0}^L \sum_{q=0}^L S_{-k,-q}(\mathbf{P})_{(l-k) \bmod m, (j-q) \bmod n} + \\ &\quad \sum_{k=0}^L \sum_{q=-L}^{-1} S_{-k,-q}(\mathbf{P})_{(l-k) \bmod m, (j-(q+n)) \bmod n} + \\ &\quad \sum_{k=-L}^{-1} \sum_{q=0}^L S_{-k,-q}(\mathbf{P})_{(l-(k+m)) \bmod m, (j-q) \bmod n} + \end{aligned}$$

Split in 4 sums

$$\mathbf{X} *_{m,n} \mathbf{Y} := \mathcal{B}(\mathbf{X}, \mathbf{Y}) = \left[\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\mathbf{X})_{i,j} (\mathbf{Y})_{(k-i) \bmod m, (\ell-j) \bmod n} \right]_{\substack{k=0, \dots, m-1 \\ \ell=0, \dots, n-1}}, \quad \mathbf{X}, \mathbf{Y} \in \mathbb{C}^{m,n}$$

$$\begin{aligned} &= \sum_{k=0}^L \sum_{q=0}^L S_{-k,-q}(\mathbf{P})_{(l-k) \bmod m, (j-q) \bmod n} + \\ &\quad \sum_{k=0}^L \sum_{q=n-L}^{n-1} S_{-k,-q+n}(\mathbf{P})_{(l-k) \bmod m, (j-q) \bmod n} + \\ &\quad \sum_{k=m-L}^{m-1} \sum_{q=0}^L S_{-k+m,-q}(\mathbf{P})_{(l-k) \bmod m, (j-q) \bmod n} + \\ &\quad \sum_{k=m-L}^{m-1} \sum_{q=n-L}^{n-1} S_{-k+m,-q+n}(\mathbf{P})_{(l-k) \bmod m, (j-q) \bmod n} \end{aligned}$$

⑥

PSF

↓

$\mathbf{C} = \mathbf{S} *_{m,n} \mathbf{P}$ with $(\mathbf{S})_{k,q} =$

↑

blurred image

$$(\mathbf{S})_{k,q} = \begin{cases} s_{-k,-q} & , \text{ for } 0 \leq k, q \leq L, \\ s_{-k+m,-q} & , \text{ for } m-L \leq k < m, \quad 0 \leq q \leq L, \\ s_{-k,q+n} & , \text{ for } 0 \leq k \leq L, \quad n-L \leq q < n, \\ s_{-k+m,-q+n} & , \text{ for } m-L \leq k < m, \quad n-L \leq q < n, \\ 0 & \text{ else.} \end{cases} \quad (4.2.5.20)$$

Theorem 4.2.5.14. 2D convolution theorem

For any $\mathbf{X}, \mathbf{Y} \in \mathbb{C}^{m,n}$, we have

$$\mathbf{X} *_{m,n} \mathbf{Y} = \text{DFT}_{m,n}^{-1}(\text{DFT}_{m,n}(\mathbf{X}) \odot \text{DFT}_{m,n}(\mathbf{Y})),$$

where $\odot$ stands for the entrywise multiplication of matrices of equal size.

$$\triangleright \quad \mathbf{P} = \text{DFT}_{m,n}^{-1} \left(\text{DFT}_{m,n}(\mathbf{C}) \odot \text{DFT}_{m,n}(\mathbf{S}) \right)$$

↑
entrywise /



what could be a problem

check for $(\text{DFT}_{m,n}(\mathbf{S}))_{e,k} \neq 0$
[in a numerically clean way]

C++11-code 4.2.5.22: DFT based deblurring → GITLAB

```
2 MatrixXd deblur(const MatrixXd &C, const MatrixXd &S, const double tol = 1e-3) {
3     const long m = C.rows(), n = C.cols(), M = S.rows(), N = S.cols();
4     const long L = (M - 1) / 2;
5     if (M != N) {
6         throw std::runtime_error("Error: S not quadratic!");
7     }
8     MatrixXd Spad = MatrixXd::Zero(m, n);
9     // Zero padding, see (4.2.5.20)
10    Spad.block(0, 0, L + 1, L + 1) = S.block(L, L, L + 1, L + 1);
11    Spad.block(m - L, n - L, L, L) = S.block(0, 0, L, L);
12    Spad.block(0, n - L, L + 1, L) = S.block(L, 0, L + 1, L);
13    Spad.block(m - L, 0, L, L + 1) = S.block(0, L, L, L + 1);
14    // Inverse of blurring operator (fft2 expects a complex matrix)
15    MatrixXcd SF = fft2(Spad.cast<complex>());
16    // Test for invertibility
17    if (SF.cwiseAbs().minCoeff() < tol * SF.cwiseAbs().maxCoeff()) {
18        std::cerr << "Error: Deblurring impossible!\n";
19    }
20    // DFT based deblurring
21    return fft2(fft2(C.cast<complex>()).cwiseQuotient(SF)).real();
22 }
```

} build $S_{e,k}^{m,n}$

Review questions 4.2.5.23

(Try to answer them without looking up formulas in the lecture/tablet notes)

7