

ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022
(C) Seminar für Angewandte Mathematik, ETH Zürich

[Link](#) to the current version of this homework collection

0.1 General Information

0.1.1 Weekly Homework Assignments

All problems will be published in this single “.pdf” file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. “Solving” the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

Problem 4-7: Modified Cosine Transform

Many so-called trigonometric transformations can be reduced to the discrete Fourier transform (DFT). One such example is studied in this problem.

Assumes familiarity with the discrete Fourier transform, C++, and the complex exponential

For a given vector $\mathbf{a} = [a_0, \dots, a_{n-1}]^\top \in \mathbb{R}^n$, $n \in \mathbb{N}$, we want to compute another vector $\mathbf{f} := [f_0, \dots, f_{n-1}]^\top \in \mathbb{R}^n$ according to the formula

$$f_k := \sum_{j=0}^{n-1} a_j \cos\left(\pi \frac{jk}{n}\right), \quad k = 0, \dots, n-1. \quad (4.7.1)$$

Using the identity $\cos x = \frac{1}{2}(\exp(-ix) + \exp(ix))$, we find the alternative formula

$$f_k = \sum_{\ell=0}^{n-1} \frac{1}{2} a_\ell \exp(-2\pi i \frac{k\ell}{2n}) + \exp(\pi i \frac{k(n-1)}{n}) \cdot \sum_{\ell=0}^{n-1} \frac{1}{2} a_{n-1-\ell} \exp(-2\pi i \frac{k\ell}{2n}) \quad (4.7.2)$$

for $k = 0, \dots, n-1$.

Recall the definition of the discrete Fourier transform (DFT) as multiplication of a vector with the Fourier matrix $\mathbf{F}_n$:

Definition [Lecture → Def. 4.2.1.18].

The linear map $\text{DFT}_n : \mathbb{C}^n \mapsto \mathbb{C}^n$, $\text{DFT}_n(\mathbf{y}) := \mathbf{F}_n \mathbf{y}$, $\mathbf{y} \in \mathbb{C}^n$, is called **discrete Fourier transform** (DFT), i.e. for $[c_0, \dots, c_{n-1}] := \text{DFT}_n(\mathbf{y})$

$$c_k := \sum_{j=0}^{n-1} y_j \exp(-2\pi i \frac{kj}{n}), \quad k = 0, \dots, n-1. \quad [\text{Lecture} \rightarrow \text{Eq. (4.2.1.19)}]$$

(4-7.a)  (10 min.)

Taking the cue from (4.7.2) we can write (“C++ indexing” throughout)

$$\mathbf{f} = (\text{DFT}_{2n}(\mathbf{x}))_{0:n-1} + \mathbf{D}(\text{DFT}_{2n}(\mathbf{y}))_{0:n-1}, \quad (4.7.3)$$

with suitable vectors $\mathbf{x}, \mathbf{y} \in \mathbb{C}^{2n}$ and a **diagonal** matrix $\mathbf{D} \in \mathbb{C}^{n,n}$. Characterize the vectors $\mathbf{x}, \mathbf{y}$ (in terms of $\mathbf{a}$) and the matrix $\mathbf{D}$.

$$(\mathbf{x})_\ell = \begin{cases} \boxed{} & \text{for } \ell \in \{ \boxed{} \}, \\ \boxed{} & \text{for } \ell \in \{ \boxed{} \}, \end{cases}, \quad (\mathbf{y})_\ell = \begin{cases} \boxed{} & \text{for } \ell \in \{ \boxed{} \}, \\ \boxed{} & \text{for } \ell \in \{ \boxed{} \}, \end{cases},$$

$$(\mathbf{D})_{k,k} = \boxed{}, \quad k = 0, \dots, n-1.$$

SOLUTION of (4-7.a):

Remark. The formula (4.7.2) is derived as follows

$$\begin{aligned} f_k &= \sum_{j=0}^{n-1} a_j \cos\left(\pi \frac{kj}{n}\right) = \frac{1}{2} \sum_{j=0}^{n-1} a_j \left(e^{-\pi i \frac{kj}{n}} + e^{\pi i \frac{kj}{n}} \right) \\ &= \sum_{j=0}^{n-1} \frac{1}{2} a_j e^{-2\pi i \frac{kj}{2n}} + \sum_{\ell=0}^{n-1} \frac{1}{2} a_{n-1-\ell} e^{2\pi i \frac{k(n-1-\ell)}{2n}} \\ &= \sum_{j=0}^{n-1} \frac{1}{2} a_j e^{-2\pi i \frac{kj}{2n}} + e^{\pi i \frac{k(n-1)}{n}} \sum_{\ell=0}^{n-1} \frac{1}{2} a_{n-1-\ell} e^{-2\pi i \frac{k\ell}{2n}} \end{aligned}$$

Matching this with (4.7.3) we find

$$(x)_j = \begin{cases} \frac{1}{2}a_j & \text{for } j \in \{0, \dots, n-1\}, \\ 0 & \text{for } j \in \{n, \dots, 2n-1\}, \end{cases}, \quad (y)_j = \begin{cases} \frac{1}{2}a_{n-1-j} & \text{for } j \in \{0, \dots, n-1\}, \\ 0 & \text{for } j \in \{n, \dots, 2n-1\}, \end{cases},$$

$$(D)_{k,\ell} = \exp\left(\pi i k \frac{n-1}{n}\right), \quad \ell = 0, \dots, n-1. \quad (4.7.4)$$

▲

(4-7.b) 🕒 (10 min.) The C++ function `modcostrf()` realizes the mapping $\mathbf{a} \rightarrow \mathbf{f}$ according to (4.7.1). Supplement the missing parts of the following listing by writing valid C++ code in the boxes.

```
Eigen::VectorXcd modcostrf(const Eigen::VectorXcd &a) {
    using Comp = std::complex<double>;
    unsigned int n = a.size();
    Eigen::VectorXcd f(n);
    Eigen::FFT<double> fft;

    Eigen::VectorXcd tmp( );

    tmp << , Eigen::VectorXcd::Zero(n);

    Eigen::VectorXcd v1 = fft. ;

    tmp << 0.5 * a.reverse(), ;

    Eigen::VectorXcd v2 = fft. ;

    Comp fac = std::exp(((n - 1) * M_PI * Comp(0, 1)) / (double)n);
    Comp d(1.0, 0.0);
    for (int k = 0; k < n; ++k) {
        f[k] = (v1[ ] + * v2[ ]).real();
        *= ;
    }
    return f;
}
```

The constant `M_PI` is the number π .

HINT 1 for (4-7.b): The EIGEN helper class **Eigen::FFT<double>** provides the member functions `fwd()` for DFT and `inv()` for inverse DFT. ┘

HINT 2 for (4-7.b): The member function `reverse()` of **Eigen::VectorXd** reverses the order of the vector components. ┘

SOLUTION of (4-7.b):

The following code is a straightforward implementation of (??) making use of the fact that the DFT is available through the method `fwd()` of an **Eigen::DFT** object. The final loop also performs the multiplication with **D**, which is a simple component-wise scaling.

Note that (4.7.4) means that

$$(\mathbf{D})_{\ell,\ell} = \left(\exp\left(\pi i \frac{(n-1)}{n}\right) \right)^\ell,$$

such that the scaling factors can be obtained by successive multiplication of 1 with `fac =  $\exp\left(\pi i \frac{(n-1)}{n}\right)$` .

C++ code 4.7.5: Function `modcostrf()`

```

2  Eigen::VectorXd modcostrf(const Eigen::VectorXd &a) {
3      using Comp = std::complex<double>;
4      unsigned int n = a.size();
5      Eigen::VectorXd f(n);
6      Eigen::FFT<double> fft;
7      Eigen::VectorXcd tmp(2 * n);
8      tmp << 0.5 * a, Eigen::VectorXcd::Zero(n);
9      Eigen::VectorXcd v1 = fft.fwd(tmp);
10     tmp << 0.5 * a.reverse(), Eigen::VectorXcd::Zero(n);
11     Eigen::VectorXcd v2 = fft.fwd(tmp);
12     Comp fac = std::exp(((n - 1) * M_PI * Comp(0, 1)) / (double)n);
13     Comp d(1.0, 0.0);
14     for (int k = 0; k < n; ++k) {
15         f[k] = (v1[k] + d * v2[k]).real();
16         d *= fac;
17     }
18     return f;
19 }
```



End Problem 4-7 , 20 min.