

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 2-11: Sequential linear systems

Consider a sequence of linear systems when all the linear systems share the same matrix $\mathbf{A}$: $\mathbf{A}\mathbf{x} = \mathbf{b}_i$. The computational cost of this problem may be reduced by performing an LU decomposition of $\mathbf{A}$ only once and reusing it for the different systems.

Related to [Lecture → Rem. 2.5.0.10]

Consider the following pseudocode with input data $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^n$:

Algorithm 2.11.1: Code solvepermb

Require: Matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, vector $\mathbf{b} \in \mathbb{R}^n$

Ensure: Matrix $\mathbf{X} = [\mathbf{X}_1, \dots, \mathbf{X}_n] \in \mathbb{R}^{n \times n}$, $\mathbf{X}_i \in \mathbb{R}^n$

while not all cyclic permutations of $\mathbf{b}$ tested yet **do**

$\mathbf{b} \leftarrow [b_n, b_1, b_2, \dots, b_{n-1}]^\top$

$\mathbf{X}_i = \mathbf{A}^{-1}\mathbf{b}$

end while

(2-11.a) ☐ What is the worst case asymptotic complexity of the function solvepermb for $n \rightarrow \infty$?

HINT 1 for (2-11.a): Think about the complexity of solving a linear system (Gaussian elimination, see [Lecture → § 2.3.1.5]) for n times. ┘

SOLUTION of (2-11.a):

This function solves the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}_i$ n times. In every iteration i (i going from 1 to n) $\mathbf{b}_i$ is the original vector shifted by $i - 1$ elements. The output is the matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$, where each column $\mathbf{X}_i$ corresponds to a solution of $\mathbf{A}\mathbf{X}_i = \mathbf{b}_i$.

The complexity of solving a linear system is generally $O(n^3)$.

The complexity of solving n linear systems is therefore $O(n^4)$.



(2-11.b) ☐ Port the function solvepermb (Code 2.11.1) to C++ using EIGEN.

HINT 1 for (2-11.b): The C++ code should perform exactly the same computations in the same order. Consider the constructor of class template FullPivLU in EIGEN to solve the linear systems. ┘

SOLUTION of (2-11.b):

C++11-code 2.11.2: Solution of (2-11.b)

```

2 void solvepermb(const Eigen::MatrixXd& A, Eigen::VectorXd& b,
3               Eigen::MatrixXd& X) {
4     // Size of b, which is the size of A
5     const unsigned int n = b.size();
6     assert(n == A.rows() && n == A.cols() && "Error: size mismatch!");
7     // TODO: (2-11.b) Port the pseudocode from (3.11.1) to C++-code.
8     // START
9     X.resize(n, n);
10

```

```

11 // For each loop iteration:
12 // 1. solve the linear system  $Ax = b$ ,
13 // 2. store the result in a column of  $X$ 
14 // 3. and shift  $b$  by one element for the next iteration.
15 for (unsigned int l = 0; l < n; ++l) {
16     X.col(l) = A.fullPivLu().solve(b);
17     shift(b);
18 }
19 // END
20 }

```

Get it on  GitLab (solvepermb.hpp).



(2-11.c)  Design an efficient C++ implementation of function `solvepermb` using EIGEN with asymptotic complexity $O(n^3)$.

HINT 1 for (2-11.c): The complexity increases by solving the linear system n times *from scratch*. However, there are ways to represent the matrix A which makes the solution of the linear problem $Ax = b$ much easier to find. One of these ways is the LU decomposition. If we compute the LU decomposition of A once, we can reuse it when solving $Ax = b_i$ for different b_i .

SOLUTION of (2-11.c):

C++11-code 2.11.3: Solution of (2-11.c)

```

2 void solvepermb_on3(const Eigen::MatrixXd& A, Eigen::VectorXd& b,
3                     Eigen::MatrixXd& X) {
4     // Size of b, which is the size of A
5     const unsigned int n = b.size();
6     assert(n == A.cols() && n == A.rows() && "Error: size mismatch!");
7     // TODO: (2-11.c) Alter the above function such that it has
8     // asymptotic
9     // complexity  $O(n^3)$ .
10    // START
11    X.resize(n, n);
12
13    // Notice that here we reuse the LU factorization
14    Eigen::FullPivLU<Eigen::MatrixXd> LU = A.fullPivLu();
15
16    for (unsigned int l = 0; l < n; ++l) {
17        X.col(l) = LU.solve(b);
18        shift(b);
19    }
20    // END
21 }

```

Get it on  GitLab (solvepermb.hpp).



End Problem 2-11