

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 2-12: Structured linear systems

In this problem we come across the example of a structured matrix, for which a linear system can be solved very efficiently, though this is not obvious.

Related to [Lecture → Section 2.6]

Consider the linear system $\mathbf{Ax} = \mathbf{b}$, where the $n \times n$ matrix $\mathbf{A}$ has the following structure:

$$\mathbf{A} = \begin{bmatrix} a_1 & 0 & 0 & \dots & 0 \\ a_1 & a_2 & 0 & \ddots & \vdots \\ a_1 & a_2 & a_3 & \ddots & \\ \vdots & \vdots & & \ddots & 0 \\ a_1 & a_2 & a_3 & \dots & a_n \end{bmatrix}$$

(2-12.a) □ Give necessary and sufficient conditions for the vector $\mathbf{a} = (a_1, \dots, a_n)^\top \in \mathbb{R}^n$ such that the matrix $\mathbf{A}$ is non-singular.

HINT 1 for (2-12.a): A matrix $\mathbf{A}$ is non-singular if and only if $\det|\mathbf{A}| \neq 0$ (which is also equivalent to $\mathbf{A}$ being invertible $\iff \text{rank}(\mathbf{A}) = n$).

It can be proven by induction that the determinant of a triangular matrix is the product of its diagonal elements, i.e. $\det|\mathbf{A}| = \prod_{i=1}^n a_i$. ┘

SOLUTION of (2-12.a):

$$a_i \neq 0 \quad \forall i = 1, \dots, n$$



(2-12.b) □ Write a C++ function that builds the matrix $\mathbf{A}$ given the vector $\mathbf{a}$:

```
Eigen::MatrixXd buildA(const Eigen::VectorXd & a);
```

You can use EIGEN classes for your code.

HINT 1 for (2-12.b): Consider a loop initialization. ┘

SOLUTION of (2-12.b):

C++11-code 2.12.1: Solution of (2-12.b)

```
2 Eigen::MatrixXd buildA(const Eigen::VectorXd& a) {
3   // Initialization
4   const unsigned int n = a.size();
5   Eigen::MatrixXd A = Eigen::MatrixXd::Zero(n, n);
6   // TODO: (2-12.b) Build the matrix A given the vector a.
7   // START
8   for (unsigned int j = 0; j < n; ++j) {
9       for (unsigned int i = j; i < n; ++i) {
```

```

10     A(i, j) = a(j);
11   }
12 }
13 // END
14 return A;
15 }

```

Get it on  GitLab (structuredLSE.hpp).



(2-12.c) ☐ Implement a function in C++ which solves the linear system $Ax = b$ by means of “structure oblivious” Gaussian elimination, for which EIGEN’s dedicated classes and methods should be used.

```

void solveA(const Eigen::VectorXd & a, const Eigen::VectorXd & b,
            Eigen::VectorXd & x);

```

The input is composed of vectors **a** and **b**.

Run this function with **a** and **b** made of random elements and explore the cases $n = 2^k$, $k = 4, \dots, 12$. What is the asymptotic complexity in n of this naive implementation?

HINT 1 for (2-12.c): Remember what you did in Code 2.11.2.

The `main()` code to test your implementation is already provided in the template.



SOLUTION of (2-12.c):

C++11-code 2.12.2: Solution of (2-12.c)

```

2 void solveA(const Eigen::VectorXd& a, const Eigen::VectorXd& b,
3             Eigen::VectorXd& x) {
4     // size of b, which is the size of a
5     const unsigned int n = b.size();
6     assert(n == a.size() && "Error: size mismatch!");
7     // TODO: (2-12.c) Solve Ax = b using "structure oblivious" Gaussian
8     // elimination.
9     // START
10    x.resize(n);
11    Eigen::MatrixXd A = buildA(a);
12    x = A.fullPivLu().solve(b);
13    // END
14 }

```

Get it on  GitLab (structuredLSE.hpp).

Even with a naive implementation, the solution of a lower triangular system corresponds to a forward substitution and should be of $O(n^2)$ complexity. However, EIGEN’s methods are not optimized and follow the general complexity of solving a linear system, which is $O(n^3)$. This can be seen from the plot in Fig. 7.

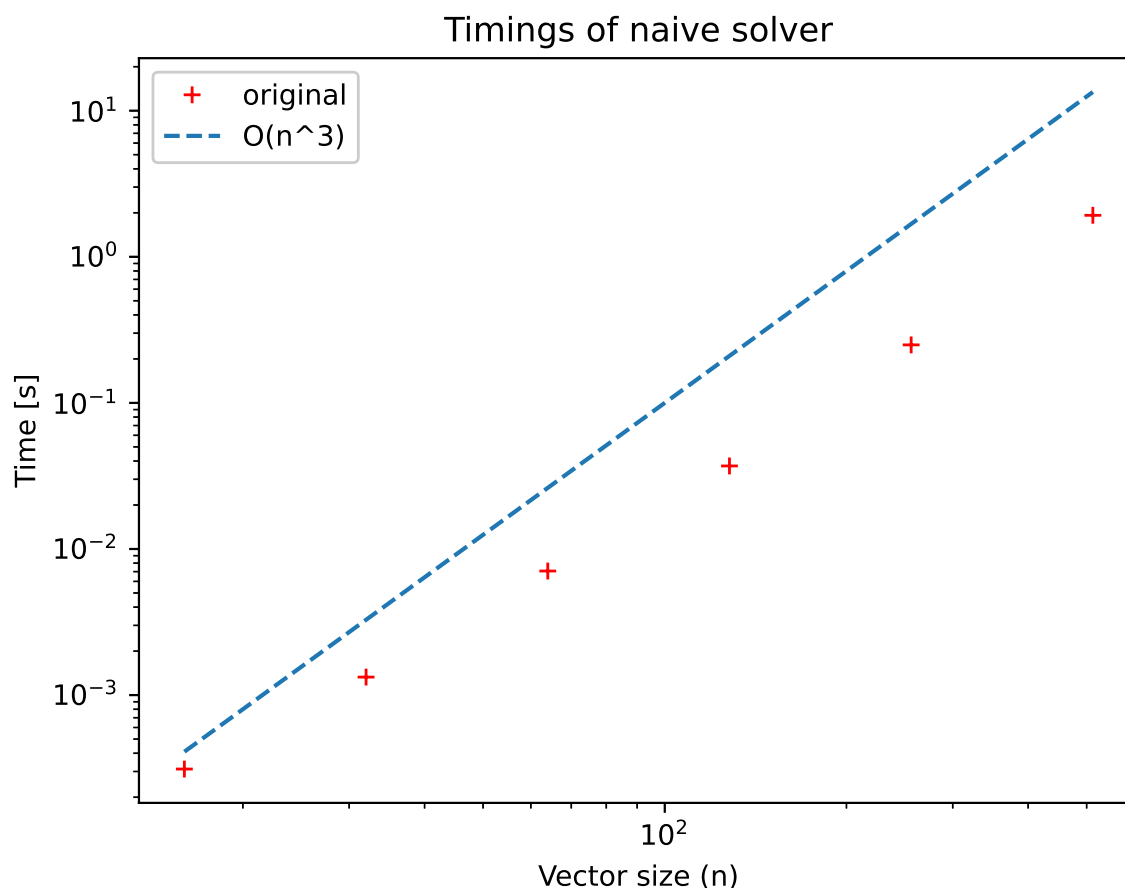


Fig. 7

(2-12.d) Given a generic vector $\mathbf{a} \in \mathbb{R}^n$, compute symbolically (i.e. *by hand*) the inverse of matrix $\mathbf{A}$ and the solution $\mathbf{x}$ of $\mathbf{Ax} = \mathbf{b}$.

HINT 1 for (2-12.d): Notice the following equality:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 1 & 1 & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & 0 & \dots & 0 \\ 0 & a_2 & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ 0 & 0 & \dots & a_n \end{bmatrix} = \mathbf{E} \cdot \mathbf{D}.$$

Then figure out an efficient way to solve a linear system of equations with coefficient matrix $\mathbf{E}$.

SOLUTION of (2-12.d):

$$\begin{aligned}
 \mathbf{A}^{-1} &= \mathbf{D}^{-1} \cdot \mathbf{E}^{-1} = \begin{bmatrix} a_1^{-1} & 0 & 0 & \dots & 0 \\ 0 & a_2^{-1} & 0 & \ddots & \vdots \\ 0 & 0 & a_3^{-1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & 0 & a_n^{-1} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ -1 & 1 & 0 & \ddots & \vdots \\ 0 & -1 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} a_1^{-1} & 0 & 0 & \dots & 0 \\ -a_2^{-1} & a_2^{-1} & 0 & \ddots & \vdots \\ 0 & -a_3^{-1} & a_3^{-1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -a_n^{-1} & a_n^{-1} \end{bmatrix}
 \end{aligned}$$

While the matrix $\mathbf{A}$ is dense, the inverse $\mathbf{A}^{-1}$ is sparse and banded with bandwidth = 2!

$$\mathbf{x} = \mathbf{D}^{-1}\mathbf{E}^{-1}\mathbf{b} \implies x_j = \begin{cases} b_1/a_1 & j = 1 \\ \frac{b_j - b_{j-1}}{a_j} & j = 2, \dots, n \end{cases}$$



(2-12.e) Implement a function in C++ which efficiently solves the linear system $\mathbf{Ax} = \mathbf{b}$, using EIGEN classes. The input consists of vectors $\mathbf{a}$ and $\mathbf{b}$.

HINT 1 for (2-12.e): Exploit the solution of (2-12.d).



SOLUTION of (2-12.e):

C++11-code 2.12.3: Solution of (2-12.e)

```

2 void solveA_fast(const Eigen::VectorXd& a, const Eigen::VectorXd& b,
3                 Eigen::VectorXd& x) {
4     // size of b, which is the size of a
5     const unsigned int n = b.size();
6     assert(n == a.size() && "Error: size mismatch!");
7     // TODO: (2-12.e) Efficiently (O(n)) solve Ax = b.
8     // START
9     x.resize(n);
10
11     x(0) = b(0) / a(0);
12     for (unsigned int i = 1; i < n; ++i) {
13         x(i) = (b(i) - b(i - 1)) / a(i);
14     }
15     // END
16 }

```

Get it on GitLab (structuredLSE.hpp).



(2-12.f) ☐ Compare the timing of the naive implementation of (2-12.c) with the efficient solution of (2-12.e).

HINT 1 for (2-12.f): The `main()` code to test your implementation is already provided in the template.

SOLUTION of (2-12.f):

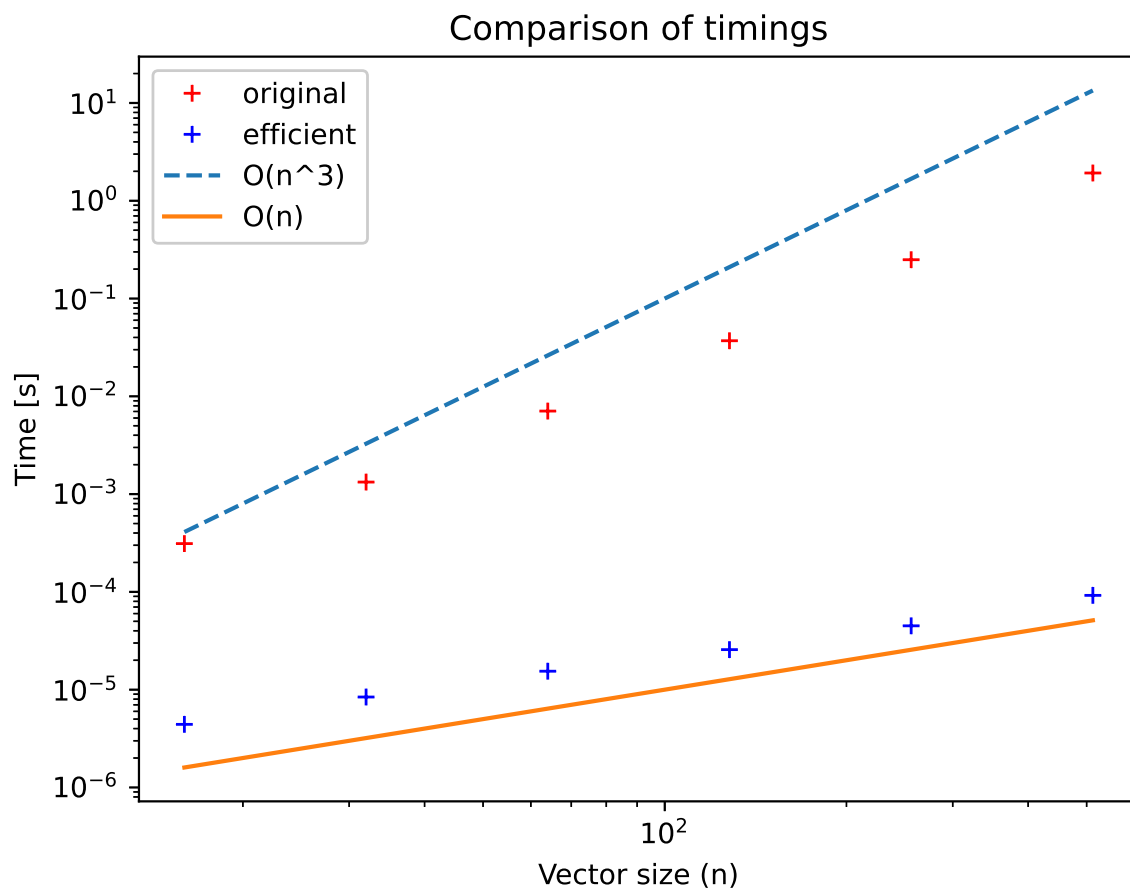


Fig. 8



End Problem 2-12