

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 2-3: Block-wise LU-decomposition

Using block matrix operations and the LU-decomposition as described in [Lecture → Rem. 2.3.2.19], you should implement a function to solve a LSE for a matrix of particular structure.

Moderate EIGEN-based implementation in C++. Connected with [Lecture → Section 2.6]

Let the matrix $\mathbf{A} \in \mathbb{R}^{n+1, n+1}$ be block-partitioned according to

$$\mathbf{A} = \begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{u}^\top & 0 \end{bmatrix}$$

where $\mathbf{v}, \mathbf{u} \in \mathbb{R}^n$ and $\mathbf{R} \in \mathbb{R}^{n, n}$ is *upper triangular* and regular.

(2-3.a) ☐ (15 min.) Refresh yourself on the definition of the LU-decomposition of a matrix [Lecture → Def. 2.3.2.3] and study [Lecture → Rem. 2.3.2.19]. ▲

(2-3.b) ☐ (10 min.) Find the LU-decomposition of $\mathbf{A}$.

SOLUTION of (2-3.b):

We can simply rely on

$$\underbrace{\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}}_{\text{block LU-factorization}} = \begin{bmatrix} \mathbf{I} & 0 \\ \mathbf{A}_{21}\mathbf{A}_{11}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{S} \end{bmatrix}, \quad \text{with Schur complement } \boxed{\mathbf{S} := \mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}}. \quad [\text{Lecture} \rightarrow \text{Eq. (2.3.2.20)}]$$

Here we have $\mathbf{A}_{11} = \mathbf{R}$, $\mathbf{A}_{12} = \mathbf{v}$, $\mathbf{A}_{21} = \mathbf{u}^\top$, $\mathbf{A}_{22} = \mathbf{O}$. Thus the sought LU-decomposition is

$$\mathbf{A} = \mathbf{LU} = \begin{bmatrix} \mathbf{I} & 0 \\ \mathbf{u}^\top \mathbf{R}^{-1} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{v} \\ 0 & -\mathbf{u}^\top \mathbf{R}^{-1} \mathbf{v} \end{bmatrix}$$

Here it is important that $\mathbf{R}$ was assumed to be invertible. ▲

(2-3.c) ☐ (10 min.) [depends on Sub-problem (2-3.b)]

Show that $\mathbf{A}$ is regular, if and only if $\mathbf{u}^\top \mathbf{R}^{-1} \mathbf{v} \neq 0$.

HINT 1 for (2-3.c): Use the LU-decomposition of $\mathbf{A}$ to compute its determinant. ┘

SOLUTION of (2-3.c):

$\mathbf{A}$ is regular if and only if $\det(\mathbf{A}) \neq 0$.

$$\det(\mathbf{A}) = \det(\mathbf{L}) \det(\mathbf{U}) = \det(\mathbf{L}) \det(\mathbf{R}) (\mathbf{u}^\top \mathbf{R}^{-1} \mathbf{v}) \quad (2.3.1)$$

$\det(\mathbf{L}) = 1$ and $\mathbf{R}$ is regular, so $\det(\mathbf{R}) \neq 0$, which leaves $\mathbf{u}^\top \mathbf{R}^{-1} \mathbf{v} \neq 0$ as necessary and sufficient condition.

(2-3.d) 🕒 (15 min.) [depends on Sub-problem (2-3.b)]

In the file `blockLU.hpp` write a C++ function

```
Eigen::VectorXd solve_LSE(const Eigen::MatrixXd& R, const
    Eigen::VectorXd& u, const Eigen::VectorXd& v, const
    Eigen::VectorXd& b);
```

that solves $Ax = b$ for A described above and returns x . Make use of the block-LU-decomposition you derived in Sub-problem (2-3.b) and only use elementary operations, in particular no built-in solver classes of EIGEN.

HINT 1 for (2-3.d): You will need to solve a LSE multiple times by backwards substitution. It can be useful to implement a function `Eigen::VectorXd solve_R(const Eigen::MatrixXd& R, const Eigen::VectorXd& c);` that does that. ┘

SOLUTION of (2-3.d):

C++ code 2.3.2: Code

```
2 Eigen::VectorXd solve_LSE(const Eigen::MatrixXd& R, const Eigen::VectorXd& v,
3                           const Eigen::VectorXd& u, const Eigen::VectorXd& b) {
4     const unsigned int n = R.rows();
5     assert(R.cols() == n && "R has to be square");
6     assert(n == v.size() && n == u.size() && n + 1 == b.size() &&
7            "Input dimensions must agree");
8     Eigen::VectorXd y(n + 1), x(n + 1);
9
10    // TODO: (2-3.d) Solve the system Ax=b by LU-Decomposition.
11    // START
12    // Solve Ly = b through forward substitution.
13    // Due to the special structure of our L,
14    // the first n entries of y are easy:
15    y.head(n) = b.head(n);
16    // The last element of y is given by  $y_n = b_n -$ 
17    //  $u^T R^{-1} y_{0:n-1}$ 
18    y(n) = b(n) - u.transpose() * solve_R(R, y.head(n));
19
20    // Solve Ux = y by backward substitution.
21    // First we build U
22    Eigen::MatrixXd U(n + 1, n + 1);
23    U << R, v, Eigen::VectorXd::Zero(n).transpose(),
24        -u.transpose() * solve_R(R, v);
25
26    // Then we solve Ux = y
27    x = solve_R(U, y);
28    //
29    // END
30    return x;
31 }
```

Get it on 🐙 [GitLab](#) (`blockLU.hpp`).

This makes use of a function to solve a LSE by backwards substitution:

C++ code 2.3.3: Code

```
2 Eigen::VectorXd solve_R(const Eigen::MatrixXd& R, const Eigen::VectorXd& c) {
```

```
3 | const unsigned int n = R.rows();
4 | assert(n == R.cols() && n == c.size() && "Input dimensions must agree");
5 | // initialize the return value
6 | Eigen::VectorXd y(n);
7 |
8 | // TODO: (optional) Solve Ry=c using backward substitution. May
9 | help
10 | // when implementing solve_LSE.
11 | // START
12 | // Since R is upper triangular, we can solve by backwards
13 | substitution
14 | for (int i = n - 1; i >= 0; --i) {
15 |     y(i) = c(i);
16 |     for (int j = n - 1; j > i; --j) {
17 |         y(i) -= R(i, j) * y(j);
18 |     }
19 |     y(i) /= R(i, i);
20 | }
21 | // END
22 | return y;
23 | }
```

Get it on 🐙 [GitLab](#) (blockLU.hpp).



End Problem 2-3 , 50 min.