

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 2-8: Lyapunov equation

Any linear system of equations with a finite number of unknowns can be written in the “canonical form” $\mathbf{Ax} = \mathbf{b}$ with a system matrix $\mathbf{A}$ and a right hand side vector $\mathbf{b}$. However, the linear system may be given in a different form and it may not be obvious how to extract the system matrix. We propose an intriguing example and also present an important *matrix equation*, the so-called **Lyapunov equation**.

Related to [Lecture → Section 2.5] and [Lecture → Section 2.7.2]

Given the square matrix $\mathbf{A} \in \mathbb{R}^{n,n}$, we consider the equation

$$\mathbf{AX} + \mathbf{XA}^\top = \mathbf{I}, \quad (2.8.1)$$

where the matrix $\mathbf{X} \in \mathbb{R}^{n,n}$ is the unknown.

(2-8.a) ☐ (10 min.) Show that for a fixed matrix $\mathbf{A} \in \mathbb{R}^{n,n}$ the following mapping is *linear*:

$$L: \begin{cases} \mathbb{R}^{n,n} & \rightarrow \mathbb{R}^{n,n} \\ \mathbf{X} & \mapsto \mathbf{AX} + \mathbf{XA}^\top \end{cases}$$

HINT 1 for (2-8.a): Recall the definition of a linear mapping between two vector spaces from linear algebra. ┘

SOLUTION of (2-8.a):

Take $\alpha, \beta \in \mathbb{R}$ and $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{n,n}$. We readily compute:

$$\begin{aligned} L(\alpha\mathbf{X} + \beta\mathbf{Y}) &= \mathbf{A}(\alpha\mathbf{X} + \beta\mathbf{Y}) + (\alpha\mathbf{X} + \beta\mathbf{Y})\mathbf{A}^\top \\ &= \alpha\mathbf{AX} + \beta\mathbf{AY} + \alpha\mathbf{XA}^\top + \beta\mathbf{YA}^\top \\ &= \alpha(\mathbf{AX} + \mathbf{XA}^\top) + \beta(\mathbf{AY} + \mathbf{YA}^\top) \\ &= \alpha L(\mathbf{X}) + \beta L(\mathbf{Y}) \end{aligned}$$

This confirms linearity of the mapping $\mathbf{X} \mapsto \mathbf{AX} + \mathbf{XA}^\top$

▲

In the following let $\text{vec}(\mathbf{M}) \in \mathbb{R}^{n^2}$ denote the column vector obtained by storing the internal coefficient array of a matrix $\mathbf{M} \in \mathbb{R}^{n,n}$ ($\rightarrow$ [Lecture → Eq. (1.2.3.5)]) in column major format, i.e. a data array with n^2 components [Lecture → Rem. 1.2.3.4]. In PYTHON, MATLAB, $\text{vec}(\mathbf{M})$ would be the column vector obtained by `reshape(M, n*n, 1)`, see [Lecture → Rem. 1.2.3.6] for the implementation based on EIGEN.

Owing to the observation made in Sub-problem (2-8.a), (2.8.1) is equivalent to the following linear system of equations:

$$\mathbf{C} \text{vec}(\mathbf{X}) = \mathbf{b} \quad (2.8.2)$$

The system matrix is $\mathbf{C} \in \mathbb{R}^{n^2, n^2}$ and the right-hand side vector $\mathbf{b} \in \mathbb{R}^{n^2}$.

(2-8.b) ☐ (10 min.) Recall the notion of *sparse matrix*: see [Lecture → Section 2.7] and, in particular, [Lecture → Notion 2.7.0.1] and [Lecture → Def. 2.7.0.3]. ▲

(2-8.c) 🕒 (15 min.) Determine $\mathbf{C}$ and $\mathbf{b}$ from Eq. (2.8.2) for $n = 2$ and

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ -1 & 3 \end{bmatrix}$$

HINT 1 for (2-8.c): Start by expressing $\mathbf{X}$ as $\begin{bmatrix} x_1 & x_3 \\ x_2 & x_4 \end{bmatrix}$.

SOLUTION of (2-8.c):

Writing $\mathbf{X} = \begin{bmatrix} x_1 & x_3 \\ x_2 & x_4 \end{bmatrix}$, we have that $\text{vec}(\mathbf{X}) = (x_1, x_2, x_3, x_4)^\top$. A direct calculation shows that (2.8.1) is equivalent to Eq. (2.8.2) with:

$$\mathbf{C} = \begin{bmatrix} 4 & 1 & 1 & 0 \\ -1 & 5 & 0 & 1 \\ -1 & 0 & 5 & 1 \\ 0 & -1 & -1 & 6 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

(2-8.d) 🕒 (25 min.) Use the Kronecker product $\otimes$ [Lecture $\rightarrow$ Def. 1.4.3.7] to find a general expression for $\mathbf{C}$ in terms of $\mathbf{A}$.

SOLUTION of (2-8.d):

By elementary properties of the matrix product:

$$\begin{aligned} \mathbf{AX} &= [\mathbf{A}(\mathbf{X})_{:,1}, \dots, \mathbf{A}(\mathbf{X})_{:,n}] \Leftrightarrow \text{vec}(\mathbf{AX}) = (\mathbf{I} \otimes \mathbf{A}) \text{vec}(\mathbf{X}), \\ (\text{vec}(\mathbf{XA}^\top))_k &= (\mathbf{XA}^\top)_{i^*,j^*} = \sum_{\ell=1}^n (\mathbf{X})_{i^*,\ell} (\mathbf{A})_{j^*,\ell} = \sum_{\ell=1}^n (\mathbf{A})_{j^*,\ell} (\text{vec}(\mathbf{X}))_{i^*+(\ell-1)n}, \end{aligned}$$

where $(\text{vec}(\mathbf{X}))_k = (\mathbf{X})_{i^*,j^*}$. Thus, we have

$$\mathbf{C} = \mathbf{I} \otimes \mathbf{A} + \mathbf{A} \otimes \mathbf{I}. \quad (2.8.3)$$

The first term is related to $\mathbf{AX}$, the second to $\mathbf{XA}^\top$.

(2-8.e) 🕒 (30 min.) [depends on Sub-problem (2-8.d)]

In the file `lyapunov.hpp` implement a C++ function that builds the EIGEN matrix $\mathbf{C}$ from $\mathbf{A}$:

```
Eigen::SparseMatrix<double> buildC(const Eigen::MatrixXd& A)
```

Make sure that the initialisation is done efficiently using an intermediate triplet format, see [Lecture $\rightarrow$ Section 2.7.2].

HINT 1 for (2-8.e): Read [Lecture $\rightarrow$ Section 2.7.2] carefully before starting.

SOLUTION of (2-8.e):

The argument matrix is assumed to be densely populated, but we have to return a matrix in sparse format. The simplest option for efficient initialization is to go via the triplet/COO format. Then go through the $n \times n$ -blocks in (2.8.3) to fill the triplets

C++ code 2.8.4: Solution of (2-8.e)

```

2  Eigen::SparseMatrix<double> buildC(const Eigen::MatrixXd &A) {
3      // Initialization
4      const unsigned int n = A.rows();
5      Eigen::SparseMatrix<double> C(n * n, n * n);
6      std::vector<Eigen::Triplet<double>> triplets;
7      Eigen::MatrixXd I = Eigen::MatrixXd::Identity(n, n);
8
9      // Iterate over n*n blocks.
10     for (unsigned int i = 0; i < n; ++i) {
11         for (unsigned int j = 0; j < n; ++j) {
12             // TODO: (2-8.e) Fill in the (i,j)-th block of C.
13             // Hint: The vector triplets should contain the position
14             // and value the non-zero entries of C.
15             // Use triplets.push_back() to add a new
16             // Triplet<double> to the list.
17             // START
18             // Set diagonal
19             if (i == j) {
20                 for (unsigned int k1 = 0; k1 < n; ++k1) {
21                     for (unsigned int k2 = 0; k2 < n; ++k2) {
22                         Eigen::Triplet<double> triplet(i * n + k1, j * n + k2,
23                                                         A(i, j) * I(k1, k2) + A(k1, k2));
24                         triplets.push_back(triplet);
25                     }
26                 }
27                 // Set other cells
28             } else {
29                 for (unsigned int k = 0; k < n; ++k) {
30                     Eigen::Triplet<double> triplet(i * n + k, j * n + k, A(i, j));
31                     triplets.push_back(triplet);
32                 }
33             }
34             // END
35         }
36     }
37     C.setFromTriplets(triplets.begin(), triplets.end());
38     C.makeCompressed();
39     return C;
40 }

```

Get it on  GitLab (lyapunov.hpp).



(2-8.f)  (20 min.) In the file `lyapunov.hpp` a C++ function that returns the solution of (2.8.1), i.e. the $n \times n$ -matrix $\mathbf{X}$ if $\mathbf{A} \in \mathbb{R}^{n,n}$ is passed in the argument $\mathbf{A}$.

```
void solveLyapunov(const Eigen::MatrixXd& A, Eigen::MatrixXd& X)
```

Use a suitable sparse elimination solver provided by EIGEN, see [Lecture → Section 2.7.3].

HINT 1 for (2-8.f): Not every invertible matrix $\mathbf{A}$ allows a solution. If $\mathbf{A}$ and $-\mathbf{A}$ have a common

eigenvalue, the system $Cx = b$ is singular. This is for example the case of the matrix $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. For an efficient solution, see Chapter 15 of Higham's book. ┘

SOLUTION of (2-8.f):

C++11-code 2.8.5: Solution of (2-8.f)

```

2 void solveLyapunov(const Eigen::MatrixXd &A, Eigen::MatrixXd &X) {
3     // To solve A*X + X*A^T = I, we must first write
4     // the LSE in "canonical form", C*vecX = b.
5     const unsigned int n = A.rows();
6     Eigen::SparseMatrix<double> C;
7     Eigen::MatrixXd I = Eigen::MatrixXd::Identity(n, n);
8     Eigen::VectorXd b(n * n);
9     Eigen::VectorXd vecX(n * n);
10
11     // TODO: (2-8.f) Fill in the entries of C, b, and vecX.
12     // Hint: Use SparseLU to solve the LSE.
13     // START
14     C = buildC(A);
15     b = Eigen::Map<Eigen::MatrixXd>(I.data(), n * n, 1);
16     Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
17     solver.compute(C);
18     vecX = solver.solve(b);
19     // END
20
21     X = Eigen::Map<Eigen::MatrixXd>(vecX.data(), n, n);
22 }

```

Get it on  GitLab (lyapunov.hpp).

(2-8.g)  (10 min.) Write a C++ function

bool testLyapunov() ;

that is meant to validate your implementation of `buildC` and `solveLyapunov` for $n = 5$ and:

$$A = \begin{bmatrix} 10 & 2 & 3 & 4 & 5 \\ 6 & 20 & 8 & 9 & 1 \\ 1 & 2 & 30 & 4 & 5 \\ 6 & 7 & 8 & 20 & 0 \\ 1 & 2 & 3 & 4 & 10 \end{bmatrix}.$$

SOLUTION of (2-8.g):

C++ code 2.8.6: Solution of (2-8.g)

```

2 bool testLyapunov() {
3     // Initialization
4     constexpr unsigned int n = 5;
5     Eigen::MatrixXd A(n, n);
6     A << 10, 2, 3, 4, 5, 6, 20, 8, 9, 1, 1, 2, 30, 4, 5, 6, 7, 8, 20, 0, 1, 2, 3,

```

```

7      4, 10;
8
9      constexpr double tol = 1e-8;
10     bool works = false;
11
12     // TODO: (2-8.g)
13     // i) Print the full system matrix C (with zeroes),
14     // ii) print the solution X, and
15     // iii) return true if and only if the error defined
16     // by || A*X + X*A^T - I || is less than tol.
17     // START
18     // Test 'buildC'
19     Eigen::SparseMatrix<double> C = buildC(A);
20     std::cout << "C = " << C << std::endl;
21
22     // Test 'solveLyapunov'
23     Eigen::MatrixXd X(n, n);
24     solveLyapunov(A, X);
25
26     std::cout << "X = " << X << std::endl;
27
28     // Verify the solution if you obtain zero
29     Eigen::MatrixXd I = Eigen::MatrixXd::Identity(n, n);
30     double error = (A * X + X * A.transpose() - I).norm();
31     std::cout << "Correct if close to 0: " << error << std::endl;
32
33     works = (error < tol);
34     // END
35
36     return works;
37 }

```

Get it on  GitLab (lyapunov.hpp).



(2-8.h)  (15 min.) Give an upper bound (as sharp as possible) for $\text{nnz}(\mathbf{C})$ in terms of $\text{nnz}(\mathbf{A})$ (nnz is the number of nonzero elements). Can $\mathbf{C}$ be legitimately regarded as a sparse matrix for large n even if $\mathbf{A}$ is dense?

SOLUTION of (2-8.h):

For general matrices $\mathbf{A}$ and $\mathbf{B}$ we have that $\text{nnz}(\mathbf{A} \otimes \mathbf{B}) = \text{nnz}(\mathbf{A}) \text{nnz}(\mathbf{B})$. This follows from the fact that the block in position (i, j) of the matrix $\mathbf{A} \otimes \mathbf{B}$ is $a_{ij}\mathbf{B}$.

In our case, we immediately obtain $\text{nnz}(\mathbf{C}) = \text{nnz}(\mathbf{I} \otimes \mathbf{A} + \mathbf{A} \otimes \mathbf{I}) \leq \text{nnz}(\mathbf{I} \otimes \mathbf{A}) + \text{nnz}(\mathbf{A} \otimes \mathbf{I}) \leq 2 \text{nnz}(\mathbf{I}) \text{nnz}(\mathbf{A})$. Summarising, $\text{nnz}(\mathbf{C}) < 2n \text{nnz}(\mathbf{A})$. The sharpness of this bound can be checked with the matrix $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

This bound implies that, even if $\mathbf{A}$ is not sparse, we have that $\text{nnz}(\mathbf{C}) < 2n^3 \ll n^4$. Therefore, $\mathbf{C}$, being an $n^2 \times n^2$ matrix, can be regarded as sparse for any $\mathbf{A}$.

Another way to see this is the following. The block in position (i, j) of matrix $\mathbf{C}$ is given by $a_{ij}\mathbf{I} + \delta_{ij}\mathbf{A}$ (δ_{ij} is Kronecker delta). Suppose now that $a_{ij} \neq 0$. We have two cases:

$i \neq j$ There are at most n nonzero entries in the block in position (i, j) of matrix $\mathbf{C}$, which is equal to $a_{ij}\mathbf{I}$, and one nonzero entry in each diagonal block. This gives a total of, at most, $2n$ nonzero

entries.

$i = j$ There is at most one nonzero entry in each diagonal block. Thus, a total of, at most, n nonzero entries.



End Problem 2-8 , 135 min.