

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 0-2: EIGEN block operations on matrices

EIGEN matrices offer a range of methods to access sub-matrices (blocks), instead of individual entries at a time. In this exercise we practice their use.

This problem practices certain operations in EIGEN and is connected with [Lecture → § 1.2.1.5], which you should read beforehand.

Remark. The codes given as solutions may leave ample room for improvement. You should have the ambition to do better than the master solution.

(0-2.a) 🕒 (15 min.) In the file `MatrixBlocks.hpp` supplement the missing lines of the function

```
Eigen::MatrixXd
zero_row_col(const Eigen::MatrixXd &A, int p, int q);
```

so that it returns a matrix that arises from A by setting to zero the p -th row and q -th column.

HINT 1 for (0-2.a): In [CODEEXPERT](#) the following test case is implemented in the `main.cpp` file: `zero_row_col(Matrix3d::Constant(-1), 0, 1);`, which should produce the matrix

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & -1 \\ -1 & 0 & -1 \end{bmatrix}.$$

SOLUTION of (0-2.a):

Use the write access methods `row()` and `col()` for a matrix.

C++ code 0.2.1: Initialization of fixed size 2×2 -matrix

```
2 Eigen::MatrixXd zero_row_col(const Eigen::MatrixXd& A, unsigned int p,
3                             unsigned int q) {
4     // assert that we do not run into Eigen assertions.
5     assert(A.rows() > p && A.cols() > q && "p or q out of bounds.");
6
7     // Make a copy of A.
8     Eigen::MatrixXd Anew(A);
9
10    // TODO: (0-2.a) Set the entries of row number p and column number q
11    // to zero.
12    // Hint: We can access rows and columns of A by A.row() and
13    // A.col().
14    // The method setZero() is useful here.
15    // START
16    Anew.row(p).setZero();
17    Anew.col(q).setZero();
18    // END
19    return Anew;
20 }
```

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

(0-2.b) (20 min.)

Again in the file `MatrixBlocks.hpp` complete the implementation of the C++ function

```
MatrixXd swap_left_right_blocks(const MatrixXd &A, int p) {
```

that returns (see [Lecture → Section 1.1.1] for explanations concerning the notations; PYTHON users beware!)

$$\left[(\mathbf{A})_{:,p+1:m}, (\mathbf{A})_{:,1:p} \right] \quad \text{for } \mathbf{A} \in \mathbb{R}^{n,m}, \quad m, n \in \mathbb{N}, \quad 1 \leq p < m.$$

The matrix $\mathbf{A}$ is passed through the argument `A`.

HINT 1 for (0-2.b): The test call issued in `main.cpp` is `swap_left_right_blocks(MatrixXd::Identity(4,3),2);` and yields the matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

SOLUTION of (0-2.b):

We can use the write-access version of the `block()` method of matrix data types.

C++ code 0.2.2: Swapping left and right parts of a matrix in EIGEN.

```
2 Eigen::MatrixXd swap_left_right_blocks(const Eigen::MatrixXd& A,
3                                         unsigned int p) {
4     assert(A.cols() >= p && "Can only swap at most A.cols() columns.");
5     Eigen::MatrixXd B, C;
6
7     // We can use .rows() and .cols() to get the number of rows and
8     // columns in A.
9     unsigned int q = A.cols() - p;
10
11    // A.block( i, j, m, n ) returns the m by n block that has its
12    // top-left corner
13    // at the index (i,j) in A. Hence, the first p columns of A can be
14    // accessed in
15    // the following way:
16    B = A.block(0, 0, A.rows(), p);
17
18    // TODO: (0-2.b) Use A.block() to define C as the matrix containing
19    // the last q
20    // columns of A.
21    // START
22    C = A.block(0, p, A.rows(), q);
23    // END
24
25    // Make a copy of A.
26    Eigen::MatrixXd Anew(A);
27    // The block() method can access arbitrary blocks within a matrix.
28    // For our purposes, it is actually simpler to use leftCols() and
29    // rightCols().
30    Anew.leftCols(q) = C;
31
32    // TODO: (1-2.b) Use A.rightCols() to fill in the remaining columns
33    // of the new
34    // matrix A.
```

```

29 // START
30 Anew.rightCols(p) = B;
31 // END
32
33 // Tip: Many more methods exist that are special cases of block(),
34 // e.g. topRows(), bottomRows(), topLeftCorner(),
35 //       bottomLeftCorner(), ...
36 // For vectors we have head(), tail(), and segment().
37
38 return Anew;
39 }

```

Get it on  GitLab (MatrixBlocks.hpp).



(0-2.c)  (20 min.) Supplement the missing lines of the function

```
MatrixXd tridiagonal(int n, double a, double b, double c);
```

in the file MatrixBlocks.hpp that generates a square **tridiagonal** matrix

$$\begin{bmatrix}
 b & c & 0 & \dots & & & \dots & 0 \\
 a & b & c & 0 & & & & \vdots \\
 0 & \ddots & \ddots & \ddots & \ddots & & & \\
 \vdots & & & & & & & \\
 & & & & & & \ddots & \vdots \\
 & & & & \ddots & \ddots & \ddots & 0 \\
 \vdots & & & & & a & b & c \\
 0 & \dots & & \dots & 0 & a & b
 \end{bmatrix} \in \mathbb{R}^{n,n},$$

with the value $b \in \mathbb{R}$ on the main diagonal, the value $a \in \mathbb{R}$ on the first sub-diagonal, and the value $c \in \mathbb{R}$ on the first super-diagonal.

HINT 1 for (0-2.c): As test case invoked from `main.cpp` we use `tridiagonal(4,-1,2,-1)`; and should get

$$\begin{bmatrix}
 2 & -1 & 0 & 0 \\
 -1 & 2 & -1 & 0 \\
 0 & -1 & 2 & -1 \\
 0 & 0 & -1 & 2
 \end{bmatrix}$$



SOLUTION of (0-2.c):

The code below use a special submatrix access function of EIGEN: `diagonal()`. However, a simple loop-based implementation based on entry access via **operator** `()` is hardly less efficient.

C++ code 0.2.3: Initialization of a triangular matrix in EIGEN.

```

2  Eigen::MatrixX<double> tridiagonal(unsigned int n, double a, double b, double c) {
3      Eigen::MatrixX<double> A;
4
5      // TODO: (0-2.c) Fill the matrix A with zeros. Then fill the
6      // subdiagonal with
7      // the value a, the diagonal with b and the superdiagonal with c.
8      // Hint: You can get the diagonal of A by A.diagonal(). Moreover,
9      // you can get
10     // the super- and subdiagonals by passing +1 or -1 as arguments to
11     // A.diagonal(). Here, however, we can use templated access because
12     // the
13     // distance to the diagonal is known at compile time. This method
14     // is usually
15     // faster.
16     // START
17     A = Eigen::MatrixX<double>::Zero(n, n);
18     A.diagonal<-1>() = Eigen::VectorX<double>::Constant(n - 1, a);
19     A.diagonal() = Eigen::VectorX<double>::Constant(n, b);
20     A.diagonal<1>() = Eigen::VectorX<double>::Constant(n - 1, c);
21     // END
22
23     return A;
24 }

```

Get it on  GitLab (MatrixBlocks.hpp).



End Problem 0-2 , 55 min.