

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 2-14: Sparse matrices in CCS format

Sparse matrices must be stored in special formats that avoid storing the many zero entries. Sometimes one has to manipulate sparse matrices on that basic level. Therefore it is important to be familiar with some details of the storage formats. This exercise focuses on the CCS format.

Related to [Lecture → Section 2.7.1]

In [Lecture → § 2.7.1.4] the so-called *CRS format* (Compressed Row Storage) was introduced. It uses three arrays (`val`, `col_ind` and `row_ptr`) to store a sparse matrix.

EIGEN can also handle the *CCS format* (Compressed Column Storage), which is like CRS for the transposed matrix. It relies on the arrays `val`, `row_ind` and `col_ptr`.

(2-14.a) 🧩 (30 min.) Implement a C++ function which for a given square matrix $A \in \mathbb{R}^{n \times n}$ computes the data describing it in CCS format:

```
void CCS(const Eigen::MatrixXd & A, Eigen::VectorXd & val,
        Eigen::VectorXd & row_ind, Eigen::VectorXd & col_ptr);
```

The CCS data are returned through the vectors `val`, `row_ind`, and `col_ptr`.

Prohibited. While you can use EIGEN classes such as `Eigen::MatrixXd`, do not use EIGEN methods to directly access `val`, `row_ind` and `col_ptr`. In other words, you must not use EIGEN's class `Eigen::SparseMatrix` and simply run `makeCompressed()`.

You may assume that A does not have a column with all elements equal to 0 (Otherwise, it may become problematic to update `col_ptr`).

In this problem you may test for exact equality with 0.0, because zero matrix entries are not supposed to be results of floating point computations.

SOLUTION of (2-14.a):

C++11-code 2.14.1: Solution of (2-14.a)

```
2 void CCS(const Eigen::MatrixXd & A, Eigen::VectorXd & val,
3         Eigen::VectorXd & row_ind, Eigen::VectorXd & col_ptr) {
4     // number of rows and columns
5     const unsigned int m = A.rows();
6     const unsigned int n = A.cols();
7
8     // TODO: (2-14.a) Implement a function that stores the matrix A
9     // in the vectors (val, row_ind, col_ptr) in CCS format
10    // Do not use Eigen methods for Sparse matrices.
11    // START
12
13    // number of nonzeros in the matrix
14    unsigned int nnz = 0;
15    for (unsigned int i = 0; i < m; ++i) {
16        for (unsigned int j = 0; j < n; ++j) {
17            if (A(i, j) != 0) ++nnz;
18        }
19    }
20
21    // initialization
22    val.resize(nnz);
```

```

23 row_ind.resize(nnz);
24 col_ptr.resize(n);
25
26 // store A in CCS format
27 unsigned int index = 0;
28 for (unsigned int j = 0; j < n; ++j) { // col iterator
29     // update 'col_ptr'
30     col_ptr(j) = index;
31
32     for (unsigned int i = 0; i < m; ++i) { // row iterator
33         if (A(i, j) != 0) {
34             // record the value to 'val'
35             val(index) = A(i, j);
36             // record row index to 'row_ind'
37             row_ind(index) = i;
38             ++index;
39         }
40     }
41 }
42 // END
43 }

```

Get it on  GitLab ([sparseCCS.hpp](#)).



(2-14.b)  (5 min.) [depends on Sub-problem (2-14.a)]

What is the computational complexity of your CCS function:

- with respect to the matrix size n , where $\mathbf{A} \in \mathbb{R}^{n \times n}$?
- with respect to nnz , which denotes the number of nonzero elements in $\mathbf{A}$?

SOLUTION of (2-14.b):

The algorithm needs to check whether each element $a_{i,j}$ of the $n \times n$ matrix is nonzero. The computational complexity is $O(n^2)$, given the double loop to build `val` and `row_ind`. Forming `col_ptr` is only $O(n)$ and does not therefore change the global complexity $O(n^2)$.

The complexity does not depend on the number nnz of nonzero elements in $\mathbf{A}$, since we have to iterate throughout all elements in any case.



End Problem 2-14 , 35 min.