LINMA2710 – Scientific Computing – 2022-2023

Project 1

Deadline: Friday 10 March 2023 18:15 (Week 5)

Welcome to the Scientific Computing course!

This document presents the first of three projects. The goal of the projects will be to familiarize yourselves with some classical tools of scientific computing.

From our experience it is possible that some of you encounter initial difficulties due to less coding experience. Also from our experience, these should not stop you from succeeding in doing the projects. Getting rapidly acquainted with the tools can usually be done by looking at straight-to-the-point exemples.

At any point if you have a question about the projects or the tools you can reach us: Hazan Daglayan and Brieuc Pinon; through Teams, by email and also by the moodle Q&A forum[1]. We will also organize help sessions for the project during the quadrimester.

1 Statement

In this project, you will code an implementation for sparse vectors and sparse matrices in a Compressed Sparse Column format (CSC) using the `C++` language.

To help you, several parts of codes are provided:

- all the code files, so you know which files you are expected to complete;
- a `Vector` class [2];
- an `AbstractVector` abstract class;
- a `test.cpp` to launch tests on your code, these tests will test all the functions you have to implement but they do not directly test memory management (see in the Instructions);
- a `Makefile` to compile everything and run the tests.

On this base, you have to code the `SparseVector` and `SparseMatrix` classes.

The `SparseVector` class should implement:

- the `AbstractVector` abstract class;
- an empty constructor `SparseVector()` and a destructor `~SparseVector()`;
- a constructor `SparseVector(int nnz, int const *rowidx, double const *nzval, int size)` for sparse vector of dimension `size` and `nnz` non-zero entries specified by `rowidx` and `nzval` (see Section 2);
- an assignation operator `SparseVector& operator=(const SparseVector& otherVector)`, it should **copy** the data in the object;
- a function with signature `SparseVector operator+(const SparseVector& v1) const` that performs the element-wise addition between two sparse vectors with a worst-case computational complexity in $O(nnz_1 + nnz_2)$flops where nnz_1, nnz_2 are the respective number of non-zero elements in the vectors.

The `SparseMatrix` class should implement the following member functions:

- a constructor `SparseMatrix(int size1, int size2)` for zero sparse matrix of `size1` rows and `size2` columns;

[1] It is forbidden to contact both of us without letting each other know about it. So set one or both of us in an email but not one email for each of us.

[2] taken from Listings 10.1 and 10.2 of Pitt-Francis & Witheley (http://dx.doi.org/10.1007/978-1-4471-2736-9, freely available from the UCLouvain network)

- a constructor for deep copy `SparseMatrix(int m, int n, int nnz, int* colptr, int* rowidx, double* nzval)`;
- a constructor `SparseMatrix(int nnz, int const *ridx, int const *cidx, double const *nzval, int size1, int size2)` for sparse matrix of `size1` rows and `size2` columns and `nnz` entries specified by `nzval`, `ridx` and `cidx` which respectively contains the non-zero values, the row indices and column indices of those values. Note that there are no conditions on the order of these elements;
- a destructor `~SparseMatrix()`;
- an accessor[3] `int GetSize(int)` where `GetSize(1)` returns the number of rows and `GetSize(2)` returns the number of columns;
- the operator `=` with another `SparseMatrix`;
- the unary `+`, unary `-` and scalar multiplication operators;

We also ask you to implement the following additional operation function:

- The `*` operator between a `SparseMatrix` in the left-hand side and a `Vector` in the right-hand side, returning the multiplication result in a `Vector`. Its worst-case computational complexity should be in $O(\max(n_1, n_2, nnz))$ where n_1/n_2 is the number of rows/columns in the matrix and nnz its number of non-zero values.

2 Background

2.1 Sparse Vectors

The sparse format for the vector stores two arrays: one for the nonzero values and one for the corresponding indices. For instance, the vector $[0, \pi, 0, 0, 0, 1.2, 0, 0.8, 0]$ is stored with an array $[\pi, 1.2, 0.8]$ of nonzero values and an array $[1, 5, 7]$ of corresponding indices.

2.2 Sparse Matrices and the compressed sparse column format (CSC)

In the CSC format, the row indices (resp. values) of the nonzero entries are stored column by column, sorted in ascending row index.

For instance, for the matrix

$$\begin{bmatrix} 2.1 & 3.5 & & 1.9 \\ & 2.2 & & \\ 1.0 & & 2.3 & \\ & 2.0 & & \end{bmatrix},$$

the row indices (resp. values) for each column are $[1, 3]$ (resp. $[2.1, 1.0]$), $[1, 2, 4]$ (resp. $[3.5, 2.2, 2.0]$), $[3]$ (resp. $[2.3]$) and $[1]$ (resp. $[1.9]$).

The row indices (resp. values) of the different columns are concatenated consecutively in an array `rowidx` (resp. `nzval`). A third array `colptr` store the indices where the different columns start and end in `rowidx` and `nzval`. In those arrays the row indices and values of column `j` are located between index `colptr[j]` (included) and `colptr[j+1]` (excluded).

Continuing our example, we have:

$$\begin{aligned} \texttt{colptr} &= [0, 2, 5, 6, 7] \\ \texttt{rowidx} &= [1, 3, 1, 2, 4, 3, 1] \\ \texttt{nzval} &= [2.1, 1.0, 3.5, 2.2, 2.0, 2.3, 1.9] \end{aligned}$$

Note that the first element of `colptr` is always 0 and the last element is always equal to the number of nonzero entries in the matrix.

[3] *Accessors* are methods that can be used to access the value of a private object member.

Instructions

1. **Fraud:** As always for this course, you must do all the writing (code, report, user manual...) individually. Never share your production. Your codes will be checked against fraud. However, you are allowed, and even encouraged, to exchange ideas on how to address the assignment.
2. **Plagiarism:** As always, you must cite all your sources.
3. **Report Submission:** Using the Moodle assignment activity, submit your report in a file called `Report_Project 1_FirstName LastName.pdf`.

   The report should be short (maximum 2 pages) and (only) contain:

   - A computational complexity analysis of your implementations of: `+` between two `SparseVector`; and `*` between a `SparseMatrix` and a `Vector`.

     By computational complexity we mean asymptotic time and space-complexity; both should respect the upper-bounds given the statement. We will make the simplifying assumption that a memory allocation is done in time $O(1)$.

     While this assumption is strong, the computational analysis is "robust" to this choice since we ask for both time and space-complexity to follow the given bounds.
   - A clear description of the memory states during the call to the "=" operator of `SparseVector`. More concretely, you should describe what the different variables at the input and output point to in memory.

4. **Code Submission:** On Inginious[4], submit your files `SparseVector.cpp` and `SparseVector.hpp` containing the implementation of the `SparseVector` class as well as `SparseMatrix.cpp` and `SparseMatrix.hpp` containing the implementation of the `SparseMatrix` class and the additional operations and functions.

   You are allowed to make as many submissions as you need, only the last submission will be taken into account.

   You are advised to verify that your submission passes the tests in Inginious early before the deadline.

   On Inginious, the files `test.cpp`, `Vector.cpp`, `Vector.hpp` `AbstractVector.cpp` and `AbstractVector.hpp` available on Moodle are compiled together with your submitted codes and results of the tests included on `test.cpp` are checked.

   As mentionned, the tests on Inginious are exactly those included in `test.cpp`. It is not sufficient to pass these tests to get a full grade. We will also check errors in memory management with `Valgrind`. These errors could be silent (not properly freeing memory, accessing unitialized values,...). We strongly advice you to run `Valgrind` on your code.

   Note that it is forbidden to bypass the described CSC format.
5. **Language:** Reports in French are accepted without penalty. However, English is strongly encouraged. The quality of the English will not impact the grade, provided that the text is intelligible.
6. **AI assisted coding** Artificial Intelligence tools for programmers are autorized (CODEX, ChatGPT, ...) but you should mention their use in the report.

Advices

Editing, compiling and running You will have to choose how to code: with an IDE, a text editor, a terminal? Something to take into account in your decision is the fact that in the third project we will ask you to connect through `ssh` to a computing facility running on a `Linux` distribution. This operation will be easier with a terminal.

[4] https://inginious.info.ucl.ac.be/course/LINMA2710/assignment1

For this reason, we encourage you to familiarize yourselves to compiling and running code through the terminal. To edit the code take something that suits your preferences VS Code, Atom, Emacs, ...

For `Windows` users to get a `Linux` terminal, check out `Windows Subsystem for Linux`.

Checking and debugging We will check your work with `Valgrind`, and you should do it too. It is a free of use memory checker. It will run your code and check memory operation against several usual bugs (e.g. uses of uninitialised values). This usually allows to catch some errors that could be invisible on classic tests, or to catch the roots of a visible error earlier in the execution (there is sometimes a big discrepancy between where an error start and when it becomes visible). You will remark that it is a powerful debugging tool, when an error is detected you can usually pass an option for it to give you more information about the problem.

For more information check the troubleshooting guide written by Benoît available on the moodle. There are also good ressources on the internet.

Note that if you don't have `Linux` (or `Windows Subsystem for Linux`), you may have trouble installing it. If that is the case we can connect you to a university server that run on Linux. Note that connecting to this server will in any case be mandatory for the third project.