Assignment 1: Sparse matrices

Implement a class `SparseMatrix`. The class should be used to store sparse matrices in row major format. Store the matrix as two `vectors` of `vectors`. One `vector` of `vectors` is of type double and stores our entries, each inner vector represents a row of the matrix. A second vector of vectors is of type int and contains the column indexes of those entries. This is reasonably efficient since the vectors representing different rows need not be of the same length. If we simply allocated a rectangular array for this data, then unless by happenstance all the rows had the same number of non-zero elements, some memory would be wasted.

Define appropriate member data to store the representation. In either case use the STL container `vector <double>` to store the matrix entries and the STL container `vector<int>` to store any necessary matrix indexes.

Implement a member method of the class `SparseMatrix` called `addEntry`. It should accept a `double` for a new entry in the matrix, as well as two `int`s to specify the new entry's location. Use this function later to populate your matrix with the example problem.

Iterative solver Implement a member method of the class `SparseMatrix` called `GaussSeidel` which will be used to invert the matrix with the Gauss-Seidel method (you can refer to the lecture notes, to the suggested books, Wikipedia or any other resource). Briefly explain the algorithm and your implementation. The method should accept as input an initial guess for the answer $x^{(0)}$ and the vector to be inverted against b. It should explicitly or implicitly return the number of iterations required to find a solution (if it can), the final residual error (in L_∞ norm) and of course the computed approximate solution itself. Generic vectors may be simply of type `vector<double>`.

Add sufficient conditions to your implementation, that if an inverse cannot be found (for whatever reason), or if the method has stagnated, that the program exits with an appropriate warning to the user.

Testing Test the implementation by solving the system $Ax = b$ with $A \in \mathbb{R}^{N\times N}$ use as starting point for the iteration $x_0 = 0$ and a tolerance of $\epsilon = 1e-6$.

For our test we define the vector $w_i = \frac{i+1}{N+1}$ and define the tridiagonal matrix $(i, j = 0, \ldots, N-1)$

$$a_{ij} = \begin{cases} -D_{i-1} & j = i-1 \\ D_i + D_{i-1} & j = i \\ -D_i & j = i+1 \\ 0 & \text{otherwise} \end{cases}$$

with $D_i = a(w_i - \frac{1}{2})^2 + \delta$ with constants $a = 4(1-\delta)$, $\delta > 0$. Set $D_{-1} = D_0$. The right hand side is given by $b_i = -2a(w_i - \frac{1}{2})w_0^2$ for $i = 0, \ldots, N-2$ and $b_{N-1} = -2a(w_i - \frac{1}{2})w_0^2 + 1$.

Do a first test run of your program with $\delta = 1$ (then all $D_i = 1$ and all $b_i = 0$ except for the last entry). In this case the exact solution is given by $x = (w_i)_{i=0}^{N-1}$. Show that your method works for $N = 100, 1000, 10000$ by outputting the error between the exact solution and the result of the iterative scheme in the maximum norm.

Try varying δ and measure the residual error at every iteration, use gnuplot, matplotlib or MATLAB to plot it.

Consider another matrix $B = A + \lambda I$, where $\lambda \in \mathbb{R}^+$. Try inverting it for some different λ and again measure the residual. Can you explain the varying performance of the algorithm with respect to the change in δ and λ?

Submission requirements

You should submit your **code** together with a **report** that discusses the problem setup, underlying theory, all tests you performed, answers the questions outlined in the assignments, and describes your observations. Remember that your code should include some **comments** to make it readable.

Prepare a **folder** with all your files related to the assignment such that everything you have done can be reproduced, including the plotting and the report LaTeXcompilation. You have to provide a potential user with a simple way to reproduce your results. Provide a `README` with instructions about how to generate your report. Please also put a **separate pdf version** of your report into this folder, so that marking can be done even if the reproduction should fail.

Combine all files together into a single **archive**, the filename of which is `u123123123-assignment1.zip` (or other ending), where the first part should be replaced by your **user-id**. Then upload the archive file to

https://files.warwick.ac.uk/ma913_2018/sendto

Please let us know in the covering note for the upload the amount of time you have spent on this assignment.

Deadline: Wed 7 November 2018, 6:00pm

Appendix: Techniques

You can manage submission by providing us a Makefile, a bash script, some other scripts or a combination thereof. You can build on the `Makefile` discussed in the exercises of the `C++` course if you want or create your own setup.

The simplest solution is probably to create a bash script (just a text file named e.g. `run.sh`) that looks like this:

```
!#/usr/bin/bash
# COMPILE COMMANDS
# examples of compilation of two programs with external headers/libraries
g++ -I INCLUDEPATH -L LIBRARYPATH -lmylib code1.cc -o program1
g++ -I INCLUDEPATH -L LIBRARYPATH -lmylib code2.cc -o program2

# RUN COMMANDS
# examples of different runs with different parameters
./program1 parameter_set1
./program1 parameter_set2
./program2 parameter_set1
./program2 parameter_set2
# parameters_set can be a list of parameters or a text file
# with the list of parameters to be read by the code

# POSTPROCESSING
gnuplot < 'load gnuplot_script.plot'                # gnuplot
python matplotlib_script.py                         # matplotlib
matlab -nojvm -nodisplay -nosplash matlab_script.m # matlab

# LATEX REPORT
pdflatex report.tex
pdflatex report.tex
```

then make it executable with `chmod u+x run.sh`. If properly written, this file should be able to automatically compile, run the code and produce the results. Nevertheless, please include also the figures and the PDF, just in case reproduction is failing. The figures in the report can be visually different from the ones produced by the script file provided but should contain exactly the same data.

Upload all your files including this running script. If you have used external libraries, you do not need to include them but you should provide within your `README` file information on the libraries you used and how you compiled and used them.

Combine all files together into a single file, e.g., like this `tgz u123123123-assignment1.tgz folder` where you use your University ID number and `folder` stands for the folder to be tar-zipped. You can also zip it (using the `zip` command) or gzip it (using the `gzip` command).