would use the explicit matrix. However, the trick here is to use A in a "virtual" form, so we need only $\mathcal{O}(n)$ memory.

Task 7 (10 points) Write a program in C++ that solves a linear system $A\boldsymbol{x} = \boldsymbol{b}$ by using Jacobi and Gauss-Seidel method. You can assume that A is invertible and hence the system has a unique solution x^*.
For a benchmark problem, we define the $n \times n$- matrix

$$A = \begin{pmatrix} +2^0 & -2^{-2} & -2^{-4} & -2^{-8} & \dots & -2^{-2^{n-1}} \\ -2^{-2} & +2^0 & -2^{-2} & -2^{-4} & \dots & -2^{-2^{n-2}} \\ -2^{-4} & -2^{-2} & +2^0 & -2^{-2} & \dots & -2^{-2^{n-3}} \\ -2^{-8} & -2^{-4} & -2^{-2} & +2^0 & \dots & -2^{-2^{n-4}} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ -2^{-2^{n-1}} & -2^{-2^{n-2}} & -2^{-2^{n-3}} & -2^{-2^{n-4}} & & +2^0 \end{pmatrix}$$

and the right hand side $\boldsymbol{b} = [1, ..., 1]^\top$. (This matrix is well suited for testing Gauß-Seidel and Jacobi method, respectively.)
Solve the linear system of equations $A\boldsymbol{x} = \boldsymbol{b}$ with the Jacobi method and the Gauß-Seidel method.
As a stopping criteria for the algorithms use the norm of the residual $r = \|\boldsymbol{b} - A\boldsymbol{x}\|_2^2 < tol$.
Version A Use a matrix of fix size for storing A. Use pointers for the matrix variable.
Version B Try to manage to never define explicitly the matrix A. You do not need pointers in this version.
Implement **one** of the variants.
The **only allowed include** is iostream.h except self-defined header for your own functions, i.e if you need functions like "power()" or "abs()", write your own functions for power, abs, etc.
You can assume that A given here fulfills the requirements of the Gauß-Seidel and Jacobian methods to converge. However, feel free to implement a test for (3) or to check symmetry.
Your main program should allow the use to set the problem size n (at runtime!) as well as set tol and $maxiter$. You program should print the respective number of iterations necessary to obtain the required residual. If the solver do not converges within $maxiter$ iterations, a message should be given.
Feel free to extend you program by an output of the solution x^* , the matrix A and the vector Ax^*.
We expect that you program is wells structured, i.e. it is divided in sub-function, implemented in separate file(s).
Test you program for $n = 5000$ and $tol = 0.001$.

Hints You have to use the explicit form of the iteration given by

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \dots, n$$

for the Jacobi method and

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \dots, n$$

for the Gauß-Seidel method, respectively. Do **not use matrix-vector operations** from any libraries.

To implement Version B, use the simple idea that e.g. in

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \dots, n$$

the mathematical term a_{ij} can be interpreted (in C++ notation) as $a[i, j]$ **or** $a(i, j)$, respectively, where the first case is calling the entries of a variable a representing the entries a_{ij} of the Matrix A, and in the second case it is the call of a function returning the values of the matrix elements a_{ij} of A. The difference between variant A and variant B is that variant A physically creates the matrix in memory, but variant B creates the entries of the matrix when the algorithm accesses them.

Note that plotting the entire matrix/vector on console, independently from its size, may be a bad idea. (In the case $n = 5000$,the matrix has 25.000.000 entries.)

In C++, to access the vectors $\boldsymbol{x}$ and $\boldsymbol{b}$ (and the matrix A, if you chose to use it explicitly), one need pointers to change their values by using functions.