CS319: Scientific Computing (with C++)

Niall Madden (Niall.Madden@NUIGalway.ie)

CS319 Lab 7: Solving Linear Systems II

13 April 2021

Goal: to develop expertise in *operator overloading* and to demonstrate this by developing a new implementation of the Gauss-Seidel method from Lab 6.

Deadline: 5pm, Tuesday 27 April

You should upload a *single archive file*, such as a `zip` or `tar`-ball, that contains all the necessary source files.
Don't forget to include your name, ID number, and NUI Galway Email address in all files!

Part 1: Recall Jacobi's method

In Lab 6 you developed implementations of the Jacobi and Gauss-Seidel algorithms for solving a linear system of N equations in N unknowns: *find* $x_1, x_2, \ldots, x_N$, *such that*

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N &= b_2 \\ &\vdots \\ a_{N1}x_1 + a_{N2}x_2 + \cdots + a_{NN}x_N &= b_N. \end{aligned}$$

We expressed this as a matrix-vector equation: *Find* $\mathbf{x}$ *such that*

$$A\mathbf{x} = \mathbf{b},$$

where A *is a* $N \times N$ *matrix, and* $\mathbf{b}$ *and* $\mathbf{x}$ *are (column) vectors with* N *entries.*

Part 1: Recall Jacobi's method

Then **Jacobi's method** is: choose $\mathbf{x}^{(0)}$ and set

$$x_1^{(k+1)} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \cdots - a_{1N}x_N^{(k)})$$
$$x_2^{(k+1)} = \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} - \cdots - a_{2N}x_N^{(k)})$$
$$\vdots$$
$$x_N^{(k+1)} = \frac{1}{a_{NN}}(b_N - a_{N1}x_1^{(k)} - \cdots - a_{N,N-1}x_{N-1}^{(k)})$$

There is also a matrix-version of this iteration. We set D and T to be the matrices

$$d_{ij} = \begin{cases} a_{ii} & i = j \\ 0 & \text{otherwise.} \end{cases} \qquad t_{ij} = \begin{cases} 0 & i = j \\ -a_{ij} & \text{otherwise.} \end{cases}$$

So $A = D - T$. Then *Jacobi's method* can be written neatly in matrix form:

$$x^{(k+1)} = D^{-1}(b + Tx^{(k)}). \tag{1}$$

Part 1: Recall Jacobi's method

In Week 8, and this week, we moved towards a neater implementation, based on operator overloading.

- We overloaded the assignment operator `=` for `Vector`s.
- We overloaded the `Vector` addition operator.
- We overloaded the `Matrix`-`Vector` multiplication operator.

Our next step is to implement Jacobi's method with overloaded versions of the vector addition operator, +, and multiplication operator, *, for matrices and vectors, and in just a few lines:

```
do
{
    count++;
    x = Dinv*(b+T*x);
    r=b-A*x; // set r=b-A*r
}   while ( r.norm() > tol);
```

This is implemented in the program `RunJacobi.cpp`. Download it, from https://bitbucket.org/niallmadden/2021-cs319/src/master/lab7/ and try it. You will need `Matrix09.h`, `Vector09.h`, `Matrix09.cpp` and `Vector09.cpp` from Week 9.

Part 2: Triangular systems

Some systems of equations are very easier to solve than others. Suppose the system is $L\mathbf{x} = \mathbf{b}$, but L is a lower triangular matrix. The associated system of equations looks like this:

$$\begin{aligned} l_{11}x_1 &= b_1 \\ l_{21}x_1 + l_{22}x_2 &= b_2 \\ l_{31}x_1 + l_{32}x_2 + l_{33}x_3 &= b_2 \\ &\vdots \\ l_{N1}x_1 + l_{N2}x_2 + \cdots + l_{NN}x_N &= b_N. \end{aligned}$$

To solve this,

- first set $x_1 = b_1/l_{11}$.
- Now substitute this into the second equation to get $x_2 = (b_2 - l_{21}x_1)/l_{22}$.
- Next we use $x_3 = (b_3 - l_{31}x_1 - l_{32}x_2)/l_{33}$,
- and so on.

In fact, this is quite like Jacobi's method, except we don't have to iterate.

Part 2: Triangular systems

Since we write $L\mathbf{x} = \mathbf{b}$, it is reasonable to write $\mathbf{x} = \mathbf{b}/L$.

Exercise

Overload the "/" operator so that, if L is **lower** *triangular, then* **x** *is computed as outlined above.*

We will make this operator a `friend` of the `matrix` class, meaning that it is not a member of the class, but is "known" to it.
Modify the `Matrix09.h` header file to include the following function prototype in the class definition:

```
friend vector operator/(vector u, matrix L);
```

Note that we are explicitly passing both arguments.

Then, in the Matrix09.cpp file, add the code for the operator function. The first line might be

```
vector operator/(vector b, matrix L){
   int N = L.size();
   vector x(N); // x solves L*x=b
.
. // you add the rest of the code here
.
}
```

Important: the friend keyword appears only in the function prototype, and not in the function definition itself.

Part 3: Gauss-Seidel, again

Recall that the **Gauss-Seidel method** is choose $\mathbf{x}^{(0)}$ and set

$$\begin{aligned}
x_1^{(k+1)} &= \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(k+1)} - a_{13}x_3^{(k)} - \cdots - a_{1N}x_N^{(k)}) \\
x_2^{(k+1)} &= \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k+1)} - \cdots - a_{2N}x_N^{(k)}) \\
&\vdots \\
x_N^{(k+1)} &= \frac{1}{a_{NN}}(b_N - a_{N1}x_1^{(k+1)} - \cdots - a_{N,N-1}x_{N-1}^{(k+1)})
\end{aligned}$$

In the same way as we did for Jacobi's method, we can write this in a succinct matrix-vector form: we set L and U to be the matrices

$$l_{ij} = \begin{cases} a_{ij} & i \geq j \\ 0 & \text{otherwise.} \end{cases} \qquad u_{ij} = \begin{cases} 0 & i \geq j \\ -a_{ij} & \text{otherwise.} \end{cases}$$

So $A = L - U$. Then the *Gauss-Seidel method* can be written as

$$Lx^{(k+1)} = b + Ux^{(k)}. \tag{2}$$

Note that this involves solving a linear system where L is the coefficient matrix. However, we have overloaded the "/" operator to do just that.

Assignment

Write a programme, based on `RunJacobi.cpp`. It should achieve all of the following.

1. Use both the Jacobi and Gauss-Seidel methods to solve the linear system;
2. Implement the Gauss-Seidel method using your overloaded "/" operator;
3. Verify that the Gauss-Seidel method is more efficient (assuming they both converge);
4. Allows the user to specify the convergence tolerance for the residual (i.e., stop when $\|b - Ax\| \leq TOL$);
5. Allows the user to specify the maximum number of iterations to use.

Submit your solution through the "`Lab 7`" section of the Blackboard module.
You should include all necessary files for your program to compile: *even if they are unchanged from the versions your downloaded from the website*. Including the project file or, even better, a Makefile, is helpful, but not necessary.
You should upload a *single archive file*, such as a `zip` or `tar`-ball, that contains all the necessary source files.
Don't forget to include your name, ID number, and NUI Galway Email address in all files!
Deadline: **5pm, Tuesday 27 April**