CS319: Scientific Computing (with C++)

CS319 Lab 7: Linear Systems 1

Week 9 (13+14 March, 2025)

Goal: write functions that implement the Jacobi and Gauss-Seidel methods, and compares the results.

Deadline: None. We'll develop this more in Lab 8, using `Matrix` and `Vector` objects.

1. Jacobi's Method

We'll start by studying Jacobi's method for solving a linear system of equations. This was discussed very briefly at the end of Wednesday's 4pm class.

Here is the idea in more detail.

We want to solve the problem: *find* $x_1, x_2, \ldots, x_N$*, such that*

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

1. Jacobi's Method

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

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

This can be programmed with two (or so) nested `for` loops.

1. Jacobi's Method

An implementation is given in
https://www.niallmadden.ie/2425-CS319/lab7/Jacobi-Lab7.cpp

It works as follows:

- The two-dimensional array `A` stores the coefficients for the left-hand side.
- The one-dimensional arrays `x` and `b` stores the true solution and left-hand side, respectively.
- The one-dimensional arrays `xk` and `xk1` represent the vectors $x^{(k)}$ and $x^{(k+1)}$.
- It sets `A` and `b` to represent the problem

$$9x_1 + 3x_2 + 3x_3 = 15 \tag{1}$$
$$3x_1 + 9x_2 + 3x_3 = 15 \tag{2}$$
$$3x_1 + 3x_2 + 9x_3 = 15 \tag{3}$$

The true solution is $x_1 = x_2 = x_3 = 1$.

1. Jacobi's Method

- Five iterations of the Jacobi method are taken.
- The estimated solution after five iterations is outputted.

Make the following improvements to the code for the Jacobi method.

- Add a function with header
  `double norm(double *x, unsigned int N);`
  that returns the vector 2-norm (i.e., square root of the sum of the squares) of the entries in the array `x` which has `N` entries.
- Add a function with header
  `double diff(double *x1, double *x2, unsigned int N);`
  that returns the vector of `v=x1-x2`.
- Add a function with header
  `void Jacobi(double **A, double *b, double *xk, unsigned int N, unsigned int &count, unsigned int MaxIts, double TIL);`
  that estimates the solution to `A*xk=b`, such that
  - `xk` is the initial guess for the method, and also the final estimate.

- It each iteration it computes the *residual*: $R = b - Ax^{(k)}$. Note that, if $x^{(k)}$ is the true solution, the norm of R is zero. If it is "small" then it is likely that $x^{(k)}$ is a good estimate for x.
- It performs iterations until `norm(R)<TOL`, or until the number of iterations exceeds `MaxIts`.
- `count` stores the number of iterations taken.

▶ Verify that the function your `Jacobi` function works. In the `main()` output the estimate it computes, the difference between `x` and `xk`, and the number of iterations taken.

2. Gauss-Seidel

Jacobi's method is not particularly efficient. Heuristically, you argue that it could be improved as follows. In Jacobi's method, we compute $x_1^{(k+1)}$ from

$$x_1^{(k+1)} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \cdots - a_{1N}x_N^{(k)})$$

We expect that it is a better estimate for x_1 than $x_1^{(k)}$.

Next we compute

$$x_2^{(k+1)} = \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} - \cdots - a_{2N}x_N^{(k)})$$

However, here we used the "old" value $x_1^{(k)}$ even though we already know the new, improved $x_1^{(k+1)}$. That is, we could use

$$x_2^{(k+1)} = \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \cdots - a_{2N}x_N^{(k)})$$

More generally, in Jacobi's method we set

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\big(b_i - \sum_{j=1, j\neq i}^{N} a_{ij} x_j^{(k)}\big).$$

The Gauss-Seidel method uses

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\big(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{N} a_{ij} x_j^{(k)}\big).$$

Implement this method as new function called `GaussSeidel`. Verify that it is more efficient than the Jacobi method, in the sense that fewer iterations are required to achieve the same level of accuracy.