CS319 – Scientific Computing

Lab 5: Vectors and Matrices I

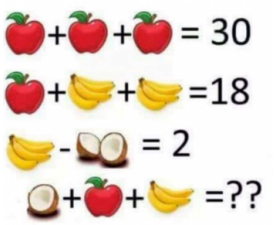
Goal: to develop some proficiency in working with the matrix and vector classes from Weeks 7, and to look forward to *operator overloading*.

1 Recall... the `vector` and `matrix` classes

In Week 7, we developed classes to represent matrices and vectors, and used these to implement Jacobi's method for solving a linear system of equations.

That the sample code is split into several source files. You'll have to work out how to combine these into a single project.

To verify that you understand how the code works, adapt the `03Jacobi.cpp` program it to solve the following problem:



2 Taking one step backwards...

Jacobi's method is designed to solve 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) vector with* N *entries.*

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

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

where $D = \mathrm{diag}(A)$ and $A = D - T$.

This gave a neat presentation of the method in terms of matrix-vector multiplication. However, a simpler, if less elegant presentation is possible: 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, rather than explicit matrix multiplication.

Implement this version that uses `for` loops, and verify that you get the same results as with the original `Jacobi` solver. (Remember: mathematically, we usually index vectors and matrices from 1 to N, but in C++ we index them from 0 to $N-1$).

3 ... and one step forwards

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

More generally, in Jacobi's method we set

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

The Gauss-Seidel method uses

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

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.

4 Exercises

You don't have to submit any work this week. Nonetheless, you should complete the following exercises before next week.

1. When we test our code on the "apple-banana-coconut" problem, Jacobi's method converges to the true solution in 3 iterations. Why?

2. Produce a working program that implements the Jacobi and Gauss-Seidel methods.

3. Test these functions for problems of at least size $N = 10$. According to the theory, these methods are guaranteed to converge if the system matrix is *diagonally dominant*. This means that, in each row, the magnitude of the diagonally entry, $|a_{ii}|$, must be greater than the sum of the other terms: i.e.,
$$|a_{ii}| < \sum_{j \neq i} |a_{ij}|.$$
Keep this in mind when you are choosing a test problem.

   Ideally, your programme should

   - Generate a random matrix that is diagonally dominant.
   - Choose a simple solution vector $\mathbf{x}$, e.g., $x = (1, 0, 1, 0, 1, \dots)$.
   - Set $\mathbf{b} = A\mathbf{x}$
   - Use your Gauss-Seidel solver to compute and estimate for $\mathbf{x}$.
   - Compute how close this estimate is to the true solution.

4. It is possible to write the Gauss-Seidel method in a neat matrix-vector form as in 1. What is this form, and explain why it does not have such an easy implementation?

5. Even though most books present the Gauss-Seidel method as an improvement upon Jacobi's, it is actually the (slightly) older of the two methods (it is from 1826, whereas Jacobi's is from 1845). Jacobi's preferred his slower method, however, because it is easier to parallelise. Explain why this is the case.