Lecture Notes, Math 170A, Winter 2020

Chapter 1.3: Forward and backward substitution

We will now start talking about how one may approach solving linear systems; one of the simplest systems to solve is a triangular one, so called because the 0-structure of the matrix makes it look like a triangle (if all entries above the main diagonal are 0, we call it lower-triangular, and if all entries below the main diagonal are 0, we call it upper-triangular).

Let $Gy = b$ be an example of a lower-triangular system (the awful notation simply reflects the textbook notation); the system looks like

$$\begin{aligned} g_{11}y_1 \qquad\qquad\qquad\qquad &= b_1 \\ g_{12}y_1 + g_{22}y_2 \qquad\qquad &= b_2 \\ &\vdots \\ g_{n1}y_1 + g_{n2}y_2 + \ldots + g_{nn}y_n &= b_n \ . \end{aligned}$$

The obvious strategy is to do <u>*forward substitution*</u>; solve for y_1 from the first equation, substitute into the second equation and solve for y_2, then substitute the values you have found for y_1, y_2 in the third equation and solve for y_3, and continue this way down until you solved all the equations.

This relies on the following formula for y_i, with $i = 1, 2, \ldots, n$, which can be evaluated sequentially from 1 through n, using the values we have already found:

$$y_i = \frac{b_i - g_{i1}y_1 - g_{i2}y_2 - \ldots - g_{i(i-1)}y_{i-1}}{g_{ii}} \ .$$

A simple way to code forward substitution into MATLAB can be seen below.

```
function y = lowtriangsolve(G,b);
y = b;
for i = 1 : n
        for j = 1 : (i − 1)
              y_i = y_i − g_ij y_j;
        end
        if g_ii = 0, error('matrix is singular'), end
        y_i = y_i/g_ii;
end
```

<u>NOTE:</u> recall that the determinant of a triangular matrix is the product of the diagonal entries (Exercise: try to think why). So if the matrix is non-singular, none of the diagonal entries can be 0. Conversely, if any diagonal entry is 0, the matrix must be singular (and the system cannot be uniquely solved).

Often, once y is found, b is no longer needed and so it is overwritten. In the textbook, the pseudocode algorithms for triangular solve overwrite b. This helps with space-saving if the matrices are huge, but it will not impact things much for the kinds of matrices we will be dealing with in this class.

Flop count. Consider the floating point operations (a.k.a. *flop*) $(+, -, *, /)$ count for forward sustitution. The inner loop effectuates 2 flops each time it is run, and it is run $(i-1)$ times, for a total of $2i-2$ flops. The division at the end adds one more flop. So for each i from 1 to n, the work done inside the first for loop is proportional to $2i-1$.

The flop count is

$$\sum_{i=1}^{n}(2i-1) = 2\sum_{i=1}^{n} i - \sum_{i=1}^{n} 1 = n(n+1) - n = n^2 \ .$$

Each time we double n, the algorithm will take 4 times as long to run.

Backward Substitution. So far we have seen how to solve lower triangular systems, but what if the system is upper triangular, i.e., looks like

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + \ldots + u_{1n}x_n &= b_1 \\ u_{22}x_2 + \ldots + u_{2n}x_n &= b_2 \\ &\vdots \\ u_{nn}x_n &= b_n \ , \end{aligned}$$

how does the strategy change? Answer: we start from the bottom, from x_n, and work our way backwards to x_1. All the rest is similar; the strategy is named *backward substitution.* You can work out the new algorithm on your own; it is very similar to the old one and you should expect to see the same flop count as before.