EMTH211-19S2 Laboratory 2 Solutions

July 22-26, 2019

2.1 Consider the system

$$\begin{aligned} 100x_1 + 1000000x_2 &= 1000000 \\ -100x_1 + 200x_2 &= 100 \end{aligned} \tag{1}$$

(a) Find the solution (with only partial pivoting) to this system using 3-digit arithmetic.

(b) Rewrite this system as

$$\begin{aligned} 1000000y_1 + 100y_2 &= 1000000 \\ 200y_1 - 100y_2 &= 100 \end{aligned}$$

Solve this system using 3-digit arithmetic (note that this is equivalent to complete pivoting).

(c) Solve (1) using 5-digit arithmetic. What is the minimum number of digits needed to get an accurate answer?

(d) Modify (1) so that 5-digit arithmetic will not give an accurate answer using only partial pivoting.

(e) Modify (1) so that 16-digit arithmetic will not give an accurate answer using only partial pivoting (this is roughly the default double-precision in MATLAB).

Solution:

(a) We have

$$\left[\begin{array}{cc|c} 100 & 1000000 & 1000000 \\ -100 & 200 & 100 \end{array}\right] \xrightarrow{\text{3-digit}} \left[\begin{array}{cc|c} 100 & 1000000 & 1000000 \\ 0 & 1000000 & 1000000 \end{array}\right]$$

and so

$$\mathbf{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

(b) In this case

$$\left[\begin{array}{cc|c} 1000000 & 100 & 1000000 \\ 200 & -100 & 100 \end{array}\right] \xrightarrow{\text{3-digit}} \left[\begin{array}{cc|c} 1000000 & 100 & 1000000 \\ 0 & -100 & 100 \end{array}\right]$$

and so

$$\mathbf{y} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Note that $y_2 = x_1$ in (1).

(c) We have

$$\left[\begin{array}{cc|c} 100 & 1000000 & 1000000 \\ -100 & 200 & 100 \end{array}\right] \xrightarrow{\text{5-digit}} \left[\begin{array}{cc|c} 100 & 1000000 & 1000000 \\ 0 & 1000200 & 1000100 \end{array}\right]$$

and so

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0.99990 \end{bmatrix}.$$

We need at least 5-digits.

(d)

$$
\begin{aligned}
10x_1 + 1000000x_2 &= 1000000 \\
-10x_1 + 20x_2 &= 10
\end{aligned}
$$

(e)

$$
\begin{aligned}
x_1 + 10^{17}x_2 &= 10^{17} \\
-x_1 + 2x_2 &= 1
\end{aligned}
$$

2.2 In the lectures, it was shown that to solve a system of equations by Gaussian elimination requires

$$\mathcal{O}(\tfrac{2}{3}\,n^3)$$

flops. This refers to a *generic* or full matrix. Some special forms are much much faster to solve. Compute the flop count for solving a system of equations whose coefficient matrix is

(a) diagonal

(b) upper triangular

(c) lower triangular

SOLUTION:

(a) n.

(b) This requires back substitution only. Therefore $\mathcal{O}(n^2)$.

(c) This is the same as case (b) (only the order is changed) and so $\mathcal{O}(n^2)$.

2.3 In Lab 1, you solved a *tridiagonal* system. You should have noticed that the speed at which this was achieved depended critically on using the `sparse` function. Compute the flop count to solve a tridiagonal system $A\mathbf{x} = \mathbf{d}$ with

$$A = \begin{bmatrix} a & b & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ b & a & b & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & b & a & b & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & b & a & b \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & b & a \end{bmatrix}$$

efficiently.

OPTIONAL Implement the algorithm you found in MATLAB.

SOLUTION:

For the first column, we need the elementary row operation $R_2 \to R_2 - \frac{b}{a}R_1$ to give

$$[A \mid \mathbf{d}] \to \begin{bmatrix} a & b & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & d_1 \\ 0 & \tilde{a} & b & 0 & 0 & \cdots & 0 & 0 & 0 & * \\ 0 & b & a & b & 0 & \cdots & 0 & 0 & 0 & d_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & b & a & b & \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & b & a & d_n \end{bmatrix}$$

This requires 5 flops. One to compute $\frac{b}{a}$ and two each to compute the $\tilde{a}$ and $*$. For the second column we need $R_3 \to R_3 - \frac{b}{\tilde{a}}R_1$ to give

$$[A \mid \mathbf{d}] \to \begin{bmatrix} a & b & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & d_1 \\ 0 & \tilde{a} & b & 0 & 0 & \cdots & 0 & 0 & 0 & * \\ 0 & 0 & * & b & 0 & \cdots & 0 & 0 & 0 & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & b & a & b & d_{n-1} \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & b & a & d_n \end{bmatrix}$$

This again takes 5 flops. Clearly each row will require the same number of flops. Since there are $n-1$ rows, we have 5(n-1) flops to obtain

$$[A \mid \mathbf{d}] \to \begin{bmatrix} a & b & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & d_1 \\ 0 & \tilde{a} & b & 0 & 0 & \cdots & 0 & 0 & 0 & * \\ 0 & 0 & * & b & 0 & \cdots & 0 & 0 & 0 & * \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & * & b & * \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & * & * \end{bmatrix}$$

The back substitution will require 1 flop for row n and 3 flops for each of the remaining $n-1$ rows. Thus we have

$$3(n-1) + 1 = 3n - 2$$

flops for back substitution. Therefore the flop count is

$$5(n-1) + 3n - 2 = \mathcal{O}(8n)$$

considerably faster than the generic reduction.

OPTIONAL

```
function x = TDsolver(a,b,d)
%a is the diagonal element b is the off diagonal element, d is the
%righthand side vector
n = length(d); % n is the number of rows

% Modify the first-row coefficients
c(1) = b / a;    % assuming a not zero.
                 % c is a vector of the super diagonal elements
                 % after reduction.
d(1) = d(1) / a;    % Reduced vector d.

for i = 2:n-1
    temp = a - b * c(i-1);
    c(i) = b / temp;
    d(i) = (d(i) - b * d(i-1))/temp;
end

d(n) = (d(n) - b * d(n-1))/( a - b * c(n-1));

% Now back substitute.
x(n) = d(n);
for i = n-1:-1:1
    x(i) = d(i) - c(i) * x(i + 1);
end
x = x'; % convert to a column vector.
end
```