CS375 HW5

Ryan Scherbarth, University of New Mexico

September 2024

1. Cost of Gaussian Elimination
   Assume that an ancient computer solves a 1000-variable, upper-triangular, linear system by back substitution in 0.5 seconds. Estimate the time needed to solve a general (full) system by Gaussian Elimination (forward elimination + back-substitution). Use the counts from the lectures: $\frac{2n^3}{3} + O(n^2)$ for elimination and n^2 for back substitution.

   Given a mxn matrix, the number of operations on row m of an upper triangular matrix is 1, as we divide $\frac{x_{m,n-1}}{x_m,n}$.
   We then move up one row. Here, we will do
   1. plug in (multiply) prev value
   2. Subtract value from pt. 1 to RHS
   3. Divide to determine second term
   This illustrates that solving an upper-right triangular matrix takes $O(n^2)$ operations. Therefore, the total operations from back-substitution and elimination will be given by

$$\frac{2}{3}(1000)^3 + (1000)^2$$

   total operations. We know that our computer can solve a 1000 upper-triangular matrix in 0.5 seconds. This tells us the computer can solve at a rate of $2(1000^2) = 2,000,000$ FLOPS, or 2 Mega-FLOPS.

   Simply dividing our total operations with the speed per second will give us the total time in seconds;

$$\begin{aligned} &= \frac{\frac{2}{3}(1000)^3 + (1000)^2}{2,000,000} \\ &= 333.83 \text{ seconds} \\ &\approx 5.6 \text{ minutes} \end{aligned}$$

2. Gaussian Elimination with and without partial pivoting

   - (a) Write a Matlab script that uses naive Gaussian Elimination to solve the linear system

$$\begin{bmatrix} a & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1+a \\ 2 \end{bmatrix}$$

     for $a = 10^{-2k}$, $k = 1, 2, \ldots, 10$. The exact solution is $x = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, regardless of the value a. Place the solutions your script produces in a table. How does the accuracy of your numerical solution behave as k gets bigger. Explain.

```
k_values = 1:10;
exact_solution = [1; 1];

solutions = zeros(2, length(k_values));
errors = zeros(1, length(k_values));

for k = k_values
    a = 10^(-2*k);

    A = [a, 1;
         1, 1];
    b = [1 + a;
         2];

    n = length(b);
    x = zeros(n,1);

    % Forward Elimination
    for i = 1:n-1
        factor = A(i+1,i) / A(i,i);
        A(i+1,:) = A(i+1,:) - factor * A(i,:);
        b(i+1) = b(i+1) - factor * b(i);
    end

    % Back Substitution
    x(n) = b(n) / A(n,n);
    for i = n-1:-1:1
        x(i) = (b(i) - A(i,i+1:n) * x(i+1:n)) / A(i,i);
    end

    solutions(:, k) = x;
    errors(k) = norm(exact_solution - x);
end
```

k	a	x_1	x_2	Error
1	1.000000×10^{-2}	1.000000	1.000000	8.881784×10^{-16}
2	1.000000×10^{-4}	1.000000	1.000000	1.101341×10^{-13}
3	1.000000×10^{-6}	1.000000	1.000000	2.875566×10^{-11}
4	1.000000×10^{-8}	1.000000	1.000000	6.077471×10^{-9}
5	1.000000×10^{-10}	1.000000	1.000000	8.274037×10^{-8}
6	1.000000×10^{-12}	0.999867	1.000000	1.331440×10^{-4}
7	1.000000×10^{-14}	0.999201	1.000000	7.992778×10^{-4}
8	1.000000×10^{-16}	2.220446	1.000000	1.220446
9	1.000000×10^{-18}	0.000000	1.000000	1.000000
10	1.000000×10^{-20}	0.000000	1.000000	1.000000

As k increases, the param a decreases, approaching closer and closer to 0. As this happens, we the naive approach we implemented allows for very small values of a, which quickly decrease the accuracy of the result.

- (b) Next, change your Matlab code to carry out partial-pivoting. What is the solution now? Give a table of solution values. Explain your result and the accuracy relative to part (a).

k	a	x_1	x_2	Error
1	1.000000×10^{-2}	1.000000	1.000000	0.000000×10^{0}
2	1.000000×10^{-4}	1.000000	1.000000	0.000000×10^{0}
3	1.000000×10^{-6}	1.000000	1.000000	0.000000×10^{0}
4	1.000000×10^{-8}	1.000000	1.000000	0.000000×10^{0}
5	1.000000×10^{-10}	1.000000	1.000000	0.000000×10^{0}
6	1.000000×10^{-12}	1.000000	1.000000	3.140185×10^{-16}
7	1.000000×10^{-14}	1.000000	1.000000	0.000000×10^{0}
8	1.000000×10^{-16}	1.000000	1.000000	1.110223×10^{-16}
9	1.000000×10^{-18}	1.000000	1.000000	0.000000×10^{0}
10	1.000000×10^{-20}	1.000000	1.000000	0.000000×10^{0}

This table shows how partial pivoting can significantly decrease our error across a similar problem set.

3. Gaussian Elimination with scaled partial pivoting - Using scaled partial pivoting without actually moving data in the matrix, show the steps required to solve the following system of equations. Calculate the scale vector (called s in the lecture). Show how the pivot row is selected at each step, and carry out the computations. At each step, include the index vector (called ℓ in lecture).

$$\begin{bmatrix} 2 & -1 & 3 & 4 & 15 \\ 4 & 2 & 0 & 7 & 11 \\ 2 & 1 & 1 & 3 & 7 \\ 6 & 5 & 4 & 17 & 31 \end{bmatrix}, s = \begin{bmatrix} 4 \\ 7 \\ 3 \\ 17 \end{bmatrix}, \ell = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

For the augmented matrix,ple

$$\begin{bmatrix} 2 & 1 & 1 & 3 & 7 \\ 0 & -2 & 2 & 1 & 8 \\ 0 & 0 & -2 & 1 & -3 \\ 0 & 0 & 3 & 9 & 18 \end{bmatrix}, s = \begin{bmatrix} 3 \\ 4 \\ 7 \\ 17 \end{bmatrix}, \ell = \begin{bmatrix} 3 \\ 1 \\ 2 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 1 & 3 & 7 \\ 0 & -2 & 2 & 1 & 8 \\ 0 & 0 & -2 & 1 & -3 \\ 0 & 0 & 0 & 7.5 & 22.5 \end{bmatrix}, s = \begin{bmatrix} 3 \\ 4 \\ 7 \\ 17 \end{bmatrix}, \ell = \begin{bmatrix} 3 \\ 1 \\ 2 \\ 4 \end{bmatrix}$$

...finish?- Answer looks like it will be a little off too