+6 for presentation

CS375 HW5

Ryan Scherbarth, University of New Mexico

September 2024

1. Write the following Matlab Functions. Each of them should use the simplest possible algorithms with nested "for" loops. None of them should involve pivoting.

   - (a) function [L,U] = naiveLU(A). This function should return the unit lower triangular matrix L and the upper triangular matrix U e.t. $LU = A$ obtained via naive forward elimination.

```
function [L, U] = naiveLU(A)
    [n, m] = size(A);

    L = eye(n);
    U = A;

    for i = 1:n-1
        for j = i+1:n
            L(j, i) = U(j, i) / U(i, i);
            for k = i:n
                U(j, k) = U(j, k) - L(j, i) * U(i, k);
            end
        end
    end
end
```

   - (b) function y = naiveLTrioSol(L, b). This function should solve the unit lower-triangular system $L\overrightarrow{y} = \overrightarrow{b}$ using forward substitution. Your code should explicitly assume that L has ones on the diagonal.

```
function y = naiveLTrioSol(L, b)
    n = length(b);

    y = zeros(n, 1);

    for i = 1:n
        y(i) = b(i);
        for j = 1:i-1
            y(i) = y(i) - L(i, j) * y(j);
        end
    end
end
```

- (c) function x = naiveUTriSol(U, y). This function should solve the upper triangular system $U\overrightarrow{x} = y$ using backward substitution.

```
function x = naiveUTriSol(U, y)
    n = length(y);

    x = zeros(n, 1);

    for i = n:-1:1
        x(i) = y(i);
        for j = i+1:n
            x(i) = x(i) - U(i, j) * x(j);
        end
        x(i) = x(i) / U(i, i);
    end
end
```

2. Prove or provide a counter-example for each of the following statements. You can use Matlab to find counter-examples. Assume A is a matrix and c is a scalar

- (a) $||cA|| = |c|||A||$

$$
\begin{aligned}
||A|| &= \sqrt{\sum_{i,j} |a_{i,j}|^2} \\
||cA|| &= \sqrt{\sum_{i,j} |ca_{i,j}|^2} \\
&= \sqrt{\sum_{i,j} |c|^2|a_{i,j}|^2} \\
&= |c|\sqrt{\sum_{i,j} |a_{i,j}|^2} \\
&= |c| * ||A||
\end{aligned}
$$

you have only proved for the 2-norm

Therefore, it is true that $||cA|| = |c|||A||$.

- (b) $k(cA) = k(A)$ for any nonzero constant c.

This statement is saying that the condition number of a matrix stays the same despite a nonzero constant being added to the equation. We know this to be true, and we can prove as follows;

$$
\begin{aligned}
k(A) &= ||A|| * ||A^-1|| \\
K(cA) &= ||cA|| * ||c(A)^{-1}|| \\
||cA|| &= |c| * ||A|| \qquad \text{ref 2a} \\
(cA)^-1 &= \frac{1}{c}A^-1 \\
(cA)^-1 &= |\frac{1}{c}| * ||A^{-1}|| \\
k(cA) &= |c| * ||A|| * |\frac{1}{c}| * ||A^-1|| \\
&= ||A|| * ||A^{-1}||
\end{aligned}
$$

Therefore, it is also true that $K(cA) = k(A)$ for any non-zero scalar c.

- (c) $k(A)$ is the same for every matrix norm

It is not true that $k(A)$ will be the same for every matrix norm. For example, consider $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. Using matlab to determine the condition number for 3 of the norms;

```
A = [1, 2; 3, 4];

k_1 = cond(A, 1);
k_2 = cond(A, 2);
k_inf = cond(A, inf);
```

We see that while the 1 norm and the infinity norm are both 21, the 2-norm in this case gives us a value 14.933.

3. For each of the following systems: verify the solution by hand, report the infinity norm condition number of $A(\text{cond}(A, \text{inf}))$, solve the system numerically using your functions from problem 1, solve the system numerically using Matlab's built in lu routine (with pivoting), and compare norm of the error of your code and Matlab's code (Use infinity norm and report just absolute error).

- (a)

$$A = \begin{bmatrix} 2 & -2 & -1 \\ 4 & 1 & -2 \\ -2 & 1 & -1 \end{bmatrix}, \overrightarrow{b} = \begin{bmatrix} -2 \\ 1 \\ -3 \end{bmatrix}, \overrightarrow{x} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

We can begin by verifying the given matrix;

$$\begin{aligned} 2(1) - 2(1) - 1(2) &== -2 \\ 4(1) + 1(1) - 2(2) &== 1 \\ -2(1) + 1(1) - 1(2) &== -3 \\ A\overrightarrow{x} &== \overrightarrow{b} \end{aligned}$$

Matlab gives us a condition norm, $\text{cond}(A, \text{inf}) = 6.3000$.

Solving with our previous functions in matlab returns the correct $\overrightarrow{x} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$

Solving numerically using lu returns the proper $\overrightarrow{x} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$

Calculating the error of each tells us that both options have an error of 0. The Matlab code for the above sections in the corresponding order is below. The functions used are those previously specified in this document and therefore are not going to be repeated here.

```
A = [2, -2, -1; 4, 1, -2; -2, 1, -1];
b = [-2; 1; -3];
x_true = [1; 1; 2];

cond_A_inf = cond(A, inf);

[L, U] = naiveLU(A);
y = naiveLTrioSol(L, b);
x_naive = naiveUTriSol(U, y);

[L, U, P] = lu(A);
y_matlab = L \ (P * b);
x_matlab = U \ y_matlab;

error_naive = norm(x_naive - x_true, inf);
error_matlab = norm(x_matlab - x_true, inf);
```

- (b)

$$A = \begin{bmatrix} 10^{-16} & 1 & -1 \\ 2 & 0 & 1 \\ 1 & 2 & -3 \end{bmatrix}, \overrightarrow{b} = \begin{bmatrix} -3\text{x}10^{-16} \\ -4 \\ -5 \end{bmatrix}, \overrightarrow{x} = \begin{bmatrix} -3 \\ 2 \\ 2 \end{bmatrix}$$

We can begin by verifying the given matrix;

$$\begin{aligned} 10^{-16}(-3) + 1(2) - 1(2) &== -3\text{x}10^{-16} \\ 2(-3) + 1(2) &== -4 \\ 1(-3) + 2(2) - 3(2) &== -5 \\ A\overrightarrow{x} &== \overrightarrow{b} \end{aligned}$$

Matlab gives us a condition norm, $\text{cond}(A, \text{inf}) = 20.0000$

Solving with our previous functions in Matlab returns the correct $\overrightarrow{x} = \begin{bmatrix} -2.2204 \\ 1.5000 \\ 1.5000 \end{bmatrix}$

Solving numerically using lu returns the proper $\overrightarrow{x} = \begin{bmatrix} -3 \\ 2 \\ 2 \end{bmatrix}$

Calculating the error of each tells us our implementation with these extreme numbers end with an error of 0.77955. Below is the corresponding Matlab code.

```
A = [1e-16, 1, -1; 2, 0, 1; 1, 2, -3];
b = [-3e-16; -4; -5];
x_true = [-3; 2; 2];

cond_A_inf = cond(A, inf);

[L, U] = naiveLU(A);
y = naiveLTrioSol(L, b);
x_naive = naiveUTriSol(U, y);

[L, U, P] = lu(A);
y_matlab = L \ (P * b);
x_matlab = U \ y_matlab;

error_naive = norm(x_naive - x_true, inf);
error_matlab = norm(x_matlab - x_true, inf);
```

4. Consider the matrix $A = \begin{bmatrix} 1 & 1 \\ b & 1 \end{bmatrix}$ where $b \neq 1$.

- (a) Find $||A||_1$ in terms of b

$$\begin{aligned} \text{col 1:} &= 1 + |b| \\ \text{col 2:} &= 2 \\ ||A||_1 &= \max(1 + |b|, 2) \end{aligned}$$

- (b) Find $||A^{-1}||_1$ in terms of b. Remember that $A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}$

$$\begin{aligned} A^{-1} &= \frac{1}{1-b} \begin{bmatrix} 1 & -1 \\ -b & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{1-b} & \frac{-1}{1-b} \\ \frac{-b}{1-b} & \frac{1}{1-b} \end{bmatrix} \\ \text{col 1:} &= \frac{1 + |b|}{|1 - b|} \\ \text{col 2:} &= \frac{2}{|1 - b|} \\ ||A^{-1}||_1 &= \max(\frac{1 + |b|}{|1 - b|}, \frac{2}{|1 - b|}) \\ &= \frac{\max(1 + |b|, 2)}{|1 - b|} \end{aligned}$$

- (c) Using the previous two parts, find $k(A)$ in terms of b.

$$\begin{aligned} k(A) &= ||A||_1 * ||A^{-1}||_1 \\ k(A) &= (\max(1 + |b|, 2))(\frac{\max 1 + |b|, 2}{|1 - b|}) \\ k(A) &= \frac{(\max(1 + |b|, 2))^2}{|1 - b|} \end{aligned}$$

- (d) Explain geometrically why the condition number grows as $b \to 1$.

As $b \to 1$, matrix A stretches vectors along an increasingly singular direction, at the same time becoming more and more sensitive to small changes. Note that we defined a condition number $k(A)$ as a matrix's sensitivity to permuatitons.

The increasing sensitivity leads to a growing condition number, indicating the matrix is also becoming ill conditioned and harder to invert.

- (e) If you solve the linear system $Ax = b$ using Gaussian elimination without pivoting, for what b should you expect to have 10 digits of accuracy, when the computations are carried out using double precision? Recall that for double precision we have $\epsilon_m \approx 2.2\text{x}10^{-16}$.

$$\begin{aligned} \frac{||\nabla x||}{||x||} &\approx k(A)\epsilon_m \\ k(A)\epsilon_m &\leq 10^{-10} \\ k(A) &\leq \frac{10^{-10}}{2.2\text{x}10^{-16}} \\ &\approx 4.55\text{x}10^5 \end{aligned}$$

Now that we've determined an estimate that satisfies $k(A)$, we just solve for b.
We find that $\frac{(1+|b|)^2}{|1-b|} \leq 4.55\text{x}10^5$

Interpreting our result, this tells us that for us to achieve a minimum of 10 digits of accuracy, b must be sufficiently far from 1.

-5

5. Let $A = \begin{bmatrix} 1 & 1+\ell \\ 1-\ell & 1 \end{bmatrix}$ where ℓ is some small number.

- (a) Find LU factorization of A without pivoting.

$$\begin{aligned}
(1-\ell) &= l_{21}(1) + 1 \\
l_{21} &= 1 - \ell \\
u_{22} &= 1 - (1-\ell)(1+\ell) \\
&= \ell^2 \\
U &= \begin{bmatrix} 1 & 1+\ell \\ 0 & \ell^2 \end{bmatrix} \\
L &= \begin{bmatrix} 1 & 0 \\ 1-\ell & 1 \end{bmatrix} \\
A &= \begin{bmatrix} 1 & 1+\ell \\ 1-\ell & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1-\ell & 1 \end{bmatrix} \begin{bmatrix} 1 & 1+\ell \\ 0 & \ell^2 \end{bmatrix}
\end{aligned}$$

- (b) Using double precision floating point arithmetic, for what value go *ell* will solving via LU factorization (without pivoting) fail?

For the LU factorization to be reliable, the diagonal entries of U need to not be too close to 0. In this case, the value we need to look at is the precision as $\ell^2 \to 0$ as the only variable on the U's diagonal.

$$\begin{aligned}
\ell^2 &\leq \epsilon_m \\
\ell &\leq \sqrt{\epsilon_m} \\
\ell &\leq \sqrt{2.2\text{x}10^{-16}} \\
&\approx 1.48\text{x}10^{-8}
\end{aligned}$$

Therefore, in double precision floating point arithmetic, the LU factorization without pivoting will become unreliable when ℓ is less than or equal to $1.48\text{x}10^{-8}$.

- (c) Find the determinant of A

$$\begin{aligned}
\det(L) &= (1)(1) - (0)(1-\ell) \\
&= 1 \\
\det(U) &= (1)(\ell^2) - (1+\ell)(0) \\
&= \ell^2 \\
\det(A) &= (1)(\ell^2) \\
&= \ell^2
\end{aligned}$$

- (d) Using double precision floating point arithmetic, for what values of *ell* will $\det(A)$ equal to 0?

Just as the factor ℓ^2 bounds our answer in part (b), we come to the same solution here. In double precision floating point arithmetic, the determinant of A will become zero. This is because for any value below this point we are going to end up rounding downward to 0. Therefore, the threshold to be zero is $\ell \leq 1.48\text{x}10^{-8}$.

6. The Hilbert Matrix of size $k, A = \text{HILB}(k)$, is defined by $a_{i,j} = \frac{1}{i+j-1}$, for example,
$A = \text{HILB}(3) = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$
Consider th elinear system $Ax = b$, where A is a Hilbert Matrix and the vector b is given by $b_i = \sum_{j=1}^{n} a_{i,j}$ (i.e. b is the row-sum of A).

- (a) Use the Matlab command hilb to create Hilbert Matricies for $8 \leq k \leq 12$. Create a table of the condition numbers of these matricies using the matlab command cond.

Matrix Size (k)	**Condition Number**
8	1.5258×10^{10}
9	4.9315×10^{11}
10	1.6025×10^{13}
11	5.2268×10^{14}
12	1.6361×10^{16}

- (b) Without using Matlab, what is the true solution, x_{true}, to the linear systems? Hint: the answer is nearly identical for any k.

The true solution will just be $x_{\text{true}} = \overrightarrow{1}$. Each row of the matrix sums to a unique value, but the solution vector always works out to $\begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}^T$.

As we increase the matrix dimensions we will be multiplying by smaller and smaller numbers, so the larger our matrix will increase the likelihood of our answer differing from the exact answer.

- (c) Without solving the system, how many decimal digits of accuracy do you expect for $8 \leq k \leq 12$ if you solve the system using Gaussian elimination without pivoting? Explain why.

Like the previous question we can use $k(A)\epsilon_m$ to estimate our accuracy. The relative error will be given by $\frac{||\nabla x||}{||x||} \approx (1.5\text{x}10^{10})(2.2\text{x}10^{-16}) = 3.3\text{x}10^{-6}$.

Repeating for each value of $k \exists [8..12]$ we estimate the corresponding number of digits of probabilities as follows; $[6, 4, 2, 0, 0]$ where we can essentially not trust the answer at all once we get to $k = 12$.what are the

- (d) Now solve the linear systems for $8 \leq k \leq 12$. Use the Matlab backslash operator for the linear solve. For each k, write down your Matlab solution. Make sure you set the format to long so that Matlab displays 16 digits of the solution.

k	Solution Vector
8	0.999999999971768 1.000000001528637 0.999999979905157 1.000000109263256 0.999999704801189 1.000000418828814 0.999999701297392
9	0.999999999724598 1.000000018952923 0.999999679825507 1.000002282376294 0.999991637642271 1.000017059494292 0.999980421913029 1.000011818523027
10	0.999999999095413 1.000000077638004 0.999998354294758 1.000014906295676 0.999929106136442 1.000194425970870 0.999681635015205 1.000307148615796 0.999838981464458
11	0.999999994843199 1.000000538857933 0.999986041359532 1.000155875384751 0.999072324686194 1.003258717986100 0.992910160557468 1.009658807364932 0.991981907673492 1.003707653813361
12	0.999999965056470 1.000004409407701 0.999861743769333 1.001879710648280 0.986241982741676 1.060375974230446 0.831939173215308 1.303973363208347 0.643862006187990 1.260684018118645 0.891666923800637

- (e) How many digits of accuracy do you see in the solution components for different values of n. Double precision has about 15 or 16 digits of accuracy. Does this agree with your answer in c?

We can determine the accurate digits with the approximation approx $\approx -\log_{10}(\text{Error})$. We can find the actual errors to be:

We notice for smaller values of k our actual accuracy is actually better than the prediction. The Matlab

Matrix Size k	**Actual accuracy**
8	10 digits
9	8 digits
10	7 - 9 digits
11	3 - 5 digits
12	0 - 3 digits

code is probably implementing some optimization techniques like pivoting by default which is helping.

For larger k's, we drop off significantly in accuracy which aligns with our prediction. Therefore, we can conclude that we had a relatively accurate prediction in the previous part.

- (f) Tabulate the relative error in the solution $\frac{||x_t - x||_2}{||x_t||_2}$ and the relative value of residual $\frac{||b - Ax||_2}{||b||_2}$, where x_t is the true solution and x is the solution returned by the Matlab backslash operator. Are both relative errors and relative residuals small as n increases? Explain your results

k	**Rel Error**	**Rel Residual**
8	2.1542265528007425e-07	1.1039170017430076e-16
9	9.9875752539065326e-06	9.0228225564539507e-17
10	1.6307255259116017e-04	1.2911145854691962e-16
11	4.6086387523204724e-03	7.7395460024724246e-17
12	1.6617999618966295e-01	1.3460611949190969e-16

Here, we see that while the relative residuals remain small, the errors increase as k increases. This corresponds to the ill-conditioning of the Hilbert matrix for larger matrix sizes.