Math 584 Homework 5
Due November 18
By Marvyn Bailly

Problem 1 *Let A be an m by m nonsingular matrix and let b be a given nonzero m-vector. Suppose x satisfies $Ax = b$ and $\hat{x}$ satisfies $A\hat{x} = \hat{b}$, where $\hat{b}$ is slightly different from b. Show that*

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \leq \kappa(A) \frac{\|\hat{b} - b\|_2}{\|b\|_2},$$

where $\kappa(A) = \|A\|_2 \|A^{-1}\|_2$ is the 2-norm condition number of A. Show that there are nonzero vectors b and $\hat{b} - b$ for which equality holds. [Note: You might want to review material in Lecture 12 of the text.]

Solution.
Let A be an $m \times m$ nonsingular matrix and let b be a given nonzero m-vector. Suppose that x solves $Ax = b$ and let $\hat{x}$ solve $A\hat{x} = \hat{b}$ where $\hat{b}$ is slightly different than $\hat{b}$. If we subtract the two equations we have

$$\begin{aligned}
A\hat{x} - Ax &= \hat{b} - b \\
A(\hat{x} - x) &= \hat{b} - b \\
\hat{x} - x &= A^{-1}(\hat{b} - b) \\
\|\hat{x} - x\|_2 &= ||A^{-1}(\hat{b} - b)||_2 \\
\frac{\|\hat{x} - x\|}{\|x\|_2} &= \frac{||A^{-1}(\hat{b} - b)||_2}{\|x\|_2} \\
\frac{\|\hat{x} - x\|}{\|x\|_2} &\leq \left\|A^{-1}\right\|_2 \frac{||(\hat{b} - b)||_2}{\|x\|_2} \\
&= \left\|A^{-1}\right\|_2 \frac{||(\hat{b} - b)||_2}{\|x\|_2} \frac{\|Ax\|_2}{\|b\|_2} \\
&= \left\|A^{-1}\right\|_2 \frac{||(\hat{b} - b)||_2}{\|x\|_2} \frac{\|Ax\|_2}{\|b\|_2} \\
&= \frac{\|Ax\|_2}{\|x\|_2} \left\|A^{-1}\right\|_2 \frac{||(\hat{b} - b)||_2}{\|b\|_2} \\
&\leq \|A\|_2 \left\|A^{-1}\right\|_2 \frac{||(\hat{b} - b)||_2}{\|b\|_2} \\
&= \kappa(A) \frac{||(\hat{b} - b)||_2}{\|b\|_2},
\end{aligned}$$

Therefore

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \leq \kappa(A) \frac{\|\hat{b} - b\|_2}{\|b\|_2}.$$

To achieve equality observe that the inequality arose form

$$\frac{\|A^{-1}(\hat{b} - b)\|_2}{\|x\|_2} \leq \|A^{-1}\|_2 \quad \text{and} \quad \frac{\|Ax\|_2}{\|x\|_2} \leq \|A\|_2.$$

The first inequality is equality when $(\hat{b} - b)/\|\hat{b} - b\|_2$ is a right singular vector of A^{-1} corresponding to the greatest singular value of A^{-1}. To get equality on the second case, we require $x/\|x\|_2$ is a right singular value of A which means that $b = \|b\|_2$ needs to be a left singular vector of A corresponding to the greatest singular value. Thus we have that

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} = \kappa(A) \frac{\|\hat{b} - b\|_2}{\|b\|_2},$$

when these conditions are met. $\square$

Problem 2 *By hand, write the following matrix in the form LU, where L is a unit lower triangular matrix and U is an upper triangular matrix:*

$$\begin{bmatrix} 2 & 1 & -1 \\ 1 & 3 & 1 \\ -1 & 1 & 2 \end{bmatrix}.$$

(You may use Matlab to check your result, but keep the exact answer by retaining square roots, etc.) Without completely redoing the calculation, write the same matrix in the form LL^T, where L is lower triangular (but probably does not *have ones on the diagonal). Explain how you can derive the LL^T-factorization from the LU factorization, and vice-versa.*

Solution. Consider the matrix

$$A = \begin{pmatrix} 2 & 1 & -1 \\ 1 & 3 & 1 \\ -1 & 1 & 2 \end{pmatrix}.$$

We wish to find the LU factorization of A by hand. Let's first eliminate the entries under 2 in the first column with L_1 to get

$$L_1 A = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 1/2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & -1 \\ 1 & 3 & 1 \\ -1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 1 & -1 \\ 0 & 5/2 & 3/2 \\ 0 & 3/2 & 3/2 \end{pmatrix}.$$

Next let's eliminate the entries under $5/2$ in the second column with L_2 to get

$$L_2 L_1 A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3/5 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 1/2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & -1 \\ 1 & 3 & 1 \\ -1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 1 & -1 \\ 0 & 5/2 & 3/2 \\ 0 & 0 & 3/5 \end{pmatrix} = U.$$

Note that

$$L_2 L_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3/5 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 1/2 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 1/2 & -3/5 & 1 \end{pmatrix}.$$

Thus we have found the LU factorization of A to be

$$A = \begin{pmatrix} 2 & 1 & -1 \\ 1 & 3 & 1 \\ -1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ -1/2 & 3/5 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & -1 \\ 0 & 5/2 & 3/2 \\ 0 & 0 & 3/5 \end{pmatrix} = L_2^{-1} L_1^{-1} U = LU.$$

Now we wish to form a LL^T factorization of A using the LU factorization. We have that

$$A = LU = LIU,$$

where I is the $m \times m$ identity matrix of the form

$$I = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} \begin{pmatrix} 1/a & 0 & 0 \\ 0 & 1/b & 0 \\ 0 & 0 & 1/c \end{pmatrix},$$

for some scalars a, b, and c. Thus A can be rewritten in the form

$$\begin{aligned} A &= LIU \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ -1/2 & 3/5 & 1 \end{pmatrix} \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} \begin{pmatrix} 1/a & 0 & 0 \\ 0 & 1/b & 0 \\ 0 & 0 & 1/c \end{pmatrix} \begin{pmatrix} 2 & 1 & -1 \\ 0 & 5/2 & 3/2 \\ 0 & 0 & 3/5 \end{pmatrix} \\ &= \begin{pmatrix} a & 0 & 0 \\ a/2 & b & 0 \\ -a/2 & 3b/5 & c \end{pmatrix} \begin{pmatrix} 2/a & 1/a & -1/a \\ 0 & 5/2b & 3/2b \\ 0 & 0 & 3/5c \end{pmatrix}. \end{aligned}$$

In order for these two matrices to be transposes of each other, $a = \frac{2}{a} \implies a = \pm\sqrt{2}$, $b = \frac{5}{2b} \implies b = \pm\sqrt{\frac{5}{2}}$, and $c = \frac{3}{5c} \implies c = \pm\sqrt{\frac{3}{5}}$. Note that a, b, and c are the square roots of the main diagonal of U. If we consider the positive case, we get

$$\begin{aligned} A &= \begin{pmatrix} \sqrt{2} & 0 & 0 \\ \sqrt{2}/2 & \sqrt{5/2} & 0 \\ -\sqrt{2}/2 & 3\sqrt{10}/10 & \sqrt{3/5} \end{pmatrix} \begin{pmatrix} 2/\sqrt{2} & 1/\sqrt{2} & -1/\sqrt{2} \\ 0 & \sqrt{10}/2 & 3\sqrt{10}/10 \\ 0 & 0 & \sqrt{15}/5 \end{pmatrix} \\ &= \begin{pmatrix} \sqrt{2} & 0 & 0 \\ \sqrt{2}/2 & \sqrt{5/2} & 0 \\ -\sqrt{2}/2 & 3\sqrt{10}/10 & \sqrt{3/5} \end{pmatrix} \begin{pmatrix} \sqrt{2} & 0 & 0 \\ \sqrt{2}/2 & \sqrt{5/2} & 0 \\ -\sqrt{2}/2 & 3\sqrt{10}/10 & \sqrt{3/5} \end{pmatrix}^T \\ &= L'(L')^T. \end{aligned}$$

And thus we have found our LL^T factorization (Cholesky factorization) from the LU factorization of A. Notice that we can always find a LL^T factorization from the LU factorization by utilizing

$$A = LIU = L \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} \begin{pmatrix} 1/a & 0 & 0 \\ 0 & 1/b & 0 \\ 0 & 0 & 1/c \end{pmatrix} U,$$

for scalars a, b, and c and then choosing their values to satisfy the factorization (the square roots of the main diagonal of U). To get the LU factorization from the LL^T we can undo the process by computing

$$A = LIL^T = L \begin{pmatrix} 1/a & 0 & 0 \\ 0 & 1/b & 0 \\ 0 & 0 & 1/c \end{pmatrix} \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} L^T,$$

for some a, b, and c and then choosing their values to satisfy the LU factorization (the values of the main diagonal of L). In both cases we will end up with a system of equations for a, b, and c that can be solved to satisfy the factorization only if the original matrix is Hermitian positive definite. This is since we know that every Hermitian positive definite matrix A has a unique Cholesky factorization that can be computed using the algorithm described in lecture 23:

$$\begin{aligned} A &= \begin{pmatrix} \alpha & 0 \\ w/\alpha & I \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & K - ww^*/a_{11} \end{pmatrix} \begin{pmatrix} \alpha & w^*/\alpha \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ w/\alpha & I \end{pmatrix} \begin{pmatrix} \alpha^2 & 0 \\ 0 & K - ww^*/a_{11} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ w^*/\alpha & I \end{pmatrix}^T \\ &= \begin{pmatrix} 1 & 0 \\ w/\alpha & I \end{pmatrix} \begin{pmatrix} a_{11} & 0 \\ 0 & K - ww^*/a_{11} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ w^*/\alpha & I \end{pmatrix}^T, \end{aligned}$$

which leads to the method that we applied to go between LU and LL^T factorization. $\square$

Problem 3 *p. 154, Exercises 20.1 and 20.2. In Exercise 20.2, count the number of operations required to compute L and U, assuming that you do not operate on entries that are* 0 *and are known to remain* 0 *after elimination.*

Solution. Consider the nonsingular matrix $A \in \mathbb{C}^{m\times m}$ of the form

$$A = a_{ij}.$$

20.1: We wish to show that A has a LU factorization if and only if for each k with $1 \leq k \leq m$, the upper-left $k \times k$ block A_k is nonsingular.

($\Leftarrow$) First let's assuming that A_k is nonsingular for all k. When $k = 1$, we have that $\det(A_1) \neq 0$ which means that $a_{11} \neq 0$. As a_{11} is nonzero, we can apply Gaussian Elimination to get

$$A = L_1^{-1}\hat{A}_1 = L_1^{-1}\begin{pmatrix} u_{11} & w_1 \\ 0 & \hat{A}_{2:m,2:m} \end{pmatrix}$$

where $w_1 = [a_{12}, \cdots, a_{1m}]$ and $(\hat{A}_{2:m,2:m})_{ij} = \hat{a}_{ij}$. Now if $\hat{a}_{22} = 0$, then

$$\hat{A}_2 = \begin{pmatrix} a_{11} & w_{11} \\ 0 & 0 \end{pmatrix},$$

which implies that $\det(\hat{A}_2) = 0$. But this is a contradiction of our assumption and thus $a_{22} \neq 0$ and we can proceed with Gaussian Elimination. Assuming that this holds until the k^{th} step, we will have gotten the matrix

$$A = L_k^1 \cdots L_1 \hat{A}_k = L_k^1 \cdots L_1 \begin{pmatrix} U_k & W \\ 0 & \hat{A}_{k+1:m,k+1:m} \end{pmatrix},$$

where U_k is a upper triangular and $(\hat{A}_{k+1:m,k+1:m})_{11} \neq 0$ as otherwise $\det \hat{A}_k = 0$ which contradicts our assumptions. Thus we can continue to the $k+1$th step. Therefore we are able to continue preforming Gaussian Elimination to obtain $A = LU$.

($\Rightarrow$) Now let's assume that $A = LU$ where L is nonsingular and lower triangular and U is upper triangular. Since the product of nonsingular matrices are nonsingular, $\det(LA) \neq 0$ which implies that $\det(U) \neq 0$. Since U is triangular, $u_{ii} \neq 0$ for all i. Therefore, L_k^{-1} and U_k are nonsingular and we have that

$$A_k = L_k^{-1}U_k,$$

which means that A_k is nonsingular for all k.

Finally we wish to show that the LU factorization is unique. Let's assume that $A = L_1U_1$ and $A = L_2U_2$ be two distinct LU factorizations of A. We know that L_1, L_2, U_1, U_2 are nonsingular and thus we have that

$$L_1U_1 = L_2U_2$$

$$\begin{aligned} &\iff L_2^{-1}L_1U_1 = U_2 \\ &\iff L_2^{-1}L_1U_1U_1^{-1} = U_2U_1^{-1} \\ &\iff L_2^{-1}L_1 = U_2U_1^{-1}, \end{aligned}$$

and since L_1 and L_2 are lower triangular and U_1 and U_2 are upper triangular, $L_2^{-1}L_1$ and $U_2U_1^{-1}$ are both diagonal matrices. Since the diagonal entries of L_1 and L_2 are one, then $L_2^{-1}L_1 = I = U_2U_1^{-1}$. Thus $L_1 = L_2$ and $U_1 = U_2$. Therefore the LU factorization is unique.

20.2: Let $A \in \mathbb{C}^{m\times m}$ such that it satisfies the condition of Exercise 20.1 and is banded with bandwidth $2p+1$, i.e., $a_{ij} = 0$ for $|i-j| > p$ and $a_{ij} \neq 0$ otherwise. Since A satisfies the conditions of exercise 20.1, we know we can carry out Gaussian elimination at every step. On every step, we only need to zero out p elements below the main diagonal. Thus we can see that U is of the form $u_{ij} = 0$ if $j - i > p$ or $i < j$, i.e., is upper triangular but banded at p above the main diagonal. While L is of the form $l_{ij} = 0$ if $i - j > p$ or $i > j$. i.e., is lower triangular but banded at p below the main diagonal. Thus we have the sparsity of U and L.

To count the operations, we know that at the jth stage of Gaussian elimination, we will need to zero the $j+1$th entry in the column to either the $j+p$th entry (since there are zeros below this point) or the mth entry (since we have reached the end of the matrix). Similarly, we will only need to modify $j+p$ or m entries in each row corresponding to the entries of the columns. Thus we have that the work in the jth step is $2 \cdot \min\{p, m-j\} \cdot \min\{p+1, m-j+1\}$. Therefore the total work is given

$$\begin{aligned} &\sum_{j=1}^{m-1} 2 \cdot \min\{p, m-j\} \cdot \min\{p+1, m-j+1\} \\ &\approx \sum_{j=1}^{m-1} (1 + 2(\min\{p+1, m-j+1\}))(\min\{p, m-j\}) \\ &= \sum_{j=1}^{m-p-1} (1+2(p+1))p + \sum_{j=m-p}^{m-1} (1+2(m-k+1))(m-k) \\ &= (m-p-1)(2p^2+3p) + \sum_{j=m-p}^{m-1} (2(m-j)^2 + 3(m-j)) \\ &= (m-p-1)(2p^2+3p) + \sum_{j=1}^{p} (2k^2+3k) \\ &= (m-p-1)(2p^2+3p) + 2\left(\frac{p(p+1)(2p+1)}{6}\right) + 3\left(\frac{p(p+1)}{2}\right), \end{aligned}$$

and collecting the highest order terms, we get the total work to be approximately $2p^2m$.

□

Problem 4 *Write a routine to generate an m by m matrix with a given 2-norm condition number. You can make your routine a* `function` *in Matlab that takes two input arguments – the matrix size m and the desired condition number* `condno` *– and produces an m by m matrix A with the given condition number as output:*

```
function A = matgen(m, condno)
```

Form A by generating two random orthogonal matrices U and V and a diagonal matrix Σ with $\sigma_{jj} = condno^{-(j-1)/(m-1)}$, and setting $A = U\Sigma V^*$. [Note that the largest diagonal entry of Σ is 1 and the smallest is $condno^{-1}$, so the ratio is `condno`*.] You can generate a random orthogonal matrix in Matlab by first generating a random matrix,* `Mat = randn(m)`*, and then computing its QR decomposition,* `[Q,R] = qr(Mat)`*. The matrix* `Q` *is then a random orthogonal matrix. You can check the condition number of the matrix you generate by using the function* `cond` *in Matlab. Turn in a listing of your code.*

For `condno`$= (1,\ 10^4,\ 10^8, 10^{12}, 10^{16})$*, use your routine to generate a random matrix* `A` *with condition number* `condno`*. Also generate a random vector* `xtrue` *of length m and compute the product* `b = A*xtrue`*.*

1. *Solve $Ax = b$ using Gaussian elimination with partial pivoting. This can be done in Matlab by typing* `x = A\b`*. Determine the 2-norm of the error*

   ```
   norm(x - xtrue)/norm(xtrue)
   ```

   in your computed solution and explain how this is related to the condition number of A. Compute the 2-norm of the residual, `norm(b-A*x)/(norm(A)*norm(x))`*. Does the algorithm for solving $Ax = b$ appear to be* backward stable *(at least for this problem); that is, is the computed solution the exact solution to a nearby problem?*

2. *Solve $Ax = b$ by inverting A and multiplying by the inverse:* `Ainv = inv(A); x = Ainv*b`*. Again look at relative errors and residuals. Does this algorithm appear to be backward stable?*

3. *Finally, solve $Ax = b$ using Cramer's rule (i.e., compute the determinant of A by typing* `det(A)` *and then compute* `x(j)` *by replacing column j of A by the right-hand side vector b, computing the determinant of the resulting matrix A_j and finding the ratio:* `det(A_j)/det(A)`*). Again look at relative errors and residuals and determine whether this algorithm is backward stable.*

Turn in a table showing the relative errors and residuals for each of the three algorithms and each of the condition numbers tested, along with a brief explanation of the results.

Solution.
I first created the MatLab function `matgen(m,condno)` which outputs an $m \times m$ matrix with condition number condno. The following code is this function

```
function A=matgen(m, condno)

%Generate a two random orthogonal m-square matrix
[U,R1] = qr(randn(m));
[V,R2] = qr(randn(m));

%Generate Sigma
v = zeros(m,1);
for j=1:m
    v(j) = condno^(-(j-1)/(m-1));
end
sigma = diag(v);

%Compute A
A = U*sigma*V';
```

Next I created the MatLab script that computes the desired values for this question:

```
function hw5q4(size)
%define condnos
condnos = [1,10^4,10^8,10^12,10^16];

%define xtrue
xtrue = randn(size,1);

%one big loop for each condnos case
for i=1:length(condnos)
    A = matgen(size,condnos(i));
    b = A*xtrue;

    %Method 1 - Guassian Elimation with Partial Pivoting
    x1 = A\b;
    error1 = norm(x1 - xtrue)/norm(xtrue);
    residual1 = norm(b-A*x1)/(norm(A)*norm(x1));

    %Method 2 - Inverting A
    x2 = inv(A)*b;
    error2 = norm(x2 - xtrue)/norm(xtrue);
    residual2 = norm(b-A*x2)/(norm(A)*norm(x2));

    %Method 3 - Cramer's Rule
    dA = det(A);
    x3 = zeros(size,1);
```

```
    for j=1:size
        Aj = A;
        Aj(:,j) = b;
        dj = det(Aj);
        x3(j) = dj/dA;
    end
    error3 = norm(x3 - xtrue)/norm(xtrue);
    residual3 = norm(b-A*x3)/(norm(A)*norm(x3));

    fprintf("For Condition number: %e \n",condnos(i))
    x = [x1,x2,x3];
    error = [error1,error2,error3];
    residual = [residual1,residual2,residual3];
    %disp("x: " + x)
    fprintf("Relative error and res for GE: %e and %e \n",error1,residual1)
    fprintf("Relative error and res for inv(A): %e and %e \n",error2,residual2)
    fprintf("Relative error and res for Cramer: %e and %e \n",error3,residual3)
    fprintf("\n")
end
```

This code outputs the following table

```
>> hw5q4(20)
For Condition number: 1.000000e+00
Relative error and res for GE: 4.717867e-16 and 4.118674e-16
Relative error and res for inv(A): 3.226785e-16 and 2.539641e-16
Relative error and res for Cramer: 2.487007e-16 and 3.576038e-16

For Condition number: 1.000000e+04
Relative error and res for GE: 2.765010e-13 and 4.161482e-17
Relative error and res for inv(A): 2.849089e-13 and 2.981683e-14
Relative error and res for Cramer: 2.821079e-13 and 2.602296e-14

For Condition number: 1.000000e+08
Relative error and res for GE: 5.663710e-10 and 5.222286e-17
Relative error and res for inv(A): 6.799182e-10 and 7.097760e-11
Relative error and res for Cramer: 7.666664e-10 and 6.237951e-11

For Condition number: 1.000000e+12
Relative error and res for GE: 5.258795e-06 and 4.091842e-17
Relative error and res for inv(A): 1.073367e-05 and 1.576008e-06
Relative error and res for Cramer: 7.524406e-06 and 1.148565e-06
```

```
For Condition number: 1.000000e+16
Relative error and res for GE: 1.579228e-03 and 1.917691e-17
Relative error and res for inv(A): 8.686767e-03 and 1.127315e-03
Relative error and res for Cramer: 3.173410e-02 and 7.940142e-04
```

From the data, we can see that all three methods across different condition numbers computes a similar relative error. But the residual of Gaussian elimination with pivoting remains small throughout the increasing condition numbers while the other two methods are inversely proportional to the condition number. Note that the 2-norm error is approximately proportional to the condition number. We know that backward stable algorithms can compute an accurate solution to a near by problem and thus we expect the residual to remain near machine epsilon regardless of the condition of A. Thus we can guess that Gaussian Elimination with pivoting is backward stable while the other methods are not. $\square$

Problem 5 *In Matlab, form a* 60 *by* 60 *matrix* A *with* 1*'s on the main diagonal and in the last column, with* -1*'s below the main diagonal, and with* 0*'s everywhere else, as in (22.4) on p. 165 in the text. Compute the 2-norm condition number of* A*:* `cond(A)`*. Set a random vector* x *of length* 60*:* `x = randn(60,1)`*. Compute* `b = A*x`*.*

1. *Solve the linear system* $Ax = b$ *using Gaussian elimination with partial pivoting by typing* `x_ge = A\b`*. Compute the 2-norm of the difference between the computed vector* `x_ge` *and the true solution* `x` *generated previously.*

2. *Solve the linear system* $Ax = b$ *using the QR factorization of* A*:* `[Q,R] = qr(A); x_qr = R\(Q'*b)`*. Compute the 2-norm of the difference between the computed vector* `x_qr` *and the true solution* `x`*. Explain the difference in accuracy between the two computed solutions* `x_ge` *and* `x_qr`*.*

3. *By hand, factor the* 5 *by* 5 *matrix in (22.4) on p. 165 using* complete *pivoting, so that* $PAQ = LU$*. What is the growth factor* ρ *in (22.2)? Would you expect to be able to solve a* 60 *by* 60 *linear system of this form to high relative accuracy (on a computer that satisfies the usual assumptions of IEEE arithmetic) using Gaussian elimination with complete pivoting? Explain why or why not.*

Solution. I created the following MatLab code to complete this assignment:

```
function hw5q5
record = 0;
%Construct A
A = eye(60);
A(:,60) = ones(60,1);
B = -1*tril(ones(60),-1);
A = A + B;
%Compute condition number of A
cA = cond(A);
%fprintf("The condition nubmer of A is %e \n",cA);

%generate x
x = randn(60,1);

%compute b
b = A*x;

%Method a - Gaussian Elimination with Pivoting
x_ge = A\b;
error_ge = norm(x_ge - x);
rel_ge = norm(x_ge - x)/norm(x);
```

```
%Method b - QR Factorization
[Q,R] = qr(A);
x_qr = R\(Q'*b);
error_qr = norm(x_qr - x);
rel_qr = norm(x_qr - x)/norm(x)

fprintf("Gaussian Elimination Error: %e\n",error_ge)
fprintf("Gaussian Elimination Relative Error: %e\n",rel_ge)
fprintf("QR Factorization Error: %e\n",error_qr)
fprintf("QR Factorization Relative Error: %e\n",rel_qr)
```

This code outputs,

```
Gaussian Elimination Error: 2.502579e+00
Gaussian Elimination Relative Error: 3.015817e-01
QR Factorization Error: 5.243449e-15
QR Factorization Relative Error: 6.318795e-16
```

We can clearly see that QR factorization is significantly more accurate than Gaussian Elimination. We also note that the relative error of QR factorization is very small compared to the condition number of A while the relative error of Gaussian Elimination is roughly proportional to the condition number of A. Thus we can conclude that QR factorization is backward stable and thus more accurate.
Now consider the matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix}.$$

We wish to preform Gaussian elimination with pivoting on A. In our first step, we do not need to pivot and thus we have

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix} \overset{L_1}{\to} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & -1 & 1 & 0 & 2 \\ 0 & -1 & -1 & 1 & 2 \\ 0 & -1 & -1 & -1 & 2 \end{pmatrix}, \quad \text{where} \quad L_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Now we need to pivot to swap the fifth column with the second

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & -1 & 1 & 0 & 2 \\ 0 & -1 & -1 & 1 & 2 \\ 0 & -1 & -1 & -1 & 2 \end{pmatrix} \overset{Q_1}{\to} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 & -1 \\ 0 & 2 & -1 & 1 & -1 \\ 0 & 2 & -1 & -1 & -1 \end{pmatrix}, \quad \text{where} \quad Q_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Now let's eliminate in the second column

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 & -1 \\ 0 & 2 & -1 & 1 & -1 \\ 0 & 2 & -1 & -1 & -1 \end{pmatrix} \xrightarrow{L_2} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & -2 \\ 0 & 0 & -1 & 1 & -2 \\ 0 & 0 & -1 & -1 & -2 \end{pmatrix}, \quad \text{where} \quad L_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{pmatrix}.$$

Next let's pivot to swap the fifth and third columns

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & -2 \\ 0 & 0 & -1 & 1 & -2 \\ 0 & 0 & -1 & -1 & -2 \end{pmatrix} \xrightarrow{Q_2} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -2 & 0 & 1 \\ 0 & 0 & -2 & 1 & -1 \\ 0 & 0 & -2 & -1 & -1 \end{pmatrix}, \quad \text{where} \quad Q_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

Elimination in the third column gives

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & -2 \\ 0 & 0 & -1 & 1 & -2 \\ 0 & 0 & -1 & -1 & -2 \end{pmatrix} \xrightarrow{L_3} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -2 & 0 & 1 \\ 0 & 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & -1 & -2 \end{pmatrix}, \quad \text{where} \quad L_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \end{pmatrix}.$$

Once more let's pivot to interchange the fourth and fifth columns

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -2 & 0 & 1 \\ 0 & 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & -1 & -2 \end{pmatrix} \xrightarrow{Q_3} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & -2 & -1 \end{pmatrix}, \quad \text{where} \quad Q_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Finally let's eliminate in the fourth column

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & -2 & -1 \end{pmatrix} \xrightarrow{L_4} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}, \quad \text{where} \quad L_4 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

Collecting all our terms we have that

$$L_4 L_3 L_2 L_1 A Q_1 Q_2 Q_3 = LAQ = U,$$

where

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 & 0 \\ -1 & 1 & 1 & 1 & 0 \\ -1 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad Q = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}.$$

We can find the max growth factor its

$$\rho = \frac{\max_{i,j} |u_{ij}|}{\max i, j |a_{ij}|} = \frac{2}{1} = 2.$$

Since ρ is of first order and the matrix A is well conditioned, I would expect that Gaussian Elimination with complete pivoting can be performed with high relative accuracy.

□