CSC 336F ASSIGNMENT #3 Due: 23 November 2018.

This assignment is due at the **start** of the class on Friday, 23 November 2019.

For the questions that require you to write a MatLab program, hand-in the program and its output as well as any written answers requested in the question. Your program and its output, as well as your written answers, will be marked. Your program should conform to the usual CS standards for comments, good programming style, etc. When first learning to program in MatLab, students often produce long, messy output. Try to format the output from your program so that it is easy for your TA to read and to understand your results. To this end, you might find it helpful to read "A short description of fprintf" on the course webpage. Marks will be awarded for well-formatted, easy-to-read output.

1. [15 marks; 5 marks for each part]

   Consider the family of linear systems $A_\gamma x = b$, where

$$A_\gamma = \begin{pmatrix} \gamma & 1-\gamma \\ 1 & 1 \end{pmatrix} \qquad b = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{1}$$

   and $\gamma \in \mathbb{R}$. The solution of this system is

$$x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

   for all $\gamma \in \mathbb{R}$.

   This question shows that, if $|\gamma|$ is small, you need to pivot when solving such systems. It also shows that, even if the LU factorization is inaccurate, you can use this inaccurate LU factorization together with *iterative refinement* (see part (c) below) to compute an accurate solution to the linear system $Ax = b$. You can find a description of iterative refinement in Section 2.4.10 of the Heath textbook. I probably won't discuss iterative refinement in class. So, you will need to read the Heath textbook. In case you don't have a copy of the textbook, I've put Section 2.4.10 (i.e., pages 83–84) on the course webpage.

   (a) If you don't pivot and $\gamma \neq 0$, then the LU factorization of A_γ is

$$L_\gamma^{(1)} = \begin{pmatrix} 1 & 0 \\ 1/\gamma & 1 \end{pmatrix} \qquad U_\gamma^{(1)} = \begin{pmatrix} \gamma & 1-\gamma \\ 0 & 2-1/\gamma \end{pmatrix}$$

   That is, $A_\gamma = L_\gamma^{(1)} U_\gamma^{(1)}$.

   For $\gamma = 10^{-2k}$, $k = 1, 2, \ldots, 10$, use MatLab and this LU factorization to compute the solution of $A_\gamma x = b$. That is, use the MatLab backslash operator \ first to solve $L_\gamma^{(1)} y = b$ for y in MatLab by $y = L_\gamma^{(1)} \setminus b$ and then to solve $U_\gamma^{(1)} x = y$ for x in MatLab by $x = U_\gamma^{(1)} \setminus y$.

Call the computed solution $\hat{x}$ and call the exact solution x. For each $k = 1, 2, \ldots, 10$, print $\gamma = 10^{-2k}$ and both components, error(1) and error(2), of the error vector, error $= \hat{x} - x$.

How does the accuracy of the computed solution behave as γ decreases?

(b) If you do pivot, the LU factorization of A_γ is

$$P_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad L_\gamma^{(2)} = \begin{pmatrix} 1 & 0 \\ \gamma & 1 \end{pmatrix} \qquad U_\gamma^{(2)} = \begin{pmatrix} 1 & 1 \\ 0 & 1-2\gamma \end{pmatrix}$$

That is, $P_2 A_\gamma = L_\gamma^{(2)} U_\gamma^{(2)}$.

For $\gamma = 10^{-2k}$, $k = 1, 2, \ldots, 10$, use MatLab and this LU factorization to compute the solution of $A_\gamma x = b$. That is, first compute $\tilde{b} = P_2 b$ in MatLab, next solve $L_\gamma^{(2)} y = \tilde{b}$ for y in MatLab by $y = L_\gamma^{(2)} \setminus \tilde{b}$ and then solve $U_\gamma^{(2)} x = y$ for x in MatLab by $x = U_\gamma^{(2)} \setminus y$.

Call the computed solution $\hat{x}$ and call the exact solution x. For each $k = 1, 2, \ldots, 10$, print $\gamma = 10^{-2k}$ and both components, error(1) and error(2), of the error vector, error $= \hat{x} - x$.

How does the accuracy of the computed solution behave as γ decreases?

What do the solutions of parts (a) and (b) tell you about the importance of pivoting in solving linear systems?

(c) Read Section 2.4.10 of the Heath textbook on "Improving Accuracy" using *iterative refinement*. In case you don't have a copy of the textbook, I've put Section 2.4.10 (i.e., pages 83–84) on the course webpage.

Repeat part (a) using the LU factorization $A_\gamma = L_\gamma^{(1)} U_\gamma^{(2)}$ computed without pivoting, but this time use one iteration of iterative refinement to improve the solution. Compute the residual in MatLab as $r = b - A_\gamma \hat{x}$, where $\hat{x}$ is the computed solution from part (a) and A_γ is the matrix given in (1). Note that, when you compute the residual, r, you should use the matrix A_γ in (1), not the LU factorization of A_γ.

In your implementation of iterative refinement, you must solve $A_\gamma e = r$ for an approximation e to the error in your computed solution. Use MatLab and the inaccurate LU factorization $A_\gamma = L_\gamma^{(1)} U_\gamma^{(1)}$ to solve this system. That is, first solve $L_\gamma^{(1)} z = r$ for z in MatLab by $z = L_\gamma^{(1)} \setminus r$ and then solve $U_\gamma^{(1)} e = z$ for e in MatLab by $e = U_\gamma^{(1)} \setminus z$. Then compute in MatLab the "better" approximate solution $\tilde{x} = \hat{x} + e$, where $\hat{x}$ is the computed solution from part (a).

For each $k = 1, 2, \ldots, 10$, print $\gamma = 10^{-2k}$ and both components, error(1) and error(2), of the error vector, error $= \hat{x} - x$.

How does the accuracy of the "better" approximate solution $\tilde{x}$ behave as γ decreases?

What do the solutions of parts (a) and (c) tell you about the effectiveness of iterative refinement in this context?

2. [15 marks; 5 marks for each part]

Consider the matrix

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

This is a famous example that shows that, in some cases, *complete pivoting* can be much more effective than *row partial-pivoting* in reducing the growth of elements during LU factorization. Read pages 75 and 76 of your textbook for a description of complete pivoting and the difference between complete pivoting and row partial-pivoting. In case you don't have a copy of the textbook, I've put pages 75–76 on the course webpage.

(a) Calculate by hand the matrices P, L and U that you obtain if you compute the LU factorization of A with row partial-pivoting. (This is the factorization $PA = LU$ that we are talking about in class.) In doing this factorization, you should never have to pivot. Hence, you should find that $P = I$ in this case. So, the L and U that you calculate should satisfy $A = LU$.

Show all your calculations.

(b) Calculate by hand the matrices P, Q, L and U that you obtain if you apply LU factorization with complete pivoting to the matrix A above. (This is the factorization $PAQ = LU$ discussed on page 75 of your textbook.) In doing this factorization, at each stage k, just before you zero the elements below the main diagonal in column k, you should interchange columns k and 5. (Do this interchange for column 1 also, even though it is not really required there.) You should never interchange any rows. Therefore, the P, Q, L and U that you calculate should satisfy $P = I$ and $AQ = LU$.

Show all your calculations.

(Note: the largest element in U should be smaller in this case than in part (a) above.)

(c) To see the ill effects of the element growth that can occur when you perform an LU factorization with row partial-pivoting, we need to consider a larger example than the one above to allow the elements to grow more.

To this end, let $n = 60$ and construct an $n \times n$ matrix similar to A above with the MatLab commands

```
A = ones(n,n);
A = A – triu(A);
A = eye(n) – A;
A = A + [ones(n–1,1); 0] * [zeros(1,n–1),1];
```

and an $n \times n$ matrix similar to Q from part (b) with the MatLab commands

Q = diag(ones(n–1,1),1);
Q(n,1) = 1;

You can compute the LU factorization with row partial-pivoting of this $n \times n$ matrix A with the MatLab command

[L1, U1, P1] = lu(A)

(Read "help lu" in MatLab.) Print U1(n,n) and verify that U1(n,n)= 2^{n-1}. Thus, we see an exponential growth in the elements of U1 with respect to n, the size of the matrix, even though all the multipliers used in the LU factorization process are of magnitude 1 or less.

You can compute the LU factorization with complete pivoting for the same $n \times n$ matrix A with the MatLab command

[L2, U2] = lu(A*Q)

(Read "help lu" in MatLab.) You can compute the largest element of U2 with the MatLab command max(max(abs(U2))). Verify that this value is 2, just as it was for the smaller version of the matrix A in part (b) above. Thus, the LU factorization with complete pivoting does not suffer from the exponential growth of elements that we saw above in the LU factorization with row partial-pivoting.

To see the ill effect of this exponential growth in the elements of U1, let x = ones(n,1) and $b = Ax$. Solve the system $Ax = b$ using the matrices computed by the lu factorization with row partial-pivoting by executing the MatLab commands

y = L1 \ b
x1 = U1 \ y

(You don't have to use $P1$ above because $P1 = I$.) Compute and print norm(x – x1, inf), where x is the exact solution of $Ax = b$.

Note that x1 is a very poor approximation to x. How poor is it?

Also, solve the system $Ax = b$ using the matrices computed by the lu factorization with complete pivoting by executing the MatLab commands

y = L2 \ b
z = U2 \ y
x2 = Q * z

(You don't need a P matrix in this case because $P = I$.) Compute and print norm(x – x2, inf), where x is the exact solution of $Ax = b$.

Note that x2 is a good approximation to x. How much better is x2 than x1?

3. [20 marks: 5 marks for each part]

Consider the matrix

$$A = \begin{pmatrix} -1 & 3 & 2 \\ 2 & -4 & -2 \\ 1 & 1 & -1 \end{pmatrix}$$

(a) Use partial pivoting to compute the LU factorization of A. That is, compute the 3×3 permutation matrix P, the 3×3 unit-lower-triangular matrix L with $|L_{ij}| \leq 1$ for $i > j$, and the 3×3 upper-triangular matrix U such that $PA = LU$. Show all your calculations.

(b) Use the LU factorization of A computed in part (a) to solve the linear system $Ax = b$, where

$$b = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Show all your calculations.

(c) Suppose we change the (3,3) element of A from -1 to 0 to yield a new matrix

$$\hat{A} = \begin{pmatrix} -1 & 3 & 2 \\ 2 & -4 & -2 \\ 1 & 1 & 0 \end{pmatrix}$$

Note that all the elements of A and $\hat{A}$ are equal except for the (3,3) element.

Find two vectors u and v such the $\hat{A} = A - uv^T$. Thus, A and $\hat{A}$ differ by a rank 1 update.

(Note: u and v are not unique. If you find one pair of vectors u and v such that $\hat{A} = A - uv^T$, then, for any $\alpha \neq 0$, if you let $\hat{u} = \alpha u$ and $\hat{v} = v/\alpha$, you also have that $\hat{A} = A - \hat{u}\hat{v}^T$.)

(d) Read Section 2.4.9 (i.e., pages 81–83) of your textbook and then use the Sherman-Morrison formula

$$(A - uv^T)^{-1} = A^{-1} + \frac{A^{-1}uv^TA^{-1}}{1 - v^TA^{-1}u} \tag{2}$$

to solve $\hat{A}\hat{x} = b$, where $\hat{A}$ is the matrix in part (c) and b is the vector in part (b).

In case you don't have a copy of the textbook, I've put Section 2.4.9 (i.e., pages 81–83) on the course webpage.

Do not compute any matrix inverses explicitly in the Sherman-Morrison formula (2). Instead, use the LU factorization from part (a) whenever you need to solve a linear system with the matrix A.

In addition, organize your computation so that there are no matrix-matrix multiplies or solves. That is, if all the vectors in this question were n-vectors and all

the matrices were $n \times n$ matrices, you should be able to solve the system $\hat{A}\hat{x} = b$ in time proportional to n^2, not time proportional to n^3.

Show all your calculations.

4. [15 marks: 5 marks for each part]

We will talk in class about using a vector $p = [p_1, p_2, \ldots, p_{n-1}]$ to represent the $n \times n$ elementary permutation matrices $P_1, P_2, \ldots, P_{n-1}$ that we use in Gaussian Elimination with row-partial-pivoting (i.e., the type of pivoting that we discussed in class and is described in Section 2.4.5 of your textbook). Recall that, in this notation, if $y = P_k\, x$, then y is the same as x, except that the elements x_k and x_j are interchanged, for some $j \geq k$. That is,

$$\begin{aligned} y_i &= x_i \quad \text{if } i \in \{1, 2, \ldots, n\} \text{ and } i \notin \{k, j\} \\ y_k &= x_j \\ y_j &= x_k \end{aligned}$$

If $j = k$, then this interchange effectively does nothing and so $P_k = I$ and $y = x$. However, if $j > k$, then $y \neq x$, unless (by chance) $x_k = x_j$.

We represent the elementary permutation matrix P_k in the vector p by setting $p_k = j$, since all you really need to know about P_k is that it is a permutation matrix that does nothing except interchange elements k and j, where $j = p_k$, when you perform the matrix-vector multiply $P_k\, x$.

For example, the vector $p = [3, 2, 4]$ represents the three 4×4 elementary permutation matrices

$$P_1 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad P_2 = I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad P_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

When we compute the LU factorization of a matrix using row-partial-pivoting, we also need to represent the $n \times n$ permutation matrix P, where

$$P = P_{n-1} P_{n-2} \cdots P_2 P_1$$

One way to represent an $n \times n$ permutation matrix P is with a n-vector q for which $q_i = j$ if and only if $P_{ij} = 1$ (i.e., row i of P has a 1 in column j). Since a permutation matrix has exactly one 1 in each row (and all other elements are zero), this representation is very effective. For example, the 5×5 permutation matrix

$$P = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

can be represented by the vector $q = [3, 4, 5, 1, 2]$.

(a) Write a MatLab function

```
function y = perm_a(p,x)
```

that takes as arguments

- an $(n-1)$-vector p that represents the $n \times n$ elementary permutation matrices $P_1, P_2, \ldots, P_{n-1}$ as described above
- an n-vector x

and computes the n-vector y satisfying

$$y = P_{n-1}P_{n-2}\cdots P_2P_1x$$

Your function should not compute the matrices $P_1, P_2, \ldots, P_{n-1}$; it should use p only to compute y from x. In particular, it should compute y in time proportional to n.

(b) Write a MatLab function

```
function q = perm_b(p)
```

that takes as its argument

- an $(n-1)$-vector p that represents the $n \times n$ elementary permutation matrices $P_1, P_2, \ldots, P_{n-1}$ as described above, and

and computes the n-vector q that represents the permutation matrix P satisfying

$$P = P_{n-1}P_{n-2}\cdots P_2P_1$$

Your function should not compute the matrices $P_1, P_2, \ldots, P_{n-1}$ or P; it should use p only to compute q. In particular, it should compute q in time proportional to n.

(c) Write a MatLab function

```
function y = perm_c(q,x)
```

that takes as arguments

- an n-vector q that represents the $n \times n$ permutation matrix P as described above, and
- an n-vector x

and computes the n-vector y satisfying

$$y = Px$$

Your function should not compute the matrix P; it should use q only to compute y from x. In particular, it should compute y in time proportional to n.

To test your functions `perm_a`, `perm_b` and `perm_c`, let

$$\texttt{p} = [5, 4, 9, 10, 6, 8, 10, 9, 10]$$

and

$$x = [1 : 10]'$$

(I.e., p is row vector with 9 elements and x is a column vector with 10 elements.)

Compute

```
y1 = perm_a(p,x)
q  = perm_b(p)
y2 = perm_c(q,x)
```

Hand in your MatLab program, the functions `perm_a`, `perm_b` and `perm_c` and their output for the test case above.

5. [10 marks: 5 marks for each part]

Do question 5 on last year's final exam. You can find the final exam on the course webpage `http://www.cs.toronto.edu/~krj/courses/336/`

In addition to answering question 5 on last year's final exam, implement your

```
function p = perm_d(q)
```

in MatLab and test it by computing

```
p1 = perm_d(q)
```

where `q` is the vector that you computed from the vector `p` in your tests in question 4 above. That is,

```
q  = perm_b(p)
```

where

$$\texttt{p} = [5, 4, 9, 10, 6, 8, 10, 9, 10]$$

Also, when you do part (b) of question 5, prove that the hint given at the end of the question is true. That is, for an elementary permutation matrix P_k described at the start of question 5, prove that

$$\det(P_k) = \begin{cases} +1 & \text{if } P_k = I \\ -1 & \text{if } P_k \neq I \end{cases}$$

Hand in your written answers to this question, your function `perm_d` and the output from the test case above.