Gaussian Elimination and LU-Decomposition

Gary D. Knott
Civilized Software Inc.
12109 Heritage Park Circle
Silver Spring MD 20906
phone:301-962-3711
email:knott@civilized.com
URL:www.civilized.com

November 5, 2013

1 Gaussian Elimination and LU-Decomposition

Solving a set of linear equations arises in many contexts in applied mathematics. At least until recently, a claim could be made that solving sets of linear equations (generally as a component of dealing with larger problems like partial-differential-equation solving, or optimization, consumes more computer time than any other computational procedure. (Distant competitors would be the Gram-Schmidt process and the fast Fourier transform computation, and the Gram-Schmidt process is a first cousin to the Gaussian elimination computation since both may be used to solve systems of linear equations, and they are both based on forming particular linear combinations of a given sequence of vectors.) Indeed, the invention of the electronic digital computer was largely motivated by the desire to find a labor-saving means to solve systems of linear equations [Smi10].

Often the subject of linear algebra is approached by starting with the topic of solving sets of linear equations, and Gaussian elimination methodology is elaborated to introduce matrix inverses, rank, nullspaces, etc.

We have seen above that computing a preimage vector $x \in \mathcal{R}^n$ of a vector $v \in \mathcal{R}^k$ with respect to the $n \times k$ matrix A consists of finding a solution $(x_1, \ldots, x_n)$ to the k linear equations:

$$\begin{aligned} A_{11}x_1 + A_{21}x_2 + \cdots + A_{n1}x_n &= v_1 \\ A_{12}x_1 + A_{22}x_2 + \cdots + A_{n2}x_n &= v_2 \\ &\vdots \\ A_{1k}x_1 + A_{2k}x_2 + \cdots + A_{nk}x_n &= v_k. \end{aligned}$$

This corresponds to $xA = v$.

If $v \in \mathcal{R}^k - \mathit{rowspace}(A)$ then there are no solutions x; the equations $xA = v$ are inconsistent. For example, $[1x_1 + 2x_2 = 0, 2x_1 + 4x_2 = 1]$. This is because $xA = x_1(A \text{ row } 1) + \cdots + x_n(A \text{ row } n) \in \mathit{rowspace}(A) \subseteq \mathcal{R}^k$ for every $x \in \mathcal{R}^n$.

Recall that $\mathit{nullspace}(A) = \{x \in \mathcal{R}^n \mid xA = 0\}$ with $\mathit{dim}(\mathit{nullspace}(A)) = n - \mathit{rank}(A)$. If $\mathit{nullspace}(A) = \{0\}$ and $v \in \mathit{rowspace}(A)$, then the linear equations $xA = v$ have the unique solution $x = vA^+$, where the $k \times n$ matrix A^+ is the Moore-Penrose pseudo-inverse matrix of A. Necessarily $k \geq n$; in case $n = k$, A is non-singular and $A^+ = A^{-1}$ so $x = vA^{-1}$. The vector vA^+ always belongs to $\mathit{colspace}(A)$ regardless of the choice of v or the dimension of $\mathit{nullspace}(A)$.

More generally, if $\mathit{dim}(\mathit{nullspace}(A)) \geq 0$ and $v \in \mathit{rowspace}(A)$, there is a $\mathit{dim}(\mathit{nullspace}(A))$-dimensional flat of solutions x. The vectors in $\mathit{nullspace}(A) + vA^+ \subseteq \mathcal{R}^n$ comprise all the solution vectors, x, that satisfy $xA = v$. The matrix A corresponds to a mapping that maps the family of parallel $(n - \mathit{rank}(A))$-dimensional flats $\{\mathit{nullspace}(A) + y \mid y \in \mathit{colspace}(A)\}$ covering $\mathcal{R}^n$ to points in $\mathit{rowspace}(A) \subseteq \mathcal{R}^k$; this flat-to-point mapping is one-to-one and onto.

Geometrically, with $\mathit{rank}(A) = r$ and $v \in \mathit{rowspace}(A)$, we have r linearly-independent hyperplanes defined by $(x, A \text{ col } j_1) = v_{j_1}$, ..., $(x, A \text{ col } j_r) = v_{j_r}$ where $A \text{ col } j_1, \ldots, A \text{ col } j_r$ are linearly-independent columns of A; these hyperplanes intersect in an $(n - r)$-dimensional flat in $\mathcal{R}^n$; this flat is the translation of $\mathit{nullspace}(A)$: $vA^+ + \mathit{nullspace}(A)$.

In general, the matrix A^+A is the $k \times k$ projection matrix onto $\mathit{rowspace}(A) \subseteq \mathcal{R}^k$, and for any vector $v \in \mathcal{R}^k$, the vector vA^+ is the unique vector in $\mathit{colspace}(A)$ such that $|v - vA^+A|$ is minimal; moreover vA^+ is a shortest minimizing vector in $\mathcal{R}^n$.

We often wish to determine which of these cases (no solution, unique solution, multiple solutions) hold for a given $n \times k$ matrix A and a given righthand-side vector $v \in \mathcal{R}^k$, and when $v \in \mathit{rowspace}(A)$, we wish to compute a solution vector $x = vA^+$ without the expense of computing the Moore-Penrose pseudo-inverse matrix A^+. The classic step-wise approach to computing x is to add a multiple of one equation to another at each step until the system of equations is reduced to a form which is easy to either solve or to see that no solution or no unique solution exists. This process is called *Gaussian elimination*, since we generally aim to eliminate successive variables from successive equations by simple algebraic modifications as was proceduralized by C. F. Gauss [Grc11a].

The form that is most commonly sought is a *triangular* system of equations. We will usually only need to deal with such a triangular system in the case where $n = k$ and we have a unique solution vector x, *i.e.*, where we have a consistent system of equations with $n = k$, and the matrix of coefficients is non-singular. In this case, we can obtain:

$$\begin{array}{rcrcrcrcrcl}
L_{11}x_1 & + & L_{21}x_2 & + & \cdots & + & L_{n-1,1}x_{n-1} & + & L_{n1}x_n & = & y_1 \\
 & & L_{22}x_2 & + & \cdots & + & L_{n-1,2}x_{n-1} & + & L_{n2}x_n & = & y_2 \\
 & & & & & & \vdots & & & & \\
 & & & & & & L_{n-1,n-1}x_{n-1} & + & L_{n,n-1}x_n & = & y_{n-1} \\
 & & & & & & & & L_{nn}x_n & = & y_n
\end{array}$$

This corresponds to $xL = y$ where L is an $n \times n$ lower-triangular matrix. Let us assume there is a unique solution, so L must be non-singular, and thus L_{ii} is necessarily non-zero for $i = 1, \ldots, n$. If

we have such a triangular form, then we have one equation involving just x_n, one involving just x_n and x_{n-1}, and so on, and it is easy to compute the solution vector x. The process of computing the solution is called "back-substitution." An algorithm for solving $xL = y$ with back-substitution is given below. Note if we try to divide by zero, this algorithm will fail; this is why we require $L_{ii} \neq 0$ for $1 \leq i \leq n$.

[for $i = n, n-1, \ldots, 1:$ $(x_i \leftarrow y_i$; for $j = i+1, \ldots, n : (x_i \leftarrow x_i - L_{ji}x_j)$; $x_i \leftarrow x_i/L_{ii})$].

Exercise 1.1: State the back-substitution algorithm that applies when we have a non-singular $n \times n$ upper-triangular matrix U with $xU = y$, so that we have one equation involving just x_1, one involving just x_1 and x_2, and so on.

Exercise 1.2: Write-out the matrix products xL and $L^{\mathrm{T}}x^{\mathrm{T}}$ and compare them.

Exercise 1.3: Let A be a $1 \times k$ matrix. What is the lower-triangle of A?

Exercise 1.4: Let L be an $n \times n$ lower-triangular matrix. Why must $L_{ii} \neq 0$ for $1 \leq i \leq n$ when L is non-singular?

Exercise 1.5: Show that the back-substitution algorithm given above uses n divisions, $n(n-1)/2$ multiplications and subtractions, and $n(n+3)/2$ assignment operations to compute the vector x.

Exercise 1.6: When is the inverse of a non-singular $n \times n$ triangular matrix, L, itself triangular? Hint: consider determining the i-th row of L^{-1} by solving $xL = e_i$.

Exercise 1.7: Show that if the i-th column of an $n \times n$ matrix L has $L_{ji} = 0$ then $(L \text{ col } i)^{\mathrm{T}}$ is normal to the natural basis vector e_j. Thus when L is a lower-triangular matrix, $(L \text{ col } i)^{\mathrm{T}}$ is normal to e_1, e_2, $\ldots$, e_{i-1} for $i = 2, \ldots, n$.

For the $n \times k$ matrix A with $A \neq 0$, we can approach solving the system of k equations with n unknowns $xA = v$ as follows. For each variable x_i with $i = 1, 2, \ldots, n$, select an "eligible" equation $(x, A \text{ col } j) = v_j$ with $A_{ij} \neq 0$; this equation is called the *pivot equation* and the coefficient A_{ij} is called the *pivot value* for the elimination of x_i. Solve this equation for x_i, and then use this expression for x_i to *eliminate* x_i in *all* the other equations.

Specifically, given $x_i = \sum_{\substack{1 \leq h \leq n \\ h \neq i}} -\frac{A_{hj}}{A_{ij}} x_h$, we eliminate x_i from the equation $(x, A \text{ col } p) = v_p$ by substituting $\sum_{\substack{1 \leq h \leq n \\ h \neq i}} -\frac{A_{hj}}{A_{ij}} x_h$ for x_i in $(x, A \text{ col } p) = v_p$ to obtain the equation

$$(x, [A \text{ col } p] - \frac{A_{ip}}{A_{ij}}[A \text{ col } j]) = v_p - \frac{A_{ip}}{A_{ij}} v_j,$$

in which the variable x_i has been eliminated, (*i.e.*, the coefficent of x_i is zero.) Note this transformation replaces equation p with the sum of equation p and a multiple of equation j. (This transformation applied to all equations p with $p \neq j$ is called a *full-elimination* or *full-pivoting*

iteration.) We then make equation j "ineligible" and proceed to try to eliminate the next variable, x_{i+1} from all but one of our equations (both eligible and ineligible,) continuing in this way until there are no eligible equations left or no variables appearing in multiple equations left to eliminate.

If we can succeed in eliminating each variable from all but one of our equations, and if each final equation thus obtained contains no more than one variable, distinct from the variables in all the other equations, then we have "diagonalized" our system of equations; each equation is now either of the form "$0 = v'_j$" or is of the form "$A'_{ij}x_i = v'_j$," with $A'_{ij} \neq 0$. If we have any equation of the form "$0 = v'_j$" with $v'_j \neq 0$, then our equations are inconsistent. Otherwise, we can easily obtain values of $x_1, \ldots, x_n$ that satisfy $xA = v$. (We have $x_i = v'_j/A'_{ij}$ when $A'_{ij} \neq 0$, and we may take x_i to be an arbitrary value when $A'_{ij} = 0$, *i.e.*, when the " equation for x_i" is $0 = 0$.)

> **Exercise 1.8:** Is the qualifying clause "distinct from the variables in all the other equations" in the prior paragraph superfluous? Hint: yes.

This method of eliminating variables by forming linear combinations of the originally-given equations, as well as variant methods which do not achieve a full "diagonalized" system of equations, are all called *Gaussian elimination*.

If we have more equations than unknowns ($n < k$,) then either these k equations are inconsistent, or at least $k - n$ of these equations are linear combinations of the other equations. If exactly n of the k consistent equations are linearly-independent, there is a unique solution; otherwise we have fewer than n independent equations and we have an infinite number of solutions. If we have at least as many unknowns as equations ($n \geq k$,) then, unless these equations are inconsistent, there is always at least one solution, and when $n > k$ there are an infinite number of solutions. (There may also be an infinite number of solutions when $n = k$ and there are fewer than n linearly-independent equations.)

> **Exercise 1.9:** What do we mean by an "eligible" equation? Why do we make an equation "ineligible" after we use it to eliminate a variable from all the other equations? (Try using the same equation in two steps to eliminate two distinct variables x_1 and x_2 from all the other equations in an example such as $[x_1 + x_2 + x_3 = 3, x_1 - x_2 - x_3 = 1, x_1 + x_2 - x_3 = 2]$.)

Alternately, we can try to "reduce" our system of equations $xA = v$ to triangular form rather than diagonal form by "partially" eliminating each variable from our equations, *i.e.*, variable x_i is eliminated from $k - i$ equations when possible. This triangular form can be as useful as the diagonal form, and even more so when $xA = v$ does not have a unique solution, and it can also be used to solve multiple systems of equations with distinct righthand-sides, with just two back-substitution steps for each such system. This is achieved by using Gaussian elimination as sketched above, but only eliminating the i-th variable x_i from the $k - i$ currently-"eligible" equations, not including the current pivot equation. This transformation is called *tail-elimination*. (What we mean by 'triangular' is that one equation contains only 1 variable, one equation contains at most 2 variables, and so on. Note, in general, neither "true" diagonal or triangular forms of coefficient matrix can be obtained unless we renumber, *i.e.*, permute our variables and/or suitably order our equations.)

It is important to note that, depending on the order of our equations and the numbering of our variables, constructing either a triangular or diagonal form may encounter very small pivot values

and this may introduce excessive round-off error in the solution of the associated set of linear equations; we may use a "pivot search" for a large pivot value during each iteration to mitigate this danger. (Large pivot values are less problematic with respect to error.) The "partial" elimination process to obtain a triangular form using round-off error-reduction measures is the subject to be addressed below.

In matrix form, adding a multiple of one equation to another in the system of equations $xA = v$ consists of adding a multiple of one column of the $n \times k$ matrix A to another, and at the same time adding that multiple of the same (single element) column of v to the other corresponding (single element) column of v. (We manipulate columns rather than rows because we take vectors to be rows rather than columns and we apply matrices to vectors by multiplying on the right.) This can be nicely organized, when desired, by appending v to the matrix A as an additional row.

As an example, consider the equations

$$\begin{array}{lrcl} [1] & 0 \cdot x_1 + 1 \cdot x_2 + 2 \cdot x_3 & = & 2 \\ [2] & -1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 & = & -1 \\ [3] & 1 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 & = & 0 \\ [4] & 0 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 & = & 1 \end{array} .$$

In matrix form these equations are $xA = v$ where $x = (x_1, x_2, x_3)$, $A = \begin{bmatrix} 0 & -1 & 1 & 0 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 3 \end{bmatrix}$ and $v = (2, -1, 0, 1)$.

If we select the first pivot equation for x_1 to be equation 2 with the pivot value -1, (*i.e.*, the coefficient of x_1 in equation 2,) then eliminating x_1 in all the other equations yields:

$$(x_1, x_2, x_3) \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 1 & 3 \end{bmatrix} = (2, -1, -1, 1).$$

Now selecting the second pivot equation for x_2 to be equation 1 with the pivot value 1, eliminating x_2 in the remaining eligible equations (equations 3 and 4,) yields:

$$(x_1, x_2, x_3) \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & -1 & -1 \end{bmatrix} = (2, -1, -3, -3).$$

Now we may select equation 3 to be our third pivot equation with the pivot element -1 corresponding to the variable x_3, and we may eliminate x_3 from the only remaining eligible equation (equation 4,) to obtain:

$$(x_1, x_2, x_3) \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & -1 & 0 \end{bmatrix} = (2, -1, -3, 0).$$

Now we may reorder our equations to obtain the triangular form:

$$(x_1,\, x_2,\, x_3) \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 2 & -1 & 0 \end{bmatrix} = (-1, 2, -3, 0).$$

And a back-substitution computation shows that $x_3 = 3$, and then $x_2 = 2 - 2 \cdot x_3 = -4$, and then $x_1 = 1 - 0 \cdot x_2 + 1 \cdot x_3 = 4$. Note that if our original equation 4 had been $0 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 = 2$, our reduced equation 4 would have been $0 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 = 1$, and our system of equations would have been inconsistent.

Also note that, rather than reorder our equations, (*i.e.*, permuting the columns of our reduced matrix of coefficients,) we could have re-labeled our variables: writing x_1 for x_2 and x_2 for x_1, (and leaving x_3 alone,) and permute the rows of our coefficient matrix accordingly to obtain

$$(x_1,\, x_2,\, x_3) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 2 & 1 & -1 & 0 \end{bmatrix} = (2, -1, -3, 0).$$

And in this case, we must remember that x_1 is the "original" x_2, and x_2 is the "original" x_1.

Exercise 1.10: Why do we say *two* back-substitution steps are required to solve each of a sequence of linear systems with differing righthand-sides? (If you can answer this question, you have probably seen the LU-decomposition idea before.)

Adding a multiple of one column of a matrix to another column can be effected by multiplying on the right by a suitable *elementary* matrix E. Define $E_k[i, j, \alpha]$ to be the $k \times k$ matrix $I + \alpha e_j^{\mathrm{T}} e_i$ where e_j is the k-vector $(0, \ldots, 0, 1, 0, \ldots, 0)$ with each component equal to 0 except component j which is 1. Now, for any k-column matrix A, $AE_k[i, j, \alpha]$ is the same matrix as A except that column i is replaced by $(A \text{ col } i) + \alpha(A \text{ col } j)$. The matrix $E_k[i, j, \alpha]$ is called an *elementary matrix*.

Exercise 1.11: Show that $e_j^{\mathrm{T}} e_i$ is the $k \times k$ matrix each of whose elements is 0 except component $[j, i]$ which is 1. Note $rank(e_j^{\mathrm{T}} e_i) = 1$.

Exercise 1.12: Show that $E_k[i, j, \alpha]^{\mathrm{T}} = E_k[j, i, \alpha]$. Thus the transpose of an elementary matrix is an elementary matrix.

Exercise 1.13: A suitable comformable elementary matrix can also be used to add a multiple of a row of an $n \times k$ matrix A to another row of A. Show that $E_n[i, j, \alpha]A$ is the same matrix as A except that row j is replaced by $(A \text{ row } j) + \alpha(A \text{ row } i)$.

Exercise 1.14: Show that $E_k[i, i, \alpha - 1] = I$ except the $[i, i]$ element is α. Show that $B = AE_k[i, i, \alpha - 1]$ has the same columns as A except B col $i = \alpha(A \text{ col } i)$.

Exercise 1.15: Show that, if $\alpha = 0$ or $i \neq j$ then $E_k[i, j, \alpha]^{-1} = E_k[i, j, -\alpha]$, if $\alpha \neq -1$ and $i = j$ then $E_k[i, j, \alpha]^{-1} = E_k[i, i, -\alpha/(1+\alpha)]$, and if $\alpha = -1$ and $i = j$ then $E_k[i, j, \alpha]$ is singular.

Exercise 1.16: Let the $k \times k$ matrix $S = E_k[i, j, -1]E_k[j, i, 1]E_k[i, j, -1]E_k[i, i, -2]$, where $1 \leq i \leq k$ and $1 \leq j \leq k$ with $i \neq j$. Show that $B = AS$ has the same columns as A except B col $i = A$ col j and B col $j = A$ col i, where $i \neq j$. What does right-multiplication by S do if $i = j$?

Recall that $transpose_k(i,j)$ denotes the permutation $\langle 1,\ldots,j,\ldots,i,\ldots,k\rangle$ where component $t=t$, except component $i=j$ and component $j=i$. The $k\times k$ column-permutation matrix corresponding to $transpose_k(i,j)$ is the matrix I col $transpose_k(i,j)$; from the exercise above, this is: $E_k[i,j,-1]\ E_k[j,i,1]\ E_k[i,j,-1]\ E_k[i,i,-2]$ when $i\neq j$. The $n\times n$ row-permutation matrix corresponding to $transpose_n(i,j)$ is the matrix I row $transpose_n(i,j)$; when $i\neq j$, this is the matrix $(E_n[i,j,-1]\ E_n[j,i,1]\ E_n[i,j,-1]\ E_n[i,i,-2])^{\mathrm{T}}$. When $i=j$, I col $transpose_k(i,i)=E_k[i,i,0]$ and I row $transpose_n(i,i)=E_n[i,i,0]$.

Note that since any transposition permutation matrix can be expressed as a product of elementary matrices and every permutation can be expressed as a composition of transpose permutations, any permutation matrix can be expressed as a product of elementary matrices.

It is convenient to define the $k\times k$ matrix $G_k[i,w]=I+e_i^{\mathrm{T}}w$, where $w\in\mathcal{R}^k$. The matrix $G_k[i,w]$ is called a *column-operation Gauss matrix*. Note $(AG_k[i,w])$ col $j=(A$ col $j)+w_j(A$ col $i)$ for $1\leq j\leq k$ where $colsize(A)=k$.

Exercise 1.17: Show that $G_k[i,w]$ row $i=e_i+w$.

Exercise 1.18: Show that $G_k[i,w]=E_k[1,i,w_1]E_k[2,i,w_2]\cdots E_k[k,i,w_k]$.

Exercise 1.19: Show that all the matrices $E_k[1,i,w_1],\ldots,E_k[i-1,i,w_{i-1}]$, $E_k[i+1,i,w_{i+1}]$, $\ldots$, $E_k[k,i,w_k]$ commute with one-another, but not necessarily with $E_k[i,i,w_i]$.

Exercise 1.20: Show that if $w_i=0$, then $G_k[i,w]^{-1}=G_k[i,-w]=I-e_i^{\mathrm{T}}w$.

Exercise 1.21: Let A be a $k\times m$ matrix. Show that $(G_k[i,w]^{\mathrm{T}}A)$ row $j=(A$ row $j)+w_i(A$ row $i)$. (The transpose of a column-operation Gauss matrix is called a *row-operation* Gauss matrix.)

Exercise 1.22: Let A be a $k\times k$ matrix with $A_{ri}\neq 0$.
Let $w=\left(\frac{-A_{r1}}{A_{ri}},\ldots,\frac{-A_{r,i-1}}{A_{ri}},\frac{1-A_{ri}}{A_{ri}},\frac{-A_{r,i+1}}{A_{ri}},\ldots,\frac{-A_{rk}}{A_{ri}}\right)$. What is $(AG_k[i,w])$ row r?

The column-operation Gauss matrix $G_k[h,w]$ can be used to convert the last $h-1$ components of a k-vector to 0, when component h of the vector is non-zero. Let $a=(a_1,a_2,\ldots,a_k)$ with $a_h\neq 0$. Take $w=(0,\ldots,0,0,-a_{h+1}/a_h,-a_{h+2}/a_h,\ldots,-a_k/a_h)$. Then $aG_k[h,w]=(a_1,a_2,\ldots,a_h,0,\ldots,0)$. This is the essential computation in tail-elimination. We use only this form of Gauss matrix below; thus, without overriding qualification, we shall henceforth consider only *restricted* Gauss matrices $G_k[h,w]$ where w col $(1:h)=0$.

Exercise 1.23: Take $s\in\{1,\ldots,k\}$ and let $v_1,\ldots,v_s$ be vectors in $\mathcal{R}^k$ such that (v_h) col $(1:h)=0$ for $h=1,\ldots,s$. Let $M_h=G_k[h,v_h]$ be the indicated restricted Gauss matrix. Show that $M_1M_2\cdots M_s=I+\sum_{1\leq i\leq s}e_i^{\mathrm{T}}v_i$ and also show that $(M_1M_2\cdots M_s)^{-1}=I-\sum_{1\leq i\leq s}e_i^{\mathrm{T}}v_i$. Hint: show that $(e_i^{\mathrm{T}}v_i)(e_j^{\mathrm{T}}v_j)=0$ when $i>j$ (and (v_i) col $(1:i)=0$).

Given an $n\times k$ matrix A, we can (1) scale any desired column of A, (2) add a multiple of one column to another column, or (3) swap any two specified columns by multiplying A on the right by one or more suitable $k\times k$ elementary matrices. Similar operations can be done to the rows of A by multiplying on the left by one or more suitable $n\times n$ elementary matrices.

We can transform an $n \times k$ matrix A into a lower-triangular form by multiplying by suitable permutation matrices and restricted Gauss matrices appropriately on the left and on the right of A. Indeed if we multiply the coefficient matrix A by suitable permutation matrices and *unrestricted* Gauss matrices appropriately on the left and on the right, we can transform A into a diagonal form. This does not mean that every system of linear equations has a solution; the equations may be inconsistent. Moreover a solution need not be unique; some diagonal elements in the matrix of the transformed equations may be 0. (Of course, we do not want to use matrix multiplication; we want to achieve the effect of multiplying by suitable matrices without incurring the cost of actually doing such multiplications. This is achieved by using Gaussian elimination which "optimizes" the implied matrix multiplications.)

The reduction of the $n \times k$ matrix A to triangular or diagonal form by multiplying by suitable elementary matrics on the left and/or right of A is equivalent to applying a sequence of the operations: (1) scaling a row or column by a constant, (2) adding a multiple of a row (or column) to another row (or column,) and (3) swapping two rows or two columns. We shall see that it is possible to do all the swapping operations *before* the other operations. However to do this, a preliminary computation is required to determine the exchanges that must be done, so doing all the exchanges before other operations is impractical as well as unnecessary.

Exercise 1.24: Let A be an $n \times k$ matrix. Show that there are $k \times k$ elementary matrices F_1, F_2, ..., F_m such that $G := AF_1 \cdots F_m$ is in *column-echelon* form where G is in column-echelon form when: (1) G col i is a covector with $i-1+j_i$ initial zero-components with $0 \leq j_1 \leq j_2 \leq \cdots \leq j_k$ and $i - 1 + j_i \leq n$ for $1 \leq i \leq k$. (2) If $i - 1 + j_i \neq n$, $G_{i-1+j_i,i} = 1$. (3) If $G_{i-1+j_i,i} = 1$, $G_{i-1+j_i,t} = 0$ for $1 \leq t < i$. Hint: decipher what a column-echelon matrix is in simpler terms.

1.1 The Gausssian-Elimination LU-Decomposition Algorithm

Gaussian elimination can be systematized and cast in a more general form by considering an associated matrix factorization called an *LU-decomposition* [GV89] [Grc11b]. Again, by multiplying by suitable permutation matrices and restricted Gauss matrices appropriately on the left and on the right of A, we can obtain a complete LU-decomposition of the $n \times k$ matrix A. Given such a decomposition, solving the linear equations $xA = v$ is done with a pair of back-substitution steps.

Let $r = rank(A)$ We will give an algorithm below based on the complete-pivoting algorithm in [GV89] that determines the rank r, and further determines $n \times n$ transposition permutation matrices $R_1, \ldots, R_r$ and $k \times k$ transposition permutation matrices $C_1, \ldots, C_r$ and $k \times k$ restricted column-operation Gauss matrices $M_1, \ldots, M_r$, and an $n \times k$ lower-triangular matrix L and a non-singular $k \times k$ upper-triangular matrix U such that

$$R_r \cdots R_1 A C_1 M_1 \cdots C_r M_r = L \quad \text{and} \quad R_r \cdots R_1 A C_1 \cdots C_r = LU.$$

The identity $R_r \cdots R_1 A C_1 \cdots C_r = LU$ is called an *LU-decomposition* for A. The matrix L is an $n \times k$ lower-triangular matrix of the form $\begin{bmatrix} J & 0 \\ K & 0 \end{bmatrix}$, where J is an $r \times r$ non-singular lower-triangular matrix and K is an $(n-r) \times r$ matrix. The upper-triangular matrix U satisfies $diag(U) = (1, \ldots, 1)$.

When $r < k$, the submatrix U row $(r+1) : k$ col $(r+1) : k$ is the $(k-r)\times(k-r)$ identity matrix. Note when $R_1 = \cdots = R_r = I_{n\times n}$ and $C_1 = \cdots = C_r = I_{k\times k}$, our LU-decomposition is just $A = LU$; this form can only be obtained for specially-structured matrices A such as symmetric positive-definite matrices. (Generally, in practice, r will be computed with finite-precision arithmetic and may thus be computed erroneously so that r will be the "computational rank" of A which is just an estimate of $rank(A)$, but, for presentation purposes, we assume exact arithmetic unless we indicate otherwise.)

This algorithm uses a *complete pivot search* in step 2. Essentially this means we search for a largest magnitude coefficient among all the terms for un-"eliminated" variables in all the "eligible" equations; this coefficent determines which variable to partially eliminate next, *i.e.*, the variable that this coefficent multiplies, and which "eligible" equation to use to effect this partial elimination, *i.e.*, the equation in which this coefficient is found. We then use this coefficient as our pivot element for our next elimination.

Using a complete pivot search introduces a large cost, and, apparently, a lot of complexity due to the need for maintaining the permuation information needed to cope with using pivot values in arbitrary positions in our coefficient matrix. However, for the most part, this complexity cannot be avoided for arbitrary coefficient matrices, and using a largest-magnitude admissble coefficient as our pivot value in each iteration generally reduces the error introduced when fixed finite-precision arithmetic is used.

LU-Decomposition by Column-Operation Gaussian Elimination with Complete-Pivoting:
input: $n \times k$ matrix A, $n \geq 1$, $k \geq 1$.
output: $L, U, b, c, r, R_1, \ldots, R_r, C_1, \ldots, C_r, M_1, \ldots, M_r$

1. $L \leftarrow A$; $U \leftarrow I_{k\times k}$; $b \leftarrow \langle 1, 2, \ldots, n\rangle$; $c \leftarrow \langle 1, 2, \ldots, k\rangle$; $h \leftarrow 1$.
2. Determine indices $p \in \{h, \ldots, n\}$ and $q \in \{h, \ldots, k\}$ such that $|L_{pq}| = max_{\substack{h\leq i\leq n\\ h\leq j\leq k}} |L_{ij}|$.
3. $a \leftarrow L_{pq}$; if $a = 0$ then ($r \leftarrow h - 1$; exit).
4. $b_h \leftarrow p$; $c_h \leftarrow q$.
   $\left[\begin{array}{l}\text{Let } u = transpose_k(h, q). \text{ Define } C_h = I \text{ col } u.\\ \text{Let } u = transpose_n(h, p). \text{ Define } R_h = I \text{ row } u.\end{array}\right]$
5. If $h \neq q$ swap L col h and L col q in L;
   If $h \neq p$ swap L row h and L row p in L.
   { Now $L_{hh} = a$.}
6. $\left\{\begin{array}{l}\text{Subtract multiples of } L \text{ col } h \text{ from } L \text{ col } (h+1), L \text{ col } (h+2), \ldots, L \text{ col } k\\ \text{to make } L \text{ row } h \text{ col } [(h+1) : k] = 0. \text{ Also compute } U \text{ row } h \text{ col } [(h+1) : k].\end{array}\right\}$
   for $j = h + 1, \ldots, k$:
   ($z \leftarrow L_{hj}/a$; $L_{hj} \leftarrow 0$; $U_{hj} \leftarrow z$; for $i = h + 1, \ldots, n : (L_{ij} \leftarrow L_{ij} - zL_{ih})$).

$$\left[\begin{array}{l}\text{Let } w \text{ col } (1:h) = 0 \text{ and } w \text{ col } ((h+1):k) = -[L \text{ row } h \text{ col } ((h+1):k)]/L_{hh}. \\ \text{Define } M_h = G_k[h, w].\end{array}\right]$$

7. if $h = n$ or $h = k$ then $(r \leftarrow h$; exit);
   $h \leftarrow h + 1$; go to step 2.

At exit, this algorithm has determined the value r, the permutation matrices $C_1, \ldots, C_r$ and $R_1, \ldots, R_r$, the restricted column-operation Gauss matrices $M_1, \ldots, M_r$, the lower-triangular matrix L, the upper-triangular matrix U, and the vectors b and c specifying permutations in transposition vector form; b corresponds to the $n \times n$ row-permutation matrix $R_r \cdots R_1$ such that $R_r \cdots R_1 = I$ row $perm(b)^{-1}$, where $perm(b)$ denotes the permutation corresponding to the transposition vector b, and c corresponds to the $k \times k$ column-permutation matrix $C_1 \cdots C_r$ such that $C_1 \cdots C_r = I$ col $perm(c)^{-1}$, where $perm(c)$ denotes the permutation corresponding to the transposition vector c. Assuming exact arithmetic, the value $r = rank(A)$.

Exercise 1.25: Show that the matrix products $R_r \cdots R_1$ and $M_1 \cdots M_r$ and $C_1 \cdots C_r$ are all products of elementary matrices.

Let $P = R_r \cdots R_1$ and let $B = C_1 M_1 \cdots C_r M_r$. Let $Q = C_1 \cdots C_r$. If $r = 0$, take $P = I$, $B = I$, and $Q = I$. We shall see that $B^{-1}Q$ is the same matrix as the $k \times k$ upper-triangular matrix U computed in the algorithm above.

The matrix P is an $n \times n$ row-permutation matrix, B is a $k \times k$ non-singular matrix, Q is a $k \times k$ column-permutation matrix, and the matrix U is a $k \times k$ non-singular upper-triangular matrix with $U_{ii} = 1$ for $i = 1, \ldots, k$. Also, the transposition vector b represents the inverse of the n-permutation that the row-permutation matrix P represents, and the transposition vector c represents the inverse of the k-permutation that the column-permutation matrix Q represents.

Step 5 in the algorithm above is equivalent to $[L \leftarrow LC_h;\ L \leftarrow R_h L]$ and step 6 is equivalent to $[L \leftarrow LM_h]$. Note L is initialized to A. Then $R_r \cdots R_1 A C_1 M_1 \cdots C_r M_r = L$, so $PAB = L$, so $PA = LB^{-1}$, so, with $B^{-1}Q = U$, $PAQ = LB^{-1}Q$, and thus $PAQ = LU$.

The relation $PAQ = LU$ is called an *LU-decomposition* for A. Given $PAQ = LU$, we may determine if the set of linear equations $xA = v$ is consistent, and if so, compute x such that $xA = v$ as follows.

Note $A = P^{-1}LUQ^{-1} = P^{\mathrm{T}}LUQ^{\mathrm{T}}$ since the inverse of a permutation matrix is its transpose. Then $xA = v$ implies $xP^{\mathrm{T}}LUQ^{\mathrm{T}} = v$ implies $xP^{\mathrm{T}}LU = vQ$. Let $y = xP^{\mathrm{T}}L$. Then $yU = vQ$. Since U is non-singular and upper-triangular, we can compute y with back-substitution. Now recall $y = xP^{\mathrm{T}}L$. Let $z = xP^{\mathrm{T}}$. Then $y = zL$.

If $n = k$ and the lower-triangular matrix L is non-singular, we can compute z with back-substitution. Otherwise, recall L is an $n \times k$ lower-triangular matrix with $L = \begin{bmatrix} J & 0 \\ K & 0 \end{bmatrix}$ where J is an $r \times r$ non-singular lower-triangular matrix and K is an $(n-r) \times r$ matrix. If y col $[(r+1):k] \neq 0$, our equations are inconsistent and x does not exist, (since $y = zL$ and $z[L \text{ col } (r+1):k] = 0$.) Otherwise we may take $z_{r+1} = z_{r+2} = \cdots = z_n = 0$ and solve for $z_1, \ldots, z_r$ in the triangular system

of linear equations $[z \text{ col } 1 : r]J = [y \text{ col } 1 : r]$ via back-substitution. Now z is determined, no matter what the value of r is.

Finally, we must appropriately permute the components of $z = xP^{\mathrm{T}}$ according to the permutation matrix P to obtain $x = zP$. (In order to compute zP within z, we may use the following algorithm. [for $i = r, r-1, \ldots, 1$: swap z_i with z_{b_i}]. Also, in order to compute vQ within v, we may use the algorithm: [for $i = 1, 2, \ldots, r$: swap v_i with v_{c_i}].) Here b and c are the transposition vectors computed in the Gaussian-elimination LU-decomposition algorithm.

Exercise 1.26: Why is the recipe for computing zP different from the recipe for computing vQ? Hint: $P = R_r \cdots R_1$ and $Q = C_1 \cdots C_r$ where $R_i = I$ row $transpose_n(i, b_i)$ and $C_j = I$ col $transpose_k(j, c_j)$. We have $zP = z(I \text{ row } perm(b)^{-1}) = z(I \text{ col } perm(b))$ and $vQ = v(I \text{ col } perm(c)^{-1})$.

Exercise 1.27: Show that $b_i \geq i$ for $i = 1, \ldots, n$ and $c_j \geq j$ for $j = 1, \ldots, k$.

We could construct *permutation* vectors b and c representing the matrices P and Q, rather than constructing transposition vectors as is done in step 4 in the Gaussian-elimination LU-decomposition algorithm. We would do this by replacing '$b_h \leftarrow p$' with 'Swap b_h with b_p' and replacing '$c_h \leftarrow q$' with 'Swap c_h with c_q'. This defines b and c as products of transpositions, with $P = I$ row b and $Q = I$ col c. Then PW is computed as W row b, WP is computed as W col b^{-1}, VQ is computed as V col c, and QV is computed as V row c^{-1}, where W and V are arbitrary conformable matrices.

Exercise 1.28: How do we compute $P^{\mathrm{T}}W$, WP^{T}, VQ^{T}, and $Q^{\mathrm{T}}V$ for conformable matrices W and V, given that the vectors b and c are permutation vectors that correspond to the row-permutation matrix P and the column-permutation matrix Q with $P = I$ row b and $Q = I$ col c? Hint: $P^{-1} = P^{\mathrm{T}}$ and $Q^{-1} = Q^{\mathrm{T}}$.

Exercise 1.29: Show that $xP^{\mathrm{T}}L = vB$, and hence $zL = vB$ where $B = QU^{-1}$.

Exercise 1.30: What is computed in the Gaussian-elimination LU-decomposition algorithm when the $n \times k$ matrix $A = 0$? What is computed when $k = 1$ and $A = e_n^{\mathrm{T}}$? What is the result of the Gaussian-elimination LU-decomposition algorithm when $n \geq k$? What is the result when $n < k$?

Exercise 1.31: What is the LU-decomposition of the $n \times k$ rank 0 matrix $O_{n \times k}$?

Exercise 1.32: What is computed in the Gaussian-elimination LU-decomposition algorithm when the $n \times k$ matrix A is $diag(v_1, v_2, \ldots, v_{\min(n,k)})$ where $v_i \in \mathcal{R}$ for $i = 1, \ldots, \min(n, k)$?

Exercise 1.33: If we have the same number of equations as unknowns, do we always have a solution for our linear equations $xA = v$? Show that if $n = k$ and $rank(A) = k$, the linear equations $xA = v$ are necessarily consistent and have a unique solution.

Exercise 1.34: Explain why U row $(r+1) : k$ col $(r+1) : k = I_{k-r,k-r}$ when $r < k$.

Exercise 1.35: Given the $n \times k$ matrix A, let $m = \max(n, k)$ and define the "augmented" $m \times m$ matrix S as $S_{ij} = A_{ij}$ for $1 \leq i \leq n$ and $1 \leq j \leq k$, and $S_{ij} = 0$ for $n + 1 \leq i \leq m$ or $k + 1 \leq j \leq m$. Explain how we can compute an LU-decomposition of the $n \times k$ matrix A, given

an LU-decomposition of S. This is equivalent to only using our LU-decomposition algorithm for square matrix inputs.

Exercise 1.36: The pivot value a computed in each iteration of the Gaussian-elimination LU-decomposition algorithm is the value of an element of L, L_{pq}, called the *pivot element* of that iteration. Why do we seek the largest magnitude element of L row $(h:n)$ col $(h:k)$ in step 2 to serve as the pivot element in iteration h? Would determining *any* non-zero element of L row $(h:n)$ col $(h:k)$ suffice?

Solution 1.36: With exact arithmetic there is no need to seek a large pivot value; any non-zero value will suffice. However, with fixed finite-precision floating-point arithmetic, each arithmetic operation may introduce error in the result, depending upon the input values. In particular, if we have two non-zero floating-point values $b = 2^r f$ and $c = 2^s g$, where f and g are p-bit binary fractions with $.5 \leq |f| < 1$ and $.5 \leq |g| < 1$, and r and s are integers, then, assuming overflow or underflow does not occur, the floating-point approximation to the product bc is $2^{r+s} fg(1+\epsilon u)$ where $|\epsilon| \leq .5$ and $u = 2^{1-p}$; u is called the *rounding-unit* of our floating-point number system; u is the least positive value such that $1+u$ rounded to a floating-point value with p bits of precision is greater than 1. (On a 64-bit IEEE floating-point machine, $u = 2^{-52}$.)

The error introduced in the product is thus $2^{r+s} fg\epsilon u = bc\epsilon u$. The larger the magnitude of the product bc is, the larger the magnitude of the error can be.

Also, if b and c are themselves rounded-off results of floating-point computations of the form $b = 2^r f(1+\beta u)$ and $c = 2^s g(1+\gamma u)$ for some values β and γ, not necessarily less than .5, then the floating-point form of the product bc is $2^{r+s} fg(1+\beta u)(1+\gamma u)(1+\epsilon u) \approx 2^{r+s} fg(1+(\beta+\gamma+\epsilon)u)$, so the error is approximately $2^{r+s} fg(\beta+\gamma+\epsilon)u$. Thus error is propagated and its magnitude is magnified as our computation progresses.

The basic computation in Gaussian-elimination LU-decomposition is of the form $L_{ij} \leftarrow L_{ij} - (L_{hj}/a)L_{ih}$. When $L_{hj} \neq 0$, the smaller the magnitude of the pivot value a is, the larger the magnitude of the multiplier L_{hj}/a is, and hence the larger the absolute error in L_{hj}/a can be. And when $L_{ih} \neq 0$, the error in the product $(L_{hj}/a)L_{ih}$ is a combination of the error in L_{hj}/a and L_{ih}, magnified by their opposite factors, and this error propagates, increasing superlinearly in each iteration. This is why we want to avoid using small pivot values!

There is also a more subtle source of error that occurs in computing the difference $L_{ij} - (L_{hj}/a)L_{ih}$ (which is also slightly alleviated by avoiding small pivot values.)

Think about the round-off error in computing a difference of the form $\alpha - \dfrac{\gamma\beta}{a}$ with fixed finite-precision arithmetic where a is a pivot value and α, β, and γ are random values. If $\dfrac{\gamma\beta}{a}$, computed with fixed finite-precision arithmetic, is about the same size as α and has the same sign, the difference will then suffer a large loss in precision, in which the error present in $\dfrac{\gamma\beta}{a}$ is magnified. By "loss of precision" we mean that $\alpha - (\alpha - \dfrac{\gamma\beta}{a})$ computed with fixed finite-precision floating-point arithmetic is very different from $\dfrac{\gamma\beta}{a}$.

Essentially, if $\frac{\gamma\beta}{a}$ is represented by $\frac{\gamma\beta}{a}(1+\epsilon u)$, then our computation results in $\left[\alpha - \frac{\gamma\beta}{a}(1+\epsilon u)\right]$ $(1+\delta u) \approx \alpha - \frac{\gamma\beta}{a} - \frac{\gamma\beta}{a}\epsilon u + \left(\alpha - \frac{\gamma\beta}{a}\right)\delta u$ where $|\delta| \leq .5$, and if $\alpha - \frac{\gamma\beta}{a}$ is small, the magnitude of the absolute error $|\frac{\gamma\beta}{a}\epsilon - \left(\alpha - \frac{\gamma\beta}{a}\right)\delta|u$ can be *comparatively* large. When $\alpha - \frac{\gamma\beta}{a}$ is small, the absolute error is approximately $|\frac{\gamma\beta}{a}\epsilon|u$, and the *relative error* $|\frac{\gamma\beta}{a}\epsilon|u/|\left(\alpha - \frac{\gamma\beta}{a}\right)|$ in our result can be large. Thus we see that it is computing sums of values of similar magnitudes and opposite signs that risk causing subsequent catastophic loss-of-precision error. Loss-of-precision error is essentially relative error, in contrast to the absolute error discussed before.

Choosing a to be larger than an "average" value means that, more often than not, $\frac{\gamma\beta}{a}$ is smaller than average and the difference $\alpha - \frac{\gamma\beta}{a}$ has less loss of precision than would be expected on average. See [Knu97] for a discussion of floating-point arithmetic and round-off error.

Exercise 1.37: Show that $R_t = I$ row $transpose_n(t, b_t)$ and $C_t = I$ col $transpose_k(t, c_t)$ where b and c are the transposition vectors computed in the Gaussian-elimination LU-decomposition algorithm. Also show that $C_t = C_t^{\mathrm{T}} = C_t^{-1}$ and $R_t = R_t^{\mathrm{T}} = R_t^{-1}$.

Exercise 1.38: Let $p = perm(b)$ and let $q = perm(c)$. Show that the transposition vectors b and c computed in the algorithm above determine the row-permutation matrix $P = R_r \cdots R_1$ such that $P = I$ row p^{-1} and determine the column-permutation matrix $Q = C_1 \cdots C_r$ such that $Q = I$ col q^{-1} respectively. Thus the matrices P and Q need not be explicitly computed.

Solution 1.38:

We have $P = R_r R_{r-1} \cdots R_1 = (I$ row $transpose_n(r, b_r)) \cdots (I$ row $transpose_n(1, b_1)) =$ I row $(transpose_n(r, b_r) \cdots transpose_n(1, b_1)) = I$ row p^{-1} where the permutation p corresponds to the transposition vector $b = [b_1, b_2, \ldots, b_n]$ with $p = transpose_n(1, b_1) \cdots transpose_n(r, b_r)$ $= transpose_n(r, b_r) \downarrow \cdots \downarrow transpose_n(1, b_1)$. (Recall $u \downarrow v = u_v = \langle u_{v_1}, u_{v_2}, \ldots, u_{v_n}\rangle$.)

Thus the matrix P is the $n \times n$ row-permutation matrix I row p^{-1} where the n-permutation p is determined by the transposition vector b via the algorithm: $[p \leftarrow \langle 1, 2, \ldots, n\rangle$: for $i = r, r-1, \ldots, 1$: swap p_i with $p_{b_i}]$ and the n-permutation p^{-1} is then computed via the algorithm: [for $i = 1, 2, \ldots, n$: $p_{p_i}^{-1} \leftarrow i$].

We have $Q = C_1 C_2 \cdots C_r = (I$ col $transpose_k(1, c_1))$ $(I$ col $transpose_k(1, c_2)) \cdots$ $(I$ col $transpose_k(r, c_r)) = I$ col $(transpose_k(r, c_r) \cdots transpose_k(1, c_1)) = I$ col q^{-1} where the permutation q corresponds to the transposition vector $c = [c_1, c_2, \ldots, c_n]$ with $q = transpose_k(1, c_1)$ $\cdots transpose_k(r, c_r) = transpose_k(r, c_r) \downarrow \cdots \downarrow transpose_k(1, c_1)$.

Thus the matrix Q is the $k \times k$ column-permutation matrix I col q^{-1} where the k-permutation q is determined by the transposition vector c via the algorithm: $[q \leftarrow \langle 1, 2, \ldots, k\rangle$: for $i = r, r-1, \ldots, 1$: swap q_i with $q_{c_i}]$ and the k-permutation q^{-1} is then computed via the algorithm: [for $i = 1, 2, \ldots, k$: $q_{q_i}^{-1} \leftarrow i$].

Exercise 1.39: Let p be an n-permutation and let q be a k-permutation. Show that $z(I \text{ row } p^{-1}) = z \text{ col } p$ and $v(I \text{ col } q^{-1}) = v \text{ col } q^{-1}$.

Exercise 1.40: Show that $J_{ii} \neq 0$ for $1 \leq i \leq r$ and show that L is non-singular if and only if $n = k = r$ and $L = J$.

Exercise 1.41: Show that if $n = k$ then $det(A) = L_{11} \cdot L_{22} \cdots L_{nn} \cdot det(P) \cdot det(Q)$. Note, $det(P) \cdot det(Q)$ is either 1 or -1. If we keep track of the number, tp, of non-identity transpositions recorded in the transposition vectors b and c, we can determine the value of $det(P) \cdot det(Q)$ as $2 \cdot (tp \bmod 2) - 1$.

Exercise 1.42: Is $R_1 = \cdots = R_r = I$ equivalent to $R_1 \cdots R_r = I$? Is $C_1 = \cdots = C_r = I$ equivalent to $C_1 \cdots C_r = I$?

Exercise 1.43: Show that each of the restricted Gauss matrices $M_1, \ldots, M_r$ computed in the Gaussian-elimination LU-decomposition algorithm is an upper-triangular matrix.

Note in practical application, none of the matrices $C_1, \ldots, C_r, R_1, \ldots, R_r$, or $M_1, \ldots, M_r$ need to be computed. They are effectively replaced by b, c, and U. Thus none of the bracketed operations in the LU-decomposition algorithm need to be done!

Exercise 1.44: Show that $|U_{ij}| \leq 1$ for $1 \leq i < j \leq k$. Hint: complete-pivoting implies that the value z in step 6 of the Gaussian-elimination LU-decomposition algorithm is no greater in magnitude than 1.

Exercise 1.45: When finite-precision floating-point arithmetic operations are used, the computed rank r may be in error. Is the computed value r more likely to be an underestimate or an overestimate? What is the probability that r is correct under suitable randomness assumptions?

Exercise 1.46: Let A be an $n \times n$ non-singular matrix. Show that the Gaussian-elimination LU-decomposition algorithm with complete-pivoting applied to the matrix A chooses pivot elements such that no two pivot elements lie in the same row or in the same column of A or any iteration instance of the matrix L derived iteration-by-iteration from A.

It remains to demonstrate that the matrix U computed in the Gaussian-elimination LU-decomposition algorithm is the same as the matrix $B^{-1}Q$. We have

$$B^{-1}Q = (C_1 M_1 C_2 M_2 \cdots C_r M_r)^{-1} C_1 C_2 \cdots C_r = M_r^{-1} C_r^{-1} M_{r-1}^{-1} C_{r-1}^{-1} \cdots C_2^{-1} M_1^{-1} C_1^{-1} C_1 C_2 \cdots C_r,$$

or equivalently,

$$B^{-1}Q = M_r^{-1}(C_r(M_{r-1}^{-1}(C_{r-1}(\cdots(C_3(M_2^{-1}(C_2 M_1^{-1} C_2))C_3))\cdots))C_r).$$

Recall that $C_i = C_i^{-1} = C_i^{\mathrm{T}}$ is a $k \times k$ permutation matrix corresponding to a transposition $transpose_k(i, j)$ where $i \leq j \leq r$.

Now, let w_h be the k-vector such that $M_h = I + e_h^{\mathrm{T}} w_h$. Recall that $M_1, M_2, \ldots, M_r$ are restricted Gauss matrices. For $h = 1, \ldots, r$, we have $(w_h) \text{ col } 1 : h = 0$ and $(w_h) \text{ col } ((h+1) : k) = -[L \text{ row } h \text{ col } ((h+1) : k)]/L_{hh}$ as computed in step 6. Then $M_h^{-1} = I - e_h^{\mathrm{T}} w_h$.

Now $C_2 M_1^{-1} C_2 = C_2(I - e_1^{\mathrm{T}} w_1) C_2 = C_2 I C_2 - (C_2 e_1^{\mathrm{T}})(w_1 C_2) = I - e_1^{\mathrm{T}}(w_1 C_2)$. This follows because the row-permutation matrix C_2 exchanges row 2 with row j where $j \geq 2$, so that $C_2 e_1^{\mathrm{T}} = e_1^{\mathrm{T}}$. And, $C_2 = C_2^{-1}$, so $C_2 I C_2 = I$.

Also note that since the column-permutation matrix C_2 exchanges column 2 with column j where $j \geq 2$, we have $w_1 C_2$ col 1 = (w_1) col 1; thus (w_1) col 1 remains 0 and $I - e_1^{\mathrm{T}}(w_1 C_2)$ remains a restricted Gauss matrix.

Next, $M_2^{-1}(C_2 M_1^{-1} C_2) = (I - e_2^{\mathrm{T}} w_2)(I - e_1^{\mathrm{T}}(w_1 C_2)) = I - e_1^{\mathrm{T}}(w_1 C_2) - e_2^{\mathrm{T}} w_2$, since $e_2^{\mathrm{T}} w_2 e_1^{\mathrm{T}}(w_1 C_2) = O_{k \times k}$. And thus, $C_3(M_2^{-1}(C_2 M_1^{-1} C_2))C_3 = I - e_1^{\mathrm{T}}(w_1 C_2 C_3) - e_2^{\mathrm{T}}(w_2 C_3)$.

Continuing in this manner, we finally obtain

$$B^{-1} Q = I - \sum_{1 \leq i \leq r} e_i^{\mathrm{T}}(w_i C_{i+1} \cdots C_r).$$

But, $[w_i C_{i+1} \cdots C_r]$ col $(1 : i) = 0$ and $[w_i C_{i+1} \cdots C_r]$ col $((i+1) : k)$ is the final value of U row i col $((i+1) : k)$ computed in the Gaussian-elimination LU-decomposition algorithm above. Thus $B^{-1} Q = U$, where U row $i = e_i + w_i C_{i+1} \cdots C_r$. [QED]

Exercise 1.47: Show that U is a $k \times k$ upper-triangular matrix, and show that $diag(U) = (1, \ldots, 1)$.

When we have obtained r and L and U and P (or equivalently b) and Q (or equivalently c) such that $PAQ = LU$, we can use the process described above to solve $xA = v$ for any given righthand-side vector v that admits a solution with two back-substitution steps, and two vector permutations.

Algorithmically, the process to solve $xP^{\mathrm{T}} L U Q^{\mathrm{T}} = v$ is:

1. Compute $v \leftarrow vQ$ by permuting v according to the inverse of the permutation determined by the transposition vector c.

2. Compute y such that $yU = v$ by back-substitution.

3. If y col $[(r+1) : k] \neq 0$ then ('$xA = v$' is inconsistent; exit.)

4. z col $[(r+1) : n] \leftarrow 0$.

5. Compute $[z$ col $(1 : r)]$ such that $[z$ col $(1 : r)][L$ row $(1 : r)$ col $(1 : r)] = [y$ col $(1 : r)]$ via back-substitution.

6. Compute $x \leftarrow zP$ by permuting z according to the permutation determined by the transposition vector b.

7. exit.

In detail, this is:

1. for $i = 1, 2, \ldots, r$: swap v_i with v_{c_i}.

2. for $i = 1, 2, \ldots, k : (y_i \leftarrow v_i$; for $j = 1, \ldots, i-1 : (y_i \leftarrow y_i - U_{ji} y_j))$.

3. If y col $[(r+1) : k] \neq 0$ then ('$xA = v$' is inconsistent; exit.)

4. z col $[(r+1) : n] \leftarrow 0$.

5. for $i = r, r-1, \ldots, 1 :$ $(z_i \leftarrow y_i$; for $j = i+1, \ldots, r : (z_i \leftarrow z_i - L_{ji} z_j)$; $z_i \leftarrow z_i / L_{ii})$.

6. $x \leftarrow z$; for $i = r, r-1, \ldots, 1$: swap x_i with x_{b_i}.

7. exit.

Note this algorithm destroys the input righthand-side vector v.

Exercise 1.48: Why does the loop in step 1 run for $i = 1, \ldots, r$ rather than $i = 1, \ldots, k$?

Exercise 1.49: Can you revise the above algorithm to permute v into the vector y during step 2 and both prevent destroying v and save some small amount of computation?

Exercise 1.50: Show that we can dispense with the vector z in the algorithm above.

Exercise 1.51: Use the fact that U row $(r+1) : k$ col $(r+1) : k = I$ to show that step 2 can be written as: for $i = 1, 2, \ldots, k : (y_i \leftarrow v_i$; for $j = 1, \ldots, \min(i-1, r) : (y_i \leftarrow y_i - U_{ji} y_j))$.

Exercise 1.52: Explain step 3.

Solution 1.52: When $r < k$, our k equations are linearly-dependent if they are consistent. In this case, L col $[(r+1) : k] = O_{n \times (k-r)}$, so the equations $zL = y$ are not satisfiable if y col $[(r+1) : k]$ is not 0, and hence does not match the lefthand-side vector (zL) col $[(r+1) : k]$.

Exercise 1.53: Show that L_{ii} is never 0 in step 5.

Exercise 1.54: Give the algorithm for solving for the vector y in $Ay^{\mathrm{T}} = w^{\mathrm{T}}$, given the LU-decomposition of the $n \times k$ matrix A and the righthand-side vector w. Here $y \in \mathcal{R}^k$ and $w \in \mathcal{R}^n$.

We have seen that, given the LU-decomposition $A = P^{\mathrm{T}} L U Q^{\mathrm{T}}$, the linear equations $xA = v$ can be written $xP^{\mathrm{T}} LU = vQ$ and when the equations $xA = v$ are consistent, x can be computed as $x = zP$ where z satisfies $zL = y := vQU^{-1}$. When the equations $xA = v$ are consistent, the n-vector z belongs to the solution flat $yL^+ + \mathit{nullspace}(L)$. Thus the solution flat of the linear equations $xA = v$ is $\{xP \in \mathcal{R}^n \mid x \in vQU^{-1}L^+ + \mathit{nullspace}(L)\}$.

Exercise 1.55: Recall that the $n \times k$ matrix $L = \begin{bmatrix} J & 0 \\ K & 0 \end{bmatrix}$ where J is an $r \times r$ non-singular lower-triangular matrix and K is an $(n-r) \times r$ matrix. Show that the $k \times n$ matrix $L^+ = \begin{bmatrix} J^{-1} & 0 \\ 0 & 0 \end{bmatrix}$. Hint: $L^+ L$ is the projection onto $\mathit{rowspace}([J\ K])$.

The solution flat $yL^+ + \mathit{nullspace}(L)$ can be constructed as follows. The linear equations $zL = y$ are just the linear equations $(z_1\ z_2) \begin{bmatrix} J & 0 \\ K & 0 \end{bmatrix} = (y_1\ 0)$ where $z = (z_1\ z_2)$, $y = (y_1\ 0)$, z_1 is an r-vector,

z_2 is an $(n-r)$-vector, J is an $r \times r$ non-singular lower-triangular matrix, K is an $(n-r) \times r$ matrix, and y_1 is an r-vector. Altogether the matrix $\begin{bmatrix} J & 0 \\ K & 0 \end{bmatrix}$ is an $n \times k$ matrix and the vector $(y_1\ 0)$ is a k-vector.

The equations $zL = y$ reduce to $z_1 J + z_2 K = y_1$. We can fix z_2 to be any vector in $\mathcal{R}^{n-r}$ and take z_1 to be the solution $z_1 = (y_1 - z_2 K)J^{-1}$ of the equations $z_1 J + z_2 K = y_1$ in order to form a solution of $zL = y$. This set of solutions of $zL = y$ is $\{((y_1 - z_2 K)J^{-1}\ z_2) \in \mathcal{R}^n \mid z_2 \in \mathcal{R}^{n-r}\}$ and this is the same set as $yL^+ + nullspace(L)$ where $yL^+ = (y_1 J^{-1}\ 0)$ and $nullspace(L) = \{((-z_2 K)J^{-1}\ z_2) \in \mathcal{R}^n \mid z_2 \in \mathcal{R}^{n-r}\}$. (Note $(-z_2 K J^{-1}\ z_2) \begin{bmatrix} J & 0 \\ K & 0 \end{bmatrix} = (-z_2 K + z_2 K\ \ 0) = 0$.)

Exercise 1.56: Recall that $nullspace(AB) = nullspace(A)$ for conformable matrices A and B when the columns of B are linearly-independent. Let A be an $n \times k$ matrix with the LU-decomposition $P^{\mathrm{T}} L U Q^{\mathrm{T}}$. Show that $nullspace(A) = nullspace(L)P$.

Solution 1.56: $PA = LUQ^{\mathrm{T}}$, so $nullspace(PA) = nullspace(L)$, and

$$\begin{aligned} nullspace(PA) &= \{x \in \mathcal{R}^n \mid xPA = 0\} \\ &= \{xP^{\mathrm{T}} \in \mathcal{R}^n \mid xP^{\mathrm{T}}PA = 0\} \\ &= \{xP^{\mathrm{T}} \in \mathcal{R}^n \mid xA = 0\} \\ &= nullspace(A)P^{\mathrm{T}}. \end{aligned}$$

Thus $nullspace(L)P = nullspace(A)$.

Exercise 1.57: Revise the back-substitution algorithm given above for computing x that satisfies $xA = v$ to compute a barycentric basis for the solution flat of $xA = v$. Hint: take z col $(r+1) : n$ to be each of the $(n-r)$-vectors $0, e_1, \ldots, e_{n-r}$ in step 4.

If we wish to solve just the single system of equations $xA = v$, we may save some time by appending v to A as A row $(n+1)$, and then using Gaussian elimination (with pivot positions and values in A) to convert the augmented matrix $\begin{bmatrix} A \\ v \end{bmatrix}$ to lower-triangular form $\begin{bmatrix} J & 0 \\ K & 0 \\ w & u \end{bmatrix}$ where the row $[w\ u]$ is the result of processing v with the Gaussian elimination steps applied to the columns of $\begin{bmatrix} A \\ v \end{bmatrix}$, and then using one back-substitution step to compute x; indeed this is often what is meant by Gaussian elimination. The vector $[w\ u]$ is equal to the vector y computed in step 2 in the back-substitution procedure above. (Does this idea, in fact, save any time?)

Exercise 1.58: Show that the equations $xA = v$ are inconsistent if and only if $rank(A) \neq rank\left(\begin{bmatrix} A \\ v \end{bmatrix}\right)$.

Exercise 1.59: We have $PAQ = LU$ where the $n \times k$ matrix $L = \begin{bmatrix} J & 0 \\ K & 0 \end{bmatrix}$ and the $k \times k$ matrix $U = \begin{bmatrix} Z & W \\ 0 & I \end{bmatrix}$ where J is an $r \times r$ non-singular lower-triangular matrix, K is an

$(n-r)\times r$ matrix, Z is an $r\times r$ non-singular upper-triangular matrix with $diag(Z)=(1,\ldots,1)$, W is an $r\times(k-r)$ matrix, and I denotes an $(n-r)\times(n-r)$ identity matrix. Show that $PAQ=\bar{L}\bar{U}$ where $\bar{L}=\begin{bmatrix} J \\ K \end{bmatrix}$ and $\bar{U}=\begin{bmatrix} Z & W \end{bmatrix}$.

Using the result of the above exercise we may re-examine solving the linear equations $xA=v$. We have $PAQ=\bar{L}\bar{U}$ where $\bar{L}=\begin{bmatrix} J \\ K \end{bmatrix}$ and $\bar{U}=\begin{bmatrix} Z & W \end{bmatrix}$ as described above. Let $z=xP^{\mathrm{T}}$ and $w=vQ$. Then $xA=v$ is equivalent to $z\bar{L}\bar{U}=w$, and thus $z\begin{bmatrix} J \\ K \end{bmatrix}\begin{bmatrix} Z & W \end{bmatrix}=w$.

Therefore $z\begin{bmatrix} JZ & JW \\ KZ & KW \end{bmatrix}=w$. Now write $z=[z_1\ z_2]$ where $z_1\in\mathcal{R}^r$ and $z_2\in\mathcal{R}^{n-r}$. Then $[z_1\ z_2]\begin{bmatrix} JZ & JW \\ KZ & KW \end{bmatrix}=[z_1JZ+z_2KZ\ z_1JW+z_2KW]=w$.

Since $rank(A)=rank(JZ)=r$, when we take $z_2=0$ we have $z_1JZ=w$ col $1:r$ and $z_1JW=w$ col $(r+1):k$, and if these two vector equations are inconsistent, *i.e.*, if the latter relation fails to hold given the former, the linear equations $xA=v$ are inconsistent. Otherwise $z_1=(w$ col $1:r)(JZ)^{-1}$ and $(w$ col $1:r)Z^{-1}W=w$ col $(r+1):k$ and $[z_1\ 0]P=x$.

It is common to use the number of floating-point arithmetic operations (flops) as a measure of the cost of a numerical algorithm.

The loop-structure of the version of the Gaussian-elimination LU-decomposition algorithm given above, not including the optional steps in brackets, is:

$$[\,\text{for } h=1,\ldots,m:\{\,\text{for } i=h+1,\ldots,k:\{\,1\text{ flop}\,\};\ \text{ for } j=h+1,\ldots,n:\{\,2\text{ flops}\,\}\}]$$

where $m=\min(n,k)$. If $m=k$, the outer-loop is effectively $[h=1,\ldots,m-1]$. If $k>n$, the inner-loop: $[j=h+1,\ldots,n]$ is empty when $h=m$, and there are just $k-n$ flops used in the $h=m$ iteration.

Thus the total cost in flops, C, for the Gaussian-elimination LU-decomposition algorithm applied to a rank m $n\times k$ matrix is: $\displaystyle\sum_{1\le h\le m-1}\ \sum_{h+1\le i\le k}\left[1+\sum_{h+1\le j\le n}2\right]+\delta_{mn}(k-n)$.

And thus,

$$\begin{aligned} C &= \sum_{1\le h\le m-1}\left[k-h+\sum_{h+1\le i\le k}2(n-h)\right]+\delta_{mn}(k-n) \\ &= \sum_{1\le h\le m-1}[k-h+2(n-h)(k-h)]+\delta_{mn}(k-n) \\ &= \sum_{1\le h\le m-1}\left[(1+2n)k+2h^2-h(1+2n+2k)\right]+\delta_{mn}(k-n) \end{aligned}$$

$$
\begin{aligned}
&= (1+2n)k(m-1) + 2\left[\sum_{1\le h\le m-1} h^2\right] - \left[\sum_{1\le h\le m-1} h\right](1+2n+2k) + \delta_{mn}(k-n) \\
&= (1+2n)k(m-1) + 2\left[\frac{m^3}{3} - \frac{m^2}{2} + \frac{m}{6}\right] - \frac{1}{2}m(m-1)(1+2n+2k) + \delta_{mn}(k-n) \\
&= (1+2n)k(m-1) + \left[\frac{2}{3}m^3 - m^2 + \frac{1}{3}m\right] - \frac{1}{2}(m^2-m)(1+2n+2k) + \delta_{mn}(k-n).
\end{aligned}
$$

When $n = k = m$, we have $C = \frac{2}{3}m^3 - \frac{1}{2}m^2 - \frac{1}{6}m$. (Can you explain why $\frac{2}{3}m^3 - \frac{1}{2}m^2 - \frac{1}{6}m$ is always a non-negative integer when $m \in \mathcal{Z}^+$? (Other than the fact that it is a *count* of operations.) Is it sufficient to consider $m \in \{0, 1, 2, 3, 4, 5\}$?)

Note the number of flops used is not a very good indicator of the total cost of the Gaussian-elimination LU-decomposition algorithm, since there is a substantial cost in searching for a maximal-magnitude pivot element in each iteration.

Exercise 1.60: What is the maximum number of times we *read* an element of the matrix L, given as a function of n and k, in the Gaussian-elimination LU-decomposition algorithm above, not including the optional steps in brackets?

Solution 1.60: Let $m = \min(n, k)$. Then the maximum number of L-reads is:
$\sum_{1\le h\le m} [(n-h+1)(k-h+1) + 1 + 2n + 2k + (k-h)(1+2(n-h))] =$
$m^3 + \frac{3}{2}(k-n+2)m^2 + 3[(n+2)(k+\frac{1}{2}) - \frac{1}{6}k]m.$
(When $n = k = m$, this is $4m^3 + 10m^2 + 3m$.)

Exercise 1.61: How should we modify the Gaussian-elimination LU-decomposition algorithm to apply to matrices and vectors with complex elements?

There are several modifications to the Gaussian-elimination LU-decomposition algorithm given above that are practically desirable in a computer program.

The first issue has to do with dividing by L_{kk} (*i.e.*, by a). Because of the possibility of computing too-large or too-small values in various computations in the Gaussian-elimination LU-decomposition algorithm or in a back-substitution algorithm, we should use an overflow handler that either substitutes the largest representable correctly-signed value in place of the unrepresentable overflowing value (with a warning,) or declares our coefficient matrix to be indecomposable. Also, we should use a machine with "soft" (unnormalized) underflow.

Because of round-off error the test "$a = 0$" in step 3 that protects us from dividing by zero might usefully be replaced by "$|a| < \alpha$", where α is a small value. What "small" should be is a complicated matter. A choice independent of the elements of the matrix A might be the rounding-unit u of our computer. (Recall that the rounding-unit u is the smallest value such that $1 < 1 + u$ in floating-point, and the "binary-precision" p is the number of bits used to represent numbers in the interval $[.5, 1)$; $p = 1 - log_2(u)$.) Really we should take α to be a value relative to the magnitude of the numbers that produced a so that α represents the error in a; a compromise might

be $\alpha = u \cdot \left\lceil \max_{\substack{1 \le i \le n \\ 1 \le j \le k}} |A_{ij}| \right\rceil /(nk)$, or even better, u times the average absolute value of the non-zero elements of A. We could compute this value during the first pivot search in our LU-decomposition algorithm.

If a is a value near the smallest normalized positive value, ϵ, of the machine then an overflow is likely to occur before too long. (For 64-bit IEEE floating-point format, $\epsilon = 2^{-1022}$.) When a is small, it is hard to know if a is an "approximation of zero" or not; this is another reason we want to do a pivot search in order to avoid small pivot values whenever possible. (There is a costly device called *interval arithmetic* that can be employed; with interval arithmetic we can know whether a computed value could be zero if it were computed exactly.)

Note that the errors that arise in the various values of a and the other elements of L due to round-off error means that the resulting elements of L and U are likely to be incorrect and the computed value of the rank of A may be incorrect. Suppose $rank(A) = r$. Due to round-off error, it can be unlikely that we will see $L_{jj} = 0$ for $j = r+1, \ldots, min(n,k)$. The best we can do using Gaussian elimination with ordinary floating-point arithmetic is to estimate the rank of A by treating final small absolute values of $diag(L)$ as zero.

> **Exercise 1.62:** Will it always be the case that $L_{jj} \neq 0$ for $j = 1, \ldots, rank(A)$ when the Gaussian-elimination LU-decomposition algorithm is executed with fixed finite-precision floating-point arithmetic?

In some cases we want to enforce a requirement that the $n \times k$ matrix A be of full-rank: $rank(A) = \min(n,k)$. We can modify A if necessary to do this. In this circumstance, we may replace the statement "if $a = 0$ then $(r \leftarrow h-1;$ exit)" in step 3 with "if $|a| < \varepsilon$ then $\{a \leftarrow Sign(a)(|a|+2 \cdot m)\}$", where $Sign(a) = 1$ if $a \geq 0$ and $Sign(a) = -1$ otherwise, and where m is a small positive value [PTVF92]. (We may also want to issue a warning or set a flag to indicate when this modification occurs.) Suitable choices for m might be the value 100ε or $\varepsilon + \min_{A_{ij} \neq 0} |A_{ij}|/100$. (What is a suitable choice for ε?) This device of forcing A to have full-rank is also a "stabilization" device since adding suitable constants to each diagonal element of an $n \times n$ matrix to force it to be non-singular usually improves its "condition" for processing by the Gaussian-elimination LU-decomposition algorithm.

The second issue has to do with the cost of searching for a pivot element. The Gaussian-elimination LU-decomposition algorithm given above processes the columns of A one-by-one; The processing of a single column consists of permuting the eligible rows and columns of A to bring a non-zero value to the diagonal position in the column being processed (*i.e.*, for column j, the diagonal position is row j of column j,) and then subtracting multiples of this column from other columns to "zero-out" the row or "tail" of the row of that diagonal position.

In each iteration, the non-zero diagonal element holding our pivot value, after permuting to bring it into the diagonal, (denoted by a in the Gaussian-elimination LU-decomposition algorithm,) is called the *pivot element* in that iteration, and the process of "zeroing-out" or *eliminating* elements in the pivot-element row is called *elimination*, specifically the process of "zeroing-out" ("eliminating") the elements in the pivot-element row following the pivot-element itself is called *tail-elimination*,) so the Gaussian-elimination LU-decomposition algorithm consists of $\min(n,k)$ or fewer tail-elimination

operations. (In general, in contrast to a tail-elimination step, a single *full-elimination* step applied to a matrix A with respect to the pivot element A_{ij} is generally taken to be just the transformation effected by the post- or pre-multiplication of A by the appropriate (non-restricted) Gauss matrix that converts A row i to e_j, or alternately A col j to e_i^{T} when a "transposed" form of Gaussian elimination is used.)

In step 2 of the Gaussian-elimination LU-decomposition algorithm, the search for the element with the largest absolute value in the sub-array L row $[h:n]$ col $[h:k]$ is called a *complete pivot search,* (or just *complete-pivoting*,) and the element $L_{pq} = a$ that is found is the pivot element at iteration h. Using the element with the largest absolute value generally results in the best numerical "stability" – we generally obtain close to the least practicable error in the resulting solution vector or vectors computed based on the lower-triangular matrix L. Complete pivot searching is time-consuming. Nevertheless, in order to sucessfully terminate and to guarantee the exact form of the matrix L specified above in the case of a singular matrix, and even for many non-singular matrices, we need some sort of pivot search; we must, at least, search for a non-zero element.

Exercise 1.63: If the matrix A has non-zero diagonal elements $A_{11}, A_{22}, \ldots, A_{mm}$ where $m = \min(n, k)$, is it necessary to use a pivot search in constructing the LU-decomposition of A? That is, can the h-th diagonal element of L as modified in the prior iteration serve as the pivot element in iteration h?

Exercise 1.64: Can we really guarantee we will get a close approximate solution to $xA = v$ in the case where A is non-singular by using the LU-decomposition algorithm given above, followed by two back-substitution computations done in 64-bit floating-point arithmetic?

Solution 1.64: It depends on what the meaning of 'close approximation' is.

When all we care about is computing the unique solution to $xA = v$ when it exists, then there is a practical compromise called *cross-row partial pivot search*, (or just *cross-row partial-pivoting*,) where we replace the pivot-value search in step 2 with: "Determine the index $p \in \{h, \ldots, n\}$ such that $|L_{ph}| = max_{h \le i \le n} |L_{ih}|$.", and then take $a = L_{ph}$ in step 3 and drop the unnecessary unexecuted statement "If $h \neq q$ swap L col h and L col q in L" in step 5. A cross-row partial pivot search effectively searches L col h, (across the rows of L col h,) for an element of maximum magnitude. Note that if L row $h:n$ col $h = 0$, our coefficient matrix A would be singular. (In fact, all of L col h would be 0.)

Exercise 1.65: Try using cross-row partial-pivoting and back-substitution to solve the linear equations $[x_1 + x_2 + x_3 = 3,\ x_1 = 1,\ x_2 + x_3 = 2]$.

Much experience has shown that, for A non-singular, computing the solution to $xA = v$ using partial-pivoting is almost always stable and it is much less costly than complete-pivoting. When the matrix A is non-singular, partial-pivoting will usually succeed in computing an acceptable approximation to the LU-decomposition of A.

Exercise 1.66: Show that when cross-row partial-pivoting is used, the permutation matrices $C_1, \ldots, C_r$ will all be the $k \times k$ identity matrix I.

Exercise 1.67: Show that the maximum number of times we read an element of the matrix

L in the LU-decomposition algorithm using cross-row partial-pivoting, as a function of n and k is: $\frac{2}{3}m^3 - (k+n)m^2 + ((2k-1) + \frac{1}{3})m - 2n\delta_{mk}$ where $m = \min(n,k)$. (When $n = k = m$, this is $\frac{2}{3}m^3 - m^2 + \frac{5}{6}m$.)

We could also replace the pivot-value search in step 2 with: "Determine the index $q \in \{h, \ldots, k\}$ such that $|L_{hq}| = max_{h \le i \le k}|L_{hi}|$.", and then take $a = L_{hq}$ in step 3 and drop the unnecessary unexecuted statement "If $h \neq p$ swap L row h and L row p in L" in step 5. In this case, the permutation matrices $R_1, \ldots, R_r$ will all be the $n \times n$ identity matrix. We shall call this variant of partial-pivoting, *cross-column partial pivot search*. When A is non-singular and cross-column partial-pivoting is used, we have $AC_1M_1 \cdots C_rM_r = L$, and $L_{ii} \neq 0$ for $1 \le i \le n$. Note, in this case, we are, in essence, changing the basis of *rowspace*(A) by multiplying by the change-of-coordinates matrix $C_1M_1 \cdots C_rM_r$ to obtain a "lower-triangular" basis for *rowspace*(A).

Exercise 1.68: Why must A be non-singular to ensure that $AC_1M_1 \cdots C_rM_r = L$ is obtained with cross-column partial-pivoting? Hint: what happens in our Gaussian-elimination LU-decomposition algorithm with a cross-column partial pivot search when a 0 pivot value arises?

Exercise 1.69: Suppose A is a non-singular matrix so that $n = k$. Show that when complete pivoting is used, $|L_{ij}| \le |L_{jj}|$ for $1 \le i \le n$. Does this hold when a cross-row partial pivot search is used? Does this hold when a cross-column partial pivot search is used?

When cross-column partial-pivoting is applied to an $n \times k$ matrix A with $rank(A) < \min(n,k)$, as the Gaussian-elimination process progresses, (using exact arithmetic,) we will have one or more rows where the value in the pivot position in that row, position (h,j), is zero, and all the values $L_{h,j+1}, \ldots, L_{h,k}$ are zero as well. If we just continue the Gaussian-elimination process with the next row of L (if any,) and the *same* column of L, (*i.e*, we take $L_{h+1,j}$ as the next pivot element position,) we will obtain a lower-triangular matrix with one or more zero diagonal elements. This lower-triangular matrix, possibly with one or more zero diagonal elements, is called an *echelon-form* matrix equivalent to the matrix A. Note if a lower-triangular echelon-form matrix L has $L_{jj} = 0$ then L row $j : n$ col $j = 0$. Any triangular matrix L equivalent to A gives the rank of A as $\min(n,k)$ − (the number of 0's in the diagonal of L). The rows of such a matrix can be permuted to obtain a lower-triangular matrix $\hat{L} = \begin{bmatrix} J & 0 \\ K & 0 \end{bmatrix}$ with J an $r \times r$ lower-triangular non-singular matrix.

We may modify our search for a pivot value in the sub-array L row $h : n$ col $h : k$ to accept the *first* non-zero value encountered. Using the first encountered non-zero element as the pivot element will be called *minimal-pivoting*. Note a pivot search is required in general, even when $rank(A) = \min(n,k)$, since L_{hh} may be zero in any iteration.

There are strategies for pivot searching beyond a minimal pivot search, a cross-column or cross-row partial pivot search, or a complete pivot search. For example, the following strategy, called a *rook pivot search*, may be used [NP92].

$[p \leftarrow h;\ q \leftarrow h;\ a \leftarrow L_{hh};\ s \leftarrow h;\ t \leftarrow 0;$
$\alpha:$ for $i = h, \ldots, k:$ if $|L_{pi}| > |a|$ then $\{q \leftarrow i; a \leftarrow L_{pi}\}$; if $t \neq q$ then $(t \leftarrow q)$ else exit;
for $i = h, \ldots, n:$ if $|L_{iq}| > |a|$ then $\{p \leftarrow i; a \leftarrow L_{iq}\}$; if $s \neq p$ then $(s \leftarrow p;$ goto $\alpha)$ else exit$]$.

This code computes p, q, and a such that $a = L_{pq}$ is the pivot element selected for iteration h. The selected pivot element is an element in L row $h : n$ col $h : k$ which is an element of maximum magnitude in *both* its row and its column. This pivot search strategy generally performs only a modest amount worse than complete-pivoting with respect to introduced round-off error and is usually much less costly than complete-pivoting with an estimated average total cost of $em(m-1)/2$ comparisons for computing *all* the pivot elements needed to construct an LU-decomposition of a "random" $n \times k$ matrix where $m = \max(n, k)$ [Fos97].

Exercise 1.70: Show that the rook pivot search algorithm given above always terminates. (Does this remind you of the result that a function of two variables has a maximum in any closed and bounded region in its domain?)

Exercise 1.71: Give an example $n \times n$ matrix where at least n^2 comparisons are required to find an element which is both row and column maximal using rook pivot searching. (The algorithm given above uses $2n^2$ comparisons in the worst case.)

Exercise 1.72: Note that the rook pivot search algorithm above does unnecessary comparisons. Can you find a nice way to avoid these comparisons?

Exercise 1.73: Let A be an $n \times k$ matrix with unequal elements. Show that there are at most $\min(n, k)$ elements of A that are both row and column maximal.

The rook pivot search algorithm, like cross-row or cross-column partial pivoting, is not guaranteed to produce a non-zero pivot value; a zero pivot value may be chosen, even when an eligible non-zero pivot value exists. This can happen when the submatrix L row $h : n$ col $h : k$ has L row h col $h : k = 0$ and L row $h : n$ col $h = 0$ with $h < \min(n, k)$. We can deal with this by reverting to a minimal pivot search for a non-zero value in the situation where the rook pivot search results in a zero pivot value. This is required to ensure that any $n \times k$ matrix A can be decomposed. (However, when A is non-singular, the rook pivot search algorithm, like cross-row or cross-column partial pivoting, will never fail to produce a non-zero pivot value, assuming exact arithemetic.)

Exercise 1.74: When can we avoid searching for a pivot value entirely, *i.e.*, when can we replace step 2 with "$p \leftarrow h; q \leftarrow h$."? (Does it suffice for A_{ii} to be non-zero for $1 \leq i \leq n$?)

Exercise 1.75: Can we save any time by searching for the *next* pivot value in L row $((h+1) : n)$ col $((h+1) : k)$ *while* we are subtracting suitable multiples of L row $((h+1) : n)$ col h from L row $((h+1) : n)$ col $((h+1) : k)$ in step 6, and not using step 2 after the first initial pivot value is determined?

Solution 1.75: Probably we can obtain a constant-factor speed-up. The exact improvement depends on how good a coder you or your compiler is. But the maximum running time of the LU-decomposition algorithm remains $O(\min(n, k)^3)$.

Third and finally, by dropping the assignment "$U \leftarrow I_{k \times k}$" in step 1 and replacing the commands "$L_{hi} \leftarrow 0; U_{hi} \leftarrow z$" with "$L_{hi} \leftarrow z$" in step 6, we can save space in the Gaussian-elimination LU-decomposition algorithm by storing the elements U_{ij} for $2 \leq i < \min(n, k)$ and $i < j \leq k$ in the strictly-upper-triangular part of the matrix L (which would otherwise be 0) as it is being formed; U_{ii} is known to be 1 for $i = 1, \ldots, k$, U_{ij} is known to be 0 for $2 \leq i > j < k$, and

U row $(n+1:k) = [O_{k-n,n}\ I_{k-n,k-n}]$ when $n < k$. Thus the $k \times k$ matrix U need not be explicitly created as a separate array. (Note, if we do not save U in the upper-triangle of L, then we can save a little time by replacing the statement "Swap L col h and L col q in L" with "Swap L row $h:n$ col h and L row $h:n$ col q in L".)

The "access" function for the elements of the matrix U stored in the upper-triangular part of L as described above is:
$[U(i,j) :=$ if $(i > j)$ return(0) else if $(i = j)$ return(1) else if $(i > n)$ return(0) else return$(L_{ij})]$.
(This algorithm assumes $1 \le i \le k$ and $1 \le j \le k$.)

Also note we may save space by dispensing with the assignment "$L \leftarrow A$" in step 1, and just overwrite the matrix A with L and U (less the main diagonal of U) when the destruction of the input matrix A is permissible.

Exercise 1.76: In machine language, or in a language like C with pointer data-types, we can avoid the assignment "$L \leftarrow A$" even when A is to be preserved; this saves nk scalar assignments, although space for L is still required. We can do this by loading L with values *during* the first iteration of the LU-decomposition where L row 1 col $2:k$ is made zero in a manner compatible with using the (strict) upper triangle of L to hold the values of U. Explain in more detail how this can be done.

Exercise 1.77: Explain how we can store an $n \times n$ lower-triangular matrix, L, in $n(n+1)/2$ locations: $mem[0 : n(n+1)/2-1]$ so that we can access the matrix element L_{ij} with the program [if $(i < j)$ return(0); $k \leftarrow \dfrac{(i-1)i}{2} + j - 1$; return$(mem[k])$.] Give the corresponding access function for an $n \times n$ upper-triangular matrix stored in $n(n+1)/2 - 1]$ locations.

Suppose we use a row-indexing vector ri and a column-indexing vector ci maintained so that $ri[j]$ is the position of the *original* row j in the matrix L and $ci[k]$ is the position of the *original* column k in the matrix L. In order to do complete-pivoting, the rows and columns of L are exchanged in step 5 of the LU-decomposition algorithm above; instead we may just swap the appropriate entries of the indexing vectors ri and ci, and not swap the rows and columns of L in step 5. Then we access the array L as $L_{ri[j],ci[k]}$ rather than L_{jk}. (With U stored in the upper-triangle of L, this device also manages U appropriately.)

If, at the end of the LU-decomposition algorithm, we restore L to its final permuted state by applying the permutation in ri to the rows of L and applying the permutation in ci to the columns of L, then we save little time overall, and if computing $L_{ri[j],ci[k]}$ rather than L_{jk} costs two additional memory accesses (assuming the indices j and k are kept in registers and have relatively small access cost,) then we lose time overall. However, if we never restore L (and U) to the permuted state where $A = P^{\mathrm{T}}LUQ^{\mathrm{T}}$ and use the indexing arrays ri and ci to access L (and U) when doing, for example, back-substitution, then we may save some time by thus avoiding the row and column swaps done in step 5 in the case where computing $L_{ri[j],ci[k]}$ rather than L_{jk} costs the equivalent of much less than two additional accesses; otherwise we still do not save time by using indexing arrays. (We might be able to keep the indexing arrays in fast memory for instance. Indeed, perhaps a special LU-decomposition chip would be worth designing.)

Exercise 1.78: Quantify the relative costs in using indexing arrays, and avoiding row and

column swapping in complete-pivoting, under various assumptions about the additional cost of computing $L_{ri[j],ci[k]}$ rather than L_{jk}.

The LU-decomposition $PAQ = LU$ for a given $n \times k$ matrix A is trivially unique in the sense that application of the complete-pivoting Gaussian-elimination LU-decomposition algorithm executes a unique sequence of computations (assuming a systematic method of resolving ties in pivot searches is used.)

The question remains as to whether there are more than one pair of lower-triangular and upper-triangular matrices L and U of the forms produced by the Gaussian-elimination LU-decomposition algorithm such that $A = P^{\mathrm{T}}LUQ^{\mathrm{T}}$ where the permutation matrices P and Q are produced in the Gaussian-elimination LU-decomposition algorithm applied to the matrix A with P and Q *fixed* to correspond to a specific admissible choice of the sequence of pivot elements.

We can see that it is plausible that this factorization is unique when $n = k$ as follows. Let $rank(A) = r$. Now consider the LU-decomposition $PAQ = LU$ with A given and the admissible permutation matrices P and Q fixed. We have $rn - (r-1)r/2$ elements of L to be determined, and $(k-1)k/2$ elements of U to be determined (remember $diag(U) = (1, \ldots, 1)$.) We have nk non-linear equations relating these $rn-(r-1)r/2+(k-1)k/2$ values and we know these equations are consistent. When $rn-(r-1)r/2+(k-1)k/2$ of our nk equations are independent, the values defining the matrices L and U are uniquely determined. This is possible when $nk \geq rn-(r-1)r+(k-1)k$, or equivalently, when $2rn-(r-1)r \leq 2kn-(k-1)k$. And since $r \leq min(n,k)$, this is always the case when $n = k$. (It turns-out, however, that a sufficent number of independent equations do not generally exist when $r < \min(n,k)$.)

Exercise 1.79: Suppose we have two LU-decompositions: $P_1AQ_1 = L_1U_1$ and also $P_2AQ_2 = L_2U_2$. Show that $P_2P_1^{\mathrm{T}}L_1U_1Q_1^{\mathrm{T}}Q_2 = L_2U_2$. Explain what this means about the uniqueness of the LU-decomposition of A.

Exercise 1.80: Show that for any fixed sequence of non-zero pivot elements determining the permutation matrices P and Q, the LU-decomposition $PAQ = LU$, with $diag(U) = (1, 1, \ldots, 1)$, is unique when the matrix A is non-singular.

Solution 1.80: Suppose A is non-singular. If $PAQ = L_1U_1$ and also $PAQ = L_2U_2$ where L_1 and L_2 are lower-triangular non-singular matrices and U_1 and U_2 are upper-triangular non-singular matrices with $diag(U_1) = diag(U_2) = (1, \ldots, 1)$ then $L_1U_1 = L_2U_2$ so $L_2^{-1}L_1 = U_2U_1^{-1}$.

And $L_2^{-1}L_1$ is lower-triangular since the product of lower-triangular matrices is lower-triangular and the inverse of a non-singular lower-triangular matrix is itself lower-triangular, and similarly $U_2U_1^{-1}$ is upper-triangular. But a matrix that is both lower-triangular and upper-triangular is a diagonal matrix so $L_2^{-1}L_1 = U_2U_1^{-1} = D$ where D is an $n \times n$ diagonal matrix where $diag(D) = (diag(L_2^{-1}), diag(L_1)) = (diag(U_2), diag(U_1^{-1}))$.

But the diagonal of the inverse of a non-singular triangular matrix M is specified by $diag(M^{-1}) = (1/M_{11}, 1/M_{22}, \ldots, 1/M_{nn})$ so $diag(D) = (1, \ldots, 1)$ since $diag(U_2) = (1, \ldots, 1)$ and $diag(U_1^{-1}) = (1, \ldots, 1)$. Thus $L_2^{-1}L_1 = U_2U_1^{-1} = I$ so $L_1 = L_2$ and $U_2 = U_1$, and thus, for given permutation matrices P and Q, the LU-decomposition $PAQ = LU$ is unique when A is non-singular and

P and Q are chosen to produce non-zero pivots in the Gaussian-elimination LU-decomposition algorithm.

Let A be an $n \times k$ rank r matrix, and consider the LU-decomposition $PAQ = LU$. We can transform this relation to write $Q^{\mathrm{T}}A^{\mathrm{T}}P^{\mathrm{T}} = U^{\mathrm{T}}L^{\mathrm{T}}$. Note L^{T} is upper-triangular and U^{T} is lower-triangular with $diag(U^{\mathrm{T}}) = (1, \ldots, 1)$. We can "reshape" this relation to obtain the LU-decomposition of A^{T} as follows.

Suppose $r \leq n \leq k$. Then L^{T} is a $k \times n$ matrix with L^{T} row $(r+1) : k = 0$ and U^{T} is a $k \times k$ lower-triangular matrix with $diag(U^{\mathrm{T}}) = (1, \ldots, 1)$. Let $\hat{U} = L^{\mathrm{T}}$ row $1 : n$; $\hat{U}$ is just L^{T} with $k - n$ zero-rows removed, so $\hat{U}$ is an $n \times n$ upper-triangular matrix. Let $\hat{L} = U^{\mathrm{T}}$ col $1 : n$; $\hat{L}$ is just U^{T} with columns $(n+1) : k$ removed, so $\hat{L}$ is a $k \times n$ lower-triangular matrix with $diag(\hat{L}) = (1, \ldots, 1)$. Now by examining a block-multiplication, we can see that $U^{\mathrm{T}}L^{\mathrm{T}} = \hat{L}\hat{U}$.

Exercise 1.81: Write-out the block-multiplication which verifies that $U^{\mathrm{T}}L^{\mathrm{T}} = \hat{L}\hat{U}$ in the case where $r \leq n \leq k$.

We have $\hat{U} = \begin{bmatrix} J^{\mathrm{T}} & K^{\mathrm{T}} \\ 0 & 0 \end{bmatrix}$ with $\hat{U}$ row $(r+1) : n = 0$; thus we may take $\hat{L}$ col $(r+1) : n$ to be 0, and then take $\hat{U}$ row $(r+1) : n$ col $(r+1) : n$ to be $I_{(n-r)\times(n-r)}$, and the product $\hat{L}\hat{U}$ remains unchanged. Now $\hat{U}$ is an $n \times n$ upper-triangular matrix with $\hat{U}_{ii} \neq 0$ for $1 \leq i \leq n$, so $\hat{U}$ is non-singular.

Now suppose $r \leq k < n$. Then define the $n \times n$ upper-triangular matrix $\hat{U} = \begin{bmatrix} L^{\mathrm{T}} \\ 0 \end{bmatrix}$; we have appended $n-k$ zero-rows to L^{T} to form $\hat{U}$. Define the $k \times n$ lower-triangular matrix $\hat{L} = \begin{bmatrix} U^{\mathrm{T}} & 0 \end{bmatrix}$; we have appended $n-k$ zero-columns to U^{T} to form $\hat{L}$. Again, by examining a block-multiplication, we can see that $U^{\mathrm{T}}L^{\mathrm{T}} = \hat{L}\hat{U}$. Also we can "transfer rank" from $\hat{L}$ to $\hat{U}$ in the same way we did before. We can replace $\hat{L}$ col $(r+1) : k$ with 0, and then replace $\hat{U}$ row $(r+1) : n$ col $(r+1) : n$ with $I_{(n-r)\times(n-r)}$. Now we still have $\hat{L}\hat{U} = L^{\mathrm{T}}U^{\mathrm{T}}$ and $\hat{U}$ is now an $n \times n$ upper-triangular matrix with $\hat{U}_{ii} \neq 0$ for $1 \leq i \leq n$, so $\hat{U}$ is non-singular. And $\hat{L}$ is a $k \times n$ lower-triangular matrix with $\hat{L}_{ii} = 1$ for $1 \leq i \leq r$.

Exercise 1.82: Write-out the block-multiplication which verifies that $U^{\mathrm{T}}L^{\mathrm{T}} = \hat{L}\hat{U}$ in the case where $r \leq k < n$.

Thus, whether $k \leq n$ or $k > n$, the matrix product $\hat{L}\hat{U}$ satisfies the criteria for being an LU-decomposition factorization, except that we do not necessarily have $diag(\hat{U}) = (1, \ldots, 1)$. But we do have $\hat{U}_{ii} \neq 0$ for $1 \leq i \leq n$, so we can fix our factorization as follows.

Let the $n \times n$ diagonal matrix $D = diag(\hat{U})$. Now define $\tilde{L} = \hat{L}D$ and define $\tilde{U} = D^{-1}\hat{U}$. This makes $diag(\tilde{U}) = (1, \ldots, 1)$ and makes $diag(\tilde{L}$ row $1 : r) = diag(D$ row $1 : r)$. Since $D_{ii} \neq 0$ for $1 \leq i \leq n$, we have $\tilde{L}_{ii} \neq 0$ for $1 \leq i \leq r$. Also $\tilde{L}\tilde{U} = \hat{L}DD^{-1}\hat{U} = \hat{L}\hat{U}$. Thus, if $PAQ = LU$ is an LU-decomposition of A, then $Q^{\mathrm{T}}A^{\mathrm{T}}P^{\mathrm{T}} = \tilde{L}\tilde{U}$ is an LU-decomposition of A^{T}.

As a practical matter, for the purpose of solving non-singular systems of linear equations, we need not insist on a particular form of LU-decomposition. The essential feature we require is just that we can write $PAQ = LU$ for the matrix A where P and Q are permutation matrices and L and

U are non-singular triangular matrices. When this is the case, we can solve the equations $xA = v$ or $Ax^{\mathrm{T}} = v^{\mathrm{T}}$ by two back-substitution computations together with the associated permutations specified by the matrices P and Q.

Note the Gaussian-elimination LU-decomposition algorithm given above can be "transposed" and recast to zero-out the "tail" parts of successive columns by subtracting suitable multiples of successive pivot rows. In this case, we construct an upper-triangular matrix in the (copy of the) input matrix A, and concommitantly construct a lower-triangular matrix of pivot-element-scaled values with an implicit diagonal of ones. This version of the Gaussian-elimination LU-decomposition algorithm performs row operations and is equivalent to multiplying A on the left by certain restricted Gauss matrices (and by pivot-positioning permutation matrices on both the left and right.) Therefore, just as in the column-operation algorithm, we obtain a factorization of the form $PAQ = LU$, where it is now the matrix L that is the non-singular factor with $diag(L) = (1, \ldots, 1)$.

Exercise 1.83: Write the row-operation version of the Gaussian-elimination LU-decomposition algorithm.

Solution 1.83:

LU-Decomposition by Row-Operation Gaussian Elimination with Complete-Pivoting:
input: $n \times k$ matrix A, $n \geq 1$, $k \geq 1$.
output: $L, U, b, c, r, R_1, \ldots, R_r, C_1, \ldots, C_r, M_1, \ldots, M_r$

1. $U \leftarrow A$; $L \leftarrow I_{n \times n}$; $b \leftarrow \langle 1, 2, \ldots, n \rangle$; $c \leftarrow \langle 1, 2, \ldots, k \rangle$; $h \leftarrow 1$.
2. Determine indices $p \in \{h, \ldots, n\}$ and $q \in \{h, \ldots, k\}$ such that $|U_{pq}| = max_{\substack{h \leq i \leq n \\ h \leq j \leq k}} |U_{ij}|$.
3. $a \leftarrow L_{pq}$; if $a = 0$ then ($r \leftarrow h - 1$; exit).
4. $b_h \leftarrow p$; $c_h \leftarrow q$.
   $\left[\begin{array}{l} \text{Let } u = transpose_k(h, q). \text{ Define } C_h = I \text{ col } u. \\ \text{Let } u = transpose_n(h, p). \text{ Define } R_h = I \text{ row } u. \end{array} \right]$
5. If $h \neq q$ swap U col h and U col q in U;
   If $h \neq p$ swap U row h and U row p in U.
   { Now $U_{hh} = a$.}
6. $\left\{ \begin{array}{l} \text{Subtract multiples of } U \text{ row } h \text{ from } U \text{ row } (h+1), U \text{ row } (h+2), \ldots, U \text{ row } n \\ \text{to make } U \text{ col } h \text{ row } [(h+1) : n] = 0. \text{ Also compute } L \text{ col } h \text{ row } [(h+1) : n]. \end{array} \right\}$
   for $i = h + 1, \ldots, n$:
   ($z \leftarrow U_{ih}/a$; $U_{ih} \leftarrow 0$; $L_{ih} \leftarrow z$; for $j = h + 1, \ldots, k$: ($U_{ij} \leftarrow U_{ij} - zU_{hj}$)).
   $\left[\begin{array}{l} \text{Let } w \text{ col } (1 : h) = 0 \text{ and } w \text{ col } ((h+1) : n) = -[U \text{ col } h \text{ row } ((h+1) : n)]/U_{hh}. \\ \text{Define } M_h = G_n[h, w]^{\mathrm{T}}. \end{array} \right]$
7. if $h = n$ or $h = k$ then ($r \leftarrow h$; exit);
   $h \leftarrow h + 1$; go to step 2.

At exit, this algorithm has determined the value r, the permutation matrices $C_1, \ldots, C_r$ and $R_1, \ldots, R_r$, the row-operation Gauss matrices $M_1, \ldots, M_r$, the lower-triangular matrix L, the upper-triangular matrix U, and the permutations b and c in transposition vector form that correspond to the permutation matrices $P = R_r R_{r-1} \cdots R_1$ and $Q = C_1 C_2 \cdots C_r$.

The value r is the rank of the matrix A (assuming exact arithmetic.) The matrices $R_1, \ldots, R_r$ are $n \times n$ transposition permutation matrices, the matrices $C_1, \ldots, C_r$ are $k \times k$ transposition permutation matrices, and the matrices $M_1, \ldots, M_r$ are $n \times n$ restricted row-operation Gauss matrices.

The matrix L is a non-singular $n \times n$ lower-triangular matrix with $diag(L) = (1, \ldots, 1)$ and the matrix U is an $n \times k$ upper-triangular matrix of the form $\begin{bmatrix} J & K \\ 0 & 0 \end{bmatrix}$, where J is an $r \times r$ non-singular upper-triangular matrix and K is an $(k - r) \times r$ matrix such that

$$M_r R_r \cdots M_1 R_1 A C_1 \cdots C_r = U \quad \text{and} \quad R_r \cdots R_1 A C_1 \cdots C_r = LU = PAQ.$$

As with the column-operation version, the outputs L, U, b, c, and r are the only essential outputs. Moreover, the variable part of the strictly-lower-triangular part of L can be returned in the strict lower-triangle of the matrix U, thus saving space.

Exercise 1.84: Let $B = M_r R_r \cdots M_1 R_1$, let $P = R_r \cdots R_1$, and let $Q = C_1 \cdots C_r$, where $M_r, \ldots, M_1$, $R_r, \ldots, R_1$, and $C_1, \ldots, C_r$ are produced by the row-operation Gaussian-elimination LU-decomposition algorithm given above. Show that B is non-singular. Let $L = PB^{-1}$. Show that L is an $n \times n$ non-singular lower-triangular matrix with $diag(L) = (1, \ldots, 1)$.

Exercise 1.85: Give the algorithm for solving the linear system $xA = v$, given the row-operation LU-decomposition of the matrix A.

Exercise 1.86: Can you construct a column-operation version and a row-operation version of the Gaussian-elimination LU-decomposition algorithm which scan along the main-diagonal of the input matrix "backwards" from lower-right to upper-left?

Exercise 1.87: Let $PAQ = LU$ be an LU-decomposition of the $n \times k$ matrix A computed by the column-operation version of the Gaussian-elimination LU-decomposition algorithm. Show that, if we apply the row-operation version of the Gaussian-elimination LU-decomposition algorithm to the $k \times n$ matrix A^{T} using the same choice of pivot elements that correspond to the row and column-permutation matrices being identical to the transposes of the column and row-permutation matrices Q and P, then the LU-decomposition we obtain is identical to the decomposition $U^{\mathrm{T}} L^{\mathrm{T}}$ which we considered above.

Exercise 1.88: Let $G = \left(\prod_{\substack{1 \le i \le n \\ i \ne r}} E_n[r, i, -A_{ic}] \right) E_n[r, r, -1 + \frac{1}{A_{rc}}]$, where A is an $n \times n$ matrix

with $A_{rc} \neq 0$. What is the matrix product GA?

1.2 Variants of LU-Decomposition

Let $m = \min(n, k)$. Suppose the $n \times k$ matrix A satisfies $rank(A) \geq m - 1$ and has its rows and columns "pre-permuted" so that A can be decomposed into the product LU with *no* pivot search, (which means the successive matrix elements L_{hh} computed during the LU-decomposition must be non-zero since they are used as pivot values.) We shall call such an LU-decomposition with no permutations involved a *direct* LU-decomposition, and we shall call an $n \times k$ matrix whose first $\min(n, k) - 1$ pivot values computed with no pivot search are non-zero a *diagonal-pivot matrix*. (Note a diagonal-pivot matrix may have its last pivot value equal to zero since the last pivot value never serves as a denominator.) When A is a diagonal-pivot matrix, the Gaussian-elimination LU-decomposition algorithm can be written as follows (omitting the computation of the rank r.)

LU-Decomposition Gaussian Elimination with No Pivot Search:
input: $n \times k$ matrix A, $n \geq 1$, $k \geq 1$.
output: $n \times n$ lower-triangular matrix L, $n \times k$ upper-triangular matrix U, if successful.

1. $L \leftarrow A$; $U \leftarrow I_{k \times k}$; $m \leftarrow \min(n, k)$.
2. for $h = 1, \ldots, m$:
   for $j = 1, \ldots, k$:
   for $i = 1, \ldots, n$:
   if $i > h$ and $j > h$ then $\{a \leftarrow L_{hh}$; if $a = 0$ then exit("0 pivot encountered."); $z \leftarrow L_{hj}/a$; if $i = n$ then $(L_{hj} \leftarrow 0$; $U_{hj} \leftarrow z)$; $L_{ij} \leftarrow L_{ij} - zL_{ih}\}$
3. exit.

Exercise 1.89: When can the loop "for $h = 1, \ldots, m$:" in the procedure above be replaced by "for $h = 1, \ldots, m - 1$:"? Hint: what if $n = k$?

This expository procedure contains unnecessary computation, but it exhibits the three "nested-loop" structure of the LU-decomposition algorithm with no pivot search. This loop structure generates mkn index-triples $\langle h, j, i \rangle$ and executes the loop body for each of these triples. (Here $m = \min(n, k)$.) Note these mkn index-triples can be generated in any order! In particular, we can order the three loops in any of six possible orders. (Does this observation allow us to develop a parallel-computation version of our LU-decomposition algorithm?)

This procedure also allows us to see how to write a straightforward recursive description of Gaussian-elimination LU-decomposition with no pivot search. For descriptive convenience, let $n = k$ and assume A is non-singular. The Gaussian-elimination LU-decomposition algorithm applied to the $n \times n$ "pre-permuted" non-singular matrix A where searching for a pivot value is not required

generates two sequences of matrices: $L^{(1)} = A$, $L^{(2)}$, ..., $L^{(n)} = L$ and $U^{(1)} = I$, $U^{(2)}$, ..., $U^{(n)} = U$. Also $U^{(h)}$ row $1 : (h-1) = U$ row $1 : (h-1)$ and $L^{(h)}$ col $1 : h = L$ col $1 : h$ for $h = 1, \ldots, n$. The matrices $L^{(h)}$ and $U^{(h)}$ are the matrices L and U in the algorithm above at the start of iteration h. For $1 \le i \le n$, $1 \le j \le n$, and $h = 1, \ldots, n-1$, we have:

$$L_{ij}^{(h+1)} = \begin{cases} 0 & \text{for } i \le h \text{ and } j > i, \\ L_{ij}^{(h)} & \text{for } j \le h, \\ L_{ij}^{(h)} - U_{hj} \cdot L_{ih}^{(h)} & \text{for } j > h \text{ and } i > h, \end{cases} \quad \text{and } U_{ij} = \begin{cases} 0 & \text{for } i > j, \\ 1 & \text{for } i = j, \\ \dfrac{L_{ij}^{(i)}}{L_{ii}^{(i)}} & \text{for } i < j. \end{cases}$$

Exercise 1.90: Show that $U_{ij}^{(h)} = \begin{cases} U_{ij} & \text{for } i < h, \\ \delta_{ij} & \text{for } j \ge h. \end{cases}$

With $n = k$, iteration h computes U row h and L col $h+1$ and updates L col $h+2$, ..., L col n.

Note, for $L_{ij}^{(h+1)}$, the defining cases above are not disjoint, but they are consistent when overlap occurs. (The cases defining $L_{ij}^{(h+1)}$ can be made disjoint by replacing "for $j \le h$," with "for $j \le h$ and $i \ge j$.")

We see that $L_{ii}^{(i)} \ne 0$ and $U_{ii} = 1 = L_{ii}^{(i)}/L_{ii}^{(i)}$ since, when $n = k$, the pre-permuted matrix A is assumed to be non-singular (or at least properly-structured of rank $n-1$,) so as to have each pivot value that is used as a denominator be non-zero. Also, because $L_{ih}^{(h)} = 0$ for $i < h$ and $U_{hj} = 0$ for $h > j$, and $U_{hj} = L_{hj}^{(h)}/L_{hh}^{(h)} = L_{hj}/L_{hh}$ for $h \le j$, we have $L^{(h+1)}$ col $j = L^{(h)}$ col $j - U_{hj}(L^{(h)}$ col $h)$ for $j \ne h$ and $0 = L_{ih}^{(h+1)} = L_{ih}^{(h)} - U_{hh}L_{ih}^{(h)}$ for $i < h$, *i.e.*, $L_{ij}^{(h+1)} = L_{ij}^{(h)} - U_{hj}L_{ih}^{(h)}$ for $j \ne h$ or $i < h$.

The fundamental recursive relations that characterize Gaussian-elimination LU-decomposition with no pivot search thus are:
for $h = 1, \ldots, n-1 : L_{ij}^{(h+1)} = L_{ij}^{(h)} - U_{hj}L_{ih}^{(h)}$ for $j \ne h$ or $i < h$, and $L_{ih}^{(h+1)} = L_{ih}^{(h)}$ for $i \ge h$ with U_{ij} defined as above. This can be more succinctly written as: for $h = 1, \ldots, n-1 : L_{ij}^{(h+1)} = L_{ij}^{(h)} - (1 - \delta_{hj})U_{hj}L_{ih}^{(h)}$. We may "solve" this recursion relation to show again that $LU = A$ as follows.

Writing our recursion relation for $h+1, h, \ldots, 2$, we have: $L_{ij}^{(h+1)} = L_{ij}^{(h)} - (1 - \delta_{hj})L_{ih}^{(h)} \cdot U_{hj}$, $L_{ij}^{(h)} = L_{ij}^{(h-1)} - (1 - \delta_{h-1,j})L_{i,h-1}^{(h-1)} \cdot U_{h-1,j}$, $L_{ij}^{(h-1)} = L_{ij}^{(h-2)} - (1 - \delta_{h-2,j})L_{i,h-2}^{(h-2)} \cdot U_{h-2,j}$, ..., $L_{ij}^{(3)} = L_{ij}^{(2)} - (1 - \delta_{h2})L_{i2}^{(2)} \cdot U_{2j}$, and $L_{ij}^{(2)} = L_{ij}^{(1)} - (1 - \delta_{h1})L_{i1}^{(1)} \cdot U_{1j}$. Now substituting the identity for $L_{ij}^{(2)}$ in the identity for $L_{ij}^{(3)}$, and so on until we substitute our obtained expression for $L_{ij}^{(h)}$ in the identity for $L_{ij}^{(h+1)}$, we obtain $L_{ij}^{(h+1)} = L_{ij}^{(1)} - \sum_{1 \le t \le h} (1 - \delta_{tj})L_{it}^{(t)}U_{tj}$.

And $L_{ij}^{(1)} = A_{ij}$ and $L_{ih}^{(h)} = L_{ih}^{(h+1)} = \cdots = L_{ih}^{(n)} = L_{ih}$, *i.e.*, $L_{ij}^{(h)} = L_{ij}$ for $j \ge h$, so $L_{ij}^{(h+1)} = A_{ij} - \sum_{1 \le t \le h} (1 - \delta_{tj})L_{it}U_{tj}$. Thus $A_{ij} = \sum_{1 \le t \le h} (1 - \delta_{tj})L_{it}U_{tj} + L_{ij}^{(h+1)}$.

Take $h = j - 1$. Then $A_{ij} = \sum_{1 \le t \le j-1} (1 - \delta_{tj}) L_{it} U_{tj} + L_{ij}^{(j)}$. And, since $L_{ij}^{(j)} = L_{ij}$ and $U_{jj} = 1$, $A_{ij} = \sum_{1 \le t \le j-1} L_{it} U_{tj} + L_{ij} U_{jj}$, so $A_{ij} = \sum_{1 \le t \le j} L_{it} U_{tj}$.

And, since $L_{it} = 0$ for $t > i$ and $U_{tj} = 0$ for $t > j$, we have $A_{ij} = \sum_{1 \le t \le \min(i,j)} L_{it} U_{tj} = \sum_{1 \le t \le n} L_{it} U_{tj} =$ (L row i)(U col j). Thus $A = LU$. (Note the utility of using the Kronecker delta function above.)

Exercise 1.91: Suppose $A \in M_{n \times k}$ with $A = LU$ where L is an $n \times k$ lower-triangular matrix and U is a $k \times k$ upper-triangular matrix. Let $m = \max(n, k)$. Extend A to an $m \times m$ matrix by adding $n - k$ columns of 0 values when $n > k$ and adding $k - n$ rows of 0 values when $n \le k$. Denote this extended square matrix by $\bar{A}$. Explain how to extend the matrix L to an $m \times m$ lower-triangular matrix $\bar{L}$ and extend the matrix U to an $m \times m$ upper-triangular matrix $\bar{U}$ such that $\bar{A} = \bar{L}\bar{U}$.

Exercise 1.92: Let A be an $n \times n$ non-singular lower-triangular matrix. Show that A^{-1} is also a lower-triangular matrix and $diag(A^{-1}) = (1/A_{11}, \ldots, 1/A_{nn})$.

Solution 1.92: Suppose every $(n-1) \times (n-1)$ non-singular lower-triangular matrix B has a lower-triangular inverse with its diagonal elements equal to the reciprocals of the corresponding diagonal elements of B.

Write the $n \times n$ matrix A as $A = \begin{bmatrix} A_{11} & 0 \\ v & B \end{bmatrix}$ where v is an $(n-1) \times 1$ matrix and B is an $(n-1) \times (n-1)$ non-singular matrix. Then $A^{-1} = \begin{bmatrix} A_{11}^{-1} & 0 \\ w & B^{-1} \end{bmatrix}$ where $w = -A_{11}^{-1} B^{-1} v$. When A is non-singular and lower-triangular, B is also non-singular and lower-triangular. Now by our induction hypothesis, B^{-1} is a lower-triangular matrix with its diagonal elements formed by the reciprocals of the diagonal elements of B, and thus A^{-1} is also a lower-triangular matrix with its diagonal elements formed by the reciprocals of the diagonal elements of A. This establishes the desired result by induction. Note the matching property holds for upper-triangular non-singular matrices by a transposition of this argument. (The set of $n \times n$ lower-triangular (or upper-triangular) non-singular matrices form a subgroup of the general linear group of $n \times n$ non-singular matrices.)

Exercise 1.93: Let A be an $n \times n$ non-singular diagonal-pivot matrix with integer elements and the LU-decomposition $A = LU$. Show that if each pivot value $L_{ii}^{(i)}$ is ± 1 then the elements of L^{-1} and U^{-1} are integers and the elements of A^{-1} are thus integers.

Solution 1.93: For $A = LU$, $diag(L) = (\pm 1, \ldots, \pm 1)$ implies that the Gaussian-elimination LU-decomposition algorithm generates integer elements in L and U. Now we may show that L^{-1} also has integer elements by induction. Write $L = L_n$ to indicate that L is an $n \times n$ lower-triangular matrix with $diag(L_n) = (\pm 1, \ldots, \pm 1)$.

Now write $L_n = \begin{bmatrix} L_{n-1} & 0 \\ a & \pm 1 \end{bmatrix}$ where $a \in \mathcal{R}^{n-1}$ with integer components. And write $L_n^{-1} =$

$\begin{bmatrix} L_{n-1}^{-1} & 0 \\ b & \pm 1 \end{bmatrix}$. (Recall L^{-1} is lower-triangular with $diag(L^{-1}) = (1/L_{11}, \ldots, 1/L_{nn})$ when L is lower-triangular.) Then $I = L_n L_n^{-1} = \begin{bmatrix} I & 0 \\ aL_{n-1}^{-1} \pm b & 1 \end{bmatrix}$, and $aL_{n-1}^{-1} \pm b = 0$. Thus, $b = \mp aL_{n-1}^{-1}$.

But a has integer components, and L_{n-1}^{-1} has integer elements by our induction hypothesis, so the product aL_{n-1}^{-1} has integer components, and thus b has integer components and hence all the elements of $L_n^{-1} = L^{-1}$ are integers.

A similar argument shows that U^{-1} also has integer elements given that U is an $n \times n$ upper-triangular matrix with $diag(U) = (\pm 1, \ldots, \pm 1)$. And when L^{-1} and U^{-1} have integer elements, A^{-1} has integer elements. [QED]

The particular loop order [for $j = 1, \ldots, n$: for $i = 1, \ldots, k$: for $h = 1, \ldots, m$:] corresponds to a formulation of LU-decomposition with no pivot search known as Crout's method; we give a variant of Crout's method below. To simplify the indexing, let us fix $n = k$. Then the variant Crout method of LU-decomposition for an $n \times n$ matrix A is given in the following algorithm. (Note, like Gaussian-elimination with no pivot search, Crout's method only applies to diagonal-pivot matrices A that have $L_{hh}^{(h)} \neq 0$ for $h = 1, \ldots, n-1$.

Exercise 1.94: Show that the $n \times n$ matrix A is a diagonal-pivot matrix if and only if each of the matrices A row 1 col 1, A row $1:2$ col $1:2$, ..., A row $1:(n-1)$ col $1:(n-1)$ are non-singular.

Crout's LU-Decomposition with No Pivot Search:
input: $n \times n$ diagonal-pivot matrix A, $n \geq 1$.
output: $n \times n$ lower-triangular matrix L, $n \times n$ non-singular upper-triangular matrix U

1. $L \leftarrow O_{n \times n}$; $U \leftarrow I_{n \times n}$.
2. for $j = 1, \ldots, n$:
   $\{$ for $i = 1, \ldots, j-1 : U_{ij} \leftarrow \dfrac{1}{L_{ii}} \left[A_{ij} - \displaystyle\sum_{1 \leq h \leq i-1} L_{ih} U_{hj} \right]$;
   for $i = j, \ldots, n : L_{ij} \leftarrow A_{ij} - \displaystyle\sum_{1 \leq h \leq j-1} L_{ih} U_{hj}$ $\}$.
3. exit.

Exercise 1.95: Do the sums in the above LU-decomposition algorithm look familiar?

Crout's method is derived as follows. If we look at the matrix product $A = LU$ where the $n \times n$ matrix L is a lower-triangular matrix and the $n \times n$ matrix U is a upper-triangular matrix, we

obtain the n^2 equations

$$A_{ij} = \sum_{1 \le h \le \min(i,j)} L_{ih}U_{hj} \text{ for } 1 \le i \le n \text{ and } 1 \le j \le n.$$

This is because $L_{ih} = 0$ for $i < h$ and $U_{hj} = 0$ for $j < h$.

The lower-triangle elements of L and the upper-triangle elements of U constitute n^2+n "unknowns", and we have n^2 (non-linear) equations $LU = A$; if we fix $U_{11} = U_{22} = \cdots = U_{nn} = 1$, we have n^2 unknowns remaining. Crout's method is a systematic procedure for solving these n^2 equations for the undetermined n^2 variables, and this method turns-out to recapitulate Gaussian-elimination LU-decomposition with no pivot search.

Let $m = \min(i, j)$, so we may write $A_{ij} = L_{i1}U_{1j} + \cdots + L_{i,m-1}U_{m-1,j} + L_{im}U_{mj}$. When $i \ge j$, $m = j$ and we have $U_{mj} = 1$, so $L_{im} = L_{ij} = A_{ij} - (L_{i1}U_{1j} + \cdots + L_{i,m-1}U_{m-1,j})$ for $i \ge j$. When $i < j$, $m = i$, so $U_{mj} = U_{ij} = \dfrac{1}{L_{im}} [A_{ij} - (L_{i1}U_{1j} + \cdots + L_{i,m-1}U_{m-1,j})]$ for $i < j$. Note if we attempt to divide by L_{ii} when $L_{ii} = 0$, Crout's algorithm will fail even though A is non-singular. We require that A be "pre-permuted" to be a diagonal-pivot matrix; this is because we started with $A = LU$ to derive Crout's method, rather than $PAQ = LU$. (Try applying Crout's method to $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.)

If we use these n^2 equations in the order specified by the loop-structure [for $j = 1, \ldots n$: for $i = 1, \ldots n$:] then we will compute L_{ij} and U_{ij} values in a sequence where all the needed L and U elements are computed prior to their use in the above equations. This results in the Crout procedure given above.

Exercise 1.96: Show that the values L_{ij} for $i \ge j$ and U_{ij} for $i < j$ are computed in the order L col 1, U col 2, L col 2, U col 3, ..., L col $(n-1)$, U col n, L col n. Explain how U col 1 is computed.

Crout's method can be optimized as follows.

Crout's LU-Decomposition with No Pivot Search:
input: $n \times n$ diagonal-pivot matrix A, $n \ge 1$.
output: $n \times n$ lower-triangular matrix L, $n \times n$ upper-triangular matrix U, if successful.

1. $L \leftarrow O_{n\times n}$; $U \leftarrow I_{n\times n}$.
2. for $j = 1, \ldots, n$:
   for $i = 1, \ldots, n$:
   $\{m \leftarrow \min(i, j); s \leftarrow A_{ij};$ for $h = 1, \ldots, m-1 : s \leftarrow s - L_{ih}U_{hj};$
   if $i \ge j$ then $L_{ij} \leftarrow s$ else { if $L_{ii} = 0$ then exit("0 pivot encountered"); $U_{ij} \leftarrow s/L_{ii}$ }}.
3. exit.

Note this procedure shows the flop-count similarity with matrix multiplication: we compute an (abbreviated) inner-product of L row i with U col j for $1 \leq i \leq n$ and $1 \leq j \leq n$, resulting in $O(n^3)$ floating-point operations.

Exercise 1.97: Show that the "optimized" Crout procedure above uses $\frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{1}{6}n$ floating-point operations, which is $2n^2$ more flops than our direct Gaussian-elimination LU-decomposition algorithm. Hint: sum the elements of the $n \times n$ matrix M defined by $M_{ij} = \min(i, j)$. Can you account for the extra $2n^2$ flops?

Exercise 1.98: Let A be an $n \times n$ orthogonal lower-triangular matrix so that $A^{-1} = A^{\mathrm{T}}$ and $A_{ij} = 0$ for $i < j$. Show that $A = I$.

For $j = 1, \ldots, n$, Crout's algorithm computes the n undetermined variables, first in U col j and then in L col j, (excluding the "pre-determined" variables U_{jj}.) A version of Crout's method including cross-row partial-pivoting and scaling is given in [PTVF92]. We shall give a similar version below. The addition of cross-row partial-pivoting makes Crout's method a viable procedure for computing the LU-decomposition of a non-singular matrix, generally without excessive contamination by round-off error.

Exercise 1.99: Show that we can modify Crout's procedure given above to overwrite the matrix A with the lower-triangle of L and the upper-triangle of U, less the main-diagonal of U. Also propose a more efficient way to achieve $diag(U) = (1, 1, \ldots, 1)$ than the assignment "$U \leftarrow I_{n \times n}$".

Exercise 1.100: Show that, for any $n \times n$ matrix A, the Gaussian-elimination LU-decomposition algorithm with no pivot search is equivalent to Crout's LU-decomposition algorithm, *i.e.*, exactly the same termination state occurs and the same output L and U matrices are computed.

We give a version of Crout's LU-decomposition algorithm with cross-row partial-pivoting below. The input is an $n \times n$ non-singular matrix A. The output is an $n \times n$ matrix B whose lower-triangle defines a non-singular lower-triangular matrix L, and whose strict upper-triangle defines a non-singular upper-triangular matrix U with $diag(U)$ understood to be $(1, 1, \ldots, 1)$. The output vector b is a transposition vector defining the $n \times n$ row-permutation matrix $P = I$ row $perm(b)^{-1}$. The obtained LU-decomposition of A is $PA = LU$.

Crout's LU-Decomposition with Cross-Row Partial-Pivoting:
input: $n \times n$ non-singular matrix A, $n \geq 1$.
output: $n \times n$ matrix B holding the lower-triangle of the $n \times n$ lower-triangular matrix L and the strict upper-triangle of the $n \times n$ upper-triangular matrix U, and the n-transposition vector b corresponding to the permutation matrix P such that $PA = LU$, if successful.

1. define $L = B$; define $U = B$.
2. $B \leftarrow A$.

3. for $j = 1, \ldots, n$:
   {for $i = 1, \ldots, n$:
   {$m \leftarrow \min(i, j)$; $s \leftarrow B_{ij}$; for $h = 1, \ldots, m-1 : s \leftarrow s - L_{ih}U_{hj}$;
   if $i \geq j$ then $\{L_{ij} \leftarrow s$; if $i = j$ or $s > v$ then $\{v \leftarrow s; x \leftarrow i\}\}$
   else $U_{ij} \leftarrow s$
   };
   if $v = 0$ then exit("A is singular");
   $b_j \leftarrow x$; if $j \neq x$ then swap B row j and B row x;
   for $i = j + 1, \ldots, n : U_{ij} \leftarrow U_{ij}/v$
   }.
4. exit.

Exercise 1.101: Explain why the above algorithm works as claimed.

Exercise 1.102: Let A be an $n \times n$ matrix and define $A_{[i]} := A$ row $1 : i$ col $1 : i$; $A_{[i]}$ is called the i-th *principal submatrix* of A. Take $A_{[0]} = [1]$. Suppose that A has an LU-decomposition computable with no pivot search, so $A = LU$ and $L_{ii} \neq 0$ for $i = 1, \ldots, n-1$. Show that $L_{ii} = det(A_{[i]})/det(A_{[i-1]})$. (This implies that the diagonal-pivot matrix A has a direct LU-decomposition if and only if $det(A_{[j]}) \neq 0$ for $j = 0, 1, \ldots, n-1$.)

Solution 1.102: We have $A = LU$ where $diag(U) = (1, \ldots, 1)$ and $L_{ii} \neq 0$ for $1 \leq i \leq n$, so

$$A = \begin{bmatrix} A_{[i]} & \sim \\ \sim & \sim \end{bmatrix} = \begin{bmatrix} L_{[i]} & 0 \\ \sim & \sim \end{bmatrix} \begin{bmatrix} U_{[i]} & \sim \\ 0 & \sim \end{bmatrix} = \begin{bmatrix} L_{[i]}U_{[i]} & \sim \\ \sim & \sim \end{bmatrix}$$

where each occurrence of the symbol '$\sim$' denotes a submatrix of the appropriate size that conforms to all the implicit constraints on its value dictated by where it appears. This means $A_{[i]} = L_{[i]}U_{[i]}$ for $1 \leq i \leq n$. Thus $A_{[i]} = \begin{bmatrix} L_{[i-1]} & 0 \\ \sim & L_{ii} \end{bmatrix} \begin{bmatrix} U_{[i-1]} & \sim \\ 0 & 1 \end{bmatrix}$ for $2 \leq i \leq n$.

Recall we have the determinant identity $det(\begin{bmatrix} X & 0 \\ \sim & Y \end{bmatrix}) = det(X)det((Y)$. Therefore $det(A_{[i]}) = det(L_{[i]})det(U_{[i]}) = \left[det(L_{[i-1]}) \cdot L_{ii}\right]\left[det(U_{[i-1]}) \cdot 1\right] = det(A_{[i-1]}) \cdot L_{ii}$ for $2 \leq i \leq n$. And $det(A_{[1]}) = A_{11} = L_{11}$ so $det(A_{[i]}) = L_{11} \cdot L_{22} \cdots L_{ii}$ for $1 \leq i \leq n$. Thus $L_{ii} = det(A_{[i]})/det(A_{[i-1]}$ where $det(A_{[0]}) = 1$.

This formula for the pivot values we obtain with no pivot search serves as a test that determines whether or not an $n \times n$ matrix A is a diagonal-pivot matrix. We compute $det(A_{[1]})$, $det(A_{[2]})$, ..., $det(A_{[n-1]})$; if all these determinant values are non-zero, A is a diagonal-pivot matrix, otherwise A is not a diagonal-pivot matrix.

Exercise 1.103: Let V_n be the $n \times n$ Vandermonde matrix $\begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ \vdots & \vdots & & & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{n-1} \end{bmatrix}$. Show that when $x_1, \ldots, x_n$ are distinct values, V_n is a diagonal-pivot matrix.

Exercise 1.104: Recall that we showed earlier that for $x_1, \ldots, x_n$ distinct, the Vandermonde matrix V_n has the unique LU-decomposition $V_n = LU$ where $L_{ij} = \pi_j(x_i)$ for $1 \leq i \leq n$ and $1 \leq j \leq n$ where $\pi_j(x)$ is the Newton polynomial $(x - x_1)(x - x_2) \cdots (x - x_{j-1})$ with $\pi_1(x) = 1$ and $\pi_j(x_i) = 0$ for $1 \leq i < j \leq n$, and $U_{ij} = h_{j-i}(x_1, \ldots, x_i)$ for $1 \leq i \leq n$ and $1 \leq j \leq n$ where, for $i \geq 0$ and $k \geq 0$, $h_i(x_1, \ldots, x_k)$ is the complete symmetric function $h_i(x_1, \ldots, x_k) = \sum_{\substack{j_1+j_2+\cdots+j_k=i \\ j_1 \geq 0, \ldots, j_k \geq 0}} x_1^{j_1} x_2^{j_2} \cdots x_k^{j_k}$. Note $h_0(x_1, \ldots, x_k) = 1$, and we define $h_i(x_1, \ldots, x_k) = 0$ for $i < 0$.

Is using this pre-established LU-decomposition always a good way to solve the linear equations $cV_n = y$ where y is a given n-vector and $c \in \mathcal{R}^n$ is the vector of coefficients of the interpolating polynomial for $(x_1, y_1), \ldots, (x_n, y_n)$ to be determined? Hint: Is the round-off error introduced while computing and using the elements of L and U in the course of back-substitution worse than the error introduced in computing the LU-decomposition of V_n using cross-row partial-pivoting? Note using cross-row partial pivoting is roughly equivalent to ordering $x_1, \ldots, x_n$ in decreasing order. Can you find efficient error-reducing ways to compute $h_{j-i}(x_1, \ldots, x_i)$ and $\pi_j(x_i)$ for $1 \leq i \leq n$?

Let A be an $n \times k$ rank r matrix with a direct LU-decomposition of the form $A = LU$ (where no permutations are involved.) Recall $rank(A) = r$ implies the submatrix L row $(r+1) : n$ col $(r+1) : k = 0$. Let $m = \min(n, k)$. The matrix A is an *extended diagonal-pivot matrix* in the sense that the LU-decomposition $A = LU$ can be computed with no pivot search, terminating when a 0 pivot value is encountered. An extension of the exercise above shows that the extended diagonal-pivot matrix A has the characterization that the principal submatrices $A_{[0]}$, $A_{[1]}$, ..., $A_{[r]}$ are all non-singular, and $A_{[r+1]}$, ..., $A_{[m]}$ are all singular.

Exercise 1.105: Let A be an $n \times k$ rank r matrix with $A_{[r]} = A$ row $1 : r$ col $1 : r$ non-singular. Let $n = \min(n, k)$. Show that $det(A \text{ row } (1 : h) \text{ col } (1 : h)) = 0$ for $r < h \leq m$. Hint: the rows of A row $(r + 1) : n$ are all linear combinations of the vectors A row 1, ..., A row r.

The LU-decomposition of the $n \times k$ rank r extended diagonal-pivot matrix A is determined by the LU-decomposition of the $r \times r$ non-singular diagonal-pivot matrix A row $1 : r$ col $1 : r = A_{[r]} =: A_{11}$. (Note that in this discussion only, L_{ij}, U_{ij}, and A_{ij} denote submatrices, not elements, of the matrices L, U, and A.)

Write $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}$ where L_{11}, U_{11}, and U_{22} are non-singular triangular matrices.

Then $L_{21} = A_{21}U_{11}^{-1}$, $U_{12} = L_{11}^{-1}A_{12}$, and L_{22} and U_{22} satisfy $L_{22}U_{22} = 0$ so we may choose $L_{22} = 0$ and $U_{22} = I_{k-r,k-r}$. We have $L_{22}U_{22} = 0$ because A is a rank r extended diagonal-pivot matrix and hence there is an $(n-r) \times r$ matrix B such that $B\,[A_{11}\ A_{12}] = [A_{21}\ A_{22}]$ (*i.e.*, $(B \text{ row } j)\,[A_{11}\ A_{12}]$ is a linear combination of the rows of $[A_{11}\ A_{12}]$,) so $BA_{11} = A_{21}$ and $BA_{12} = A_{22}$ and

$$\begin{aligned} A_{22} &= L_{21}U_{12} + L_{22}U_{22} \\ &= A_{21}U_{11}^{-1}L_{11}^{-1}A_{12} + L_{22}U_{22} \\ &= BA_{11}U_{11}^{-1}L_{11}^{-1}A_{12} + L_{22}U_{22} \\ &= BA_{11}A_{11}^{-1}A_{12} + L_{22}U_{22} \end{aligned}$$

$$\begin{aligned} &= BA_{12} + L_{22}U_{22} \\ &= A_{22} + L_{22}U_{22} \end{aligned}$$

and thus $L_{22}U_{22} = 0$. Thus the extended diagonal-pivot matrix A has an LU-decomposition computable with no pivot search, *i.e.*, once L_{11} and U_{11} are determined, the entire LU-decomposition of A is determined.

We can now see that the effect of the permutation matrices P and Q produced in our Gausssian-elimination LU-decomposition algorithm is to permute the columns *and* the rows of the $n \times k$ matrix A so that the principal submatrices $(PAQ)_{[i]}$ are all non-singular for $i = 1, \ldots, rank(A)$, and the remaining principal submatrices $(PAQ)_{[i]}$ for $i = rank(A) + 1, \ldots, \min(n, k)$ are all singular, *i.e.*, PAQ is an extended diagonal-pivot matrix.

Although any extended diagonal-pivot matrix has a direct LU-decomposition, not all matrices with direct LU-decompositions are extended diagonal-pivot matrices when we relax the forms of the factors L and U to merely be lower-triangular and upper-triangular matrices respectively without further structural conditions. Of course such a relaxed LU-decomposition is less useful than the strict form we focus on herein. Such relaxed direct LU-decompositions arise when either the first r rows or the first r columns of a rank r matrix are not linearly-independent.

Exercise 1.106: Show that $\begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}$.

Pavel Okunev and Charles Johnson [OJ97] show that an $n \times n$ matrix A has a direct (relaxed) LU-decomposition $A = LU$ if and only if $rank(A_{[h]}) + h \geq rank(A \text{ row } 1 : h \text{ col } 1 : n) + rank(A \text{ row } 1 : n \text{ col } 1 : h)$ for $h = 1, \ldots, n$. They also give the following algorithm to compute such a direct LU-decomposition.

Okunev and Johnson's Relaxed LU-Decomposition with No Pivot Search:
input: $n \times n$ Okunev matrix A, $n \geq 1$.
output: $n \times n$ lower-triangular matrix L, $n \times n$ upper-triangular matrix U, if successful.

1. $L \leftarrow O_{n \times n}$; $U \leftarrow O_{n \times n}$.
2. for $h = 1, \ldots, n$:
   [for $i = 1, \ldots, n$:
   [for $j = i, \ldots, n$:
   [if $A_{ij} \neq 0$ then $\{L \text{ col } h \leftarrow A \text{ col } j;\ U \text{ row } h \leftarrow (A \text{ row } i)/A_{ij};\ \text{ goto } \alpha\}$;
   if $A_{ji} \neq 0$ then $\{L \text{ col } h \leftarrow A \text{ col } i;\ U \text{ row } h \leftarrow (A \text{ row } j)/A_{ji};\ \text{ goto } \alpha\}$;
   ]
   ]
   α: $A \leftarrow A - (L \text{ col } h)(U \text{ row } h)$.
   ]

3. exit.

Exercise 1.107: Explain why the above algorithm works. (Recall for $n \times n$ matrices A and B, the product $C = AB$ satisfies $C = \sum_{1 \leq h \leq n} (A \text{ col } h)(B \text{ row } h)$.)

Froilán Dopico, Charles Johnson and Juan Molera [DJM06] have given a characterization of those $n \times n$ matrices that have multiple *semi-relaxed* LU-decompositions with $diag(L) = (1, \ldots, 1)$ (or equivalently, with $diag(U) = (1, \ldots, 1)$,) and shown how to generate all these LU-decompositions in terms of the elements of L (or equivalently, U.)

Exercise 1.108: Show that $\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$ has an infinite number of semi-relaxed LU-decompositions.

Solution 1.108: $\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 1-a \end{bmatrix} \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$ for $a \in \mathcal{R}$. How many semi-relaxed LU-decompositions does $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$ possess? (Note $\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{a} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & a \\ 0 & 1 \end{bmatrix}$ for $a \in \mathcal{R} - \{0\}$.)

Exercise 1.109: Let $LT_{n \times n}$ be the subset of all $n \times n$ non-singular lower-triangular matrices in the set $GL_{n \times n}(\mathcal{R}^n)$. Show that $LT_{n \times n}$ is a normal subgroup of the group $GL_{n \times n}(\mathcal{R}^n)$ of all $n \times n$ non-singular matrices on $\mathcal{R}^n$. Hint: use the QR-decomposition of an $n \times n$ non-singular matrix A^{T} to show that for any $L \in LT_{n \times n}$ there exists $\bar{L} \in LT_{n \times n}$ such that $AL = \bar{L}A$, and also show that for $A \in GL_{n \times n}(\mathcal{R}^n)$ and $L_1, L_2 \in LT_{n \times n}$, $AL_1 = AL_2$ implies $L_1 = L_2$. Can you find some subgroups of $LT_{n \times n}$?

Exercise 1.110: What is the dimension of $LT_{n \times n}$?

Solution 1.110: The set of all $n \times n$ lower-triangular matrices is an algebra of dimension $n(n+1)/2$, and hence $dim(LT_{n \times n}) = n(n+1)/2$. Note the set of all $n \times n$ strictly lower-triangular matrices with diagonal $(0, \ldots, 0)$ is an algebra of dimension $(n-1)n/2$ with the interesting property that $L^n = O_{n \times n}$ for every member L.

When an $n \times n$ matrix A is a non-singular diagonal-pivot matrix, it has an LU-decomposition $A = LU$, computable by the Gaussian-elimination LU-decomposition algorithm with no pivot search, and all the diagonal elements of the lower-triangular matrix L that serve as the successive pivot values are non-zero. This means we can "factor" the matrix L into a matrix product $\hat{L}D$ where D is the diagonal matrix $diag(L_{11}, \ldots, L_{nn})$ and $\hat{L}$ is the non-singular lower-triangular matrix specified by $\hat{L}$ col $i = (L \text{ col } i)/L_{ii}$ for $i = 1, \ldots, n$; this results in $diag(\hat{L}) = (1, \ldots, 1)$. In this situation, we have $A = \hat{L}DU$, or relabeling to reuse the symbol L to denote $\hat{L}$, we have $A = LDU$ where $diag(L)$ and $diag(U)$ are both $(1, 1, \ldots, 1)$ and D is a diagonal matrix.

Exercise 1.111: Let A be an $n \times n$ rank r extended diagonal-pivot matrix with the LU-decomposition $A = LU$. Show that A can be written as $\hat{L}DU$ where $\hat{L}$ is a lower-triangular matrix with $diag(\hat{L}) = (1, \ldots, 1)$ and $D = diag(L_{11}, \ldots, L_{r,r}, 0, \ldots, 0)$.

Now if A is a symmetric matrix as well as a non-singular diagonal-pivot matrix, then we can write $A = LDL^{\mathrm{T}}$ where L is a lower-triangular matrix with $diag(L) = (1, \ldots, 1)$ and D is a diagonal matrix with non-zero diagonal elements. We may derive this decomposition from the decomposition $A = LDU$.

When A is a symmetric non-singular diagonal-pivot matrix with $A = LDU$ as described above, we have $(U^{\mathrm{T}})^{-1}AU^{-1} = (U^{\mathrm{T}})^{-1}LD$. And $[(U^{\mathrm{T}})^{-1}AU^{-1}]^{\mathrm{T}} = (U^{\mathrm{T}})^{-1}AU^{-1}$ so $(U^{\mathrm{T}})^{-1}AU^{-1}$ is symmetric. But $(U^{\mathrm{T}})^{-1}LD$ is the product of lower-triangular matrices so $(U^{\mathrm{T}})^{-1}LD$ is itself lower-triangular as well as symmetric. This means $(U^{\mathrm{T}})^{-1}LD$ is is a diagonal matrix. But then $(U^{\mathrm{T}})^{-1}L$ is diagonal since D is diagonal.

We also have $diag((U^{\mathrm{T}})^{-1}) = (1, \ldots, 1)$ and $diag(L) = (1, \ldots, 1)$ so $(U^{\mathrm{T}})^{-1}L = I$. Therefore $L = U^{\mathrm{T}}$ and hence $A = LDL^{\mathrm{T}}$ when A is a symmetric non-singular diagonal-pivot matrix. [QED]

Now when A is a symmetric non-singular diagonal-pivot matrix with $A = LDL^{\mathrm{T}}$ where L is a lower-triangular matrix with $diag(L) = (1, \ldots, 1)$ and D is a diagonal matrix with $D_{ii} > 0$ for $i = 1, 2, \ldots, n$, then we can define $D^{\frac{1}{2}} = diag(\sqrt{D_{11}}, \ldots, \sqrt{D_{nn}})$ and write $A = LD^{\frac{1}{2}}D^{\frac{1}{2}}L^{\mathrm{T}}$. (We shall call a symmetric non-singular diagonal-pivot matrix where each pivot value is positive a *positive-definite matrix*.)

But $LD^{\frac{1}{2}}D^{\frac{1}{2}}L^{\mathrm{T}} = LD^{\frac{1}{2}}(D^{\frac{1}{2}})^{\mathrm{T}}L^{\mathrm{T}} = GG^{\mathrm{T}}$ where $G = LD^{\frac{1}{2}}$. The decomposition $A = GG^{\mathrm{T}}$ where G is a lower-triangular matrix is called the *Cholesky decomposition* of A. Since the LU-decomposition of a non-singular matrix is unique, the Cholesky decomposition of a positive-definite matrix is unique. (Note if we admit complex elements, any symmetric non-singular matrix has a Cholesky decomposition.)

When A is a positive-definite matrix (*i.e.*, a symmetric non-singular positive diagonal-pivot matrix,) with an LDL^{T} decomposition where $D_{ii} > 0$ for $i = 1, 2, \ldots, n$, the Cholesky decomposition of A can be computed with the following algorithm.

Cholesky-Decomposition:
input: $n \times n$ symmetric non-singular positive-definite matrix A, $n \geq 1$.
output: $n \times n$ lower-triangular matrix G such that $A = GG^{\mathrm{T}}$

1. $G \leftarrow O_{n \times n}$.
2. for $j = 1, \ldots, n$:
$$\{G_{jj} \leftarrow \left(A_{jj} - \sum_{1 \leq t < j} A_{jt}^2 \right)^{\frac{1}{2}} \; ; \; \text{for } i = 1, \ldots, n : G_{ij} \leftarrow \frac{1}{G_{jj}} \left[A_{ij} - \sum_{1 \leq t < j} A_{it}A_{jt} \right] \}.$$
3. exit.

Exercise 1.112: Explain why the above algorithm is correct. Hint: derive the equations from $A = GG^{\mathrm{T}}$ in the way Crout's method was obtained or see [GV89].

Exercise 1.113: Explain how to use the Cholesky decomposition of a symmetric non-singular positive diagonal-pivot matrix A to solve the equations $xA = v$.

Exercise 1.114: Compute the number of flops needed to compute the $n \times n$ Cholesky matrix G in the above algorithm assuming that computing a square-root is s times more costly than a floating-point multiply.

We should include a test in the above algorithm to ensure that round-off error does not result in trying to compute the square-root of a negative number, although for a well-conditioned matrix, this is unlikely. We should also guard against dividing by a too-small value as we discussed with respect to the Gaussian-elimination LU-decomposition algorithm. Golub and Van Loan [GV89] discuss the "failure modes" of the above algorithm and discuss the error in G that is expected to be introduced in terms of the elements of the input matrix A.

Note the QR-decomposition and the Cholesky decomposition are related. Let A be an $n \times n$ matrix with the QR-decomposition $A = QR$ where Q is an $n \times n$ orthogonal matrix and R is an $n \times n$ upper-triangular matrix. Then $A^{\mathrm{T}}A = R^{\mathrm{T}}Q^{\mathrm{T}}QR = R^{\mathrm{T}}R$; this is a form of LU-decomposition of $A^{\mathrm{T}}A$ as well as a generalized Cholesky decomposition. When A is non-singular, the Cholesky decomposition of $A^{\mathrm{T}}A$ is unique and $R^{\mathrm{T}}R$ is that Cholesky decomposition.

Exercise 1.115: Let A be an $n \times n$ non-singular matrix. Show how to use the Cholesky decomposition of the $n \times n$ matrix $A^{\mathrm{T}}A$ to compute the QR-factorization of A.

Exercise 1.116: Given the $n \times k$ rank r matrix A, suppose we have the LU-decomposition $PAQ = LU$. Describe how we can easily obtain the decomposition $A = FG^{\mathrm{T}}$ where F is an $n \times r$ rank r matrix with linearly-independent columns, and G is a $k \times r$ rank r matrix with linearly-independent columns. (Recall that this decomposition is the starting point for constructing the Moore-Penrose pseudo-inverse matrix A^{+}.)

Exercise 1.117: Show how to use the LU-decomposition $PAQ = LU$ of the $n \times k$ matrix A together with four back-substitution steps to solve the normal equations $xAA^{\mathrm{T}} = bA^{\mathrm{T}}$ where $b \in \mathcal{R}^k$ and $x \in \mathcal{R}^n$. How much does this cost in comparison to computing $B = AA^{\mathrm{T}}$ and then solving $xB = bA^{\mathrm{T}}$ by computing an LU-decomposition of B? Note AA^{T} is symmetric, so B can be computed at less cost than just multiplying A by A^{T}. When can we employ a Cholesky decomposition to solve the normal equations $xAA^{\mathrm{T}} = bA^{\mathrm{T}}$? (Recall that a vector x that satisfies $xAA^{\mathrm{T}} = bA^{\mathrm{T}}$ minimizes $|xA - b|$, and $x = bA^{+}$ is such a vector with least norm.)

Exercise 1.118: Suppose A is an $n \times n$ non-singular matrix. Explain how to use the LU-decomposition $PAQ = LU$ to compute A^{-1}. Hint: look at $A^{-1}A = I$ as n sets of linear equations: $xA = e_1, \ldots, xA = e_n$.

Exercise 1.119: Explain how to use the LU-decomposition of the $n \times n$ non-singular matrix A to efficiently compute the matrix product BA^{-1} where B is a given $k \times n$ matrix. Hint: consider solving $x_iA = B$ row i for $i = 1, \ldots, k$.

We stated above that when the $n \times n$ matrix A is non-singular, use of cross-column partial-pivoting ensures that we can write $AC_1M_1 \cdots C_rM_r = L$, where L is an $n \times n$ non-singular lower-triangular matrix (so that $L_{ii} \neq 0$ for $1 \leq i \leq n$.)

By multiplying at most $v := n(n+1)/2$ particular non-singular elementary matrices on the right of L, the non-singular matrix L can be reduced to the $n \times n$ identity matrix. These elementary matrices $H_1, \ldots, H_v$ are chosen to effect the same transformations as achieved by the following program: [for $i = n, n-1, \ldots, 2$: (for $j = 1, 2 \ldots, i-1$: $(L_{ij} \leftarrow L_{ij} - (L_{ij}/L_{ii})L_{ii})$);
for $i = 1, 2 \ldots, n$: $(L_{ii} \leftarrow L_{ii}/L_{ii})$].

Exercise 1.120: Let $1 \leq j < i \leq n$. Show that the elementrary matrix that zeros L_{ij} is $E_n[i, j, -L_{ij}/L_{ii}]$. Then show that for $1 \leq k \leq n(n-1)/2$, the elementary matrix H_k is $E_n[i, j, -L_{ij}/L_{ii}]$ where $i = p+1$ and $j = s - i(i-1)/2$ with $s = 1+n(n-1)/2-k$ and $p = \lfloor\sqrt{2s}\rfloor$. (Also, for $n(n-1)/2+1 \leq k \leq n(n+1)/2$, $H_k = E_n[r, r, L_{rr}^{-1} - 1]$ where $r = k - n(n-1)/2$.)

Thus $AC_1M_1 \cdots C_rM_rH_1 \cdots H_v = I$, so $A^{-1} = C_1M_1 \cdots C_rM_rH_1 \cdots H_v$ and
$A = H_v^{-1} \cdots H_1^{-1}\ C_r^{-1}H_v^{-1}M_r^{-1}C_r^{-1} \cdots M_1^{-1}C_1^{-1}$. Note that $H_1^{-1}, \ldots, H_v^{-1}, M_1^{-1}, \ldots, M_r^{-1}$, and C_1^{-1}, ..., C_r^{-1} are all elementary matrices or products of elementary matrices; thus A is written as a product of elementary matrices. (What is the minimum number of $n \times n$ elementary matrices that must be multiplied to guarantee that any non-singular $n \times n$ matrix can be produced?)

Exercise 1.121: Let A be an $n \times n$ non-singular matrix. Show that application of the algorithm below to the matrix A exits at step 7 and that, at exit, B is indeed A^{-1}.

1. $h \leftarrow 1$; $B \leftarrow I_{n \times n}$.
2. $a \leftarrow 0$; for $i = h, \ldots, n$: if $(A_{hi} \neq 0)$ then $\{a \leftarrow A_{hi}$; $q \leftarrow i$; goto step 3$\}$.
3. if $a = 0$ then exit("A is singular.").
4. if $(q \neq h)$ then {swap A col h and A col q in A; swap B row h and B row q in B; }.
5. A col $h \leftarrow (A$ col $h)/a$; B col $h \leftarrow (B$ col $h)/a$.
6. for $i = 1, \ldots, n$:
   if $(i \neq h)$ then $\{A$ col $i \leftarrow (A$ col $i) - A_{hi}(A$ col $i); B$ col $i \leftarrow (B$ col $i) - A_{hi}(B$ col $i)\}$.
7. if $h = n$ then exit("$B = A^{-1}$.").
8. $h \leftarrow h + 1$; go to step 2.

Exercise 1.122: Can any $n \times n$ matrix, not necessarily non-singular, be represented by a product of elementary matrices, as defined here?

Exercise 1.123: Let A be an $n \times n$ symmetric matrix. Show that there exists a non-singular matrix F such that FAF^{T} is a diagonal matrix of the form $diag(1, \ldots, 1, -1, \ldots, -1, 0, \ldots, 0)$ where there are s_1 ones, followed by s_2 minus-ones, followed by s_3 zeros, with $s_1 \geq 0$, $s_2 \geq 0$, $s_3 \geq 0$, and $s_1 + s_2 + s_3 = n$. Hint: Let B denote the matrix $C_1M_1 \cdots C_rM_r$ in the identity $AC_1M_1 \cdots C_rM_r = L$ expressing the reduction of A to lower-triangular form via cross-column partial-pivoting. Now express B as a product of elementary matrices $E_k \cdots E_2E_1$. Then define $F^{\mathrm{T}} = BS_1^{\mathrm{T}} \cdots S_n^{\mathrm{T}}$ where $S_i = E_n[i, i, |L_{ii}|^{1/2} - 1]$. The values s_1, s_2 and s_3, are symmetric-congruence invariants; that is, if A and X are congruent $n \times n$ symmetric matrices, then (s_1, s_2, s_3) for A is the same as (s_1, s_2, s_3) for X.

Exercise 1.124: Give a *recursive* version of the Gaussian-elimination LU-decomposition algorithm where the LU-decomposition $P^{\mathrm{T}}LUQ^{\mathrm{T}}$ of an $n \times k$ matrix A is computed from the similar LU-decomposition of the $(n-1) \times (k-1)$ matrix A row $2 : n$ col $2 : k$.

Neither Gaussian elimination with complete-pivoting nor Gaussian elimination with partial-pivoting together with back-substitution are guaranteed effective algorithms per se since the relative error introduced in the solution vector x that is intended to satisfy $xA = v$ can be arbitrarily large in certain "ill-conditioned" cases. (The error in a solution $\hat{x}$ to $xA = v$ is measured by the quantity $|x - \hat{x}|$ and the relative error is $|x - \hat{x}|/|x|$ when $|x| \neq 0$.) It may be more appropriate to say that Gaussian elimination with complete-pivoting or partial-pivoting is effective for well-conditioned problems.

The *condition* of an $n \times n$ system of linear equations $xA = v$ is measured by the *condition number* $\kappa(A)$ of the coefficient matrix A. The condition number depends on the norm being used, but all norms are comparable in a sense that we shall make precise below. The ∞-norm condition number of the $n \times n$ matrix A is defined as $\kappa_\infty(A) = \left(\max\limits_{1 \leq i \leq n} \sum\limits_{1 \leq j \leq n} |A_{ij}| \right) \cdot \left(\max\limits_{\leq i \leq n} \sum\limits_{1 \leq j \leq n} |(A^{-1})_{ij}| \right)$ when A is non-singular and ∞ when A is singular; $\kappa(A)$ essentially measures how close to singular the coefficient matrix A is. The problem $xA = v$ is ill-conditioned when $\kappa(A)$ is large (what 'large' means is dependent on the accuracy desired.)

One way to improve the condition of the matrix A, other than "damaging" A by adding to its diagonal elements, is to replace A by AS and v by vS, where the $k \times k$ matrix S is a diagonal matrix that scales the j-th equation $x(A \text{ col } j) = v_j$ by the constant S_{jj} to obtain the equivalent equation $x(A \text{ col } j)S_{jj} = v_j S_{jj}$. Generally we want to choose the scaling factors $S_{11}, S_{22}, \ldots, S_{kk}$ to make each equation similar in "size". For example, we could use $S_{jj} = 1/|A \text{ col } j|$, or $S_{jj} = 1/\max_{1 \leq i \leq n} |A_{ij}|$. Note this scaling could be done, either explicitly or implicitly, within the Gaussian-elimination LU-decomposition algorithm given above. If we choose the elements of the scale matrix S to be powers of 2, we will not directly introduce any additional round-off error in our LU-decomposition. We can obtain more flexiblity by scaling both rows and columns of A; given scale matrices T and S, we can use the LU-decomposition of TAS to solve $yTAS = vS$ for y and then compute $x = yT$. Again, it is convenient to take the scale matrices T and S to be diagonal matrices.

One reasonable way to define the scaling matrices T and S is as follows. Let c_i denote an index j such that $|A_{ij}| = \max_m |A_{im}|$ and let r_j denote an index i such that $|A_{ij}| = \max_m |A_{mj}|$. Then we may define $T_{ij} = S_{ij} = 0$ for $i \neq j$ and $T_{ii} = f(|A_{i,c_i}|^{-1/2})$ and $S_{jj} = f(|A_{r_j,j}|^{-1/2})$ where $f(x) = \delta_{x0} + x$.

Note that using scaling subverts the notion of using an eligible element of maximum magnitude in the current coefficient matrix as the pivot value in each iteration. However using large magnitude pivot values in early iterations might, in some circumstances, cause us to have to use excessively small magnitude elements as the pivot values in later iterations; scaling may ameliorate this potential problem. It is difficult to know in advance when scaling will be beneficial, and, in that case, what scale factors should be used. An algorithm for computing scale factors has been proposed by Olschowka and Neumaier [ON96].

Note that scaling with diagonal scale matrices T and S can be "virtually" implemented without actually scaling any of the coefficients in the matrix A; this avoids introducing round-off error due to scaling. All that such *implicit scaling* really does is change our choice of pivot elements when using a complete or partial or rook pivot search. (Although generally we only consider scaling when we are using a cross-row or cross-column pivot search.)

The fundamental computations in our Gaussian-elimination LU-decomposition algorithm are

$$L_{ij} \leftarrow L_{ij} - \frac{L_{hj}}{L_{hh}} L_{ih} \text{ and } U_{hj} \leftarrow \frac{L_{hj}}{L_{hh}}.$$

When diagonal scaling matrices T and S are used, these computations become

$$L_{ij} \leftarrow T_{\sigma_i,\sigma_i} L_{ij} S_{\rho_j,\rho_j} - \frac{T_{\sigma_h,\sigma_h} L_{hj} S_{\rho_j,\rho_j}}{T_{\sigma_h,\sigma_h} L_{hh} S_{\rho_h,\rho_h}} T_{\sigma_i,\sigma_i} L_{ih} S_{\rho_h,\rho_h} \text{ and } U_{hj} \leftarrow \frac{T_{\sigma_h,\sigma_h} L_{hj} S_{\rho_j,\rho_j}}{T_{\sigma_h,\sigma_h} L_{hh} S_{\rho_h,\rho_h}},$$

where L row i holds the elements derived from A row σ_i and L col j holds the elements derived from A col ρ_j during iteration h. Thus σ and ρ are permutation vectors that keep track of the row and column exchanges done to bring the selected pivot element to the diagonal position L_{hh} at iteration h for $h = 1, \ldots, n$.

These computations reduce to

$$L_{ij} \leftarrow T_{\sigma_i,\sigma_i} \left[L_{ij} - \frac{L_{hj}}{L_{hh}} L_{ih} \right] S_{\rho_h,\rho_h} \text{ and } U_{hj} \leftarrow \frac{L_{hj} S_{\rho_j,\rho_j}}{L_{hh} S_{\rho_h,\rho_h}}.$$

Thus L_{ij} is computed as $T_{\sigma_i,\sigma_i} S_{\rho_j,\rho_j}$ times the unscaled value that would normally be assigned to L_{ij}, and U_{hj} is computed as $(S_{\rho_h,\rho_h})^{-1} S_{\rho_j,\rho_j}$ times the unscaled value that would normally be assigned to U_{hj}.

Since this computation of L_{ij} holds in the initial iteration with $L = A$, it holds in every subsequent iteration as well. Thus we can dispense with multiplying L_{ij} by $T_{\sigma_i,\sigma_i} S_{\rho_h,\rho_h}$ and multiplying U_{hj} by $S^{-1}_{\rho_h,\rho_h} S_{\rho_j,\rho_j}$ altogether as long as we take into account this "virtual" implicit scaling when selecting the pivot element that is permuted into L_{hh} in each iteration!

All we need do to employ implicit scaling with the diagonal scale matrices T and S is to initialize the permutation vector σ to $\langle 1, 2, \ldots, n \rangle$ and initialize the permutation vector ρ to $\langle 1, 2, \ldots, k \rangle$ in step 1, and replace step 2 in our Gaussian-elimination LU-decomposition algorithm with the statements

{Determine indices $p \in \{h, \ldots, n\}$ and $q \in \{h, \ldots, k\}$ such that $|L_{pq}| = max_{\substack{h \le i \le n \\ h \le j \le k}} |T_{\sigma_i,\sigma_i} L_{ij} S_{\rho_j,\rho_j}|$;
Swap σ_p and σ_h; Swap ρ_q and ρ_h}.

This, in effect, chooses a sequence of pivot elements that sacrifices maximal magnitudes for lower variation. (Note there is no reason to maintain both the permutations σ and ρ and also the transposition vectors b and c in our LU-decomposition algorithm.)

We can justify this conclusion as follows. Let $PAQ = LU$ be the LU-decomposition of the matrix A computed using the pivot elements determined by the pivot search procedure using the scale

matrices T and S given above. We have $P = I$ row σ and $Q = I$ col ρ. Also let $PTASQ = L_sU_s$ be the LU-decomposition of the scaled matrix TAS computed with the ordinary complete-pivoting version of our Gaussian-elimination LU-decomposition algorithm. Note the permutation matrices P and Q are the same in both decompositions.

For T and S non-singular diagonal matrices, we have effectively shown that $L_s = T_\sigma L S_\rho$ and $U_s = S_\rho^{-1} U S_\rho$ where $T_\sigma = diag(T_{\sigma_1,\sigma_1}, T_{\sigma_2,\sigma_2}, \ldots, T_{\sigma_n,\sigma_n}) = PTP^{\mathrm{T}}$ and $S_\rho = diag(S_{\rho_1,\rho_1}, S_{\rho_2,\rho_2}, \ldots, S_{\rho_k,\rho_k}) = Q^{\mathrm{T}}SQ$.

Thus $PTASQ = L_sU_s = T_\sigma L S_\rho S_\rho^{-1} U S_\rho = PTP^{\mathrm{T}} L U Q^{\mathrm{T}} SQ$.

And then $TAS = TP^{\mathrm{T}} L U Q^{\mathrm{T}} S$, so we may "unscale" by multiplying on the left by T^{-1} and on the right by S^{-1} to obtain $A = P^{\mathrm{T}} L U Q^{\mathrm{T}}$, or equivalently, $PAQ = LU$.

Thus computing the LU-decomposition $PAQ = LU$ with pivot values determined by implicit scaling is equivalent to computing the LU-decomposition $PTASQ = L_sU_s$ and then "un-scaling" by computing $(PT)^{-1} L_s U_s (SQ)^{-1}$.

Exercise 1.125: Can you give a "dynamic" scaled pivot search where you strike a balance between partial-pivoting with scaling and complete-pivoting with no scaling by accepting the pivot element L_{pq} when L_{pq} is not too small, but making another choice when L_{pq} is too small?

Exercise 1.126: Since the average amount of computation for solving an $(n+1) \times (n+1)$ system of linear equations increases by $O(2n^2)$ operations from the cost of solving an $n \times n$ system of linear equations, is it the case that almost all "random" square systems of linear equations $xA = v$ will exhibit unbounded relative error in x as $n \to \infty$ when fixed finite-precision arithmetic is used?

Exercise 1.127: Let A_n be the $n \times n$ matrix defined by $(A_n)_{ij} = 1/(i+j-1)$. This matrix is called a *Hilbert* matrix [Knu73]. Let $v_n = e_1 + e_2 + \cdots + e_n$. Write a program that uses an LU-decomposition to solve $xA_n = v_n$, both with complete-pivoting and with cross-row partial-pivoting, and with and without implicit scaling.

Compare your results for $n = 2, 3, \ldots, 9$. Note you cannot represent a Hilbert matrix exactly in floating-point format and this will cause some of the difficulties in computation that you will observe. Why don't all inexactly-represented matrices cause trouble?

Exercise 1.128: Another way to compute $x \in \mathcal{R}^n$ such that $xA = v$, where A is an $n \times n$ non-singular matrix and $v \in \mathcal{R}^n$, is to use the QR-factorization, or more precisely, its transpose, the LQ^{T}-factorization. We may write $A = LQ^{\mathrm{T}}$ where here L is an $n \times n$ lower-triangular rank n matrix and Q is an $n \times n$ orthogonal matrix with $Q^{-1} = Q^{\mathrm{T}}$. Then $xLQ^{\mathrm{T}} = v$ so $xL = vQ$, and this is a triangular system of linear equations, and thus we can compute x via back-substitution.

In practice, the orthogonal matrix Q is computed as a product of Householder reflection matrices that, when multiplying A on the right, produce a lower-triangular matrix. An alternate way of computing L and Q^{T} essentially involves applying the Gram-Schmidt procedure to the rows of A, keeping each intermediate inner-product and the final set of orthonormal vectors from which we form L and Q^{T}.

Explain why this method for solving $xA = v$ is not cost-competitive with either direct Gaussian elimination or use of an LU-decomposition. Research when use of the QR-factorization approach to solving a set of linear equations is justified.

Exercise 1.129: There are a variety of other methods for solving systems of linear equations besides Gaussian elimination. (Look-up the Gauss-Seidel and conjugate gradient iterations.) An iterative method that generates a sequence of approximate solution vectors $x^{(0)}, x^{(1)}, \ldots, x^{(s)}, \ldots$ succeeds when this sequence converges to a solution x of the linear system $xA = v$. Assuming convergence occurs with exact arithmetic, can such a method be guaranteed to yield a solution whose relative error is bounded as $n \to \infty$ when implemented in fixed finite-precision arithmetic? Hint: the convergence test itself must be done with the same fixed finite-precision arithmetic.

Exercise 1.130: Since the accuracy of the solution is sometimes poor, why is Gaussian elimination with complete-pivoting or partial-pivoting often used in practice?

A version of our LU-decomposition algorithm incorporating implicit scaling and using a test for a zero pivot value involving an "approximation to zero" is given below

---

LU-Decomposition by Column-Operation Gaussian Elimination with Implicit Scaling and Complete-Pivoting:
input: $n \times k$ matrix A, $n \geq 1$, $k \geq 1$.
output: L, U overwritten in A, σ, ρ, r

1. Define $L = A$; Define $U = A$.

2. $\sigma \leftarrow \langle 1, 2, \ldots, n \rangle$; $\rho \leftarrow \langle 1, 2, \ldots, k \rangle$; $h \leftarrow 1$;
   $\alpha \leftarrow 0$; $\beta \leftarrow 0$; $z \leftarrow 1$; for $j = 1, \ldots, k : S_j \leftarrow 0$; for $i = 1, \ldots, n : T_i \leftarrow 0$.

3. $\left\{ \begin{array}{l} \text{Compute the diagonals } T[1:n] \text{ and } S[1:k] \text{ of the diagonal scale matrices, and} \\ \text{compute the "approximation to zero" bound } \alpha. \end{array} \right\}$
   for $i = 1, \ldots, n$:
   (for $j = 1, \ldots, k$:
   ($T_i \leftarrow \max(T_i, |A_{ij}|)$; $S_j \leftarrow \max(S_j, |A_{ij}|)$;
   if $A_{ij} \neq 0$ then ($z \leftarrow z + 1$; $\alpha \leftarrow \alpha + |A_{ij}|$)
   );
   if $T_i = 0$ then $T_i \leftarrow 1$ else $T_i \leftarrow T_i^{-1/2}$
   );
   for $j = 1, \ldots, k$: if $S_j = 0$ then $S_j \leftarrow 1$ else $S_j \leftarrow S_j^{-1/2}$;
   $\alpha \leftarrow u \cdot \alpha / z$.

4. Determine indices $p \in \{h, \ldots, n\}$ and $q \in \{h, \ldots, k\}$ such that $|L_{pq}| = max_{\substack{h \leq i \leq n \\ h \leq j \leq k}} |T_{\sigma_i} L_{ij} S_{\rho_j}|$.

5. $a \leftarrow L_{pq}$; if $|a| \leq \beta$ then ($r \leftarrow h - 1$; exit("pivot value near 0")).

6. If $h \neq q$ then (swap ρ_h and ρ_q; swap L col h and L col q in L);
   If $h \neq p$ then (swap σ_h and σ_p; swap L row h and L row p in L).
   { Now $L_{hh} = a$.}

7. $\left\{ \begin{array}{l} \text{Subtract multiples of } L \text{ col } h \text{ from } L \text{ col } (h+1),\ L \text{ col } (h+2),\ \ldots,\ L \text{ col } k \\ \text{to make } L \text{ row } h \text{ col } [(h+1):k] = 0. \text{ Also compute } U \text{ row } h \text{ col } [(h+1):k]. \end{array} \right\}$
   for $j = h+1, \ldots, k$:
   $(z \leftarrow L_{hj}/a;\ U_{hj} \leftarrow z;$ for $i = h+1, \ldots, n : (L_{ij} \leftarrow L_{ij} - zL_{ih}))$.

8. if $h = n$ or $h = k$ then $(r \leftarrow h$; exit);
   $h \leftarrow h+1$; $\beta \leftarrow \alpha$; go to step 4.

At exit we have computed the $n \times k$ lower-triangular matrix L and the $k \times k$ upper-triangular matrix U as specified below. We have also computed the permutation vectors σ and ρ and the computational rank r. The $n \times k$ input matrix A has been overwritten with values that determine the matrices L and U.

The $n \times k$ lower-triangular matrix L is formed as L row i col $1 : i = A$ row i col $1 : i$ for $i = 1, \ldots, r$, L row i col $1 : r = A$ row i col $1 : r$ for $i = r+1, \ldots, n$, and L row $1 : n$ col $(r+1) : k = 0$. The $k \times k$ upper-triangular matrix U is formed as $U_{ii} = 1$ for $i = 1, \ldots, k$, U row i col $(i+1) : k = A$ row i col $(i+1) : k$ for $i = 1, \ldots, r$, and U row i col $(i+1) : k = 0$ for $i = r+1, \ldots, k-1$.

The permutation matrix $P = I$ row σ and the permutation matrix $Q = I$ col ρ, together with the matrices L and U, constitute an LU-decomposition of A: $PAQ = LU$.

Exercise 1.131: Give a version of the above program with implicit scaling, but with rook-pivoting in place of complete-pivoting.

Exercise 1.132: Can you devise a version of our Gaussian-elimination LU-decomposition algorithm that "pre-determines" the pivot elements? Specifically, give an algorithm that computes the permutation vectors σ and ρ by examining the $n \times k$ input matrix A. If this can be done, we can then construct $(I$ row $\sigma)A(I$ col $\rho)$ and compute the corresponding LU-decomposition with *no* pivoting. Hint: recall rook-pivoting plus the fact that the pivot elements must lie in r distinct rows and r distinct columns when $rank(A) = r$.

There is a sometimes helpful device called *iterative improvement* [PTVF92] for improving our solution x for $xA = v$ obtained using the LU-decomposition of A. The idea is as follows. Suppose we have an approximate solution y with $y = x + d$, where d is the error in the vector y. Then $yA = (x+d)A = v + c$ where $c = dA = yA - v$. Thus we can compute the error d by solving for d in $dA = yA - v$ (and we can do this using the already-obtained LU-decomposition of A.) Then $y - d$ is an improved solution to $xA = v$, (although Golub and Van Loan [GV89] point-out that d may be predominantly noise unless arithmetic with higher precision, *e.g.*, twice the precision used in computing the LU-decomposition, is used to compute the inner-products that arise in the matrix-vector products involved in computing the residual $yA - v$. Usually this means a specially-crafted inner-product routine implementing multi-precision arithmetic needs to be written.)

Although computing one error vector is usually sufficient (and, indeed, if the vector d that we compute is very small, it need not be used at all,) we can repeat this process until the error vector is suitably small, *e.g.,* until $\max_{1\leq i\leq n}|d_i| \leq u \cdot \max_{1\leq i\leq n}|y_i|$ where the "rounding-unit" u is the smallest (positive) floating-point value such that $1 + u > 1$ in machine arithmetic.

Exercise 1.133: Is it possible for iterative improvement to diverge? That is, is it possible that $y - d$ is a worse solution to $xA = v$ than y itself is? Hint: because of round-off error, the LU-decomposition of A is generally inexact.

Exercise 1.134: The problem of solving a system of n linear equations with an $n\times n$ *tridiagonal* coefficient matrix A such that $A_{i+1,i} = a_i$ for $1 \leq i \leq n-1$, $A_{i,i+1} = c_i$ for $1 \leq i \leq n-1$, $A_{ii} = d_i$ for $1 \leq i \leq n$, and $A_{ij} = 0$ for $1 \leq i \leq n$ and $1 \leq j \leq n$ with $\mid i - j \mid \geq 2$ is commonly encountered. Here the elements of the sequences $a_1, \ldots, a_{n-1}$, $c_1, \ldots, c_{n-1}$, and $d_1, \ldots, d_n$ are given real values.

Devise a specialized algorithm to solve such a tridiagonal system $xA = v$. First assume the coefficient matrix A is "diagonally-dominant" and no pivot search is required. (The matrix A is diagonally-dominant when $|A_{ii}| > |A_{i1}|+|A_{i2}|+\cdots+|A_{in}|$ for $1 \leq i \leq n$.) Then assume nothing except that the coefficient matrix A is non-singular. Then assume nothing. Is LU-decomposition competive in any of these situations?

1.3 Error Analysis for Gaussian Elimination

When fixed finite-precision floating-point arithmetic is used to solve $xA = v$, either via the LU-decomposition and back-substitution method, or via direct Gaussian elimination, issues of accuracy arise. Our increased understanding of the sources and properties of the error in solving $xA = v$ with Gaussian elimination is one of the many accomplishments of modern numerical analysis [Grc11b].

Given the $n \times k$ matrix A and the k-vector v, solving for an n-vector x such that $xA = v$ exactly using fixed finite-precision floating-point arithmetic is almost always impossible due to the usual occurrence of round-off error. (We also have the possibility of overflow and the less-serious possibility of underflow. We will ignore these complications here.) Indeed, even if we could somehow use exact arithmetic, if the elements of the given input A and v are inexact, or are "rounded-off", (for example in order to represent them in a fixed finite-precision floating-point format,) we would be unable to compute x exactly. (For the analysis below, we will assume A is exactly represented in our floating-point format.)

For some matrices A and righthand-side vectors v, the error seen in the computed vector $\hat{x}$ that approximates the solution vector x where $xA = v$ can be severe or even catastrophic. There are various ways to exhibit the error in $\hat{x}$; one way is to look at the *residual vector* $v - \hat{x}A$, and another way is to look at the *relative error* $|x - \hat{x}|/|x|$. In this latter case x must be non-zero, and it seems (incorrectly) that to compute the relative error, x must somehow be known.) Which error measure is more relevant depends on the purpose to which the computed solution vector $\hat{x}$ is to be put. The amazing fact is that having a residual vector with a small norm does not guarantee a small relative error!

Exercise 1.135: Let A be an $n \times n$ non-singular matrix. Show that the error vector $x - \hat{x} = (v - \hat{x}A)A^{-1}$.

Upper bounds on the "length" of the residual $|v - \hat{x}A|$ and on the relative error $|x - \hat{x}|/|x|$ that depend only on the input A and v and the error introduced by round-off have been derived using various vector norms $|\cdot|$, not necessarily the usual 2-norm [vNG47] [Wil63] [GV89] [DB74] [IK66] [Mey00]. These bounds involve measures of the "length" or "size" of a *matrix*; such "matrix-size" functions are called *matrix norms* and they are defined by extending the ideas of vector norms.

Given a vector norm function $|\cdot|$ defined on $\mathcal{R}^k$, we can define an associated norm function for $n \times k$ matrices A as $\|A\| = \max\limits_{x \in \mathcal{R}^k, |x|=1} |xA^{\mathrm{T}}|$. Note we use the traditional notation $\|\cdot\|$ to denote a generic matrix norm. The value $\|A\|$ specifies the "size" of A as the maximum scale factor by which the length of a k-vector in the surface of the unit-ball $\{x \in \mathcal{R}^k \mid |x| \leq 1\}$ defined with respect to the vector norm $|\cdot|$ is changed by the application of A^{T} producing an n-vector. (Since $\{x \in \mathcal{R}^k \mid |x| = 1\}$ is closed and bounded, the value $\|A\| = \max\limits_{x \in \mathcal{R}^k, |x|=1} |xA^{\mathrm{T}}|$ is guaranteed to exist.) The matrix norm $\|\cdot\|$ defined as above in terms of the vector norm $|\cdot|$ is called the matrix norm *induced* by the vector norm $|\cdot|$. (We have some awkwardness in our definition of the matrix norm $\|\cdot\|$ induced by the vector norm $|\cdot|$ because we want to use the same names and defining formulas of common matrix norms as are found in most texts, and these are based on taking vectors to be columns rather than rows as we do here.)

When the given vector norm is specifically determined, *e.g.*, as the 1-norm $|x|_1 = \max_{1 \leq i \leq k} |x_i|$, we will generally use an identifying subscript to make this clear, and we will use the same subscript with the induced matrix norm. Without a subscript or other qualification, we are referring to *any* vector or matrix norm; this is in contrast to our usual convention for vector norms where $|x|$ denotes the 2-norm of the vector x when no identifying subscript is present. Also, we shall use the matrix norm notation $\|\cdot\|$ for k-vectors considered as members of $M_{1 \times k}$ or for their transposes considered as members of $M_{k \times 1}$.

Note when we specify a vector norm or a matrix norm, we are generally defining an entire family of norms on all the domains where the definition makes sense, *e.g.*, $\mathcal{R}^k$ for $k \geq 1$ or $M_{n \times k}$ for $n \geq 1$ and $k \geq 1$.

A matrix norm $\|\cdot\|$ on $M_{n \times k}$ satisfies the same properties as vector norms:
(1) $\|A\| \geq 0$ and $\|A\| = 0$ if and only if $A = O_{n \times k}$,
(2) $\|\alpha A\| = |\alpha| \cdot \|A\|$ for $\alpha \in \mathcal{R}$,
(3) $\|A + B\| \leq \|A\| + \|B\|$,
plus one additional property:
(4) $\|AB\| \leq \|A\| \cdot \|B\|$ whenever the matrices A and B are conformable.

Note this last property involves the *family* of matrix norms on $M_{n \times k}$ for all $n, k \in \mathcal{Z}^+$ since A and B need not have the same dimensions. This last property implies that, for $A \in M_{n \times k}$, $\|yA\| \leq \|y\| \cdot \|A\|$ for $y \in \mathcal{R}^n$ and $\|Ax^{\mathrm{T}}\| \leq \|A\| \cdot \|x^{\mathrm{T}}\|$ for $x \in \mathcal{R}^k$. Any induced matrix norm defined as $\|A\| = \max\limits_{x \in \mathcal{R}^k, |x|=1} |xA^{\mathrm{T}}|$ with respect to a vector norm $|\cdot|$ is a matrix norm satisfying (1), (2), (3) and (4) above. Since a matrix norm is also a vector norm on $M_{n \times k}$ treated as an

nk-dimensional vector space, any matrix norm $\|\cdot\|$ provides a way to measure how close two $n \times k$ matrices A and B are by computing $\|A - B\|$.

Exercise 1.136: Show that when A is an $n \times n$ non-singular matrix, $\min_{x \in \mathcal{R}^n, |x|=1} |xA^{\mathrm{T}}| = \dfrac{1}{\|A^{-1}\|}$. What does this statement mean geometrically?

We will use the matrix norm induced by the vector ∞-norm $|x|_\infty = \max_{1 \le i \le k} |x_i|$ below. This matrix norm is

$$\|A\|_\infty = \max_{x \in \mathcal{R}^k, |x|_\infty = 1} |xA^{\mathrm{T}}|_\infty = \max_{1 \le i \le n} \sum_{1 \le j \le k} |A_{ij}|.$$

The matrix norm $\|\cdot\|_\infty$ is also called the maximum absolute row-sum norm.

The complementary dual matrix norm is the matrix norm induced by the vector 1-norm:

$$\|A\|_1 = \max_{x \in \mathcal{R}^k, |x|_1 = 1} |xA^{\mathrm{T}}|_1 = \max_{1 \le j \le k} \sum_{1 \le i \le n} |A_{ij}|,$$

where $|x|_1 = |x_1| + \cdots + |x_k|$ for $x \in \mathcal{R}^k$. The matrix norm $\|\cdot\|_1$ is also called the maximum absolute column-sum norm. Note $\|A\|_\infty = \|A^{\mathrm{T}}\|_1$.

Not every matrix norm is induced from a vector norm. The *Frobenius norm* $\|A\|_F := [trace(AA^{\mathrm{T}})]^{\frac{1}{2}}$ is a matrix norm that is not induced by any vector norm.

The vector 1-norm $|\cdot|_1$ can be *extended* to the sets of matrices $M_{n \times k}$ by $|A|_1 = \sum_{1 \le i \le n} \sum_{1 \le j \le k} |A_{ij}|$. This norm is, in fact, a matrix norm since $|AB|_1 \le |A|_1 \cdot |B|_1$ whenever A and B are conformable matrices. Note this matrix norm is *not* the same as the matrix norm $\|\cdot\|_1$ *induced* by the vector norm $|\cdot|_1$. Indeed, like the Frobenius norm, the matrix norm $|\cdot|_1$ is not induced by any vector norm. (Because of the confusing and myriad relationships between vector norms, matrix norms obtained by extending vector norms, and matrix norms induced by vector norms, alternate notations are often employed to distinguish among such norms. For example, $|\cdot|$, $\|\cdot\|$, and $|||\cdot|||$ are sometimes used for the three categories of norms listed above. Moreover, sometimes the notation $|A|$ is used to indicate the matrix whose (i,j)th element is $|A_{ij}|$; we shall write $abs(A)$ instead.)

Exercise 1.137: Show that the Frobenius norm is the extension of the Euclidean norm to the sets of matrices $M_{n \times k}$, *i.e.*, $\|A\|_F = \left[\sum_{1 \le i \le n} \sum_{1 \le j \le k} |A_{ij}|^2 \right]^{1/2}$. (This is not the same as the matrix norm *induced* by the vector 2-norm.)

Exercise 1.138: Show that $\|x\|_\infty = |x|_1$ and $\|x\|_1 = |x|_\infty$ for $x \in \mathcal{R}^n$.

Exercise 1.139: Is the extension of the ∞-norm $|\cdot|_\infty$ to $M_{n \times k}$ a matrix norm?

Exercise 1.140: Show that $\|\cdot\|_\infty$ is a matrix norm. Hint: show that $\max_p \sum_h |B_{ph}| \le \sum_h \max_p |B_{ph}|$ for $B \in M_{k \times m}$.

Solution 1.140: Let A be an $n \times k$ matrix and let B be an $k \times m$ matrix. Then $\|AB\|_\infty = \max_i \sum_h |\sum_j A_{ij}B_{jh}| \leq \max_i \sum_h \sum_j |A_{ij}B_{jh}| \leq \max_i \sum_j \max_p \sum_h |A_{ij}| \cdot |B_{jh}|$
$\leq \left(\max_i \sum_j |A_{ij}|\right)\left(\max_p \sum_h |B_{ph}|\right)$. Note in particular, $\|xA\|_\infty \leq \|x\|_\infty \cdot \|A\|_\infty$ where $x \in \mathcal{R}^n$.

Exercise 1.141: Show that $\|\cdot\|_1$ is a matrix norm.

An important property of vector and matrix norms is that all norms on a given finite-dimensional vector space V are equivalent where two norms $|\cdot|_a$ and $|\cdot|_b$ are *equivalent* when there exist positive constants μ_{ab} and ν_{ab} such that $\mu_{ab} \leq \dfrac{|x|_a}{|x|_b} \leq \nu_{ab}$ for $x \in V - \{0\}$. This means we can convert inequalities involving specified norms to similar inequalities involving other norms.

We can show that any two norms $|\cdot|_a$ and $|\cdot|_b$ on a finite-dimensional vector space V are equivalent as follows. Let $S_b = \{y \in V \mid |y|_b = 1\}$ and let $\mu = \min_{y \in S_b} |y|_a$. Note $\mu > 0$ because S_b is closed and bounded away from 0.

Now if $\dfrac{x}{|x|_b} \in S_b$ then $|x|_a = |x|_b \left|\dfrac{x}{|x|_b}\right|_a \geq |x|_b \min_{y \in S_b} |y|_a = |x|_b \mu$. Thus $|x|_a \geq \mu |x|_b$.

Similarly, let $S_a = \{y \in V \mid |y|_a = 1\}$ and let $\nu = \min_{y \in S_a} |y|_b$. Note $\nu > 0$.

Now if $\dfrac{x}{|x|_a} \in S_a$ then $|x|_b = |x|_a \left|\dfrac{x}{|x|_a}\right|_b \geq |x|_a \min_{y \in S_a} |y|_b = |x|_a \nu$. Thus $|x|_b \geq \nu |x|_a$.

Therefore, $\mu |x|_b \leq |x|_a \leq \dfrac{1}{\nu}|x|_b$, and taking $\mu_{ab} = \mu$ and $\nu_{ab} = \dfrac{1}{\nu}$, we have $\mu_{ab} \leq \dfrac{|x|_a}{|x|_b} \leq \nu_{ab}$ for $x \in V - \{0\}$ where μ_{ab} and ν_{ab} are positive constants.

Exercise 1.142: Show that when the subscript a denotes the ∞-norm so that $\|A\|_a = \|A\|_\infty$ and the subscript b denotes the 1-norm so that $\|A\|_b = \|A\|_1$, then the equivalent-norm constants relating the norms $\|\cdot\|_a$ and $\|\cdot\|_b$ satisfy $\mu_{ab} = \nu_{ba}$ and $\nu_{ab} = \mu_{ba}$. For $\|\cdot\|_\infty$ and $\|\cdot\|_1$ on $M_{n\times n}$, we have $\mu_{ab} = \dfrac{1}{n}$ and $\nu_{ab} = n$. Thus $\dfrac{1}{n}\|A\|_\infty \leq \|A\|_1 \leq n\|A\|_\infty$ and $\dfrac{1}{n}\|A\|_1 \leq \|A\|_\infty \leq n\|A\|_1$ for $A \in M_{n\times n}$.

Exercise 1.143: Show that $\|A\|_1 \leq |A|_1 \leq n\|A\|_1$ for $A \in M_{n\times n}$.

Now returning to the analysis of errors in the solution of a set of linear equations, let $xA = v$, and assume we have computed an approximation $\hat{x}$ to x using LU-decomposition and back-substitution. Also suppose we have an $n \times k$ matrix E such that $\hat{x}(A + E) = v$. In other words, we look at $\hat{x}$ as the *exact* solution of a "nearby" system of linear equations, as well as an approximate solution of the given system of linear equations.

This "natural" idea can be realized, *i.e.*, such a matrix E always exists when $\hat{x} \neq 0$, (generally many error matrices exist such that $\hat{x}(A + E) = v$.) We compute $\hat{x}$ which "solves" $xA = v$ with error. Thus $\hat{x}A = v + \tilde{v}$ for some vector $\tilde{v}$. Now we introduce a matrix E such that $\hat{x}E = -\tilde{v}$.

Note this is always possible when $\hat{x} \neq 0$. Then $\hat{x}(A+E) = v$. This introduction of an error matrix is possible for *any* consistent linear system of equations $xA = v$ where the computed solution $\hat{x}$ is non-zero. (Note if $\hat{x}$ is close to 0 and v is far from 0, E may need to be large.)

Introducing an error matrix E "converts" the error in $\hat{x}$ to become error in A instead. This error matrix E is now taken to be due to the round-off error that occurs in computing the LU-decomposition of A and using back-substitution to compute $\hat{x}$.

For the purpose of error analysis, we may assume the permutation matrices P and Q that bring the pivot elements of A onto the diagonal are identity matrices since permuting rows and columns during the computation for L and U introduces *no* errors. Thus, for the purpose of error analysis, we may assume A is "pre-permuted" and we may write $LU = A$ and similarly write the LU-decomposition of A computed with fixed finite-precision arithmetic in terms of error matrices without permutation matrix factors.

The Gaussian-elimination LU-decomposition algorithm applied to the $n \times n$ non-singular matrix A generates two sequences of matrices: $L^{(1)} = A$, $L^{(2)}$, ..., $L^{(n)} = L$ and $U^{(1)} = I$, $U^{(2)}$, ..., $U^{(n)} = U$, where $L^{(h+1)}$ and $U^{(h+1)}$ are computed in iteration h. When fixed finite-precision arithmetic is used, corresponding sequences $\hat{L}^{(1)} = A$, $\hat{L}^{(2)}$, ..., $\hat{L}^{(n)} = \hat{L}$ and $\hat{U}^{(1)} = I$, $\hat{U}^{(2)}$, ..., $\hat{U}^{(n)} = \hat{U}$ are computed instead, where $\hat{L}^{(h)}$ and $\hat{U}^{(h)}$ correspond to $L^{(h)}$ and $U^{(h)}$ with round-off error included. Thus the LU-decomposition of A computed with fixed finite-precision arithmetic consists of a lower-triangular matrix $\hat{L}$ and an upper-triangular matrix $\hat{U}$ with $diag(\hat{U}) = (1, 1, \ldots, 1)$ such that $\hat{L}\hat{U} = A + \tilde{A}$ for some error matrix $\tilde{A}$ (under our assumption that A is "pre-permuted", and no pivot search is used.)

Let the error matrices $\bar{L}$ and $\bar{U}$ be defined by $L + \bar{L} = \hat{L}$ and $U + \bar{U} = \hat{U}$. Then $\tilde{A} = \bar{L}U + L\bar{U} + \bar{L}\bar{U}$. Note when L and U are uniquely determined by A, the error matrix $\tilde{A}$ is also uniquely determined by A since $\hat{L}$ and $\hat{U}$ are uniquely determined by A by their construction.

Further error is introduced when computing $\hat{x}$ by back-substitution. Specifically, we want to solve $yU = v$ and $xL = y$, and, with fixed finite-precision arithmetic, this results in $\hat{y}$ and $\hat{x}$ where $\hat{y}(\hat{U} + U') = v$ and $\hat{x}(\hat{L} + L') = \hat{y}$. The error matrices L' and U' are due to round-off error in the back-substitution computation.

Thus $\hat{x}(\hat{L}+L')(\hat{U}+U') = v$. Let $\tilde{L} = \bar{L}+L'$ and $\tilde{U} = \bar{U}+U'$. Then, in terms of these combined error matrices, we have $\hat{x}(L+\tilde{L})(U+\tilde{U}) = v$ and thus $A+E = (L+\tilde{L})(U+\tilde{U}) = LU+\tilde{L}U+L\tilde{U}+\tilde{L}\tilde{U}$, so $E = \tilde{L}U+L\tilde{U}+\tilde{L}\tilde{U}$. Altogether then, $\hat{x}$ can be taken as an exact vector that satisfies $\hat{x}(A+E) = v$.

Exercise 1.144: Show that a specific error matrix E such that $\hat{x}(A+E) = v$ can be *computed* in terms of $\hat{x}$, A, and v as $E = \dfrac{\hat{x}^{\mathrm{T}} d}{\hat{x}\hat{x}^{\mathrm{T}}}$ where $d = v - \hat{x}A$.

When we study how close the matrix $A + E$ is to A, *i.e.*, how "big" E is, we are doing a *backwards error analysis*. We are transferring our attention from the problem of estimating the error in $\hat{x}$ caused by round-off error to the problem of what nearby problem has $\hat{x}$ as its *exact* solution and how close together the original problem and the nearby problem are.

If $A + E$ is sufficiently close to A for A in some class of matrices W then we say that our LU-decomposition and back-substitution algorithm for solving systems of linear equations is *(back-*

wards) stable for inputs $(A, v) \in W \times \mathcal{R}^k$; this means that, for the problem $xA = v$, our algorithm outputs the exact solution to a quantifiably-nearby problem. This is realistically the most we can hope for in the presence of round-off error. (The stability class W depends on what we take "sufficently close" to mean.) Note, since the matrix E such that $\hat{x}(A + E) = v$ is not generally unique, we may restrict our choice of E to be minimal in some matrix norm if desired.

Exercise 1.145: Suppose W_1 is the class of matrices for which Gaussian-elimination LU-decomposition with minimal pivoting is stable (according to some fixed criterion.) And suppose W_2 is the class of matrices for which Gaussian-elimination LU-decomposition with cross-row partial pivoting is stable (using the same criterion,) and W_3 is the class of matrices for which Gaussian-elimination LU-decomposition with complete pivoting is stable (again using the same criterion.) Is it the case that $W_1 \subseteq W_2 \subseteq W_3$?

Our LU-decomposition and back-substitution algorithm often yields a small residual $v - \hat{x}A$ when $(A, v) \in W \times \mathcal{R}^k$. However having $|v - \hat{x}A|$ small does *not* guarantee that the relative error $|x - \hat{x}|/|x|$ in the computed solution $\hat{x}$ is small. For the relative error to be small, other conditions on the matrix A must hold. It is also possible to have a large residual and a small relative error.

Now let us take $k = n$ so that A is a square matrix, and let A be non-singular. Then with A non-singular, we can obtain Wilkinson's bound [Wil63] [IK66] [DB74] [GV89] [Mey00]:

$$\|E\|_\infty \leq k_2(n)u \cdot \|A\|_\infty,$$

where u is the *rounding-unit* of our floating-point number system and where $k_2(n) = (3n^3 + 2n^2)g_n$ with $g_n \leq 2^{n-1}$ when cross-column partial-pivoting is used. This bound is sharp, *i.e.*, this bound is attained for certain $n \times n$ matrices when the exact value of g_n is used as defined below. And $g_n \leq n^{\frac{1}{2}} \left[2 \cdot 3^{\frac{1}{2}} \cdots n^{\frac{1}{n-1}}\right]^{\frac{1}{2}} \leq 2n^{1/2} n^{log(n)/4}$ when complete-pivoting is used. When rook-pivoting is employed, Foster shows that $g_n \leq 1.5n^{4+3\,log(n)/4}$ which is comparible to the complete-pivoting bound [Fos97]. (Recall that the rounding-unit u is the smallest positive floating-point value such that $1 + u > 1$ in machine arithmetic. The associated "binary-precision" is $p = 1 - log_2(u)$ bits and the corresponding decimal precision is $[1 - log_2(u)]/\, log_2(10)$ digits.)

The function g_n is defined with respect to the sequence of matrices $A^{(h)} = R_h \cdots R_1 A C_1 M_1 \cdots C_h M_h + E^{(h)}$ for $h = 1, \ldots, n$ arising in the iterations of the LU-decomposition algorithm where $E^{(h)}$ is the matrix of errors introduced during iterations 1 through h by using fixed finite-precision arithmetic. Recall that for the purpose of error analysis, we may take the permutation matrices R_1, ..., R_n and C_1, ..., C_n to be identity matrices. (Note $A^{(h)} = \hat{L}^{(h)}$ where the matrices $A = \hat{L}^{(1)}, \hat{L}^{(2)}, \ldots, \hat{L}^{(n)}$ are computed in the successive iterations of the Gaussian-elimination LU-decomposition algorithm using fixed finite-precision arithmetic.) Specifically, assuming A has computational rank n, so that the LU-decomposition algorithm has n iterations, we have

$$g_n = \left[\max_{1 \leq h \leq n} \max_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} |A_{ij}^{(h)}|\right] \Big/ \left[\max_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} |A_{ij}|\right].$$

The function g_n measures the relative "growth" of elements in the sequence of matrices $A^{(1)}, A^{(2)}$, ..., $A^{(n)}$, (*i.e.*, $\hat{L}^{(1)}, \hat{L}^{(2)}$, ..., $\hat{L}^{(n)}$,) and specifically, g_n bounds the relative growth in the suc-

cessive pivot values used in the Gaussian-elimination LU-decomposition algorithm, when fixed finite-precision arithmetic is used.

The value g_n depends on the choice of pivot values; this is why the bounds for g_n are different for complete-pivoting and partial-pivoting. (We require A have computational rank n to avoid an undefined g_n multiplier value.) (Since with either cross-column partial-pivoting or complete-pivoting, $|U_{ij}| \leq 1$, it is the potential growth in the intermediate results in $L^{(h)}$ for $h = 1, 2, \ldots, n$ that we are concerned with.

Exercise 1.146: We have $|U_{ij}| \leq 1$. Do we also have $|\hat{U}_{ij}| \leq 1$, where $\hat{U}$ is the estimate of U computed with fixed finite-precision arithmetic. Hint: look at the definition of rounding with the rounding unit u: $round(2^m f) = sign(f) \cdot 2^m \cdot u \cdot \left\lfloor \frac{|f|}{u} + \frac{1}{2} \right\rfloor$ where $f = 0$ or $\frac{1}{2} \leq |f| < 1$ and $m \in \mathcal{Z}$.

(When $round(2^m f)$ is to be involved in a further computation with other rounded numbers, a better statistically-unbiased rounding formula is obtained by treating the case $\frac{|f|}{u} \bmod 1 = \frac{1}{2}$ specially. If $\frac{|f|}{u} \bmod 1 = \frac{1}{2}$, we add $\frac{1}{2}$ to $\frac{|f|}{u}$ to round up only if the integer $u \left\lfloor \frac{|f|}{u} \right\rfloor$ is odd; otherwise we add 0.)

Although sharp, Wilkinson's bound is almost always conservative. Also the bounds for the function g_n given above are generally far too large; for complete-pivoting g_n is a slow-growing function of n, and empirically, for most matrices, $g_n < 10$ for $n < 400$ even for partial-pivoting [GV89] [DB74]. (Does $g_n \to \infty$ as $n \to \infty$, either with $\|A\|_\infty < \alpha$ for α fixed or when $\|A\|_\infty$ is unconstrained and the elements of A are randomly-chosen according to some convenient (semi-realistic) distribution, when complete-pivoting is employed?) Prior to 1992 it was conjectured that $g_n \leq n$ for complete-pivoting, however, counterexamples have been provided by Edelman [Ede92]. Obtaining a tighter bound on g_n, potentially given in terms of the elements of the matrix A, is one of the outstanding problems in numerical analysis.

Note as a practical matter, we could directly compute g_n for any particular matrix A while executing the LU-decomposition algorithm.

Wilkinson's bound directly indicates how far the matrix $A + E$ is from the matrix A, and thus Wilkinson's bound allows us to define the class W of matrices for which we will deem LU-decomposition and back-substitution to be stable in terms of the values n and $\|A\|_\infty$.

With the non-singular matrix A "pre-permuted" so that A is a diagonal-pivot matrix and no pivot search is required, then starting with $L^{(1)} = A$, and using exact arithmetic, the Gaussian-elimination LU-decomposition algorithm generates the sequence of matrices $L^{(1)}$, $L^{(2)}$, ..., $L^{(n)} = L$, as well as the sequence of matrices $U^{(1)} = I_{n \times n}$, $U^{(2)}$, ..., $U^{(n)} = U$ using the successive diagonal elements $L^{(1)}_{11}$, $L^{(2)}_{22}$, ... as pivot values. The elements of U are accumulated row by row, and once entered in $U^{(h)}$ are never changed; and most importantly, $|U_{ij}| \leq 1$. The elements of the matrices $L^{(1)}$, $L^{(2)}$, ..., $L^{(n)} = L$, however, may grow large in magnitude as the LU-decomposition algorithm progresses iteration by iteration.

Wilkinson's bound arises from looking closely at the error in the output $\hat{L}$ and $\hat{U}$ of the Gaussian-elimination LU-decomposition algorithm using fixed finite-precision arithmetic. To do this we must examine the errors in the sequence of matrices $\hat{L}^{(1)}$, $\hat{L}^{(2)}$, ..., $\hat{L}^{(n)} = \hat{L}$, and in the matrix $\hat{U}^{(n)} = \hat{U}$, where $\hat{L}^{(h)}$ is the estimate of the matrix $L^{(h)}$ and $\hat{U}^{(h)}$ is the similar estimate of the matrix $U^{(h)}$ obtained with fixed finite-precision arithmetic. Recall that with exact arithmetic, we have:

$$L_{ij}^{(h+1)} = \begin{cases} 0 & \text{for } i \le h \text{ and } j > i, \\ L_{ij}^{(h)} & \text{for } j \le h \text{ and } i \ge j, \\ L_{ij}^{(h)} - U_{hj} \cdot L_{ih}^{(h)} & \text{for } j > h \text{ and } i > h, \end{cases} \quad \text{and } U_{ij} = \begin{cases} 0 & \text{for } i > j, \\ 1 & \text{for } i = j, \\ \dfrac{L_{ij}^{(i)}}{L_{ii}^{(i)}} & \text{for } i < j. \end{cases}$$

When fixed finite-precision arithmetic is used, we compute the matrices $\hat{L}$ and $\hat{U}$ rather than L and U, and we have:

$$\hat{L}_{ij}^{(h+1)} = \begin{cases} 0 & \text{for } i \le h \text{ and } j > i, \\ \hat{L}_{ij}^{(h)} & \text{for } j \le h \text{ and } i \ge j, \\ \left[\hat{L}_{ij}^{(h)} - \hat{U}_{hj} \cdot \hat{L}_{ih}^{(h)} \cdot (1 + \alpha_{ij}^{(h+1)} u)\right] (1 + \beta_{ij}^{(h+1)} u) & \text{for } j > h \text{ and } i > h, \end{cases}$$

$$\text{and } \hat{U}_{ij}^{(h+1)} = \begin{cases} 0 & \text{for } i < j, \\ 1 & \text{for } i = j, \\ \left[\dfrac{\hat{L}_{ij}^{(i)}}{\hat{L}_{jj}^{(i)}}\right] (1 + \gamma_{ij}^{(h+1)} u) & \text{for } i < j, \end{cases}$$

where u is our rounding unit and $1 + \alpha_{ij}^{(h)} u$, $1 + \beta_{ij}^{(h)} u$, and $1 + \gamma_{ij}^{(h)} u$, with $|\alpha_{ij}^{(h)}| \le 1$, $|\beta_{ij}^{(h)}| \le 1$, and $|\gamma_{ij}^{(h)}| \le 1$, are values that encode the rounding done in the operations they multiply during iteration h. Recall $\hat{U} = \hat{U}^{(n)}$.

We may convert our accounting for error in $\hat{L}_{ij}^{(h+1)}$ to additive error by writing

$$\hat{L}_{ij}^{(h+1)} = \begin{cases} 0 & \text{for } i \le h \text{ and } j > i, \\ \hat{L}_{ij}^{(h)} & \text{for } j \le h \text{ and } i \ge j, \\ \hat{L}_{ij}^{(h)} - \hat{U}_{hj} \cdot \hat{L}_{ih}^{(h)} + \epsilon_{ij}^{(h+1)} & \text{for } j > h \text{ and } i > h, \end{cases}$$

and we may show that $|\epsilon_{ij}^{(h+1)}| \le 2 g_n u \max_{i,j} |A_{ij}|$ for $i > h+1$, $j > h+1$.

By looking at the error matrix $\tilde{A} = \hat{L}\hat{U} - A$, and using the recursion relation $\hat{L}_{ij}^{(h+1)} = \hat{L}_{ij}^{(h)} - \hat{U}_{hj} \cdot \hat{L}_{ih}^{(h)} + \epsilon_{ij}^{(h+1)}$ for $j > h$ and $i > h$, we can obtain $|\tilde{A}_{ij}| = \sum_{1 \le h \le i} |\epsilon_{ij}^{(h+1)}|$ for $i < j$ and $|\tilde{A}_{ij}| = \sum_{1 \le h < j} |\epsilon_{ij}^{(h+1)}|$ for $i \ge j$. Let $\alpha = \max_{i,j} |A_{ij}|$. Then the bound $|\epsilon_{ij}^{(h+1)}| \le 2 g_n u \alpha$ for $i > h+1$, $j > h+1$ yields $|\tilde{A}_{ij}| \le \begin{cases} i 2\alpha g_n u & \text{for } i < j, \\ (j-1) 2\alpha g_n u & \text{for } i \ge j, \end{cases}$ so $\|\tilde{A}\|_\infty \le n(n-1)\alpha g_n u$.

Now we may bound the magnitudes of the elements of the error matrix T' accounting for the

round-off error in solving an $n \times n$ lower-triangular system $xT = y$ by back-substitution (*i.e.*, $\hat{x}(T + T') = y$.) This bound is $|T'_{ij}| \leq \max(2, |i - j + 1|) \cdot |T_{ij}| \cdot u$ with $nu \leq 1$. We can use this bound together with the bound on $\|\tilde{A}\|_\infty$ to obtain Wilkinson's bound: $\|E\|_\infty \leq k_2(n)u\|A\|_\infty$ where $k_2(n) = (3n^3 + 2n^2)g_n$. A full proof of Wilkinson's bound is given in [IK66].

We know from practice and experience that using a minimal pivot search results in large errors relatively often; thus we take it as accepted that LU-decomposition and back-substitution with minimal-pivoting is unstable outside a relatively-small class of inputs. For any chosen stability criteria, the stability class for LU-decomposition and back-substitution with minimal-pivoting is much smaller than the stability class for LU-decomposition and back-substitution with either partial-pivoting or complete-pivoting.

Note both LU-decomposition and back-substitution with complete-pivoting and LU-decomposition and back-substitution with partial-pivoting become less stable as $n \to \infty$. More precisely, the class of matrices for which either algorithm is stable diminishes in size as $n \to \infty$, but the class of matrices for which LU-decomposition and back-substitution with partial-pivoting is stable shrinks in size much faster than the class of matrices for which complete-pivoting is stable.

There are two relative errors we may consider: $|x - \hat{x}|/|x|$ is the *forward* relative error of $\hat{x}$ and $|x - \hat{x}|/|\hat{x}|$ is the *backward* relative error of $\hat{x}$ with respect to the vector norm $|\cdot|$. (We assume that $x \neq 0$ and $\hat{x} \neq 0$.) These relative error quantities are related.

Specifically, define δ by $\delta|x| = |x - \hat{x}|$ and define γ by $\gamma|\hat{x}| = |x - \hat{x}|$ where $|\cdot|$ is a generic vector norm. Then $\delta = |\Delta|/|x|$, and $\gamma = |\Delta|/|\hat{x}|$ where $\Delta = \hat{x} - x$. Note $|\Delta| = |-\Delta|$. Thus δ is the forward relative error of $\hat{x}$ and γ is the backward relative error of $\hat{x}$ with respect to the vector norm $|\cdot|$. Also, $\dfrac{\delta}{1+\delta} = \dfrac{|\Delta|/|x|}{1+|\Delta|/|x|} = \dfrac{|\Delta|}{|x|+|\Delta|} \leq \dfrac{|\Delta|}{|\hat{x}|} = \gamma$ since $|\hat{x}| = |x + \Delta| \geq |x| + |\Delta|$. Similarly, $\dfrac{\gamma}{1+\gamma} = \dfrac{|\Delta|/|\hat{x}|}{1+|\Delta|/|\hat{x}|} = \dfrac{|\Delta|}{|\hat{x}|+|\Delta|} \leq \dfrac{|\Delta|}{|x|} = \delta$.

Exercise 1.147: Show that $|x-\hat{x}|_1/|x|_1 = \|x-\hat{x}\|_\infty/\|x\|_\infty$ and $|x-\hat{x}|_\infty/|x|_\infty = \|x-\hat{x}\|_1/\|x\|_1$.

Now to obtain a bound on the backward relative error $|x - \hat{x}|/|\hat{x}| = |\Delta|/|\hat{x}|$ where $\Delta = \hat{x} - x$, we may proceed as follows.

Suppose $n = k$ and suppose A is non-singular. We have $xA = v$ and $\hat{x}(A + E) = v = (x + \Delta)(A + E) = xA + xE + \Delta A + \Delta E$, so $\Delta A + \hat{x}E = 0$. Thus $\Delta = -\hat{x}EA^{-1}$, so for any generic matrix norm $\|\cdot\|$, we have $\|\Delta\| \leq \|\hat{x}\| \cdot \|E\| \cdot \|A^{-1}\|$, and thus

$$\frac{\|\Delta\|}{\|\hat{x}\|} \leq \kappa(A)\frac{\|E\|}{\|A\|}$$

where $\kappa(A) = \|A\| \cdot \|A^{-1}\|$.

The value $\kappa(A)$ is called the *condition number* of the non-singular matrix A with respect to the matrix norm $\|\cdot\|$. (In some circumstances we can take $\kappa(A) = \|A\| \cdot \|A^+\|$ for arbitrary $n \times k$ matrices where A^+ is the Moore-Penrose pseudo-inverse of A.) Note the condition number κ is dependent on the matrix norm employed. When we have a specific norm ω in mind, we shall write κ_ω.

Note when we take our generic matrix norm to be $\|\cdot\|_\infty$, Wilkinson's bound yields

$$\frac{\|\Delta\|_\infty}{\|\hat{x}\|_\infty} \leq \kappa_\infty(A) k_2(n) u.$$

This inequality shows that the backward relative error in $\hat{x}$ measured with respect to the maximum-absolute-rowsum matrix norm is small when n and the condition number $\kappa_\infty(A)$ are not too large. If the ∞-matrix norm of the matrix of measurement errors in the coefficient matrix A is greater than $k_2(n)u\kappa_\infty(A)$ then we may justifiably claim that $\hat{x}$ is as accurate as possible since we start with an uncertainty equal to or greater than that produced by round-off error.

Now since $|z|_1 = \|z\|_\infty$ and $|z|_\infty = \|z\|_1$ for $z \in \mathcal{R}^n$, it is convenient to let the vector norm $|\cdot|$ denote the vector norm $|\cdot|_1$ and the matrix norm $\|\cdot\|$ denote the matrix norm $\|\cdot\|_\infty$, *or* let the vector norm $|\cdot|$ denote the vector norm $|\cdot|_\infty$ and the matrix norm $\|\cdot\|$ denote the matrix norm $\|\cdot\|_1$. Then, with this understanding,

$$\frac{|\Delta|}{|\hat{x}|} = \frac{\|\Delta\|}{\|\hat{x}\|} \leq \kappa(A) \frac{\|E\|}{\|A\|}$$

where $\kappa(A) = \|A\| \cdot \|A^{-1}\|$.

Exercise 1.148: Show that $\dfrac{|\Delta|}{|\hat{x}|} \leq \dfrac{\|v - \hat{x}A\|}{\|\hat{x}\| \cdot \|A\|}$. Hint: $\|x - \hat{x}\| \leq \|v - \hat{x}A\| \cdot \|A^{-1}\|$.

Now recall $\dfrac{|\Delta|}{|\hat{x}|} = \gamma \geq \delta/(1+\delta)$. Let $\beta = \kappa(A)\dfrac{\|E\|}{\|A\|}$. Then $\beta \geq \gamma$ and thus $\beta \geq \delta/(1+\delta)$. Thus $(1+\delta)\beta \geq \delta$ so $(\beta - 1)\delta \geq -\beta$ and $(1-\beta)\delta \leq \beta$. Now if $\beta = \kappa(A)\dfrac{\|E\|}{\|A\|} < 1$, then $\delta \leq \beta/(1-\beta)$. Thus we have the upper bound $\kappa(A)\dfrac{\|E\|}{\|A\|} \Big/ \left(1 - \kappa(A)\dfrac{\|E\|}{\|A\|}\right)$ for the forward relative error $\dfrac{|\Delta|}{|x|}$ when $\kappa(A)\dfrac{\|E\|}{\|A\|} < 1$, *i.e.*, when $1 - \kappa(A)\dfrac{\|E\|}{\|A\|} = 1 - \beta > 0$ with A non-singular, we have

$$\frac{|\Delta|}{|x|} \leq \kappa(A)\frac{\|E\|}{\|A\|} \Big/ \left(1 - \kappa(A)\frac{\|E\|}{\|A\|}\right) = \frac{\kappa(A)\|E\|}{\|A\| - \kappa(A)\|E\|} = \frac{\kappa(A)}{\|A\|/\|E\| - \kappa(A)}$$

where $|\cdot|$ and $\|\cdot\|$ are either $|\cdot|_1$ and $\|\cdot\|_\infty$, or $|\cdot|_\infty$ and $\|\cdot\|_1$, respectively. ($\beta < 1$ is equivalent to $\|A^{-1}\| \cdot \|E\| < 1$, and this implies $A + E$ is non-singular.)

As $\kappa(A)$ or $\|E\|$ or both increase so that $\kappa(A)\dfrac{\|E\|}{\|A\|}$ approaches 1, this bound approaches ∞, (and if $\kappa(A)\dfrac{\|E\|}{\|A\|} \geq 1$, $\dfrac{|\Delta|}{|x|}$ is potentially unbounded, *i.e.*, $A + E$ might be singular.)

If we fix our matrix norm to be $\|\cdot\|_\infty$, and use Wilkinson's bound and assume the maximum amount of round-off error occurs so that $\|E\|_\infty = k_2(n)u\|A\|_\infty$, then $\dfrac{\|\Delta\|_\infty}{\|x\|_\infty} \leq \kappa_\infty(A)k_2(n)u/(1 - \kappa_\infty(A)k_2(n)u)$ when $\kappa_\infty(A) < \dfrac{1}{k_2(n)u}$. This reveals that with $\mu \in (0, \infty)$, if $\kappa_\infty(A) < \dfrac{1}{k_2(n)u} \cdot \dfrac{\mu}{1+\mu}$ then $\dfrac{\|\Delta\|_\infty}{\|x\|_\infty} \leq \mu$.

Exercise 1.149: Let $\alpha = k_2(n)u$ and let $\mu \in (0, \infty)$. Show that, with $|\cdot| = |\cdot|_1$, $\delta \leq \mu$ when $\kappa_\infty(A) \leq \frac{1}{\alpha} \cdot \frac{\mu}{1+\mu}$.

A matrix A for which $\kappa(A)$ is large is called *ill-conditioned*. A system of linear equations $xA = v$ with an ill-conditioned coefficient matrix and a non-zero righthand-side vector has the property that a small change in A can produce a large change in the computed solution $\hat{x}$. An ill-conditioned set of linear equations is essentially a set of linearly-dependent or almost linearly-dependent equations. (We should make every effort to avoid having to compute a solution for $xA = v$ where A has less than full rank.) When the rows of A are "robustly" independent, the condition number $\kappa(A)$ will not be large.

If a sufficiently-small pivot value arises during execution of the LU-decomposition algorithm then the coefficient matrix A is ill-conditioned. The converse is *not* true. A matrix A may not have any overly-small or overly-large non-zero elements, and it may be that the magnitudes of all the pivot values that arise are far from zero, but still $\kappa(A)$ is large.

Exercise 1.150: Let the $n \times n$ matrix $B = \begin{bmatrix} 1 & 0 & & \cdots & 0 \\ -1 & 1 & 0 & \cdots & 0 \\ -1 & -1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -1 & -1 & \cdots & -1 & 1 \end{bmatrix}$. Show that B^{-1} is defined by $(B^{-1}))_{ij} = 0$ for $i < j$, $(B^{-1}))_{ij} = 1$ for $i = j$, and $(B^{-1}))_{ij} = 2^{i-1-j}$ for $i > j$. Then show that $\kappa_\infty(B) = \kappa_1(B) = n \cdot 2^{n-1}$.

This example shows that a nice-looking matrix like B, whose pivot values for either minimal-pivoting, cross-column partial-pivoting, or complete-pivoting, are all of similar magnitude can have a large condition number. Although computing a solution to $xB = v$ can be done exactly or almost exactly, computing a solution to $x(B + P) = v$, for P a perturbation matrix with small elements is much more problematic for large n.

In order to assess the condition of a matrix A we need to estimate $\kappa(A) = \|A\| \cdot \|A^{-1}\|$ using some matrix norm. If we had to compute A^{-1} it would be costly, and also potentially unreliable if $\kappa(A)$ were in fact large. Golub and Van Loan [GV89] discuss an $O(n^2)$ algorithm for estimating $\kappa_\infty(A)$, and with this algorithm we have a reasonable (but not guaranteed) way to assess the relative error (either forward or backward) in a computed solution $\hat{x}$ of the linear system $xA = v$ by estimating the relevant relative error upper-bound with respect to the ∞ matrix norm. (Experiments show that we may generally take $k_2(n) \leq 10$ when $n \leq 100$, however this is not guaranteed.) Another way to assess the relative error in $\hat{x}$ is to perturb A, re-solve $xA = v$, and see if the new $\hat{x}$ is close to or far from the old $\hat{x}$.

Exercise 1.151: Explain why round-off error will generally cause the computational rank of an $n \times n$ coefficient matrix to be the value n when the LU-decomposition algorithm is applied.

Since $\frac{\|\Delta\|_\infty}{\|\hat{x}\|_\infty} \leq \frac{10}{2^p}\kappa_\infty(A)$ is usually approximately true when p is the binary precision of our floating-point format, we may define 'ill-conditioned' as $\kappa_\infty(A) > 2^{p/2}$, say. In general we can

expect h-decimal-digit accuracy in $\hat{x}$ when $\kappa_\infty(A) < 2^{d-h}$ where $d = p\,log_2(10)$, and we cannot expect h-decimal-digit accuracy otherwise.

We have $xA = v$ and $\hat{x}(A+E) = v$ so the residual $d := v - \hat{x}A = v - \hat{x}(A+E) + \hat{x}E = \hat{x}E$ and thus $\|d\| \leq \|\hat{x}\| \cdot \|E\|$. For A non-singular, Wilkinson's bound yields $\|d\|_\infty \leq k_2(n)u\|A\|_\infty\|\hat{x}\|_\infty$.

Since this bound on the length of the residual depends on $\|\hat{x}\|_\infty$ as well as $\|A\|_\infty$, and does not directly depend on $\kappa_\infty(A)$, we see that the size of d is not well-correlated with the accuracy of $\hat{x}$ as measured by either the forward relative error $\dfrac{\|\Delta\|_\infty}{\|x\|_\infty}$ or the backward relative error $\dfrac{\|\Delta\|_\infty}{\|\hat{x}\|_\infty}$.

Exercise 1.152: Show that $d = -\Delta A$ where $\Delta = \hat{x} - x$ and show that $\|\Delta\| \geq \|d\|/\|A\|$.

Exercise 1.153: (Dahlquist [DB74]) Assume $xA = v$ where A is an $n \times n$ non-singular matrix and $v \neq 0$. Define e by $\hat{x}A = v + e$ where $\hat{x}$ is the approximate solution of $xA = v$ produced by the LU-decomposition and back-substitution algorithm with complete-pivoting. (Here we ascribe the result of round-off error to be equivalent to a perturbation of the righthand-side vector v.)

Let $\Delta = \hat{x} - x$. Show that $\|\Delta\| \leq \|A^{-1}\| \cdot \|e\|$. Show that $\dfrac{\|\Delta\|}{\|x\|} \leq \kappa(A)\dfrac{\|e\|}{\|v\|} = \kappa(A)\dfrac{\|v - \hat{x}A\|}{\|v\|}$.

References

[DB74] Germund Dahlquist and Åke Bjørck. *Numerical Methods.* Prentice-Hall, 1974.

[DJM06] Froilán Dopico, Charles Johnson, and Juan Molera. Multiple LU factorizations of a singular matrix. *Lin. Alg. and its Applications*, (419):24–36, 2006.

[Ede92] Alan Edelman. The complete pivoting conjecture for Gaussian elimination is false. *The Mathematica Journal*, 2:58–61, 1992.

[Fos97] Leslie V. Foster. The growth factor and efficiency of Gaussian elimination with rook pivoting. *J. Computational and Applied Math.*, 86(1):177–194, 1997.

[Grc11a] Joseph F. Grcar. How ordinary elimination became Gausssian elimination. *Historia Math.*, 38:163–218, 2011.

[Grc11b] Joseph F. Grcar. John von Neumann's analysis of Gausssian elimination and the origins of modern numerical analysis. *SIAM Review*, 53(4):607–682, 2011.

[GV89] Gene H. Golub and Charles F. Van Loan. *Matrix Computations, second edition.* John Hopkins University Press, 1989.

[IK66] Eugene Issacson and Herbert Bishop Keller. *Analysis of Numerical Methods.* Wiley, New York, 1966.

[Knu73] Donald E. Knuth. *The Art of Computer Programming: Volume 1, Fundamental Algorithms, second edition.* Addison-Wesley, New York, 1973.

[Knu97] Donald E. Knuth. *The Art of Computer Programming: Volume 2, Seminumerical Algorithms*. Addison-Wesley, New York, 1997.

[Mey00] Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2000.

[NP92] Larry Neal and George Poole. A geometric analysis of gaussian elimination II. *Linear Algebra Appl.*, 173:249–272, 1992.

[OJ97] Pavel Okunev and Charles R. Johnson. Necessary and sufficient conditions for existence of the LU factorization of an arbitrary matrix. *arxiv.org, technical report, Dept. of Mathematics*, College of William and Mary, July 1997.

[ON96] Markus Olschowka and Arnold Neumaier. A new pivoting strategy for Gaussian elimination. *Linear Algebra Appl.*, 240:131–151, 1996.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in FORTRAN - The Art of Scientific Computing (2nd. ed.)*. Cambridge Univ. Press, N.Y., 1992.

[Smi10] Jane Smiley. *The Man who invented the Computer: the Biography of John Atanasoff*. Doubleday, New York, 2010.

[vNG47] John von Neumann and Herman H. Goldstine. Numerical inverting of matrices of high order. *Bull. Amer. Math. Soc.*, 53:1021–1099, 1947.

[Wil63] James H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice-Hall, 1963.