

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

January 6, 2012

1 Gaussian Elimination

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 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, \dots, 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 $v \in \mathcal{R}^k - \text{rowspace}(A)$ 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 $xA = x_1(A \text{ row } 1) + \cdots + x_n(A \text{ row } n) \in \text{rowspace}(A) \subseteq \mathcal{R}^k$ for every $x \in \mathcal{R}^n$.

Recall that $\text{nullspace}(A) = \{x \in \mathcal{R}^n \mid xA = 0\}$ with $\dim(\text{nullspace}(A)) = n - \text{rank}(A)$. If $\text{nullspace}(A) = \{0\}$ and $v \in \text{rowspace}(A)$, then there is a 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 $\text{colspace}(A)$ regardless of the choice of v or the dimension of $\text{nullspace}(A)$.

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

Geometrically, with $\text{rank}(A) = r$ and $v \in \text{rowspace}(A)$, we have r linearly-independent hyperplanes defined by $(x, A \text{ col } j_1) = v_{j_1}, \dots, (x, A \text{ col } j_r) = v_{j_r}$ where $A \text{ col } j_1, \dots, A \text{ col } j_r$ 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 $\text{nullspace}(A)$: $vA^+ + \text{nullspace}(A)$.

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

We often wish to determine which of these cases hold for a given $n \times k$ matrix A and a given right-hand-side vector $v \in \mathcal{R}^k$, and when $v \in \text{rowspace}(A)$, we wish to compute the 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.

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:

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

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 $i = 1, \dots, n$. 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.

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

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 $L^T x^T$ and compare them.

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

Exercise 1.4: Let L be a 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.

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 $xL = e_i$.

We can approach solving the system of equations $xA = v$ as follows. For each variable x_i with $i = 1, 2, \dots, n$, select an “eligible” equation $(x, A \text{ col } j) = v_j$ with $A_{ij} \neq 0$. Solve this equation for x_i , and then use this expression for x_i to *eliminate* x_i in *all* the other equations.

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. Note this transformation replaces equation p with the sum of equation p and a multiple of equation j . (This transformation is called *full-pivoting*.) 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.

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, then we have “diagonalized” our system of equations; each equation is now either of the form “ $0 = v'_j$ ” or is of the form “ $A'_{ij}x_i = v'_j$.” 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, \dots, x_n that satisfy $xA = v$.

Exercise 1.7: 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.)

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. This triangular form can be as useful as the diagonal form, even 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. (Note, in general, neither “true” diagonal or triangular forms are obtained unless we renumber, *i.e.* permute, our equations.)

It is important to note that constructing either form may introduce excessive round-off error in the solution of the associated set of linear equations; we may use a “pivot search” to mitigate this danger. The “partial” elimination process using round-off error-reduction steps is the subject to be addressed below.

Exercise 1.8: Why do we say *two* back-substitution steps are required to solve each of a sequence of linear systems with differing right-hand-sides? (If you can answer this question, you have probably seen the LU-decomposition idea before.)

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 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.

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 $I + \alpha e_j^T e_i$ where e_j is the k -vector $(0, \dots, 0, 1, 0, \dots, 0)$ 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 $(A \text{ col } i) + \alpha(A \text{ col } j)$.

Exercise 1.9: Show that $e_j^T e_i$ is the $k \times k$ matrix each of whose elements is 0 except component $[j, i]$ which is 1.

Exercise 1.10: Show that $E_k[i, j, \alpha]^T = E_k[j, i, \alpha]$. Thus the transpose of an elementary matrix is an elementary matrix.

Exercise 1.11: A suitable conformable elementary matrix can also be used to add a multiple of a row of an $n \times m$ 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 1.12: 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 $B \text{ col } i = \alpha(A \text{ col } i)$.

Exercise 1.13: 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 1.14: 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 $B \text{ col } i = A \text{ col } j$ and $B \text{ col } j = A \text{ col } i$, where $i \neq j$. What does right-multiplication by S do if $i = j$?

Recall that $\text{transpose}_k(i, j)$ denotes the permutation $\langle 1, \dots, j, \dots, i, \dots, k \rangle$ where component $t = t$, except component $i = j$ and component $j = i$. The $k \times k$ column permutation matrix corresponding to $\text{transpose}_k(i, j)$ is the matrix $I \text{ col } \text{transpose}_k(i, j)$; 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 $\text{transpose}_n(i, j)$ is the matrix $I \text{ row } \text{transpose}_n(i, j)$; when $i \neq j$, this is the matrix $(E_n[i, j, -1]E_n[j, i, 1]E_n[i, j, -1]E_n[i, i, -2])^T$. When $i = j$, $I \text{ col } \text{transpose}_k(i, i) = E_k[i, i, 0]$ and $I \text{ row } \text{transpose}_n(i, i) = E_n[i, i, 0]$.

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

Exercise 1.15: Show that $G_k[i, w] \text{ row } i = e_i + w$.

Exercise 1.16: 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 1.17: Show that all the matrices $E_k[1, i, w_1], \dots, E_k[i-1, i, w_{i-1}], E_k[i+1, i, w_{i+1}], \dots, E_k[k, i, w_k]$ commute with one-another, but not necessarily with $E_k[i, i, w_i]$.

Exercise 1.18: Show that if $w_i = 0$, then $G_k[i, w]^{-1} = G_k[i, -w] = I - e_i^T w$.

Exercise 1.19: Let A be a $k \times m$ matrix. Show that $(G_k[i, w]^T A) \text{ row } j = (A \text{ row } j) + w_i(A \text{ row } i)$. (The transpose of a column-operation Gauss matrix is called a *row-operation Gauss matrix*.)

Exercise 1.20: Let A be a $k \times k$ matrix with $A_{ri} \neq 0$.

Let $w = \left(\frac{-A_{r1}}{A_{ri}}, \dots, \frac{-A_{r,i-1}}{A_{ri}}, \frac{1 - A_{ri}}{A_{ri}}, \frac{-A_{r,i+1}}{A_{ri}}, \dots, \frac{-A_{rk}}{A_{ri}} \right)$. What is $(AG_k[i, w]) \text{ row } r$?

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 $a = (a_1, a_2, \dots, a_k)$ with $a_h \neq 0$. Take $w = (0, \dots, 0, 0, -a_{h+1}/a_h, -a_{h+2}/a_h, \dots, -a_k/a_h)$. Then $aG_k[h, w] = (a_1, a_2, \dots, a_h, 0, \dots, 0)$. We use only this form of Gauss matrix below; thus we shall henceforth consider only *restricted Gauss matrices* $G_k[h, w]$ where $w \text{ col } (1 : h) = 0$.

Exercise 1.21: Take $s \in \{1, \dots, k\}$ and let v_1, \dots, v_s be vectors in \mathcal{R}^k such that $(v_h) \text{ col } (1 : h) = 0$ for $h = 1, \dots, s$. Let $M_h = G_k[h, v_h]$ be the indicated restricted Gauss matrix. Show that $M_1 M_2 \cdots M_s = I + \sum_{1 \leq i \leq s} e_i^T v_i$ and also show that $(M_1 M_2 \cdots M_s)^{-1} = I - \sum_{1 \leq i \leq s} e_i^T v_i$. Hint: show that $(e_i^T v_i)(e_j^T v_j) = 0$ when $i > j$ (and $(v_i) \text{ col } (1 : i) = 0$).

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 . Note 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.

Gaussian elimination can be systematized and cast in a more general form by considering an associated matrix factorization called an *LU-decomposition* [GV89]. 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 .

Let $r = \text{rank}(A)$ (Generally, in practice, r will be computed with error and will be the “computational rank” of A which is just an estimate of $\text{rank}(A)$, but, for presentation purposes, we assume exact arithmetic.) 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, \dots, R_r and $k \times k$ transposition permutation matrices C_1, \dots, C_r and $k \times k$ restricted column-operation Gauss matrices M_1, \dots, 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 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 identity $R_r \cdots R_1 A C_1 \cdots C_r = LU$ is called an *LU-decomposition* for A .

LU-Decomposition by Column-Operation Gaussian Elimination with Complete-Pivoting:

input: $n \times k$ matrix A , $n \geq 1$, $k \geq 1$.

output: $L, U, b, c, r, R_1, \dots, R_r, C_1, \dots, C_r, M_1, \dots, M_r$

1. $L \leftarrow A$; $h \leftarrow 1$; $U \leftarrow I_{k \times k}$; $b \leftarrow \langle 1, 2, \dots, n \rangle$; $c \leftarrow \langle 1, 2, \dots, k \rangle$.
2. Determine indices $p \in \{h, \dots, n\}$ and $q \in \{h, \dots, k\}$ 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. $b_h \leftarrow p$; $c_h \leftarrow q$.
 $\left[\begin{array}{l} \text{Let } u = \text{transpose}_k(h, q). \text{ Define } C_h = I \text{ col } u. \\ \text{Let } u = \text{transpose}_n(h, p). \text{ Define } R_h = I \text{ row } u. \end{array} \right]$
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. $\left\{ \begin{array}{l} \text{Subtract multiples of } L \text{ col } h \text{ from } L \text{ col } (h+1), L \text{ col } (h+2), \dots, L \text{ col } k \\ \text{to make } L \text{ row } h \text{ col } [(h+1) : k] = 0. \text{ Also compute } U \text{ row } h \text{ col } [(h+1) : k]. \end{array} \right\}$
 for $i = h+1, \dots, k$:
 $(z \leftarrow L_{hi}/a; L_{hi} \leftarrow 0; U_{hi} \leftarrow z; \text{ for } j = h+1, \dots, n : (L_{ji} \leftarrow L_{ji} - zL_{jh})).$
 $\left[\begin{array}{l} \text{Let } w \text{ col } (1 : h) = 0 \text{ and } w \text{ col } ((h+1) : k) = -[L \text{ row } h \text{ col } ((h+1) : k)]/L_{hh}. \\ \text{Define } M_h = G_k[h, w]. \end{array} \right]$
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, \dots, C_r and R_1, \dots, R_r , the restricted column-operation Gauss matrices M_1, \dots, 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 $R_r \cdots R_1 = I \text{ row } perm(b)^{-1}$, 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 $C_1 \cdots C_r = I \text{ col } perm(c)^{-1}$, where $perm(c)$ denotes the permutation corresponding to the transposition vector c . Assuming exact arithmetic, the value $r = rank(A)$.

Exercise 1.22: 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 $i = 1, \dots, k$. 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.

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 $A = P^{-1}LUQ^{-1} = P^T LUQ^T$. Then $xA = v$ implies $xP^T LUQ^T = v$ implies $xP^T LU = vQ$. Let $y = xP^T L$. Then $yU = vQ$. Since U is non-singular and upper-triangular, we can compute y with back-substitution. Now recall $y = xP^T L$. Let $z = xP^T$. 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 $y \text{ col } [(r+1) : k] \neq 0$, our

equations are inconsistent and x does not exist. Otherwise we may take $z_{r+1} = z_{r+2} = \cdots = z_n = 0$ and solve for z_1, \dots, 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 $z = xP^T$ according to the permutation matrix P to obtain $x = zP$. (In order to compute zP within z , we may use the following algorithm. [for $i = r, r-1, \dots, 1$: swap z_i with z_{b_i}]. Also, in order to compute vQ within v , we may use the algorithm: [for $i = 1, 2, \dots, r$: swap v_i with v_{c_i}].)

Exercise 1.23: 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 $R_i = I \text{ row } transpose_n(i \ b_i)$ and $C_j = I \text{ col } transpose_k(j \ c_j)$. We have $zP = z(I \text{ row } perm(b)^{-1}) = z(I \text{ col } perm(b))$ and $vQ = v(I \text{ col } perm(c)^{-1})$.

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 $P = I \text{ row } b$ and $Q = I \text{ col } c$. Then PW is computed as $W \text{ row } b$, WP is computed as $W \text{ col } b^{-1}$, VQ is computed as $V \text{ col } c$, and QV is computed as $V \text{ row } c^{-1}$, where W and V are arbitrary conformable matrices.

Exercise 1.24: How do we compute $P^T W$, WP^T , VQ^T , and $Q^T V$ 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 $P = I \text{ row } b$ and $Q = I \text{ col } c$? Hint: $P^{-1} = P^T$ and $Q^{-1} = Q^T$.

Exercise 1.25: Show that $xP^T L = vB$, and hence $zL = vB$ where $B = QU^{-1}$.

Exercise 1.26: 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 $A = e_n^T$? What is the result of the Gaussian elimination LU-decomposition algorithm when $n \geq k$? What is the result when $n < k$?

Exercise 1.27: What is computed in the Gaussian elimination LU-decomposition algorithm when the $n \times k$ matrix A is $diag(v_1, v_2, \dots, v_{\min(n,k)})$ where $v_i \in \mathcal{R}$ for $i = 1, \dots, \min(n, k)$?

Exercise 1.28: Show that if $n = k$ and $r = k$, the linear equations $xA = v$ are necessarily consistent and have a unique solution.

Exercise 1.29: The value a computed in each iteration of the Gaussian elimination LU-decomposition algorithm is 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? Hint: think about the round-off error in computing a difference of the form $\alpha - \frac{\gamma\beta}{a}$ where α , β , and γ are random values, usually of comparable magnitude. If a is very small in comparison to $\gamma\beta$, then $\frac{\gamma\beta}{a}$ will be much larger than α and the difference will suffer a large loss in precision. By “loss of precision” we mean that $\alpha - (\alpha - \frac{\gamma\beta}{a})$ computed in fixed-precision floating-point arithmetic

is very different from $\frac{\gamma\beta}{a}$, i.e., $[\alpha - (\alpha - \frac{\gamma\beta}{a})]/[\frac{\gamma\beta}{a}]$ computed in floating-point arithmetic is far from 1. (It is sums and differences of values of differing magnitudes that cause catastrophic round-off error. See [Knu97] for a discussion of ideas to minimize round-off error.

Exercise 1.30: Show that $R_t = I \text{ row transpose}_n(t, b_t)$ and $C_t = I \text{ col transpose}_k(t, c_t)$ where b and c are the transposition vectors computed in the Gaussian elimination LU-decomposition algorithm. Also show that $C_t = C_t^T = C_t^{-1}$ and $R_t = R_t^T = R_t^{-1}$.

Exercise 1.31: Let $p = \text{perm}(b)$ and let $q = \text{perm}(c)$. 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 $P = I \text{ row } p^{-1}$ and determine the column permutation matrix $Q = C_1 \cdots C_r$ such that $Q = I \text{ col } q^{-1}$ respectively. Thus the matrices P and Q need not be explicitly computed.

Solution 1.31:

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

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

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

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

Exercise 1.32: 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 $v(I \text{ col } q^{-1}) = v \text{ col } q$.

Exercise 1.33: 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 1.34: 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 1.35: Show that each of the restricted Gauss matrices M_1, \dots, M_r computed in the Gaussian elimination LU-decomposition algorithm is an upper-triangular matrix.

Note in practical application, none of the matrices $C_1, \dots, C_r, R_1, \dots, R_r$, or M_1, \dots, 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 1.36: Show that $|U_{ij}| \leq 1$ for $1 \leq i < j \leq k$.

Exercise 1.37: 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?

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 $C_i = C_i^{-1} = C_i^T$ 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 $M_h = I + e_h^T w_h$. Recall that M_1, M_2, \dots, M_r are restricted Gauss matrices. For $h = 1, \dots, r$, 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 $M_h^{-1} = I - e_h^T w_h$.

Now $C_2 M_1^{-1} C_2 = C_2 (I - e_1^T w_1) C_2 = C_2 I C_2 - (C_2 e_1^T) (w_1 C_2) = I - e_1^T (w_1 C_2)$. This follows because the row permutation matrix C_2 exchanges row 2 with row j where $j \geq 2$, so that $C_2 e_1^T = e_1^T$. 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 $w_1 C_2 \text{ col } 1 = (w_1) \text{ col } 1$; thus $(w_1) \text{ col } 1$ remains 0 and $I - e_1^T (w_1 C_2)$ remains a restricted Gauss matrix.

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

Continuing in this manner, we finally obtain

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

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

Exercise 1.38: Show that U is a $k \times k$ upper-triangular matrix, and show that $diag(U) = (1, \dots, 1)$.

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 right-side vector v that admits a solution with two back-substitution steps, and two vector permutations.

Algorithmically, the process to solve $xP^T LUQ^T = v$ 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 $y \text{ col } [(r+1) : k] \neq 0$ then ($'xA = v'$ is inconsistent; exit.)
4. $z \text{ col } [(r+1) : n] \leftarrow 0$.
5. Compute $[z \text{ col } (1 : r)]$ such that $[z \text{ col } (1 : r)][L \text{ row } (1 : r) \text{ col } (1 : r)] = [y \text{ col } (1 : r)]$ 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 $i = 1, 2, \dots, r$: swap v_i with v_{c_i} .
2. for $i = 1, 2, \dots, k$: ($y_i \leftarrow v_i$; for $j = 1, \dots, i-1$: ($y_i \leftarrow y_i - U_{ji}y_j$)).
3. If $y \text{ col } [(r+1) : k] \neq 0$ then ($'xA = v'$ is inconsistent; exit.)
4. $z \text{ col } [(r+1) : n] \leftarrow 0$.
5. for $i = r, r-1, \dots, 1$: ($z_i \leftarrow y_i$; for $j = i+1, \dots, r$: ($z_i \leftarrow z_i - L_{ji}z_j$); $z_i \leftarrow z_i/L_{ii}$).
6. $x \leftarrow z$; for $i = r, r-1, \dots, 1$: swap x_i with x_{b_i} .
7. exit.

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

Exercise 1.39: Show that we can dispense with the vector z in the algorithm above.

Exercise 1.40: Explain step 3.

Solution 1.40: When $r < k$, our k equations are linearly-dependent, if they are consistent. In this case, $L \text{ col } [(r+1) : k] = O_{n \times (k-r)}$, so the equations $zL = y$ are not satisfiable if $y \text{ col } [(r+1) : k]$ is not 0, and hence does not match the left-hand-side vector $(zL) \text{ col } [(r+1) : k]$.

Exercise 1.41: Give the algorithm for solving $Ay^T = w^T$, given the LU-decomposition of the $n \times k$ matrix A . Here $y \in \mathcal{R}^k$ and $w \in \mathcal{R}^n$.

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 to convert $\begin{bmatrix} A \\ v \end{bmatrix}$ 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 Gaussian elimination. (The vector $[w \ u]$ is equal to the vector y computed in step 2, above.) (Does this idea, in fact, save any time?)

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:

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

where $m = \min(n, k)$. If $m = k$, the outer-loop is effectively $[h = 1, \dots, m - 1]$. If $k > n$, the inner-loop: $[j = h + 1, \dots, n]$ is empty, 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:
$$\sum_{1 \leq h \leq m-1} \sum_{h+1 \leq i \leq k} \left[1 + \sum_{h+1 \leq j \leq n} 2 \right] + \delta_{mn}(k - n).$$

And thus,

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

When $n = k = m$, we have $C = \frac{2}{3}m^3 - \frac{1}{2}m^2 - \frac{1}{6}m$.

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 1.42: 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 1.42: Let $m = \min(n, k)$. Then the maximum number of L -reads is:

$$\sum_{1 \leq h \leq m} [(n-h+1)(k-h+1) + 1 + 2n + 2k + (k-h)(1 + 2(n-h))] = m^3 + \frac{3}{2}(k-n+2)m^2 + 3[(n+2)(k+\frac{1}{2}) - \frac{1}{6}k]m.$$

Exercise 1.43: How should we modify the Gaussian elimination LU-decomposition algorithm to apply to complex matrices and vectors?

Exercise 1.44: 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 $A = LQ^T$ where here L is an $n \times n$ lower-triangular rank n matrix and Q^T is an $n \times n$ orthogonal matrix with $Q^{-1} = Q^T$. Then $xLQ^T = v$ so $xL = vQ$, and this is a triangular system of linear equations, and thus we can compute x via back-substitution.

Recall that 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 competitive with either direct Gaussian elimination or use of the LU -decomposition.

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} . Because of round-off error and the possibility of computing too-large or too-small values, 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. Moreover the test “ $a = 0$ ” in step 3 might usefully be replaced by “ $|a| < \epsilon$ ”, where ϵ is a small value near the smallest normalized positive value of the machine. Note that the errors that arise in the various values of a due to round-off error means that the computed value of the rank r may be incorrect.

In some cases we want to enforce a requirement that the $n \times k$ matrix A be of full-rank: $\text{rank}(A) = \min(n, k)$. 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 $|a| < \epsilon$ then ($a \leftarrow a + 2 \cdot mv$)”, where mv is a small positive value [PTVF92]. Suitable choices for mv are the value ϵ or $\epsilon + \min_{A_{ij} \neq 0} |A_{ij}|/100$. (What is a suitable choice for ϵ ?) This device of forcing A to have full-rank is similar to the related device of adding suitable constants to each diagonal element of an $n \times n$ matrix to force it to be non-singular, and to improve its “condition” for processing by the Gaussian elimination LU-decomposition algorithm. (Another way to improve the “condition” of the matrix A 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 $S_{11}, S_{22}, \dots, S_{kk}$ 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 within the Gaussian elimination LU-decomposition algorithm given above.)

The second issue has to do with efficiency. 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 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 scaling this column to introduce the value 1 as the diagonal position component, and finally, subtracting multiples of the scaled column from other columns to “zero-out” the row or “tail” of the row of that diagonal component.

The non-zero diagonal element (denoted by a in the Gaussian elimination algorithm) that we scale to one is called the *pivot element*, and the process of “zeroing-out” the pivot-element row off-diagonal elements is called *pivoting* (specifically *tail-pivoting*.) so the Gaussian elimination LU-decomposition algorithm consists of $\min(n, k)$ tail-pivoting operations. (In general, in contrast to a tail-pivoting step, a single *full-pivoting* 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 obtain close to the least practicable error in the resulting solution vector or vectors computed based on the lower-triangular matrix L . However, complete pivot searching is time-consuming. Nevertheless, if we want to guarantee the exact form of the matrix L specified above in the case of a singular matrix, we need to use complete-pivoting or, at least, search for a non-zero element.

Exercise 1.45: Can we really guarantee we will get an (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 1.45: It depends on what the meaning of ‘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 $p \in \{h, \dots, n\}$ such that $|L_{ph}| = \max_{h \leq i \leq n} |L_{ih}|$.”, and then take $a = L_{ph}$ in step 3. Much experience has shown that 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 generally succeed in stably computing the LU-decomposition.

Exercise 1.46: Show that when cross-row partial-pivoting is used, the permutation matrices C_1, \dots, C_r will all be the $k \times k$ identity matrix I .

We could also replace the pivot-value search in step 2 with: “Determine the index $q \in \{h, \dots, k\}$ such that $|L_{hq}| = \max_{h \leq i \leq k} |L_{hi}|$.”, and then take $a = L_{hq}$ in step 3. In this case, the permutation

matrices R_1, \dots, 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 $1 \leq i \leq n$. Note, in this case, we are, in essence, changing the basis of $\text{rowspace}(A)$ by multiplying by the change-of-coordinates matrix $C_1M_1 \cdots C_rM_r$ to obtain a “lower-triangular” basis for $\text{rowspace}(A)$.

Exercise 1.47: Why must A be non-singular to ensure that $AC_1M_1 \cdots C_rM_r = L$ is obtained with cross-column partial-pivoting?

Exercise 1.48: 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 1.49: 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 1.49: 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 $O(\min(n, k)^3)$.

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 $i = 1, \dots, k$, 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:

$[U(i, j) := \text{if } (i > j) \text{ return}(0) \text{ else if } (i = j) \text{ return}(1) \text{ else if } (i > n) \text{ return}(0) \text{ else return}(L_{ij})]$.
(This algorithm assumes $1 \leq i \leq k$ and $1 \leq j \leq k$.)

Exercise 1.50: Explain how we can store an $n \times n$ lower-triangular matrix, L , in $n(n+1)/2$ locations: $\text{mem}[0 : n(n+1)/2 - 1]$ so that we can access the matrix element L_{ij} with the program
[if $(i < j)$ return(0); $k \leftarrow \frac{(i-1)i}{2} + j - 1$; return($\text{mem}[k]$).] Give the corresponding access function for an $n \times n$ upper-triangular matrix stored in $n(n+1)/2 - 1$ locations.

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- and upper-triangular matrices L and U of the forms produced by the Gaussian elimination LU-decomposition algorithm

such that $A = P^T L U Q^T$ 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 any 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 $\text{rank}(A) = r$. 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 $\text{diag}(U) = (1, \dots, 1)$.) We have nk non-linear equations relating these $rn - (r - 1)r/2 + (k - 1)k/2$ values. Assuming these nk equations are independent, the $rn - (r - 1)r/2 + (k - 1)k/2$ values defining the matrices L and U are uniquely determined when $2rn - (r - 1)r \leq 2kn - (k - 1)k$. And since $r \leq \min(n, k)$, this is always the case when $n = k$.

Exercise 1.51: Show that for any fixed sequence of pivot elements determining the permutation matrices P and Q , the LU-decomposition $PAQ = LU$, with $\text{diag}(L) = (1, 1, \dots, 1)$, is unique when the matrix A is non-singular. Hint: assume $PAQ = L_1 U_1$ and also $PAQ = L_2 U_2$ and deduce that $L_1 = L_2$ and $U_1 = U_2$.

Exercise 1.52: Suppose we have two LU-decompositions: $P_1 A Q_1 = L_1 U_1$ and also $P_2 A Q_2 = L_2 U_2$. Show that $P_2 P_1^T L_1 U_1 Q_1^T Q_2 = L_2 U_2$. Explain what this means about the uniqueness of the LU-decomposition of A .

Suppose $\text{rank}(A) = r$. Due to round-off error, it can be unlikely that we will see $L_{jj} = 0$ for $j = r + 1, \dots, \min(n, k)$. The best we can do is to estimate the rank of A by treating small absolute values of $\text{diag}(L)$ as zero. What “small” should be is a complicated matter. A choice independent of the elements of the matrix A might be the “precision” p of our computer (p is the largest value such that $1 = 1 + p$ in floating-point.)

Exercise 1.53: Can we guarantee that $L_{jj} \neq 0$ for $j = 1, \dots, \text{rank}(A)$ when the Gaussian elimination LU-decomposition algorithm is executed with fixed-precision floating-point arithmetic?

Let A be an $n \times k$ rank r matrix, and consider the LU-decomposition $PAQ = LU$. We can transform this relation to write $Q^T A^T P^T = U^T L^T$. Note L^T is upper-triangular and U^T is lower-triangular with $\text{diag}(U^T) = (1, \dots, 1)$. 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 $\text{diag}(U^T) = (1, \dots, 1)$. Let $\hat{U} = L^T$ 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 $\hat{L} = U^T$ 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 $\text{diag}(\hat{L}) = (1, \dots, 1)$. Now by examining a block-multiplication, we can see that $U^T L^T = \hat{L} \hat{U}$.

Exercise 1.54: Write-out the block-multiplication which verifies that $U^T L^T = \hat{L} \hat{U}$ in the case where $r \leq n \leq k$.

We have $\hat{U} = \begin{bmatrix} J^T & K^T \\ 0 & 0 \end{bmatrix}$ 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 $\hat{U} = \begin{bmatrix} L^T \\ 0 \end{bmatrix}$; we have appended $n-k$ zero-rows to L^T to form \hat{U} . Define the $k \times n$ lower-triangular matrix $\hat{L} = \begin{bmatrix} U^T & 0 \end{bmatrix}$; we have appended $n-k$ zero-columns to U^T to form \hat{L} . Again, by examining a block-multiplication, we can see that $U^T L^T = \hat{L}\hat{U}$. 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 $\hat{L}\hat{U} = L^T U^T$ 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 1.55: Write-out the block-multiplication which verifies that $U^T L^T = \hat{L}\hat{U}$ 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 $\text{diag}(\hat{U}) = (1, \dots, 1)$. 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 $D = \text{diag}(\hat{U})$. Now define $\tilde{L} = \hat{L}D$ and define $\tilde{U} = D^{-1}\hat{U}$. This makes $\text{diag}(\tilde{U}) = (1, \dots, 1)$ and makes $\text{diag}(\tilde{L} \text{ row } 1 : r) = \text{diag}(D \text{ row } 1 : r)$. 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 $\tilde{L}\tilde{U} = \hat{L}D D^{-1}\hat{U} = \hat{L}\hat{U}$. Thus, if $PAQ = LU$ is an LU-decomposition of A , then $Q^T A^T P^T = \tilde{L}\tilde{U}$ is an LU-decomposition of A^T .

As a practical matter, for the purpose of solving 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 triangular matrices. When this is the case, we can solve the equations $xA = b$ or $Ax^T = b^T$ by two back-substitution computations together with the associated permutations specified by the matrices P and Q , in the situation where A is non-singular.

Note the Gaussian elimination 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 concomitantly construct a lower-triangular matrix of pivot-element-scaled values with an implicit diagonal of ones. This version of the Gaussian elimination 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 $\text{diag}(L) = (1, \dots, 1)$.

Exercise 1.56: Write the row-operation version of the Gaussian elimination LU-decomposition algorithm.

Solution 1.56:

LU-Decomposition by Row-Operation Gaussian Elimination with Complete-Pivoting:

input: $n \times k$ matrix A , $n \geq 1$, $k \geq 1$.

output: $L, U, b, c, r, R_1, \dots, R_r, C_1, \dots, C_r, M_1, \dots, M_r$

1. $U \leftarrow A$; $h \leftarrow 1$; $L \leftarrow I_{n \times n}$; $b \leftarrow \langle 1, 2, \dots, n \rangle$; $c \leftarrow \langle 1, 2, \dots, k \rangle$.
 2. Determine indices $p \in \{h, \dots, n\}$ and $q \in \{h, \dots, k\}$ 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. $b_h \leftarrow p$; $c_h \leftarrow q$.

$$\left[\begin{array}{l} \text{Let } u = \text{transpose}_k(h, q). \text{ Define } C_h = I \text{ col } u. \\ \text{Let } u = \text{transpose}_n(h, p). \text{ Define } R_h = I \text{ row } u. \end{array} \right]$$
 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. $\left\{ \begin{array}{l} \text{Subtract multiples of } U \text{ row } h \text{ from } U \text{ row } (h+1), U \text{ row } (h+2), \dots, U \text{ row } n \\ \text{to make } U \text{ col } h \text{ row } [(h+1) : n] = 0. \text{ Also compute } L \text{ col } h \text{ row } [(h+1) : n]. \end{array} \right\}$
 for $i = h+1, \dots, n$:
 $(z \leftarrow U_{ih}/a; U_{ih} \leftarrow 0; L_{ih} \leftarrow z; \text{ for } j = h+1, \dots, k : (U_{ij} \leftarrow U_{ij} - zU_{hj}))$.

$$\left[\begin{array}{l} \text{Let } w \text{ col } (1 : h) = 0 \text{ and } w \text{ col } ((h+1) : n) = -[U \text{ col } h \text{ row } ((h+1) : n)]/U_{hh}. \\ \text{Define } M_h = G_n[h, w]^T. \end{array} \right]$$
 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, \dots, C_r and R_1, \dots, R_r , the row-operation Gauss matrices M_1, \dots, 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, \dots, R_r are $n \times n$ transposition permutation matrices, the matrices C_1, \dots, C_r are $k \times k$ transposition permutation matrices, and the matrices M_1, \dots, 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 $\text{diag}(L) = (1, \dots, 1)$ 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 1.57: 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 $M_r, \dots, M_1, R_r, \dots, R_1$, and C_1, \dots, C_r are produced by the LU-decomposition row-operation Gaussian elimination 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 $\text{diag}(L) = (1, \dots, 1)$.

Exercise 1.58: Give the algorithm for solving the linear system $xA = v$, given the row-operation LU-decomposition of the matrix A .

Exercise 1.59: 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 1.60: Let $PAQ = LU$ be an LU-decomposition of the $n \times k$ matrix A computed by the column-operation version of the Gaussian elimination algorithm. Show that, if we apply the row-operation version of the Gaussian elimination 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 $U^T L^T$ which we considered above.

Exercise 1.61: Let $G = \left(\prod_{\substack{1 \leq i \leq n \\ i \neq r}} E_n[r, i, -A_{ic}] \right) E_n[r, r, -1 + \frac{1}{A_{rc}}]$, where A is an $n \times n$ matrix with $A_{rc} \neq 0$. What is the matrix product GA ?

Exercise 1.62: 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 $A = FG^T$ 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 1.63: 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: $xA = e_1, \dots, xA = e_n$.

Exercise 1.64: 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 $x_i A = B$ row i for $i = 1, \dots, k$.

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 $AC_1 M_1 \cdots C_r M_r = L$, 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, \dots, H_v are chosen to effect the same transformations as achieved by the following program: [for $i = n, n-1, \dots, 2$: (for $j = 1, 2, \dots, i-1$: ($L_{ij} \leftarrow L_{ij} - (L_{ij}/L_{ii})L_{ii}$)); for $i = 1, 2, \dots, n$: ($L_{ii} \leftarrow L_{ii}/L_{ii}$)].

Exercise 1.65: Let $1 \leq j < i \leq n$. Show that the elementary 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 $H_1^{-1}, \dots, H_v^{-1}, M_1^{-1}, \dots, M_r^{-1}$, and $C_1^{-1}, \dots, C_r^{-1}$ 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 elementary matrices that must be multiplied to guarantee that any non-singular matrix can be produced?)

Exercise 1.66: 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 $i = h, \dots, n$: 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 $i = 1, \dots, n$:
if ($i \neq h$) then $\{A$ col $i \leftarrow (A$ col $i) - A_{hi}(A$ col $h)$; B col $i \leftarrow (B$ col $i) - A_{hi}(B$ col $h)$ }.
7. if $h = n$ then exit(" $B = A^{-1}$.").
8. $h \leftarrow h + 1$; go to step 2.

Exercise 1.67: Can any $n \times n$ matrix, not necessarily non-singular, be represented by a product of elementary matrices, as defined here?

Exercise 1.68: 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 $\text{diag}(1, \dots, 1, -1, \dots, -1, 0, \dots, 0)$ 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 $F^T = BS_1^T \cdots S_n^T$ 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 .

There is an often 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 generally still approximate when ordinary floating-point arithmetic is used.)

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 $\max_{1 \leq i \leq n} |d_i| \leq p \cdot \max_{1 \leq i \leq n} |y_i|$ where the “precision” p is the largest floating-point value such that $1 + p = 1$ in machine arithmetic. (On a 64-bit IEEE floating-point machine, $p = 2^{-53}$.)

Exercise 1.69: 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: the LU-decomposition of A is generally inexact.

References

- [GV89] Gene H. Golub and Charles F. Van Loan. *Matrix Computations, second edition*. John Hopkins University Press, 1989.
- [Knu97] Donald E. Knuth. *The Art of Computer Programming: Volume 2, Seminumerical Algorithms*. Addison-Wesley, New York, 1997.
- [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.